Semantics and Verification of Software

Summary summer semester 2021

Merlin Denker

Contents

1 Introduction

1.1 Aspects of programming languages

- **Syntax**: *"How does a program look like?"* (Lecture *Compiler Construction*)
  - hierarchical composition of programs from structual components
- **Semantics**: *"What does a program mean?"* (**This lecture**)
  - output/behaviour/... in dependence of input/environment/...
- **Pragmatics**: *"Is the programming language practically usable?"* (Lecture *Software Engineering*)
  - length and understandability of programs,
    learnability of programming language,
    appropriateness for specific applications

1.2 Kinds of formal semantics

- Operational semantics
  - Describes **computation** of the program on some abstract machine
  - Example:

$$\text{(seq)}\ \frac{\langle c_1, \sigma \rangle \to \sigma' \qquad \langle c_2, \sigma' \rangle \to \sigma''}{\langle c_1; c_2, \sigma \rangle \to \sigma''}$$

  - Application: **Implementation** of programming languages (compilers, interpreters, ...)
- Denotional semantics
  - Mathematical definition of **input/output relation** of the program by induction on its syntactic structure
  - Example: $\mathfrak{C}[\![.]\!] : \mathsf{Cmd} \to (\Sigma \to \Sigma) : \mathfrak{C}[\![c_1; c_2]\!] := \mathfrak{C}[\![c_2]\!] \circ \mathfrak{C}[\![c_1]\!]$
  - Application: Program **analysis**; often used as reference semantics
- Axiomatic semantics
  - Formalisation of special properties of programs by **logical formulae** (assertions / proof rules)
  - Example:

$$\text{(seq)}\ \frac{\{A\}c_1\{C\} \qquad \{C\}c_2\{B\}}{\{A\}c_1; c_2\{B\}}$$

  - Application: Program **verification**

1.3 The imperative model language WHILE

WHILE is a simple imperative programming language without procedures or advanced data structures.

1.3.1 Syntactic categories

Category	Domain	Meta variable
Numbers	$\mathbb{Z} = \{0, 1, -1, ...\}$	z
Truth values	$\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$	t
Variables	$\mathsf{Var} = \{x, y, ...\}$	x
Arithemtic expressions	AExp	a
Boolean expressions	BExp	b
Commands (statements)	Cmd	c

1.3.2 Syntax of WHILE

Definition 1.2 (Syntax of WHILE)

The **syntax of WHILE Programs** is defined by the following context-free grammar:

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

We assume that

- the syntax of numbers, truth values and variables is predefined (i.e., no ”lexical analysis”)
- the syntactic interpretation of ambiguous constructs (expressions) is uniquely determined (by brackets or priorities)

2 Operational Semantics of WHILE

2.1 Idea

We define the meaning of programs by specifying its behaviour being executed on an (abstract) machine. Here this is done by defining an **evaluation/execution relation** for program fragments (expressions, commands).

We employ **derivation rules** of the form

$$(\text{Name})\ \frac{\text{Premise(s)}}{\text{Conclusion}}\ [\text{side conditions}]$$

Meaning: If every premise [and all side conditions] are fulilled, then the conclusion can be drawn. A rule with no premises is called an **axiom**.

2.2 Program States

Definition 2.1 (Program state)

A **(program) state** is an element of the set

$$\Sigma := \{\sigma \mid \sigma : \mathsf{Var} \to \mathbb{Z}\}$$

called the **space state**.

Thus $\sigma(x)$ denotes the value of $x \in \mathsf{Var}$ in state $\sigma \in \Sigma$.

2.3 Evaluation of Arithmetic Expressions

Definition 2.2 (Evaluation relation for arithmetic expressions)

If $a \in \mathsf{AExp}$ and $\sigma \in \Sigma$, then $\langle a, \sigma \rangle$ is called a **configuration**.
Expression a **evaluates to** $z \in \mathbb{Z}$ in state σ (notation: $\langle a, \sigma \rangle \to z$) if this relationshop is derivable by means of the following rules:

Axioms

$$\text{(const)}\ \frac{}{\langle z, \sigma \rangle \to z}$$

$$\text{(var)}\ \frac{}{\langle x, \sigma \rangle \to \sigma(x)}$$

Rules

$$\text{(plus)}\ \frac{\langle a_1, \sigma \rangle \to z_1 \qquad \langle a_2, \sigma \rangle \to z_2}{\langle a_1 + a_2, \sigma \rangle \to z} \text{ where } z := z_1 + z_2$$

$$\text{(minus)}\ \frac{\langle a_1, \sigma \rangle \to z_1 \qquad \langle a_2, \sigma \rangle \to z_2}{\langle a_1 - a_2, \sigma \rangle \to z} \text{ where } z := z_1 - z_2$$

$$\text{(times)}\ \frac{\langle a_1, \sigma \rangle \to z_1 \qquad \langle a_2, \sigma \rangle \to z_2}{\langle a_1 * a_2, \sigma \rangle \to z} \text{ where } z := z_1 \cdot z_2$$

2.3.1 Determinism of arithmetic evaluation relation

Lemma 3.5(1) (Determinism of arithmetic evaluation relation)

For every $a \in \mathsf{AExp}$, $\sigma \in \Sigma$, and $z, z' \in \mathbb{Z}$:

$\langle a, \sigma \rangle \to z$ and $\langle a, \sigma \rangle \to z'$ implies $z = z'$

2.4 Free Variables

Definition 2.4 (Free variables)

The set of **free variables** of an expression is given by the function

$$\mathsf{FV} : \mathsf{AExp} \to 2^{\mathsf{Var}}$$

where

$$\mathsf{FV}(z) := \varnothing \qquad \mathsf{FV}(a_1 - a_2) := \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2)$$
$$\mathsf{FV}(x) := \{x\} \qquad \mathsf{FV}(a_1 * a_2) := \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2)$$
$$\mathsf{FV}(a_1 + a_2) := \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2)$$

TODO: Are there definitions for Free Variables of boolean expressions or commands?

2.5 Evaluation of Boolean Expressions

Definition 2.6 ((Strict) evaluation relation for Boolean Expressions)

For $b \in \mathsf{BExp}$, $\sigma \in \Sigma$ and $t \in \mathbb{B}$, the **evaluation relation** $\langle b, \sigma \rangle \to t$ is defined by:

Axioms

$$\frac{}{\langle t, \sigma \rangle \to t}$$

$$\frac{\langle a_1, \sigma \rangle \to z \qquad \langle a_2, \sigma \rangle \to z}{\langle a_1 = a_2, \sigma \rangle \to \mathsf{true}}$$

$$\frac{\langle a_1, \sigma \rangle \to z_1 \qquad \langle a_2, \sigma \rangle \to z_2}{\langle a_1 = a_2, \sigma \rangle \to \mathsf{false}} \text{ if } z_1 \neq z_2$$

$$\frac{\langle a_1, \sigma \rangle \to z_1 \qquad \langle a_2, \sigma \rangle \to z_2}{\langle a_1 > a_2, \sigma \rangle \to \mathsf{true}} \text{ if } z_1 > z_2$$

$$\frac{\langle a_1, \sigma \rangle \to z_1 \qquad \langle a_2, \sigma \rangle \to z_2}{\langle a_1 > a_2, \sigma \rangle \to \mathsf{false}} \text{ if } z_1 \leqslant z_2$$

Rules

$$\frac{\langle b, \sigma \rangle \to \mathsf{false}}{\langle \neg b, \sigma \rangle \to \mathsf{true}}$$

$$\frac{\langle b, \sigma \rangle \to \mathsf{true}}{\langle \neg b, \sigma \rangle \to \mathsf{false}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{true} \qquad \langle b_2, \sigma \rangle \to \mathsf{true}}{\langle b_1 \wedge b_2, \sigma \rangle \to \mathsf{true}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{true} \qquad \langle b_2, \sigma \rangle \to \mathsf{false}}{\langle b_1 \wedge b_2, \sigma \rangle \to \mathsf{false}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{false} \qquad \langle b_2, \sigma \rangle \to \mathsf{true}}{\langle b_1 \wedge b_2, \sigma \rangle \to \mathsf{false}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{false} \qquad \langle b_2, \sigma \rangle \to \mathsf{false}}{\langle b_1 \wedge b_2, \sigma \rangle \to \mathsf{false}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{true} \qquad \langle b_2, \sigma \rangle \to \mathsf{true}}{\langle b_1 \vee b_2, \sigma \rangle \to \mathsf{true}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{true} \qquad \langle b_2, \sigma \rangle \to \mathsf{false}}{\langle b_1 \vee b_2, \sigma \rangle \to \mathsf{true}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{false} \qquad \langle b_2, \sigma \rangle \to \mathsf{true}}{\langle b_1 \vee b_2, \sigma \rangle \to \mathsf{true}}$$

$$\frac{\langle b_1, \sigma \rangle \to \mathsf{false} \qquad \langle b_2, \sigma \rangle \to \mathsf{false}}{\langle b_1 \vee b_2, \sigma \rangle \to \mathsf{false}}$$

2.5.1 Determinism of boolean evaluation relation

Lemma 3.5(2) (Determinism of boolean evaluation relation)

For every $b \in \mathsf{BExp}$, $\sigma \in \Sigma$, and $t, t' \in \mathbb{B}$:

$$\langle b, \sigma \rangle \to t \text{ and } \langle b, \sigma \rangle \to t' \text{ implies } t = t'$$

2.6 Execution of Commands

The effect of a command is the **modification of a program state**.

Definition 3.2 (Execution relation for commands)

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

Axioms

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

Rules

$$\text{(seq)}\ \frac{\langle c_1, \sigma \rangle \to \sigma' \qquad \langle c_2, \sigma' \rangle \to \sigma''}{\langle c_1; c_2, \sigma \rangle \to \sigma''} \qquad \text{(if-f)}\ \frac{\langle b, \sigma \rangle \to \mathsf{false} \qquad \langle c_2, \sigma \rangle \to \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}$$

$$\text{(if-t)}\ \frac{\langle b, \sigma \rangle \to \mathsf{true} \qquad \langle c_1, \sigma \rangle \to \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'} \qquad \text{(wh-f)}\ \frac{\langle b, \sigma \rangle \to \mathsf{false}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma \rangle \to \sigma}$$

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

2.6.1 Non-Terminating Executions

Corollary 3.4

The execution relation for commands is **not total**, i.e. there exist $c \in \mathsf{Cmd}$ and $\sigma \in \Sigma$ such that $\langle c, \sigma \rangle \to \sigma'$ for no $\sigma' \in \Sigma$.

Example: $c = \mathsf{while\ true\ do\ skip\ end}$ (with arbitrary initial state $\sigma \in \Sigma$).
Proof by contradiction: assume there ex. $\sigma' \in \Sigma$ such that $\langle c, \sigma \rangle \to \sigma'$.
Then there must exist a **finite** derivation tree s for $\langle c, \sigma \rangle \to \sigma'$.
As $c = \mathsf{while\ true\ do}\ \ldots\ \mathsf{end}$ and $\langle \mathsf{true}, \sigma \rangle \to \mathsf{true}$ by Definition, s must be of the form

$$\frac{\dfrac{}{\langle \mathsf{true}, \sigma \rangle \to \mathsf{true}} \qquad \text{(skip)}\ \dfrac{}{\langle \mathsf{skip}, \sigma \rangle \to \sigma} \qquad \text{(wh-t)}\ \dfrac{s'}{\langle \mathsf{while\ true\ do\ skip\ end}, \sigma \rangle \to \sigma'}}{\langle \mathsf{while\ true\ do\ skip\ end}, \sigma \rangle \to \sigma'}$$

for some derivation tree s', which clearly contradicts the finiteness of s.

2.6.2 Determinism of execution relation

Theorem 4.1 (Determinism of execution relation)

The execution relation for commands is **deterministic**, i.e. whenever $c \in \mathsf{Cmd}$ and $\sigma, \sigma', \sigma'' \in \Sigma$ such that $\langle c, \sigma \rangle \to \sigma'$ and $\langle c, \sigma \rangle \to \sigma''$ then $\sigma' = \sigma''$.

Proof of Theorem 4.1:

We show $\sigma' = \sigma''$ by induction on the structure of the derivation tree for $\langle c, \sigma \rangle \to \sigma'$.

- **Induction base:**
  - Case $\text{(skip)}\ \dfrac{}{\langle \mathsf{skip}, \sigma \rangle \to \sigma}$ (i.e. $c = \mathsf{skip}$ and $\sigma' = \sigma$)

  Since this axiom is the only applicable rule, it follows that $\sigma'' = \sigma = \sigma'$.

- **Induction step:**
  - Case $\text{(asgn)}\ \dfrac{\langle a, \sigma \rangle \to z}{\langle x := a, \sigma \rangle \to \sigma[x \mapsto z]}$ (i.e. $c = (x := a)$ and $\sigma' = \sigma[x \mapsto z]$):

    Here the derivation for $\langle c, \sigma \rangle \to \sigma''$ must be of the form

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

    such that Lemma 3.5(1) (p. 7) implies $z' = z$ and therefore

    $$\sigma'' = \sigma[x \mapsto z'] = \sigma[x \mapsto z] = \sigma'$$

  - Case $\text{(seq)}\ \dfrac{\langle c_1, \sigma \rangle \to \sigma_1 \qquad \langle c_2, \sigma_1 \rangle \to \sigma'}{\langle c_1; c_2, \sigma \rangle \to \sigma'}$ (i.e. $c = c_1; c_2$):

    Here the derivation for $\langle c, \sigma \rangle \to \sigma''$ must be of the form

    $$\text{(seq)}\ \frac{\langle c_1, \sigma \rangle \to \sigma_2 \qquad \langle c_2, \sigma_2 \rangle \to \sigma''}{\langle c_1; c_2, \sigma \rangle \to \sigma''}$$

    such that the induction hypotheses for $\langle c_1, \sigma \rangle$ and $\langle c_2, \sigma_1 \rangle$ respectively yield $\sigma_2 = \sigma_1$ and then $\sigma'' = \sigma'$.

  - Case $\text{(if-t)}\ \dfrac{\langle b, \sigma \rangle \to \mathsf{true} \qquad \langle c_1, \sigma \rangle \to \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}$ (i.e. $c = \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}$):

    Here the derivation for $\langle c, \sigma \rangle \to \sigma''$ must be of the form

    $$\frac{\langle b, \sigma \rangle \to t \qquad \langle c_i, \sigma \rangle \to \sigma''}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma''}$$

    where $t \in \mathbb{B}$ and $i = 1/2$ for $t = \mathsf{true}/\mathsf{false}$. Now Lemma 3.5(2) (p. 9) yields $t = \mathsf{true}$ and thus $i = 1$, and therefore the induction hypothesis for $\langle c_1, \sigma \rangle$ implies $\sigma'' = \sigma'$.

- Case (if-f) $\dfrac{\langle b,\sigma\rangle \to \mathsf{false} \qquad \langle c_1,\sigma\rangle \to \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma\rangle \to \sigma'}$ is analogous to the previous case (if-t)
- Case (wh-f) $\dfrac{\langle b,\sigma\rangle \to \mathsf{false}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma\rangle \to \sigma}$ (i.e. $c = \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}$ and $\sigma' = \sigma$):

  In the derivation for $\langle c,\sigma\rangle \to \sigma''$, only one of the two while rules can be used, which both first evaluate $\langle b,\sigma\rangle$. According to Lemma 3.5(2) (p. 9), the result must again be false, meaning that rule (wh-f) is the only applicable. Hence $\sigma'' = \sigma = \sigma'$.
- Case (wh-t) $\dfrac{\langle b,\sigma\rangle \to \mathsf{true} \qquad \langle c_0,\sigma\rangle \to \sigma_1 \qquad \langle \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end}, \sigma_1\rangle \to \sigma'}{\langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma\rangle \to \sigma'}$
  (i.e. $c = \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end}$ and $\sigma' = \sigma$):
  As before, the derivation for $\langle c,\sigma\rangle \to \sigma''$ must be of the same form:

  $$\text{(wh-t)}\ \frac{\langle b,\sigma\rangle \to \mathsf{true} \qquad \langle c_0,\sigma\rangle \to \sigma_2 \qquad \langle \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end}, \sigma_2\rangle \to \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma\rangle \to \sigma''}$$

  Now the induction hypothesis for $\langle c_0,\sigma\rangle$ yields $\sigma_2 = \sigma_1$, and applying it once more to $\langle \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end}, \sigma_1\rangle$ we obtain $\sigma'' = \sigma'$.

2.7 Proof by structural induction

Given: an inductive set, i.e. a set S whose elements are either

- atomic or
- obtained from atomic elements by (finite) application of certain operations

To show: property $P(s)$ applies to every $s \in S$

Proof: we verify:

- **Induction base:** $P(s)$ holds for every atomic element s
- **Induction hypothesis:** assume that $P(s_1)$, $P(s_2)$ etc.
- **Induction step:** then $P(f(s_1, ..., s_n))$ holds for every operation f of arity n

Structural induction is a special case of **well-founded induction**.

Generalisation: **complete** (**strong**, **course-of-values**) induction

2.7.1 Structural induction on arithemtic expressions

Definition: AExp is the least set which

- contains all integers $z \in \mathbb{Z}$ and all variables $x \in \mathsf{Var}$ and
- contains $a_1 + a_2$, $a_1 - a_2$ and $a_1 * a_2$ whenever $a_1, a_2 \in \mathsf{AExp}$

Proof that property P holds for every $a \in \mathsf{AExp}$:

- **Induction base:** $P(z)$ and $P(x)$ holds (for every $z \in \mathbb{Z}$ and $x \in \mathsf{Var}$)
- **Induction hypothesis:** $P(a_1)$ and $P(a_2)$ holds
- **Induction step:** $P(a_1 + a_2)$, $P(a_1 - a_2)$ and $P(a_1 * a_2)$ holds

2.7.2 Structural induction on boolean expressions

Definition: BExp is the least set which

- contains the truth values $t \in \mathbb{B}$ and, for every $a_1, a_2 \in \mathsf{AExp}$, $a_1 = a_2$ and $a_1 > a_2$, and
- contains $\neg b_1$, $b_1 \wedge b_2$ and $b_1 \vee b_2$ whenever $b_1, b_2 \in \mathsf{BExp}$

Proof that property P holds for every $b \in \mathsf{BExp}$:

- **Induction base:** $P(t)$, $P(a_1 = a_2)$ and $P(a_1 > a_2)$ holds (for every $t \in \mathbb{B}$, $a_1, a_2 \in \mathsf{AExp}$)
- **Induction hypothesis:** $P(b_1)$ and $P(b_2)$ holds
- **Induction step:** $P(\neg b_1)$, $P(b_1 \wedge b_2)$ and $P(b_1 \vee b_2)$ holds

2.7.3 Structural induction on WHILE commands

Definition: Cmd is the least set which

- contains skip and, for every $x \in \mathsf{Var}$ and $a \in \mathsf{AExp}$, $x := a$, and
- contains $c_1; c_2$, $\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}$ and $\mathsf{while}\ b\ \mathsf{do}\ c_1\ \mathsf{end}$ whenever $b \in \mathsf{BExp}$ and $c_1, c_2 \in \mathsf{Cmd}$

Proof that property P holds for every $c \in \mathsf{Cmd}$:

- **Induction base:** $P(\mathsf{skip})$ and $P(x := a)$ holds (for every $x \in \mathsf{Var}$ and $a \in \mathsf{AExp}$)
- **Induction hypothesis:** $P(c_1)$ and $P(c_2)$ holds
- **Induction step:** $P(c_1; c_2)$, $P(\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end})$ and $P(\mathsf{while}\ b\ \mathsf{do}\ c_1\ \mathsf{end})$ holds (for every $b \in \mathsf{BExp}$)

2.7.4 Structural induction on derivation trees of the execution relation

Proof that property P holds for every derivation tree s of an expression:

- **Induction base:** $P(\frac{}{\langle \mathsf{skip}, \sigma \rangle \to \sigma})$ holds for every $\sigma \in \Sigma$, and $P(s)$ holds for every derivation tree s of an expression.
- **Induction hypothesis:** $P(s_1)$, $P(s_2)$ and $P(s_3)$ hold
- **Induction step:** it also holds that
  - $P(\ (\text{asgn})\ \frac{s_1}{\langle x := a, \sigma \rangle \to \sigma[x \mapsto z]}\ )$
  - $P(\ (\text{seq})\ \frac{s_1 \quad s_2}{\langle c_1; c_2, \sigma \rangle \to \sigma''}\ )$
  - $P(\ (\text{if-t})\ \frac{s_1 \quad s_2}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}\ )$
  - $P(\ (\text{if-f})\ \frac{s_1 \quad s_2}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}\ )$
  - $P(\ (\text{wh-t})\ \frac{s_1 \quad s_2 \quad s_3}{\langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma \rangle \to \sigma''}\ )$
  - $P(\ (\text{wh-f})\ \frac{s_1}{\langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma \rangle \to \sigma}\ )$

2.7.5 Well-founded Induction

Definition Ex1Task4 (well-foundedness)

A binary relation $<\subseteq S \times S$ is **well-founded** if every non-empty subset $X \subseteq S$ has a minimal element with respect to $<$.

Lemma Ex1Task4 (well-founded induction)

Given a well-founded relation $<\subseteq S \times S$ and a Property P. Then the principle of **well-founded induction** states:
In order to show that $P(s)$ holds for all elements $s \in S$, it suffices to prove for all $s \in S$ that $P(s)$ holds under the assumption that $P(s')$ holds for all $s' < s$.

2.8 Functional of the Operational Semantics

Definition 4.2 (Operational functional)

The **functional of the operational semantics**

$$\mathfrak{O}[\![.]\!] : \mathsf{Cmd} \to (\Sigma \to \Sigma)$$

assigns to every command $c \in \mathsf{Cmd}$ a **partial state transformation** $\mathfrak{O}[\![c]\!] : \Sigma \to \Sigma$, which is defined as follows:

$$\mathfrak{O}[\![c]\!]\sigma := \begin{cases} \sigma' & \text{if } \langle c, \sigma \rangle \to \sigma' \text{ for some } \sigma' \in \Sigma \\ \text{undefined} & \text{otherwise} \end{cases}$$

$\mathfrak{O}[\![c]\!]\sigma$ can indeed be undefined (consider e.g. $c =$ while true do skip end).

2.8.1 Operational equivalence

Definition 4.3 (Operational Equivalence)

Two commands $c_1, c_2 \in \mathsf{Cmd}$ are called **(operationally) equivalent** (notation: $c_1 \sim c_2$) iff

$$\mathfrak{O}[\![c_1]\!] = \mathfrak{O}[\![c_2]\!]$$

Thus:

- $c_1 \sim c_2$ iff $\mathfrak{O}[\![c_1]\!]\sigma = \mathfrak{O}[\![c_2]\!]\sigma$ for every $\sigma \in \Sigma$
- In particular, $\mathfrak{O}[\![c_1]\!]\sigma$ is undefined iff $\mathfrak{O}[\![c_2]\!]\sigma$ is undefined

2.8.2 Example: Unwinding of loops

Simple application of command equivalence: The test of the execution condition in a while loop can be represented by an if command.

Lemma 4.4

For every $b \in \mathsf{BExp}$ and $c \in \mathsf{Cmd}$:

$$\text{while } b \text{ do } c \text{ end} \sim \text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ end else skip end}$$

This can be proven via operational equivalence.

Let $c_1 :=$ while b do c end and $c_2 :=$ if b then $c; c_1$ else skip end. We show the mutual inclusion of the function graphs of $\mathfrak{O}[\![c_1]\!]$ and $\mathfrak{O}[\![c_2]\!]$.

First, let $\mathfrak{O}[\![c_1]\!] = \sigma'$, i.e. $\langle c_1, \sigma\rangle \to \sigma'$. Two definitions are possible:

- **(wh-t)** Here the derivation tree is of the form

$$\text{(wh-t)}\ \frac{\langle b, \sigma\rangle \to \mathsf{true} \qquad \langle c, \sigma\rangle \to \sigma'' \qquad \langle c_1, \sigma''\rangle \to \sigma'}{\langle c_1, \sigma\rangle \to \sigma'}$$

  This implies that also the following derivation tree is valid:

$$\text{(if-t)}\ \frac{\langle b, \sigma\rangle \to \mathsf{true} \qquad \text{(seq)}\ \dfrac{\langle c, \sigma\rangle \to \sigma'' \qquad \langle c_1, \sigma''\rangle \to \sigma'}{\langle c; c_1, \sigma\rangle \to \sigma'}}{\langle c_2, \sigma\rangle \to \sigma'}$$

  implying that also $\mathfrak{O}[\![c_2]\!]\sigma = \sigma'$

- **(wh-f)** Here we have

$$\text{(wh-f)}\ \frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle c_1, \sigma\rangle \to \sigma}$$

  and hence $\sigma' = \sigma$. Correspondingly,

$$\text{(if-f)}\ \frac{\langle b, \sigma\rangle \to \mathsf{false} \qquad \text{(skip)}\ \dfrac{}{\langle skip, \sigma\rangle \to \sigma}}{\langle c_2, \sigma\rangle \to \sigma}$$

  implying that also $\mathfrak{O}[\![c_2]\!]\sigma = \sigma = \sigma'$.

For the reverse inclusion, let $\mathfrak{O}[\![c_2]\!]\sigma = \sigma'$, i.e. $\langle c_1, \sigma\rangle \to \sigma'$. Again we have two cases:

- **(if-t)** Here the derivation tree is of the form

$$\text{(if-t)}\ \frac{\langle b, \sigma\rangle \to \mathsf{true} \qquad \langle c; c_1, \sigma\rangle \overset{(*)}{\to} \sigma'}{\langle c_2, \sigma\rangle \to \sigma'}$$

  where $(*)$ implies that there ex. $\sigma'' \in \Sigma$ such that $\langle c, \sigma\rangle \to \sigma''$ and $\langle c_1, \sigma''\rangle \to \sigma'$. Thus:

$$\text{(wh-t)}\ \frac{\langle b, \sigma\rangle \to \mathsf{true} \qquad \langle c, \sigma\rangle \to \sigma'' \qquad \langle c_1, \sigma''\rangle \to \sigma'}{\langle c_1, \sigma\rangle \to \sigma'}$$

  and hence $\mathfrak{O}[\![c_1]\!]\sigma = \sigma'$.

- **(if-t)** Here we have

$$\text{(if-f)}\ \frac{\langle b, \sigma\rangle \to \mathsf{false} \qquad \text{(skip)}\ \dfrac{}{\langle \mathsf{skip}, \sigma\rangle \to \sigma}}{\langle c_2, \sigma\rangle \to \sigma}$$

  Thus $\sigma' = \sigma$ and

$$\text{(wh-f)}\ \frac{\langle b, \sigma\rangle \to \mathsf{false}}{\langle c_1, \sigma\rangle \to \sigma}$$

  which implies $\mathfrak{O}[\![c_1]\!] = \sigma = \sigma'$.

2.9 The Abstract Machine

Definition 5.1 (Abstract machine)

The **abstract machine (AM)** is given by

- **programs** $P \in \mathsf{Code}$ and **instructions** p:

$$P ::= p^*$$
$$p ::= \mathsf{PUSH}(z) \mid \mathsf{PUSH}(t) \mid \mathsf{ADD} \mid \mathsf{SUB} \mid \mathsf{MULT} \mid \mathsf{EQ} \mid \mathsf{GT} \mid \mathsf{NOT} \mid \mathsf{AND} \mid \mathsf{OR} \mid \mathsf{LOAD}(x) \mid \mathsf{STO}(x) \mid \mathsf{JMP}(k) \mid \mathsf{JMPF}(k)$$

  (where $z, k \in \mathbb{Z}$, $t \in \mathbb{B}$ and $x \in \mathsf{Var}$)
- **configurations** of the form $\langle \mathsf{pc}, e, \sigma \rangle \in \mathsf{Cnf}$ where
  - $\mathsf{pc} \in \mathbb{Z}$ is the **program counter** (i.e. address of next instruction to be executed)
  - $e \in \mathsf{Stk} := (\mathbb{Z} \cup \mathbb{B})^*$ is the **evaluation stack** (top to the right)
  - $\sigma \in \Sigma = (\mathsf{Var} \to \mathbb{Z})$ is the **(storage) state**

  (thus $\mathsf{Cnf} = \mathbb{Z} \times \mathsf{Stk} \times \Sigma$)
- **initial configurations** of the form $\langle 0, \epsilon, \sigma \rangle$
- **final configurations** of the form $\langle |P|, e, \sigma \rangle$

2.9.1 Transition relation of AM

Definition 5.2 (Transition relation of AM)

For $P = p_0; \ldots; p_{n-1} \in \mathsf{Code}$ and $0 \leqslant \mathsf{pc} < n$, the **transition relation** $\rhd \subseteq \mathsf{Cnf} \times \mathsf{Cnf}$ is given by

$$
\begin{aligned}
P \vdash \langle \mathsf{pc}, e, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : z, \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{PUSH}(z) \\
P \vdash \langle \mathsf{pc}, e, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : t, \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{PUSH}(t) \\
P \vdash \langle \mathsf{pc}, e : z_1 : z_2, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (z_1 + z_2), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{ADD} \\
P \vdash \langle \mathsf{pc}, e : z_1 : z_2, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (z_1 - z_2), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{SUB} \\
P \vdash \langle \mathsf{pc}, e : z_1 : z_2, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (z_1 \cdot z_2), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{MULT} \\
P \vdash \langle \mathsf{pc}, e : z_1 : z_2, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (z_1 = z_2), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{EQ} \\
P \vdash \langle \mathsf{pc}, e : z_1 : z_2, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (z_1 > z_2), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{GT} \\
P \vdash \langle \mathsf{pc}, e : t, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (\neg t), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{NOT} \\
P \vdash \langle \mathsf{pc}, e : t_y : t_2, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (t_1 \wedge t_2), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{AND} \\
P \vdash \langle \mathsf{pc}, e : t_y : t_2, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : (t_1 \vee t_2), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{OR} \\
P \vdash \langle \mathsf{pc}, e, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e : \sigma(x), \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{LOAD}(x) \\
P \vdash \langle \mathsf{pc}, e : z, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e, \sigma[x \mapsto z] \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{STO}(x) \\
P \vdash \langle \mathsf{pc}, e, \sigma \rangle &\rhd \langle \mathsf{pc}+k, e, \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{JMP}(k) \\
P \vdash \langle \mathsf{pc}, e : \mathsf{true}, \sigma \rangle &\rhd \langle \mathsf{pc}+1, e, \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{JMPF}(k) \\
P \vdash \langle \mathsf{pc}, e : \mathsf{false}, \sigma \rangle &\rhd \langle \mathsf{pc}+k, e, \sigma \rangle && \text{if } p_{\mathsf{pc}} = \mathsf{JMPF}(k)
\end{aligned}
$$

2.9.2 Terminating and looping computations

Corollary 5.3

$\rhd$ is **not total**, i.e. there exists $\gamma \in \mathsf{Cnf}$ such that

$$\gamma \ntriangleright \gamma'$$

for all $\gamma' \in \mathsf{Cnf}$

Proof: Possible cases are:

- γ is **final** (that is, $\gamma = \langle |P|, e, \sigma \rangle$)
- γ is stuck
  - e.g. $\gamma = \langle \mathsf{pc}, 1, \sigma \rangle$ with $p_{\mathsf{pc}} = \mathsf{ADD}$ or $p_{\mathsf{pc}} = \mathsf{JMPF}(k)$ (inappropriate arguments)
  - or $\gamma = \langle \mathsf{pc}, e, \sigma \rangle$ with $\mathsf{pc} \notin \{0, \ldots, |P|\}$ (program counter out of bounds)

Definition 5.4 (AM computations)

- A **finite computation** is a finite configuration sequence of the form

  $$\gamma_0, \gamma_1, ..., \gamma_k$$

  where $k \in \mathbb{N}$ and $\gamma_{i-1} \rhd \gamma_i$ for each $i \in \{1, ..., k\}$.
- If, in addition, there is no γ such that $\gamma_k \rhd \gamma$, then $\gamma_0, \gamma_1, ...\gamma_k$ is called **terminating**.
- A **looping computation** is an infinite configuration sequence of the form

  $$\gamma_0, \gamma_1, \gamma_2, ...$$

  where $\gamma_i \rhd \gamma_{i+1}$ for each $i \in \mathbb{N}$.

Note: according to (the proof of) Corollary 5.3 (p. 19), a terminating computation may end in a final or in a stuck configuration.

2.9.3 Determinism of Execution

Lemma 5.6 (Determinism of AM semantics)

The semantics of AM is **deterministic**: for all $\gamma, \gamma', \gamma'' \in \mathsf{Cnf}$,

$$P \vdash \gamma \rhd \gamma' \text{ and } P \vdash \gamma \rhd \gamma'' \text{ implies } \gamma' = \gamma''$$

Proof:

- Instruction to be executed is unambiguously given by program counter
- Topmost stack entries and storage state then yield unique successor configuration

Thus the following function is well defined:

Definition 5.7 (Semantics of AM Programs)

The **semantics of an AM program** is given by $\mathfrak{M}[\![.]\!] : \mathsf{Code} \to (\Sigma \to \Sigma)$ as follows:

$$\mathfrak{M}[\![P]\!]\sigma := \begin{cases} \sigma' & \text{if } P \vdash \langle 0, \epsilon, \sigma\rangle \rhd^* \langle |P|, e, \sigma'\rangle \text{ for some } e \in \mathsf{Stk} \\ \text{undefined} & \text{otherwise} \end{cases}$$

2.9.4 Translation of Arithmetic expressions

Definition 6.1 (Translation of arithmetic expressions)

The translation function

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

is given by

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

Example 6.2

$$\begin{aligned}
\mathfrak{T}_a[\![x + 1]\!] &= \mathfrak{T}_a[\![x]\!]; \mathfrak{T}_a[\![1]\!]; \mathsf{ADD} \\
&= \mathsf{LOAD}(x); \mathsf{PUSH}(1); \mathsf{ADD}
\end{aligned}$$

Lemma 7.2 (Correctness of $\mathfrak{T}_a[\![.]\!]$)

For every $a \in \mathsf{AExp}$, $\sigma \in \Sigma$ and $z \in \mathbb{Z}$,

$$\langle a, \sigma \rangle \to z \text{ implies } \mathfrak{T}_a[\![a]\!] \vdash \langle 0, \epsilon, \sigma \rangle \rhd^* \langle |\mathfrak{T}_a[\![a]\!]|, z, \sigma \rangle$$

Note: The implication is sufficient to ensure soundness and completeness as the expression evaluation is **total** and the semantics of machine code is **deterministic** (see Lemma 5.6 on page 20).

Proof of Lemma 7.2:

Let $a \in \mathsf{AExp}$, $P := \mathfrak{T}_a[\![a]\!]$, $\sigma \in \Sigma$ and $z \in \mathbb{Z}$ such that $\langle a, \sigma \rangle \to z$.
By structural induction on a, we show that $P \vdash \langle 0, \epsilon, \sigma \rangle \rhd^* \langle |\mathfrak{T}_a[\![a]\!]|, z, \sigma \rangle$:

- **Induction base**
  - $a = z \in \mathbb{Z}$:
    Here $P = 0 : \mathsf{PUSH}(z)$, such that $P \vdash \langle 0, \epsilon, \sigma \rangle \rhd \langle 1, z, \sigma \rangle$
  - $a = x \in \mathsf{Var}$:
    Here $z = \sigma(x)$ and $P = 0 : \mathsf{LOAD}(x)$, such that $P \vdash \langle 0, \epsilon, \sigma \rangle \rhd \langle 1, z, \sigma \rangle$
- **Induction step**
  - $a = a_1 + a_2$:
    Here $z = z_1 + z_2$ where $\langle a_i, \sigma \rangle \to z_i$ and $P = P_1; P2; \mathsf{ADD}$ for $P_i := \mathfrak{T}_a[\![a_i]\!]$ $(i = 1, 2)$. Thus,

$$\begin{aligned} P \vdash \langle 0, \epsilon, \sigma \rangle &\rhd^* \langle |P_1|, z_1, \sigma \rangle && \text{(ind. hyp. for } a_1 \text{ and Lm.7.1)} \\ &\rhd^* \langle |P_1| + |P_2|, z_1 : z_2, \sigma \rangle && \text{(ind. hyp. for } a_2 \text{ and Lm.7.1)} \\ &\rhd^* \langle |P|, z, \sigma \rangle && \text{(ADD at address } |P_1| + |P_2| \text{ and Lm.7.1)} \end{aligned}$$

    Note: See page 29 for Lemma 7.1
  - $a = a_1 - a_2$ and $a = a_1 * a_2$:
    Analogous to $a = a_1 + a_2$

2.9.5 Translation of Boolean expressions

Definition 6.3 (Translation of Boolean expressions)

The translation function

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

is given by

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

Lemma 7.3 (Correctness of $\mathfrak{T}_b[\![.]\!]$)

For every $b \in \mathsf{BExp}$, $\sigma \in \Sigma$ and $t \in \mathbb{B}$,

$$\langle b, \sigma \rangle \to t \text{ implies } \mathfrak{T}_b[\![b]\!] \vdash \langle 0, \epsilon, \sigma \rangle \rhd^* \langle |\mathfrak{T}_b[\![b]\!]|, t, \sigma \rangle$$

Note: Again, the implication is sufficient to ensure soundness and completeness as the expression evaluation is **total** and the semantics of machine code is **deterministic** (see Lemma 5.6 on page 20).

The proof of Lemma 7.3 can be done by induction on the syntactic structure of b.

2.9.6 Translation of Commands

Definition 6.4 (Translation of commands)

The translation function

$$\mathfrak{T}_c[\![.]\!] : \mathsf{Cmd} \to \mathsf{Code}$$

is given by

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

Theorem 7.4 (Correctness of $\mathfrak{T}_c[\![.]\!]$)

For every $c \in \mathsf{Cmd}$,

$$\mathfrak{O}[\![c]\!] = \mathfrak{M}[\![\mathfrak{T}_c[\![c]\!]]\!]$$

The Proof is carried out in two steps:

- **Completeness (Lemma 7.5)**: from source to machine code
- **Soundness (Lemma 7.6)**: from machine to source code

Lemma 7.5 (Completeness of $\mathfrak{T}_c[\![.]\!]$)

For every $c \in \mathsf{Cmd}$ and $\sigma, \sigma' \in \Sigma$,

$$\langle c, \sigma \rangle \to \sigma' \text{ implies } \mathfrak{T}_c[\![c]\!] \vdash \langle 0, \epsilon, \sigma \rangle \rhd^* \langle |\mathfrak{T}_c[\![c]\!]|, \epsilon, \sigma' \rangle$$

Proof of Lemma 7.5

Let $\langle c, \sigma \rangle \to \sigma'$ and $P := \mathfrak{T}_c[\![c]\!]$. Possible cases according to Definition 3.2 (p. 10):

- Case $\text{(skip)}\ \dfrac{}{\langle \mathsf{skip}, \sigma \rangle \to \sigma}$ (i.e. $c = \mathsf{skip}$ and $\sigma' = \sigma$):

Here $P = \epsilon$ and hence

$$P \vdash \langle 0, \epsilon, \sigma \rangle \rhd^0 \langle |P|, \epsilon, \sigma' \rangle$$

- Case (asgn) $\dfrac{\langle a, \sigma \rangle \to z}{\langle x := a, \sigma \rangle \to \sigma[x \mapsto z]}$ (i.e. $c = (x := a)$ and $\sigma' = \sigma[x \mapsto z]$):

  Here $P = \mathfrak{T}_a[\![a]\!]; \mathsf{STO}(x)$ and hence

$$\begin{aligned} P \vdash \langle 0, \epsilon, \sigma \rangle &\rhd^* \langle |\mathfrak{T}_a[\![a]\!]|, z, \sigma \rangle \text{ (Lemma 7.2 and 7.1)} \\ &\rhd \langle |P|, \epsilon, \sigma' \rangle \end{aligned}$$

- Case (seq) $\dfrac{\langle c_1, \sigma \rangle \to \sigma'' \qquad \langle c_2, \sigma'' \rangle \to \sigma'}{\langle c_1; c_2, \sigma \rangle \to \sigma'}$ (i.e. $c = c_1; c_2$):

  Here $P = \mathfrak{T}_c[\![c_1]\!]; \mathfrak{T}_c[\![c_2]\!]$ such that

$$\begin{aligned} P \vdash \langle 0, \epsilon, \sigma \rangle &\rhd^* \langle |\mathfrak{T}_c[\![c_1]\!]|, \epsilon, \sigma'' \rangle \text{ (ind. hyp. for } \langle c_1, \sigma \rangle \text{ and Lemma 7.1)} \\ &\rhd^* \langle |\mathfrak{T}_c[\![c_1]\!]| + |\mathfrak{T}_c[\![c_2]\!]|, \epsilon, \sigma' \rangle \text{ (ind. hyp. for } \langle c_2, \sigma'' \rangle \text{ and Lemma 7.1)} \end{aligned}$$

- Case (if-t) $\dfrac{\langle b, \sigma \rangle \to \mathsf{true} \qquad \langle c_1, \sigma \rangle \to \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}$ (i.e. $c = \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}$):

  Here

$$\begin{aligned} P =&\mathfrak{T}_b[\![b]\!]; \\ &k : \mathsf{JMPF}(k_1 + 2); \\ &k + 1 : \mathfrak{T}_c[\![c_1]\!]; \\ &k + k_1 + 1 : \mathsf{JMP}(k_2 + 1); \\ &k + k_1 + 2 : \mathfrak{T}_c[\![c_2]\!]; \\ &k + k_1 + k_2 + 2 : \end{aligned}$$

  for $k := |\mathfrak{T}_b[\![b]\!]|$, $k_1 := |\mathfrak{T}_c[\![c_1]\!]|$ and $k_2 := |\mathfrak{T}_c[\![c_2]\!]|$, and hence

$$\begin{aligned} P \vdash \langle 0, \epsilon, \sigma \rangle &\rhd^* \langle k, \mathsf{true}, \sigma \rangle \text{ (Lemma 7.3 and 7.1)} \\ &\rhd \langle k + 1, \epsilon, \sigma \rangle \\ &\rhd^* \langle k + k_1 + 1, \epsilon, \sigma' \rangle \text{ (ind.hyp. for } \langle c_1, \sigma \rangle \text{ and Lemma 7.1)} \\ &\rhd \langle k + k_1 + k_2 + 2, \epsilon, \sigma' \rangle \end{aligned}$$

- Case (if-f) $\dfrac{\langle b, \sigma \rangle \to \mathsf{false} \qquad \langle c_1, \sigma \rangle \to \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}$ is analogous to the previous case (if-t)

- Case (wh-t) $\dfrac{\langle b,\sigma\rangle \to \mathsf{true} \qquad \langle c_0,\sigma\rangle \to \sigma'' \qquad \langle \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end},\sigma''\rangle \to \sigma'}{\langle \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end},\sigma\rangle \to \sigma'}$
  (i.e. $c = \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end}$):
  Here

$$
\begin{aligned}
P =& \mathfrak{T}_b[\![b]\!];\\
&k : \mathsf{JMPF}(k_0+2);\\
&k+1 : \mathfrak{T}_c[\![c_0]\!];\\
&k+k_0+1 : \mathsf{JMP}(-(k+k_0+1));
\end{aligned}
$$

  for $k := |\mathfrak{T}_b[\![b]\!]|$ and $k_0 := |\mathfrak{T}_c[\![c_0]\!]|$, and thus

$$
\begin{aligned}
P \vdash \langle 0,\epsilon,\sigma\rangle &\rhd^* \langle k,\mathsf{true},\sigma\rangle \text{ (Lemma 7.3 and 7.2)}\\
&\rhd \langle k+1,\epsilon,\sigma\rangle\\
&\rhd^* \langle k+k_0+1,\epsilon,\sigma''\rangle \text{ (ind. hyp. for } \langle c_0,\sigma\rangle \text{ and Lemma 7.1)}\\
&\rhd \langle 0,\epsilon,\sigma''\rangle\\
&\rhd^* \langle k+k_0+2,\epsilon,\sigma'\rangle \text{ (ind. hyp. for } \langle c,\sigma''\rangle)
\end{aligned}
$$

- Case (wh-f) $\dfrac{\langle b,\sigma\rangle \to \mathsf{false}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end},\sigma\rangle \to \sigma}$ is analogous to the previous case (wh-t)

Lemma 7.6 (Soundness of $\mathfrak{T}_c[\![.]\!]$)

For every $c \in \mathsf{Cmd}$, $\sigma,\sigma' \in \Sigma$, and $e \in \mathsf{Stk}$,

$$\mathfrak{T}_c[\![c]\!] \vdash \langle 0,\epsilon,\sigma\rangle \rhd^* \langle |\mathfrak{T}_c[\![c]\!]|, e, \sigma'\rangle \text{ implies } \langle c,\sigma\rangle \to \sigma' \text{ and } e = \epsilon$$

The proof is done by induction on the length of the computation sequence $\langle 0,\epsilon,\sigma\rangle \rhd^* \langle |\mathfrak{T}_c[\![c]\!]|, e, \sigma'\rangle$.
TODO: See proof in exercises

2.9.7 Example: Translation of factorial program

Example 6.5 (Translation of factorial program)

$$
\begin{aligned}
&\mathfrak{T}_c[\![y := 1; \mathsf{while}\ \neg(x = 1)\ \mathsf{do}\ y := y * x; x := x - 1\ \mathsf{end}]\!]\\
=&\mathfrak{T}_c[\![y := 1]\!]; \mathfrak{T}_c[\![\mathsf{while}\ \neg(x = 1)\ \mathsf{do}\ y := y * x; x := x - 1\ \mathsf{end}]\!]\\
=&\mathsf{PUSH}(1); \mathsf{STO}(y);\\
&\mathfrak{T}_b[\![\neg(x = 1)]\!]; \mathsf{JMPF}(|\mathfrak{T}_c[\![y := y * x; x := x - 1]\!]| + 2)\\
&\mathfrak{T}_c[\![c]\!]; \mathsf{JMP}(-(|\mathfrak{T}_b[\![\neg(x = 1)]\!]| + |\mathfrak{T}_c[\![y := y * x; x := x - 1]\!]| + 1))\\
=&\mathsf{PUSH}(1); \mathsf{STO}(y);\\
&\mathsf{LOAD}(x); \mathsf{PUSH}(1); \mathsf{EQ}; \mathsf{NOT}; \mathsf{JMPF}(8 + 2);\\
&\mathsf{LOAD}(y); \mathsf{LOAD}(x); \mathsf{MULT}; \mathsf{STO}(y);\\
&\mathsf{LOAD}(x); \mathsf{PUSH}(1); \mathsf{SUB}; \mathsf{STO}(x); \mathsf{JMP}(-(4 + 8 + 1))\\
=&\mathsf{PUSH}(1); \mathsf{STO}(y);\\
&\mathsf{LOAD}(x); \mathsf{PUSH}(1); \mathsf{EQ}; \mathsf{NOT}; \mathsf{JMPF}(10);\\
&\mathsf{LOAD}(y); \mathsf{LOAD}(x); \mathsf{MULT}; \mathsf{STO}(y);\\
&\mathsf{LOAD}(x); \mathsf{PUSH}(1); \mathsf{SUB}; \mathsf{STO}(x); \mathsf{JMP}(-13)
\end{aligned}
$$

2.9.8 Example: Execution of factorial program

Example 6.6 (Execution of factorial program)

Let

$$\begin{aligned} P :=& 0 : \mathsf{PUSH}(1); 1 : \mathsf{STO}(y); 2 : \mathsf{LOAD}(x); 3 : \mathsf{PUSH}(1); 4 : \mathsf{EQ}; 5 : \mathsf{NOT}; \\ & 6 : \mathsf{JMPF}(10); 7 : \mathsf{LOAD}(y); 8 : \mathsf{LOAD}(x); 9 : \mathsf{MULT}; 10 : \mathsf{STO}(y); \\ & 11 : \mathsf{LOAD}(x); 12 : \mathsf{PUSH}(1); 13 : \mathsf{SUB}; 14 : \mathsf{STO}(x); 15 : \mathsf{JMP}(-13) \end{aligned}$$

and $\sigma \in \Sigma$ with $\sigma(x) = 2$.

$\langle 0, \epsilon, \sigma \rangle$
$\rhd \langle 1, 1, \sigma \rangle$
$\rhd \langle 2, \epsilon, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 3, 2, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 4, 2 : 1, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 5, \mathsf{false}, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 6, \mathsf{true}, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 7, \epsilon, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 8, 1, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 9, 1 : 2, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 10, 2, \sigma[y \mapsto 1] \rangle$
$\rhd \langle 11, \epsilon, \sigma[y \mapsto 2] \rangle$
$\rhd \langle 12, 2, \sigma[y \mapsto 2] \rangle$
$\rhd \langle 13, 2 : 1, \sigma[y \mapsto 2] \rangle$
$\rhd \langle 14, 1, \sigma[y \mapsto 2] \rangle$
$\rhd \langle 15, \epsilon, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\rhd \langle 2, \epsilon, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\rhd \langle 3, 1, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\rhd \langle 4, 1 : 1, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\rhd \langle 5, \mathsf{true}, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\rhd \langle 6, \mathsf{false}, \sigma[x \mapsto 1, y \mapsto 2] \rangle$
$\rhd \langle 16, \epsilon, \sigma[x \mapsto 1, y \mapsto 2] \rangle$

2.9.9 Induction on Finite AM computations

We introduce a new induction principle on finite AM computations as defined in Def. 5.4 (p. 20).

- **Definition:** a finite computation $\gamma_0, \gamma_1, ..., \gamma_k$ has **length** k
- **Induction base:** property holds for all computations of length 0
- **Induction hypothesis:** property holds for all computations of length $\leqslant k$
- **Induction step:** property holds for all computations of length $k+1$

2.9.10 Embedding of Code and Stack

Lemma 7.1

If $P \vdash \langle \mathsf{pc}, e, \sigma \rangle \rhd^* \langle \mathsf{pc}', e', \sigma' \rangle$, then

$$P_1; P; P_2 \vdash \langle |P_1| + \mathsf{pc}, e_0 : e, \sigma \rangle \rhd^* \langle |P_1| + \mathsf{pc}', e_0 : e', \sigma' \rangle$$

for all $P_1, P_2 \in \mathsf{Code}$ and $e_0 \in \mathsf{Stk}$.

Interpretation: both the code and the stack component can be extended without actually changing the behaviour of the machine.

Proof:
Let $P \vdash \langle \mathsf{pc}, e, \sigma \rangle \rhd^k \langle \mathsf{pc}', e', \sigma' \rangle$ for some $k \in \mathbb{N}$, and let $P_1, P_2 \in \mathsf{Code}$ and $e_0 \in \mathsf{Stk}$. By induction on k we show that

$$P_1; P; P_2 \vdash \langle |P_1| + \mathsf{pc}, e_0 : e, \sigma \rangle \rhd^k \langle |P_1| + \mathsf{pc}', e_0 : e', \sigma' \rangle$$

- $k = 0$: Here $\mathsf{pc} = \mathsf{pc}'$, $e = e'$ and $\sigma = \sigma'$, which immediately proves the claim.
- $k \rightsquigarrow k+1$: $P \vdash \langle \mathsf{pc}, e, \sigma \rangle \rhd^{k+1} \langle \mathsf{pc}', e', \sigma' \rangle$ implies that there ex. $\mathsf{pc}'' \in \{0, ..., |P|\}$, $e'' \in \mathsf{Stk}$ and $\sigma'' \in \Sigma$ such that

  $$P \vdash \langle \mathsf{pc}, e, \sigma \rangle \rhd \langle \mathsf{pc}'', e'', \sigma'' \rangle \rhd^k \langle \mathsf{pc}', e', \sigma' \rangle$$

  Hence,

  $$P_1; P; P_2 \vdash \langle \mathsf{pc} + |P_1|, e_0 : e, \sigma \rangle \rhd \langle \mathsf{pc}'' + |P_1|, e_0 : e'', \sigma'' \rangle$$

  as the instruction at address pc in P is equal to the instruction at address $\mathsf{pc} + |P_1|$ in $P_1; P; P_2$ and e'' is fully determined by e and thus independent from e_0.
  By induction hypothesis, it follows that

  $$P_1; P; P_2 \vdash \langle \mathsf{pc}'' + |P_1|; e_0 : e'', \sigma'' \rangle \rhd^k \langle \mathsf{pc}' + |P_1|, e_0 : e', \sigma' \rangle$$

  which proves the claim.

2.9.11 Decomposition Lemma for AM programs

Lemma Ex3Task2 (Decomposition Lemma)

Let $c_1, c_2 \in \mathsf{Cmd}$ and $\mathsf{pc} \in \{0, ..., |\mathfrak{T}_c[\![c_1]\!]| - 1\}$. If

$$\mathfrak{T}_c[\![c_1]\!]; \mathfrak{T}_c[\![c_2]\!] \vdash \langle \mathsf{pc}, e, \sigma \rangle \rhd^k \langle |\mathfrak{T}_c[\![c_1]\!]; \mathfrak{T}_c[\![c_2]\!]|, e'', \sigma'' \rangle$$

then there exists a configuration $\langle \mathsf{pc}', e', \sigma' \rangle$ and $k_1, k_2 \in \mathbb{N}$ with $k = k_1 + k_2$ such that

$$\mathfrak{T}_c[\![c_1]\!] \vdash \langle \mathsf{pc}, e, \sigma \rangle \rhd^{k_1} \langle |\mathfrak{T}_c[\![c_1]\!]|, e', \sigma' \rangle$$

and

$$\mathfrak{T}_c[\![c_1]\!]; \mathfrak{T}_c[\![c_2]\!] \vdash \langle |\mathfrak{T}_c[\![c_1]\!]|, e', \sigma' \rangle \rhd^{k_2} \langle |\mathfrak{T}_c[\![c_1]\!]; \mathfrak{T}_c[\![c_2]\!]|, e'', \sigma'' \rangle$$

3 Denotional Semantics of WHILE

The primary aspect of a program is its "effect", i.e. the **input/output behaviour**. In operational semantics the semantic functional

$$\mathfrak{O}[\![.]\!] : \mathsf{Cmd} \to (\Sigma \to \Sigma)$$

was defined **indirect** by referring to the execution relation ("$\mathfrak{O}[\![c]\!]\sigma := \sigma'$ iff $\langle c, \sigma \rangle \to \sigma'$").

Now we **abstract** from operational details. The **Denotional semantics** are a direct definition of effects by induction on the syntactic structure of a program.

3.1 Denotional semantics of arithmetic expression

Definition 8.1 (Denotional semantics of arithmetic expression)

The **(denotional) semantic functional for arithmetic expressions**,

$$\mathfrak{A}[\![.]\!] : \mathsf{AExp} \to (\Sigma \to \mathbb{Z})$$

is given by:

$$\begin{aligned} \mathfrak{A}[\![z]\!]\sigma &:= z & \mathfrak{A}[\![a_1 + a_2]\!]\sigma &:= \mathfrak{A}[\![a_1]\!]\sigma + \mathfrak{A}[\![a_2]\!]\sigma \\ \mathfrak{A}[\![x]\!]\sigma &:= \sigma(x) & \mathfrak{A}[\![a_1 - a_2]\!]\sigma &:= \mathfrak{A}[\![a_1]\!]\sigma - \mathfrak{A}[\![a_2]\!]\sigma \\ & & \mathfrak{A}[\![a_1 * a_2]\!]\sigma &:= \mathfrak{A}[\![a_1]\!]\sigma \cdot \mathfrak{A}[\![a_2]\!]\sigma \end{aligned}$$

3.2 Denotional semantics of Boolean expressions

Definition 8.2 ((denotional) semantic functional for Boolean expressions)

The **(denotional) semantic functional for Boolean expressions**

$$\mathfrak{B}[\![.]\!] : \mathsf{BExp} \to (\Sigma \to \mathbb{B})$$

is given by:

$$\mathfrak{B}[\![t]\!]\sigma := t$$

$$\mathfrak{B}[\![a_1 = a_2]\!]\sigma := \begin{cases} \mathsf{true} & \mathsf{if}\ \mathfrak{A}[\![a_1]\!]\sigma > \mathfrak{A}[\![a_2]\!]\sigma \\ \mathsf{false} & \mathsf{otherwise} \end{cases}$$

$$\mathfrak{B}[\![\neg b]\!]\sigma := \begin{cases} \mathsf{true} & \mathsf{if}\ \mathfrak{B}[\![b]\!]\sigma = \mathsf{false} \\ \mathsf{false} & \mathsf{otherwise} \end{cases}$$

$$\mathfrak{B}[\![a_1 \wedge a_2]\!]\sigma := \begin{cases} \mathsf{true} & \mathsf{if}\ \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \mathsf{true} \\ \mathsf{false} & \mathsf{otherwise} \end{cases}$$

$$\mathfrak{B}[\![a_1 \vee a_2]\!]\sigma := \begin{cases} \mathsf{false} & \mathsf{if}\ \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \mathsf{false} \\ \mathsf{true} & \mathsf{otherwise} \end{cases}$$

3.3 Denotional semantics of Commands

The goal is to define the semantic function

$$\mathfrak{C}[\![.]\!] : \mathsf{Cmd} \to (\Sigma \dashrightarrow \Sigma)$$

which is the same type as the operational function

$$\mathfrak{O}[\![.]\!] : \mathsf{Cmd} \to (\Sigma \dashrightarrow \Sigma)$$

In Fact, both will turn out to be the same, which will result in the equivalence of operational and denotional semantics.

3.3.1 Auxiliary Functions

The inductive definition of $\mathfrak{C}[\![.]\!]$ employs the following auixiliary functions:

- **Identity** on state (for skip):

  $$\mathsf{id}_\Sigma : \Sigma \to \Sigma : \sigma \mapsto \sigma$$

- **(Strict) composition** of partial state transformations (for sequential composition):

  $$\circ : (\Sigma \dashrightarrow \Sigma) \times (\Sigma \dashrightarrow \Sigma) \to (\Sigma \dashrightarrow \Sigma)$$

  where for every $f, g : \Sigma \dashrightarrow \Sigma$ and $\sigma \in \Sigma$

  $$(g \circ f)(\sigma) := \begin{cases} g(f(\sigma)) & \text{if } f(\sigma) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Semantic conditional** (for if):

  $$\mathsf{cond} : (\Sigma \to \mathbb{B}) \times (\Sigma \dashrightarrow \Sigma) \times (\Sigma \dashrightarrow \Sigma) \to (\Sigma \dashrightarrow \Sigma)$$

  where for every $p : \Sigma \to \mathbb{B}$, $f, g : \Sigma \dashrightarrow \Sigma$ and $\sigma \in \Sigma$

  $$\mathsf{cond}(p, f, g)(\sigma) := \begin{cases} f(\sigma) & \text{if } p(\sigma) = \text{true} \\ g(\sigma) & \text{otherwise} \end{cases}$$

3.3.2 Denotional semantic functional for commands

Definition 8.3 ((denotional) semantic functional for commands)

The **(denotional) semantic functional for commands**

$$\mathfrak{C}[\![.]\!] : \mathsf{Cmd} \to (\Sigma \to \Sigma)$$

is given by:

$$\begin{aligned}
\mathfrak{C}[\![\mathsf{skip}]\!] &:= \mathsf{id}_\Sigma \\
\mathfrak{C}[\![x := a]\!] &:= \lambda\sigma.\sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma] \\
\mathfrak{C}[\![c_1; c_2]\!] &:= \mathfrak{C}[\![c_2]\!] \circ \mathfrak{C}[\![c_1]\!] \\
\mathfrak{C}[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}]\!] &:= \mathsf{cond}(\mathfrak{B}[\![b]\!], \mathfrak{C}[\![c_1]\!], \mathfrak{C}[\![c_2]\!]) \\
\mathfrak{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!] &:= \mathsf{fix}(\Phi)
\end{aligned}$$

where $\Phi : (\Sigma \to \Sigma) \to (\Sigma \to \Sigma) : f \mapsto \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$

The λ operator in $\mathfrak{C}[\![x := a]\!] := \lambda\sigma.\sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma]$ denotes **functional abstraction**:

$$\mathfrak{C}[\![c := a]\!]\sigma = \sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma]$$

3.4 Fixpoint semantics

3.4.1 Why Fixpoints?

The goal is to preserve the **validity of equivalence** as in Lemma 4.4 (p. 16):

$$\mathfrak{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!] \overset{(*)}{=} \mathfrak{C}[\![\mathsf{if}\ b\ \mathsf{then}\ c;\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{end}]\!]$$

Using the known parts of Definition 8.3, we obtain:

$$\begin{aligned}
&\mathfrak{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!]\\
\overset{(*)}{=}\ &\mathfrak{C}[\![\mathsf{if}\ b\ \mathsf{then}\ c;\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{end}]\!]\\
\overset{\text{Def. 8.3}}{=}\ &\mathsf{cond}(\mathfrak{B}[\![b]\!], \mathfrak{C}[\![c;\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!], \mathfrak{C}[\![\mathsf{skip}]\!])\\
\overset{\text{Def. 8.3}}{=}\ &\mathsf{cond}(\mathfrak{B}[\![b]\!], \mathfrak{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!] \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)
\end{aligned}$$

Abbreviating $f := \mathfrak{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!]$ this yields

$$f \overset{(**)}{=} \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$$

Hence f must be a **solution** of this recursive equation. In other words: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \to \Sigma) \to (\Sigma \to \Sigma) : f \mapsto \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$$

(since $(**)$ can be stated as $f = \Phi(f)$)

3.4.2 Well-Definedness of Fixpoint Semantics

The Fixpoint property is not sufficient to obtain a well-defined semantics.

Potential problems:

- **Existence:** There does not need to exist any fixpoint.
  Example: $\phi_1 : \mathbb{N} \to \mathbb{N} : n \mapsto n + 1$
  Solution: in our setting, **fixpoints always exist**
- **Uniqueness:** There might exist several fixpoints.
  Example: $\phi_2 : \mathbb{N} \to \mathbb{N} : n \mapsto n^2$ has fixpoints 0,1
  Solution: Uniqueness guaranteed by **choosing a special fixpoint**
  Question: Which is the right one?

3.4.3 Definedness

For the characterisation of the fixpoint $\mathsf{fix}(\Phi)$ we will also need the **definedness relation** $\sqsubseteq$:

Definition L9S13 (Definedness)

Given $f, g : \Sigma \to \Sigma$, let

$$f \sqsubseteq g \iff \text{for every } \sigma, \sigma' \in \Sigma : f(\sigma) = \sigma' \implies g(\sigma) = \sigma'$$

(g is "at least as defined" as f)

This is equivalent to requiring

$$\mathsf{graph}(f) \subseteq \mathsf{graph}(g)$$

where

$$\mathsf{graph}(h) := \{(\sigma, \sigma') \mid \sigma \in \Sigma, \sigma' = h(\sigma) \text{ defined}\} \subseteq \Sigma \times \Sigma$$

for every $h : \Sigma \to \Sigma$

Example 9.1 (Definedness)

Let $x \in \mathsf{Var}$ be fixed, and let $f_0, f_1, f_2, f_3 : \Sigma \to \Sigma$ be given by

$$f_0(\sigma) := \text{undefined}$$

$$f_1(\sigma) := \begin{cases} \sigma & \text{if } \sigma(x) \text{ even} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$f_2(\sigma) := \begin{cases} \sigma & \text{if } \sigma(x) \text{ odd} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$f_3(\sigma) := \sigma$$

(i.e. f_0, f_1, f_2, f_3 are (partial) identities).
This implies

$$f_0 \sqsubseteq f_1 \sqsubseteq f_3$$

$$f_0 \sqsubseteq f_2 \sqsubseteq f_3$$

$$f_1 \not\sqsubseteq f_2$$

$$f_2 \not\sqsubseteq f_1$$

3.4.4 Characterisation of fix(Φ)

Let while b do c end be a while loop (with $b \in$ BExp and $c \in$ Cmd)
Let $\Phi(f) := \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$ be the corresponding semantic function
Let $f_0 : \Sigma \to \Sigma$ be a **fixpoint** of Φ, i.e. $\Phi(f_0) = f_0$
Given some initial state $\sigma_0 \in \Sigma$, we will distinguish the following cases:

1. loop while b do c end terminates after n iteration ($n \in \mathbb{N}$))
2. body c diverges in the n-th iteration ($n \geqslant 1$) (as it contains a non-terminating while command)
3. loop while b do c end itself diverges

What can be deduced for f_0 in each of those cases?

Case 1: Termination of Loop
Loop while b do c end terminates after n iteration ($n \in \mathbb{N}$))

Formally: there exist $\sigma_1, ..., \sigma_n \in \Sigma$ such that

$$\mathfrak{B}[\![b]\!]\sigma_i = \begin{cases} \text{true} & \text{if } 0 \leqslant i < n \\ \text{false} & \text{if } i = n \end{cases} \text{ and}$$

$$\mathfrak{C}[\![c]\!]\sigma_i = \sigma_{i+1} \text{ for every } 0 \leqslant i < n$$

Now the definition $\Phi(f) := \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$ implies, for every $0 \leqslant i < n$:

$$\begin{aligned} \Phi(f_0)(\sigma_i) &= (f_0 \circ \mathfrak{C}[\![c]\!])(\sigma_i) \\ &= f_0(\sigma_{i+1}) \\ \Phi(f_0)(\sigma_n) &= \sigma_n \end{aligned}$$

Since $\Phi(f_0) = f_0$ it follows that

$$f_0(\sigma_i) = \begin{cases} f_0(\sigma_{i+1}) & \text{if } 0 \leqslant i < n \\ \sigma_n & \text{if } i = n \end{cases}$$

and hence

$$f_0(\sigma_0) = f_0(\sigma_1) = ...f_0(\sigma_n) = \sigma_n$$

Thus **all fixpoints** f_0 **coincide on** σ_0 (with result σ_n)!

Case 2: Divergence of Body

Body c diverges in the n-th iteration ($n \geqslant 1$) (as it contains a non-terminating while command)

Formally: There exists $\sigma_1, ..., \sigma_{n-1} \in \Sigma$ such that

$$\mathfrak{B}[\![b]\!]\sigma_i = \mathsf{true}$$

$$\mathfrak{C}[\![c]\!]\sigma_i = \begin{cases} \sigma_{i+1} & \text{if } 0 \leqslant i \leqslant n-2 \\ \mathsf{undefined} & \text{if } i = n-1 \end{cases}$$

Just as in the previous case (setting $\sigma_n := \mathsf{undefined}$) it follows that

$$f_0(\sigma_0) = \mathsf{undefined}$$

Again all fixpoints f_0 coincide on σ_0 (with undefined result)!

Case 3: Divergence of Loop

Loop while b do c end itself diverges

Formally: There exist $\sigma_1, \sigma_2, ... \in \Sigma$ such that

$$\mathfrak{B}[\![b]\!]\sigma_i = \mathsf{true}$$

$$\mathfrak{C}[\![c]\!]\sigma_i = \sigma_{i+1} \text{ for every } i \in \mathbb{N}$$

Here only derivable:

$$\Phi(f_0)(\sigma_i) = f_0(\sigma_{i+1}) \text{ for every } i \in \mathbb{N}$$

and thus (as $\Phi(f_0) = f_0$)

$$f_0(\sigma_0) = f_0(\sigma_i) \text{ for every } i \in \mathbb{N}$$

Thus **the value of $f_0(\sigma_0)$ is not determined**!

Summary For $\Phi(f_0) = f_0$ and initial state $\sigma_0 \in \Sigma$, the case distinction yields:

1. Loop while b do c end terminates after n iteration ($n \in \mathbb{N}$)) $\implies f_0(\sigma_0) = \sigma_n$
2. body c diverges in the n-th iteration $\implies f_0(\sigma_0) = \mathsf{undefined}$
3. loop while b do c end itself diverges $\implies f_0(\sigma_0)$ not determined

This is not surprising since, e.g. for the loop while true do skip end, **every** $f : \Sigma \to \Sigma$ is a fixpoint:

$$\Phi(f) = \mathsf{cond}(\mathfrak{B}[\![\mathsf{true}]\!], f \circ \mathfrak{C}[\![\mathsf{skip}]\!], \mathsf{id}_\Sigma) = f$$

On the other hand, out operational understanding requires, for every $\sigma_0 \in \Sigma$:

$$\mathfrak{C}[\![\text{while true do skip end}]\!]\sigma_0 = \mathsf{undefined}$$

Conclusion: $\mathsf{fix}(\Phi)$ is the **least defined fixpoint** of Φ.

Corollary L9S15 (Characterisation of $\mathsf{fix}(\Phi)$)

$\mathsf{fix}(\Phi)$ can be characterised by:

- $\mathsf{fix}(\Phi)$ is a **fixpoint** of Φ, i.e.

$$\Phi(\mathsf{fix}(\Phi)) = \mathsf{fix}(\Phi)$$

- $\mathsf{fix}(\Phi)$ is **minimal** with respect to $\sqsubseteq$, i.e. for every $f_0 : \Sigma \dashrightarrow \Sigma$ such that $\Phi(f_0) = f_0$:

$$\mathsf{fix}(\Phi) \sqsubseteq f_0$$

Example 9.2 (Fixpoint)

For `while true do skip end` we obtain for every $f : \Sigma \dashrightarrow \Sigma$:

$$\Phi(f) = \mathsf{cond}(\mathfrak{B}[\![\mathsf{true}]\!], f \circ \mathfrak{C}[\![\mathsf{skip}]\!], \mathsf{id}_\Sigma) = \mathsf{cond}(\mathsf{true}, f \circ \mathsf{id}_\Sigma, \mathsf{id}_\Sigma) = \mathsf{cond}(\mathsf{true}, f, \mathsf{id}_\Sigma) = f$$

This imples $\mathsf{fix}(\Phi) = f_\varnothing$ where $f_\varnothing(\sigma) := \mathsf{undefined}$ for every $\sigma \in \Sigma$ (that is: $\mathsf{graph}(f_\varnothing) = \varnothing$)

Now our goal is to prove the **existence** of $\mathsf{fix}(\Phi)$ for $\Phi(f) = \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$ and to show how it can be "computed" (more exactly: **approximated"**).

Sufficient conditions:

- on domain $\Sigma \dashrightarrow \Sigma$: **chain-complete partial order**
- on function Φ: **monotonicity** and **continuity**

3.4.5 Partial orders

Definition 10.1 (Partial order)

A **partial order (PO)** $(D, \sqsubseteq)$ consists of a set D, called **domain**, and of a relation $\sqsubseteq \subseteq D \times D$ such that, for every $d_1, d_2, d_3 \in D$:

- reflexivity: $d_1 \sqsubseteq d_1$
- transitivity: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_3 \implies d_1 \sqsubseteq d_3$
- antisymmetry: $d_1 \sqsubseteq d_2$ and $d_2 \sqsubseteq d_1 \implies d_1 = d_2$

It is called **total** if, in addition, always $d1 \sqsubseteq d_2$ or $d_2 \sqsubseteq d_1$.

Example 10.2 (Partial order)

1. $(\mathbb{N}, \leqslant)$ is a total partial order
2. $(2^{\mathbb{N}}, \subseteq)$ is a (non-total) partial order
3. $(\mathbb{N}, <)$ is not a partial order (since not reflexive)

Lemma 10.3

$(\Sigma \dashrightarrow \Sigma, \sqsubseteq)$ is a partial order.

Proof of Lemma 10.3:

Using the equivalence $f \sqsubseteq g \iff \mathsf{graph}(f) \subseteq \mathsf{graph}(g)$ and the partial-order property of $\subseteq$.

3.4.6 Chains and Least Upper Bounds

Definition 10.4 (Chain, (least) upper bound)

Let $(D, \sqsubseteq)$ be a partial order and $S \subseteq D$.

1. S is called a **chain** in D, if for every $s_1, s_2 \in S$:

   $$s_1 \sqsubseteq s_2 \text{ or } s_2 \sqsubseteq s_1$$

   (that is, S is a totally ordered subset of D)
2. An element $d \in D$ is called an **upper bound** of S if $s \sqsubseteq d$ for every $s \in S$ (notation: $S \sqsubseteq d$)
3. An upper bound d of S is called **least upper bound (LUB)** or **supremum** of S if $d \sqsubseteq d'$ for every upper bound d' of S (notation: $d = \bigsqcup S$)

Example 10.5 (Chains and Least upper bounds)

1. Every subset $S \subseteq \mathbb{N}$ is a chain in $(\mathbb{N}, \leqslant)$.
   It has a supremum (its greatest element) iff it is finite.
2. $\{\varnothing, \{0\}, \{0,1\}, ...\}$ is a chain in $(2^{\mathbb{N}}, \subseteq)$ with supremum $\mathbb{N}$
3. Let $x \in \mathsf{Var}$ be fixed, and let $f_i : \Sigma \to \Sigma$ for every $i \in \mathbb{N}$ be given by

   $$f_i(\sigma) := \begin{cases} \sigma[x \mapsto \sigma(x) + 1 & \text{if } \sigma(x) \leqslant i \\ \text{undefined} & \text{otherwise} \end{cases}$$

   Then $\{f_0, f_1, f_2, ...\}$ is a chain in $(\Sigma \to \Sigma, \sqsubseteq)$, since for every $i \in \mathbb{N}$ and $\sigma, \sigma' \in \Sigma$:

   $$\begin{aligned} & f_i(\sigma) = \sigma' \\ \Longrightarrow\ & \sigma(x) \leqslant i, \sigma' = \sigma[x \mapsto \sigma(x) + 1] \\ \Longrightarrow\ & \sigma(x) \leqslant i + 1, \sigma' = \sigma[x \mapsto \sigma(x) + 1] \\ \Longrightarrow\ & f_{i+1}(\sigma) = \sigma' \\ \Longrightarrow\ & f_i \sqsubseteq f_{i+1} \end{aligned}$$

3.4.7 Chain Completeness

Definition 10.6 (Chain completeness)

A partial order is called **chain complete (CCPO)** if each of its chains has a least upper bound.

Example 10.7 (Chain completeness)

1. $(2^{\mathbb{N}}, \subseteq)$ is a CCPO with $\bigsqcup S = \bigcup_{M \in S} M$ for every chain $S \subseteq 2^{\mathbb{N}}$
2. $(\mathbb{N}, \leqslant)$ is not chain complete (since e.g. the chain $\mathbb{N}$ has no upper bound)

Corollary 10.8

Every CCPO has a least element $\bigsqcup \varnothing$.

Proof of Corollary 10.8:

Let $(D, \sqsubseteq)$ be a CCPO.

- By definition, $\varnothing$ is a chain in D.
- By definition, every $d \in D$ is an upper bound of $\varnothing$.
- Thus $\bigsqcup \varnothing$ exists and is the least element of D.

Lemma 10.9

$(\Sigma \to \Sigma, \sqsubseteq)$ is a CCPO with least element $f_\varnothing$ where $\mathsf{graph}(f_\varnothing) = \varnothing$.
In particular, for every chain $S \subseteq \Sigma \to \Sigma$, $\mathsf{graph}(\bigsqcup S) = \bigcup_{f \in S} \mathsf{graph}(f)$.

Proof of Lemma 10.9

According to Lemma 10.3 (p. 40), $(\Sigma \to \sigma, \sqsubseteq)$ is a partial order.
It therefore suffices to prove that $\mathsf{graph}(\bigsqcup S) = \bigcup_{f \in S} \mathsf{graph}(f)$.

- We first show that $G := \bigcup_{f \in S} \mathsf{graph}(f)$ is the graph of a partial function $f_0 : \Sigma \to \Sigma$.
  To this aim, let $(\sigma, \sigma'), (\sigma, \sigma'') \in G$.
  Hence, there ex. $f_1, f_2 \in S$ such that $f_1(\sigma) = \sigma'$ and $f_2(\sigma) = \sigma''$.
  Since S is a chain, it holds that $f_1 \sqsubseteq f_2$ or $f_2 \sqsubseteq f_1$. In both cases $\sigma' = f_1(\sigma) = f2(\sigma) = \sigma''$.
- On the other hand, f_0 is an upper bound of S since, for every $f \in S$, $\mathsf{graph}(f) \subseteq \mathsf{graph}(f_0)$.
- It remains to show that f_0 is minimal. To this aim, let f_1 be another upper bound of S.

$$\begin{aligned}
&\Longrightarrow f \sqsubseteq f_1 \text{ for every } f \in S \\
&\Longrightarrow \mathsf{graph}(f) \subseteq \mathsf{graph}(f_1) \text{ for every } f \in S \\
&\Longrightarrow \mathsf{graph}(f_0) = \bigcup_{f \in S} \mathsf{graph}(f) \subseteq \mathsf{graph}(f_1) \\
&\Longrightarrow f_0 \sqsubseteq f_1 \\
&\Longrightarrow \text{claim}
\end{aligned}$$

Example 10.10 (Least upper bound)

Let $x \in Var$ be fixed, and let $f_i : \Sigma \to \Sigma$ for every $i \in \mathbb{N}$ be given by

$$f_i(\sigma) := \begin{cases} \sigma[x \mapsto \sigma(x) + 1] & \text{if } \sigma(x) \leqslant i \\ \text{undefined} & \text{otherwise} \end{cases}$$

Then $S := \{f_0, f_1, f_2, ...\}$ is a chain (cp. Example 10.5(3) (p. 41)) with $\bigsqcup S = f$ where

$$f : \Sigma \to \Sigma : \sigma \mapsto \sigma[x \mapsto \sigma(x) + 1]$$

3.4.8 Monotonicity

Definition 11.1 (Monotonicity)

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be partial orders, and let $F : D \to D'$. F is called **monotonic** (w.r.t $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$) if, for every $d_1, d_2 \in D$

$$d_1 \sqsubseteq d_2 \implies F(d_1) \sqsubseteq' F(d_2)$$

Interpretation: monotonic function ”preserve information”

Example 11.2 (Monotonicity)

1. Let $T := \{S \subseteq \mathbb{N} \mid S \text{ finite}\}$. Then

   $$F_1 : T \to \mathbb{N} : S \mapsto \sum_{n \in S} n$$

   is monotonic w.r.t. $(2^{\mathbb{N}}, \subseteq)$ and $(\mathbb{N}, \leqslant)$
2. The function

   $$F_2 : 2^{\mathbb{N}} \to 2^{\mathbb{N}} : S \mapsto \mathbb{N} \backslash S$$

   is not monotonic w.r.t. $(2^{\mathbb{N}}, \subseteq)$ (since e.g. $\varnothing \subseteq \mathbb{N}$ but $F_2(\varnothing) = \mathbb{N} \nsubseteq F_2(\mathbb{N}) = \varnothing$)

Lemma 11.3 (Monotonicity of Φ)

Let $b \in \mathsf{BExp}$, $c \in \mathsf{Cmd}$ and $\Phi : (\Sigma \dashrightarrow \Sigma) \to (\Sigma \dashrightarrow \Sigma)$ with $\Phi(f) := \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$.
Then Φ is monotonic w.r.t. $(\Sigma \dashrightarrow \Sigma, \sqsubseteq)$.

Proof of Lemma 11.3

Let $f, g : \Sigma \dashrightarrow \Sigma$ and $\sigma, \sigma' \in \Sigma$ such that $f \sqsubseteq g$ and $\Phi(f)(\sigma) = \sigma'$.
We have to show that $\Phi(f) \sqsubseteq \Phi(g)$, i.e. that $\Phi(g)(\sigma) = \sigma'$.
To this aim, we distinguish two cases:

- $\mathfrak{B}[\![b]\!]\sigma = \mathsf{true}$:

$$\begin{aligned}\sigma' &= \Phi(f)(\sigma) \text{ (premise)}\\ &= f(\mathfrak{C}[\![c]\!]\sigma) \text{ (definition of } \Phi)\\ &= g(\mathfrak{C}[\![c]\!]\sigma) \ (f \sqsubseteq g)\\ &= \Phi(g)(\sigma) \text{ (definition of } \Phi)\end{aligned}$$

- $\mathfrak{B}[\![b]\!]\sigma = \mathsf{false}$:

$$\begin{aligned}\sigma' &= \Phi(f)(\sigma) \text{ (premise)}\\ &= \sigma \text{ (definition of } \Phi)\\ &= \Phi(g)(\sigma) \text{ (definition of } \Phi)\end{aligned}$$

Lemma 11.4

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ be CCPOs, $F : D \to D'$ monotonic, and $S \subseteq D$ a chain in D.
Then:

1. $F(S) := \{F(d) \mid d \in S\}$ is a chain in D'
2. $\bigsqcup F(S) \sqsubseteq' F(\bigsqcup S)$

Proof of Lemma 11.4

1. Given $d'_1, d'_2 \in F(S)$, there ex. $d_1, d_2 \in S$ such that $F(d_1) = d'_1$, $F(d_2) = d'_2$ and (since S is a chain) $d_1 \sqsubseteq d_2$ or $d2 \sqsubseteq d_1$.
Since F is monotonic, this implies $F(d_1) \sqsubseteq F(d_2)$ or $F(d_2) \sqsubseteq F(d_1)$ and thus $d'_1 \sqsubseteq d'_2$ or $d'_2 \sqsubseteq d'_1$, which proves the claim.
2. Since $S \sqsubseteq \bigsqcup S$ by definition, monotonicity of F implies $F(S) \overset{(*)}{\sqsubseteq} F(\bigsqcup S)$.
As $F(S)$ is a chain (1) and D' a CCPO, $\bigsqcup F(S)$ exists in D'.
By $(*)$, $F(\bigsqcup S)$ is an upper bound of $F(S)$, implying that $\bigsqcup F(S) \sqsubseteq' F(\bigsqcup S)$.

3.4.9 Continuity

A function F is continuous if applying F and taking suprema is commutable:

Definition 11.5 (Continuity)

Let $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ by CCPOs and $F : D \to D'$ monotonic. Then F is called **continuous** (w.r.t $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$) if, for every non-empty chain $S \subseteq D$,

$$F(\bigsqcup S) = \bigsqcup F(S)$$

Remark:
According to Lemma 11.4(1) (p. 45), the monotonicity of F guarantees the existence of $\bigsqcup F(S)$.

Lemma 11.6 (Continuity of Φ)

Let $b \in \mathsf{BExp}$, $c \in \mathsf{Cmd}$ and $\Phi(f) : \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$.
Then Φ is continuous w.r.t. $(\Sigma \to \Sigma, \sqsubseteq)$.

Proof of Lemma 11.6

Let $\emptyset \neq S \subseteq \Sigma \to \Sigma$ be a chain. We have to show that $\Phi(\bigsqcup S) = \bigsqcup \Phi(S)$.

- "$\bigsqcup \Phi(s) \sqsubseteq \Phi(\bigsqcup S)$":
  Follows from Lemmata 11.3 (monotonicity, p. 45) and 11.4(2) ("$\sqsubseteq$", p. 45).
- "$\Phi(\bigsqcup S) \sqsubseteq \bigsqcup \Phi(s)$":
  By Lemma 10.9 (p. 43), this is equivalent to

  $$\mathsf{graph}(\Phi(\bigsqcup S)) \subseteq \bigcup_{f \in S} \mathsf{graph}(\Phi(f))$$

  To prove this, let $(\sigma, \sigma') \in \mathsf{graph}(\Phi(\bigsqcup S))$.
  We have to determine $f \in S$ such that $\Phi(f)(\sigma) = \sigma'$.
  - If $\mathfrak{B}[\![b]\!]\sigma = \mathsf{false}$, then $\Phi(\bigsqcup S)(\sigma) = \sigma = \sigma'$ and also $\Phi(f)(\sigma) = \sigma = \sigma'$ for every $f \in S$, which proves the claim.
  - If $\mathfrak{B}[\![b]\!]\sigma = \mathsf{true}$, then $\Phi(\bigsqcup S)(\sigma) = (\bigsqcup S)(\sigma'') = \sigma'$ for $\sigma'' := \mathfrak{C}[\![c]\!]\sigma$.
    Since $\mathsf{graph}(\bigsqcup S) = \bigcup_{f \in S} \mathsf{graph}(f)$ by Lemma 10.9 (p. 43), ex. $f \in S$ such that $f(\sigma'') = \sigma'$.
    Hence, $\Phi(f)(\sigma) = f(\mathfrak{C}[\![c]\!]\sigma) = f(\sigma'') = \sigma'$, which proves the claim.

3.4.10 The Fixpoint Theorem

Theorem 12.1 (Fixpoint Theoreme by Kleene)

Let $(D, \sqsubseteq)$ be a CCPO and $F : D \to D$ continuous. Then

$$\mathsf{fix}(F) := \bigsqcup \{F^n(\bigcup \varnothing) \mid n \in \mathbb{N}\}$$

is the **least fixpoint** of F where $F^0(d) := d$ and $F^{n+1}(d) := F(F^n(d))$.

Example 12.2 (Fixpoint Theorem)

- **Domain:** $(2^{\mathbb{N}}, \subseteq)$ (CCPO with $\bigsqcup S = \bigcup_{N \in S} N$, see Example 10.7 (p. 42))
- **Function:** $F : 2^{\mathbb{N}} \to 2^{\mathbb{N}} : N \mapsto N \cup A$ for some fixed $A \subseteq \mathbb{N}$
  - F is monotonic: $M \subseteq N \implies F(M) = M \cup A \subseteq N \cup A = F(N)$
  - F is continuous: $F(\bigsqcup S) = F(\bigcup_{N \in S} N) = (\bigcup_{N \in S} N) \cup A = \bigcup_{N \in S}(N \cup A) = \bigcup_{N \in S} F(N) = \bigsqcup F(S)$
- **Fixpoint iteration:** calculate $N_n := F^n(\bigsqcup \varnothing)$ where $\bigsqcup \varnothing = \varnothing$ (least element)
  - $N_0 = \bigsqcup \varnothing = \varnothing$
  - $N_1 = F(N_0) = \varnothing \cup A = A$
  - $N_2 = F(N_1) = A \cup A = A = N_n$ for every $n \geqslant 1$

  $\implies \mathsf{fix}(F) = A$ (least $N \subseteq \mathbb{N}$ such that $N \cup A = N$)
- **Alternatively:** $F(N) := N \cap A$

  $\implies \mathsf{fix}(F) = \varnothing$ (least $N \subseteq \mathbb{N}$ such that $N \cap A = N$)

Remark: in general, the fixpoint is only reached in the limit (see Example 12.4, p. 49)

TODO: Maybe add proof of fixpoint theorem here

3.4.11 Application to fix(Φ)

Altogether this completes the definition of $\mathfrak{C}[\![.]\!]$. In particular, for the while command:

Corollary 12.3

Let $b \in \mathsf{BExp}$, $c \in \mathsf{Cmd}$ and $\Phi(f) := \mathsf{cond}(\mathfrak{B}[\![b]\!], f \circ \mathfrak{C}[\![c]\!], \mathsf{id}_\Sigma)$. Then

$$\mathsf{graph}(\mathsf{fix}(\Phi)) = \bigcup_{n \in \mathbb{N}} \mathsf{graph}(\Phi^n(f_\varnothing))$$

Proof of Corollary 12.3:

Using

- Lemma 10.9 (p. 43)
  - $(\Sigma \dashrightarrow \Sigma, \sqsubseteq)$ CCPO with least element $f_\varnothing$
  - LUB = union of graphs
- Lemma 11.6 (Φ continuous, p. 46)
- Theorem 12.1 (Fixpoint theorem, p. 47)

3.4.12 Closedness

Lemma Ex5Task3 (Closedness)

Let $(D, \sqsubseteq)$ be a CCPO. A set $C \subseteq D$ is **closed** iff for each chain $G \subseteq C$,

$$\bigsqcup G \in C$$

3.4.13 Park's Lemma

Lemma Ex5Task3.2 (Park's Lemma)

Let $(D, \sqsubseteq)$ be a CCPO and $f : D \to D$ a continuous function. Then for every $x \in D$:

$$f(x) \sqsubseteq x \text{ implies } \mathsf{fix}(f) \sqsubseteq x$$

3.4.14 Example: Denotional semantics of Factorial Program

Example 12.4 (Denotional semantics of Factorial Program)

- Let $c \in \mathsf{Cmd}$ be given by $y := 1; \mathsf{while}\ \neg(x = 1)\ \mathsf{do}\ y := y * x; x := x - 1\ \mathsf{end}$
- For every initial state $\sigma_0 \in \Sigma$, Definition 8.3 (p. 34) yields:

$$\mathfrak{C}[\![c]\!](\sigma_0) = \mathsf{fix}(\Phi)(\sigma_1)$$

where $\sigma_1 := \sigma_0[y \mapsto 1]$ and, for every $f : \Sigma \to \Sigma$ and $\sigma \in \Sigma$,

$$\begin{aligned}\Phi(f)(\sigma) &= \mathsf{cond}(\mathfrak{B}[\![\neg(x = 1)]\!], f \circ \mathfrak{C}[\![y := y * x; x := x - 1]\!], \mathsf{id}_\Sigma)(\sigma)\\ &= \begin{cases}\sigma & \text{if } \sigma(x) = 1\\ f(\sigma') & \text{otherwise}\end{cases}\end{aligned}$$

with $\sigma' := \sigma[y \mapsto \sigma(y) * \sigma(x), x \mapsto \sigma(x) - 1]$.
- Approximations of least fixpoint of Φ according to Theorem 12.1 (p. 47):

$$\mathsf{fix}(\Phi) = \bigsqcup\{\Phi^n(f_\emptyset) | n \in \mathbb{N}\}$$

(where $\mathsf{graph}(f_\emptyset) = \emptyset$)
- Performing fixpoint iteration:

$$\begin{aligned}f_0(\sigma) &:= \Phi^0(f_\emptyset)(\sigma)\\ &= f_\emptyset(\sigma)\\ &= \text{undefined}\end{aligned}$$

$$\begin{aligned}f_1(\sigma) &:= \Phi^1(f_\emptyset)(\sigma)\\ &= \Phi(f_0)(\sigma)\\ &= \begin{cases}\sigma & \text{if } \sigma(x) = 1\\ f_0(\sigma') & \text{otherwise}\end{cases}\\ &= \begin{cases}\sigma & \text{if } \sigma(x) = 1\\ \text{undefined} & \text{otherwise}\end{cases}\end{aligned}$$

(Example continues on next page)

Example 12.4 (Denotional semantics of Factorial Program)

- Continued fixpoint iteration from previous page:

$$
\begin{aligned}
f_2(\sigma) &:= \Phi^2(f_\emptyset)(\sigma) \\
&= \Phi(f_1)(\sigma) \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ f_1(\sigma') & \text{otherwise} \end{cases} \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ \sigma' & \text{if } \sigma(x) \neq 1, \sigma'(x) = 1 \\ \text{undefined} & \text{if } \sigma(x) \neq 1, \sigma'(x) \neq 1 \end{cases} \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ \sigma' & \text{if } \sigma(x) = 2 \\ \text{undefined} & \text{if } \sigma(x) \neq 1, \sigma(x) \neq 2 \end{cases} \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ \sigma[y \mapsto 2 * \sigma(y), x \mapsto 1] & \text{if } \sigma(x) = 2 \\ \text{undefined} & \text{if } \sigma(x) \neq 1, \sigma(x) \neq 2 \end{cases}
\end{aligned}
$$

(Example continues on next page)

Example 12.4 (Denotional semantics of Factorial Program)

- Continued fixpoint iteration from previous page:

$$
\begin{aligned}
f_3(\sigma) &:= \Phi^3(f_\emptyset)(\sigma) \\
&= \Phi(f_2)(\sigma) \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ f_2(\sigma') & \text{otherwise} \end{cases} \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ \sigma' & \text{if } \sigma(x) \neq 1, \sigma'(x) = 1 \\ \sigma'[y \mapsto 2 * \sigma'(y), x \mapsto 1] & \text{if } \sigma(x) \neq 1, \sigma'(x) = 2 \\ \text{undefined} & \text{if } \sigma(x) \neq 1, \sigma'(x) \neq 1, \sigma'(x) \neq 2 \end{cases} \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ \sigma' & \text{if } \sigma(x) = 2 \\ \sigma'[y \mapsto 2 * \sigma'(y), x \mapsto 1] & \text{if } \sigma(x) = 3 \\ \text{undefined} & \text{if } \sigma(x) \notin \{1, 2, 3\} \end{cases} \\
&= \begin{cases} \sigma & \text{if } \sigma(x) = 1 \\ \sigma[y \mapsto 2 * \sigma(y), x \mapsto 1] & \text{if } \sigma(x) = 2 \\ \sigma[y \mapsto 3 * 2 * \sigma(y), x \mapsto 1] & \text{if } \sigma(x) = 3 \\ \text{undefined} & \text{if } \sigma(x) \notin \{1, 2, 3\} \end{cases}
\end{aligned}
$$

(Example continues on next page)

Example 12.4 (Denotional semantics of Factorial Program)

- Continued example from previous page:
- n-th approximation:

$$\begin{aligned} f_n(\sigma) &:= \Phi^n(f_\varnothing)(\sigma) \\ &= \begin{cases} \sigma[y \mapsto \sigma(x) * (\sigma(x)-1) * \ldots * 2 * \sigma(y), x \mapsto 1] & \text{if } 1 \leqslant \sigma(x) \leqslant n \\ \text{undefined} & \text{if } \sigma(x) \notin \{1, \ldots, n\} \end{cases} \\ &= \begin{cases} \sigma[y \mapsto (\sigma(x))! * \sigma(y), x \mapsto 1] & \text{if } 1 \leqslant \sigma(x) \leqslant n \\ \text{undefined} & \text{if } \sigma(x) \notin \{1, \ldots, n\} \end{cases} \end{aligned}$$

- Fixpoint:

$$\mathfrak{C}[\![c]\!](\sigma_0) = \mathsf{fix}(\Phi)(\sigma_1) = \begin{cases} \sigma[y \mapsto (\sigma(x))! * \sigma(y), x \mapsto 1] & \text{if } \sigma(x) \geqslant 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

4 Equivalence of operational and denotional semantics

Theorem 13.1 (Coincidence Theorem)

For every $c \in \mathsf{Cmd}$,

$$\mathfrak{O}[\![c]\!] = \mathfrak{C}[\![c]\!]$$

i.e. $\langle c, \sigma\rangle \to \sigma'$ iff $\mathfrak{C}[\![c]\!](\sigma) = \sigma'$, and thus $\mathfrak{O}[\![.]\!] = \mathfrak{C}[\![.]\!]$.

The proof of Theorem 13.1 employs the following axiliary propositions:

Lemma 13.2

1. For every $a \in \mathsf{AExp}$, $\sigma \in \Sigma$ and $z \in \mathbb{Z}$:

$$\langle a, \sigma\rangle \to z \iff \mathfrak{A}[\![a]\!](\sigma) = z$$

2. For every $b \in \mathsf{BExp}$, $\sigma \in \Sigma$ and $t \in \mathbb{B}$:

$$\langle b, \sigma\rangle \to t \iff \mathfrak{B}[\![b]\!](\sigma) = t$$

Proof of Lemma 13.2

TODO: Both via structural induction on a/b, see exercises

Proof of Theorem 13.1

TODO (see L13 pages 8 and 9)

5 Axiomatical Semantics of WHILE

5.1 Idea

Example 14.1

- Let $c \in \mathsf{Cmd}$ be given by

$$s := 0; n := 1; \mathsf{while}\ \neg(n > N)\ \mathsf{do}\ s := s + n; s := n + 1\ \mathsf{end}$$

- How to show that, after termination of c in state σ,

$$\sigma(s) = \sum_{k=1}^{\sigma(N)} k$$

- "Running" c according to the operational semantics is insufficient: every change of $\sigma(N)$ requires a **new proof**
- Wanted: a more abstract, "**symbolic**" way of reasoning

Obviously c satisfies the following **assertions** (after execution of the respective statement):

$$s := 0;$$
$$\{s = 0\}$$
$$n := 1;$$
$$\{s = 0 \wedge n = 1\}$$
$$\mathsf{while}\ \neg(n > N)\ \mathsf{do}\ s := s + n; n := n + 1\ \mathsf{end}$$
$$\{s = \sum_{k=1}^{n} k \wedge n > N\}$$

where, e.g. "$s = 0$" means "$\sigma(s) = 0$ in the current state $\sigma \in \Sigma$"

How to prove the **validity** of assertions?

- Assertions following **assignments** are evident ("$s = 0$")
- Also, "$n > N$" follows directly from the loop's **execution condition**
- But how to obtain the final value of s?
- Answer: at the loop's header, the **invariant** $s = \sum_{k=1}^{n-1} k$ is satisfied
  - holds initially
  - preserved by loop iterations
- Goal: establish such assertions by a **proof system**
- Employs **partial correctness properties** of the form $\{A\}c\{B\}$ with assertions A, B and $c \in \mathsf{Cmd}$

- Interpretation depends on expected termination behaviour of c:
  partial correctness: nothing is said about c if it fails to terminate
  total correctness: c terminates on all inputs satisfying $\{A\}$

5.2 The Assertion Language

5.2.1 Syntax of assertions

Assertions = Boolean expressions + **quantification over (additional) variables**

- to memorise previous values of program variables
- to formulate more involved state properties
- usually no occuring in program (use $i, k, ...$)

Definition 14.2 (Syntax of assertions)

The **syntax of Assn** is defined by the following context-free grammar:

$$a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in \mathsf{AExp} \textbf{ (as before)}$$
$$A ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \forall i.A \in \mathsf{Assn}$$

Thus: $\mathsf{BExp} \subsetneq \mathsf{Assn}$.

The following (and other) **abbreviations** will be employed:

$$A_1 \implies A_2 := \neg A_1 \vee A_2$$
$$\exists i.A := \neg(\forall i.\neg A)$$
$$a_1 \geqslant a_2 := a_1 > a_2 \vee a_1 = a_2$$
$$\vdots$$

5.2.2 Semantics of Assertions

- Formalized by a **satisfaction relation** of the form $\sigma \models A$ (where $\sigma \in \Sigma$ and $A \in \mathsf{Assn}$)
- Non-terminating computations captured by **undefined state** $\bot$

Definition 14.3 (Semantics of assertions)

Let $A \in \mathsf{Assn}$ and $\sigma \in \Sigma$. The relation "σ **satisfies** A" (notation $\sigma \models A$) is inductively defined by:

$$\begin{array}{lll}
\sigma & \models \mathsf{true} & \\
\sigma & \models a_1 = a_2 & \text{if } \mathfrak{A}[\![a_1]\!]\sigma = \mathfrak{A}[\![a_2]\!]\sigma \\
\sigma & \models a_1 > a_2 & \text{if } \mathfrak{A}[\![a_1]\!]\sigma > \mathfrak{A}[\![a_2]\!]\sigma \\
\sigma & \models \neg A & \text{if not } \sigma \models A \\
\sigma & \models A_1 \wedge A_2 & \text{if } \sigma \models A_1 \text{ and } \sigma \models A_2 \\
\sigma & \models A_1 \vee A_2 & \text{if } \sigma \models A_1 \text{ or } \sigma \models A_2 \\
\sigma & \models \forall i.A & \text{if } \sigma[i \mapsto z] \models A \text{ for every } z \in \mathbb{Z}
\end{array}$$

Furthermore, we let $[\![A]\!] := \{\sigma \in \Sigma \mid \sigma \models A\}$ ("semantics of formula A" or "all models of formula A"). A is called **valid** ($\models A$) if $[\![A]\!] = \Sigma$.

Example 14.4 (Semantics of assertions)

The following assertion expresses that, in the current state $\sigma \in \Sigma$, $\sigma(y)$ is the greatest divisor of $\sigma(x)$ (excluding $\sigma(x)$):

$$\underbrace{(\exists i.i > 1 \wedge i * x = x)}_{y \text{ divides } x} \wedge \underbrace{\forall j.\forall k.(j > 1 \wedge j * k = x \implies k \leqslant y)}_{y \text{ is maximal}}$$

Together with the fact that $\mathsf{BExp} \subseteq \mathsf{Assn}$, Definition 8.2 (denotional semantics of Boolean expressions, p. 32) yields:

Corollary 14.5

For every $b \in \mathsf{BExp}$ and $\sigma \in \Sigma$:

$$\sigma \models b \iff \mathfrak{B}[\![b]\!]\sigma = \mathsf{true}$$

5.3 Partial Correctness

5.3.1 Partial Correctness Properties

Definition 15.1 (Partial correctness properties)

Let $A, B \in \mathsf{Assn}$ and $c \in \mathsf{Cmd}$.

- An expression of the form

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

  is called a **partial correctness property (PCP)** with **precondition** A and **postcondition** B.
- Given $\sigma \in \Sigma$, we let

$$\sigma \models \{A\}c\{B\}$$

  if $\sigma \models A$ implies that $\mathfrak{C}[\![c]\!]\sigma \models B$ or $\mathfrak{C}[\![c]\!]\sigma = \bot$.
- $\{A\}c\{B\}$ is called **valid** (notation: $\models \{A\}c\{B\}$) if $\sigma \models \{A\}c\{B\}$ for every $\sigma \in \Sigma$.

Example 15.2 (Partial correctness properties)

- Let $x, i \in \mathsf{Var}$. We have to show:

$$\models \{i \leqslant x\}x := x + 1\{i < x\}$$

- According to Definition 15.1 (p. 58), this is equivalent to

$$\sigma \models \{i \leqslant x\}x := x + 1\{i < x\}$$

  for every $\sigma \in \Sigma$, which is entailed by the following implications:

$$\begin{aligned} & \sigma \models (i \leqslant x) \\ \Longrightarrow\ & \mathfrak{A}[\![i]\!]\sigma \leqslant \mathfrak{A}[\![x]\!]\sigma \text{ (Definition 14.3)} \\ \Longrightarrow\ & \sigma(i) \leqslant \sigma(x) \text{ (Definition 8.1)} \\ \Longrightarrow\ & \sigma(i) < \sigma(x) + 1 = (\mathfrak{C}[\![x := x + 1]\!]\sigma)(x) \\ \Longrightarrow\ & (\mathfrak{C}[\![x := x + 1]\!]\sigma) \models (i < x) \\ \Longrightarrow\ & \text{claim} \end{aligned}$$

5.4 Hoare Logic

Goal: syntactic derivation of valid partial correctness properties. Here $A[x \mapsto a]$ denotes the syntactic replacement of every free occurence of x by a in A.

Definition 15.3 (Hoare Logic)

The **Hoare rules** are given by

$$\text{(skip)}\ \frac{}{\{A\}\mathsf{skip}\{A\}} \qquad \text{(if)}\ \frac{\{A \wedge b\}c_1\{B\} \qquad \{A \wedge \neg b\}c_2\{B\}}{\{A\}\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}\{B\}}$$

$$\text{(asgn)}\ \frac{}{\{A[x \mapsto a]\}x := a\{A\}}$$

$$\text{(seq)}\ \frac{\{A\}c_1\{C\} \qquad \{C\}c_2\{B\}}{\{A\}c_1; c_2; \{B\}} \qquad \text{(while)}\ \frac{\{A \wedge b\}c\{A\}}{\{A\}\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\{A \wedge \neg b\}}$$

$$\text{(cons)}\ \frac{\models (A \implies A') \qquad \{A'\}c\{B'\} \qquad \models (B' \implies B)}{\{A\}c\{B\}}$$

A partial correctness property is **provable** (notation: $\vdash \{A\}c\{B\}$) if it is derivable by the Hoare rules. In rule (while), A is called a **(loop) invariant**.

Example 15.4 (Factorial program in Hoare Logic)

Proof of $\{A\}c\{B\}$ where $A := (x > 0 \wedge x = i)$, $B := (y = i!)$ and c given by:

$\{x > 0 \wedge x = i\} \implies$

$\{C[y \mapsto 1]\}$

$y := 1;$

$\{C\}$

while $\neg(x = 1)$ do

$\{\neg(x = 1) \wedge C\} \implies$

$\{C[x \mapsto x - 1, y \mapsto y * x]\}$

$y := y * x;$

$\{C[x \mapsto x - 1]\}$

$x := x - 1$

$\{C\}$

end

$\{\neg\neg(x = 1) \wedge C\} \implies$

$\{y = i!\}$

5.4.1 Discovering invariants

Goal: Prove PCP $\{A\}\textsf{while } b \textsf{ do } c \textsf{ end}\{B\}$ by identifying invariant C:

$$\text{(while)}\ \frac{\{C \land b\}c\{C\}}{\{C\}\textsf{while } b \textsf{ do } c \textsf{ end}\{C \land \neg b\}}$$

This may require some ingenuity, but there are a few hints on how to do that:

- In general, there are several invariants but most of them are useless (for example, true is always an invariant)
- A suitable invariant has to be
  - **weak enough** to be implied by the precondition: $\models (A \implies C)$
  - **strong enough** to imply the postcondition: $\models (C \land \neg b \implies B)$
- In general, looking at the **logical structure of the postcondition** will help
- Often a suitable invariant is found by **generalising the postcondition**, replacing a constant by a variable that is changed in the body of the loop
- It can be helpful to **"trace" the loop** and inspect the values of the variables at every iteration

Example 15.5 (Invariant)

1. $\{y \geqslant 0 \land y = i\}z := 1; \textsf{while } \neg(y = 0) \textsf{ do } y := y - 1; z := z * x \textsf{ end}\{z = x^i\}$
   - Invariant: $C = (z = x^{i-y})$
   - Precondition: $y \geqslant 0 \land y = i \land z = 1 \implies C$
   - Postcondition: $C \land y = 0 \implies z = x^i$
2. $\{x \geqslant 0 \land y > 0 \land x = i\}z := 0; \textsf{while } y <= x \textsf{ do } x := x - y; z := z + 1 \textsf{ end}\{i = z * y + x\}$
   - Invariant: $C = (i = z * x + x)$
   - Precondition: $x \geqslant 0 \land y > 0 \land x = i \land z = 0 \implies C$
   - Postcondition: $C \land y > x \implies i = z * y + x$

5.4.2 Soundness

Soundness: no wrong propositions can be derived, i.e. every (syntactically) provable partial correctness property is also (semantically) valid.

For the corresponding proof we use:

Lemma 16.1 (Substitution lemma)

For every $A \in \mathsf{Assn}$, $x \in \mathsf{Var}$, $a \in \mathsf{AExp}$ and $\sigma \in \Sigma$:

$$\sigma \models A[x \mapsto a] \iff \sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma] \models A$$

Proof by structural induction over $A \in \mathsf{Assn}$ (omitted)

Theorem 16.2 (Soundness of Hoare Logic)

For every partial correctness property $\{A\}c\{B\}$,

$$\vdash \{A\}c\{B\} \implies \models \{A\}c\{B\}$$

Proof of Theorem 16.2:

Let $\vdash \{A\}c\{B\}$. By induction over the structure of the corresponding proof tree we show that, for every $\sigma \in \Sigma$ with $\sigma \models A$, $\mathfrak{C}[\![c]\!]\sigma = \bot$ or $\mathfrak{C}[\![c]\!]\sigma \models B$.

- Case $\text{(skip)}\ \dfrac{}{\{A\}\mathsf{skip}\{A\}}$ (i.e. $c = \mathsf{skip}$, $B = A$):
  $\sigma \models A$ implies $\mathfrak{C}[\![c]\!]\sigma = \sigma \models A = B$.
- Case $\text{(asgn)}\ \dfrac{}{\{B[x \mapsto a]\}x := a\{B\}}$ (i.e. $c = (x := a)$, $A = B[x \mapsto a]$):
  $\sigma \models B[x \mapsto a]$ implies $\mathfrak{C}[\![c]\!]\sigma = \sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma] \models B$ (Lemma 16.1).
- Case $\text{(seq)}\ \dfrac{\{A\}c_1\{C\} \quad \{C\}c_2\{B\}}{\{A\}c_1; c_2\{B\}}$ (i.e. $c = c_1; c_2$):
  The induction hypothesis for $\{A\}c_1\{C\}$ and $\{C\}c_2\{B\}$ respectively yields $\models \{A\}c_1\{C\}$ and $\models \{C\}c_2\{B\}$, such that $\mathfrak{C}[\![c_1]\!]\underbrace{\sigma}_{\models A} \models C$ or $\mathfrak{C}[\![c_1]\!]\sigma = \bot$.
  In the second case, $\mathfrak{C}[\![c]\!]\sigma = \bot$. Otherwise, $\mathfrak{C}[\![c]\!]\sigma = \mathfrak{C}[\![c_2]\!](\underbrace{\mathfrak{C}[\![c_1]\!]\sigma}_{\models C}) \models B$ (or $= \bot$).
- Case $\text{(if)}\ \dfrac{\{A \wedge b\}c_1\{B\} \quad \{A \wedge \neg b\}c_2\{B\}}{\{A\}\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}\{B\}}$ (i.e. $c = \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}$):
  If $\mathfrak{B}[\![b]\!]\sigma = \mathsf{true}$, then $\sigma \models A$ and Corollary 14.5 (p. 57) imply that $\sigma \models A \wedge b$.
  By induction hypothesis, $\mathfrak{C}[\![c]\!]\sigma = \mathfrak{C}[\![c_1]\!]\sigma \models B$ (or $= \bot$).
  The case for $\mathfrak{B}[\![b]\!]\sigma = \mathsf{true}$ is analogous.
- Case $\text{(while)}\ \dfrac{\{A \wedge b\}c_0\{A\}}{\{A\}\mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end}\{A \wedge \neg b\}}$ (i.e. $c = \mathsf{while}\ b\ \mathsf{do}\ c_0\ \mathsf{end}$, $B = A \wedge \neg b$):

Here $\mathfrak{C}[\![c]\!]\sigma = \mathsf{fix}(\Phi)(\sigma)$ where $\Phi(f)(\sigma) = \begin{cases} f(\mathfrak{C}[\![c_0]\!]\sigma & \text{if } \mathfrak{B}[\![b]\!]\sigma = \text{true} \\ \sigma & \text{otherwise} \end{cases}$

If $\mathfrak{C}[\![c]\!]\sigma \neq \bot$, then there ex. $\sigma' \in \Sigma$ and $n \geqslant 1$ such that $\mathfrak{C}[\![c]\!]\sigma = \Phi^n(f_\varnothing)(\sigma) = \sigma'$.

By complete induction over n, it follows that $\sigma' \models A \wedge \neg b$.

- Case (cons) $\dfrac{\models (A \implies A') \qquad \{A'\}c\{B'\} \qquad \models (B' \implies B)}{\{A\}c\{B\}}$

  Here $\sigma \models A$ implies $\sigma \models A'$, such that the induction hypothesis yields $\mathfrak{C}[\![c]\!]\sigma \models B'$ (or $= \bot$). In the first case, also $\mathfrak{C}[\![c]\!]\sigma \models B$.

5.5 Completeness

5.5.1 Incompleteness

Theorem 16.3 (Gödel's Incompleteness Theorem)

The set of all valid assertions

$$\{A \in \mathsf{Assn} \mid \models A\}$$

is not recursively enumerable, i.e. there exists no proof system for Assn in which all valid assertions are systematically derivable.

Corollary 16.4

There is no proof system in which all valid partial correctness properties can be enumerated.

Proof of Corollary 16.4:

Given $A \in \mathsf{Assn}$, $\models A$ is obviously equivalent to $\{\mathsf{true}\}\mathsf{skip}\{A\}$. Thus the enumerability of all valid partial correctness properties would imply the enumerability of all valid assertions.

5.5.2 Relative Completeness

Theorem 17.1 (Cook's Completeness Theorem)

Hoare Logic is **relatively complete**, i.e. for every partial correctness property $\{A\}c\{B\}$:

$$\models \{A\}c\{B\} \implies \vdash \{A\}c\{B\}$$

Thus: if we know that a partial correctness property is valid, then we know that there is a corresponding proof.

Proof of Theorem 17.1:

We have to show that Hoare Logic is relative complete, i.e. that

$$\models \{A\}c\{B\} \implies \vdash \{A\}c\{B\}$$

Proof:

- Lemma 17.8 (p. 67): $\vdash \{A_c, B\}c\{B\}$
- Corollary 17.3 (p. 65): $\models \{A\}c\{B\} \implies \models (A \implies A_{c,B})$
- $\text{(cons)}\ \dfrac{\models (A \implies A_{c,B}) \qquad \{A_{c,B}\}c\{B\} \qquad \models (B \implies B)}{\{A\}c\{B\}}$

5.6 Weakest liberal precondition

Definition 17.2 (Weakest liberal precondition)

Given $c \in \mathsf{Cmd}$ and $S \subseteq \Sigma$, the **weakest (liberal) precondition** of S with respect to c collects all states σ such that running c in σ does not terminate or yields a state in S:

$$\mathsf{wlp}[\![c]\!](S) := \{\sigma \in \Sigma \mid \mathfrak{C}[\![c]\!]\sigma \in S \cup \{\bot\}\}$$

Corollary 17.3

For every $c \in \mathsf{Cmd}$ and $A, B \in \mathsf{Assn}$:

1. $\models \{A\}c\{B\} \iff [\![A]\!] \subseteq \mathsf{wlp}[\![c]\!]([\![B]\!])$
2. If $A_0 \in \mathsf{Assn}$ such that $[\![A_0]\!] = \mathsf{wlp}[\![c]\!]([\![B]\!])$, then

$$\models \{A\}c\{B\} \iff \models (A \implies A_0)$$

Remarks:

- Corollary 17.3 justifies the notion of **weakest** precondition:
  it is entailed by every precondition A that makes $\{A\}c\{B\}$ valid.
- Here, pre- and postconditions are understood as **semantic predicates** $S, \mathsf{wlp}[\![c]\!](S) \subseteq \Sigma$ (”extensional” approach - later: ”intensional” approach by ”syntactification”)

Lemma 17.4 (Weakest liberal precondition transformer)

Weakest liberal preconditions $\mathsf{wlp}[\![.]\!]() : \mathsf{Cmd} \to (2^\Sigma \to 2^\Sigma)$ can be computed as follows:

$$\begin{aligned}
\mathsf{wlp}[\![\mathsf{skip}]\!](S) &= S \\
\mathsf{wlp}[\![x := a]\!](S) &= \{\sigma \in \Sigma \mid \sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma] \in S\} \\
\mathsf{wlp}[\![c_1; c_2;]\!](S) &= \mathsf{wlp}[\![c_1]\!](\mathsf{wlp}[\![c_2]\!](S)) \\
\mathsf{wlp}[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}]\!](S) &= ([\![b]\!] \cap \mathsf{wlp}[\![c_1]\!](S)) \cup ([\![\neg b]\!] \cap \mathsf{wlp}[\![c_2]\!](S)) \\
\mathsf{wlp}[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!](S) &= \mathsf{FIX}(\Psi)
\end{aligned}$$

where $\mathsf{FIX}(\Psi)$ denotes the greates fixpoint (w.r.t $(2^\Sigma, \subseteq)$) of

$$\Psi : 2^\Sigma \to 2^\Sigma : T \mapsto ([\![b]\!] \cap \mathsf{wlp}[\![c]\!](T)) \cup ([\![\neg b]\!] \cap (S))$$

Remark: $\mathsf{FIX}(\Psi)$ of function Ψ on $(2^\Sigma, \subseteq)$ can be computed by fixpoint iteration starting from the greatest element $\bigsqcap \varnothing$.

Example 17.5

Using Lemma 17.4, we want to determine the weakest liberal precondition for

$$\{?\}\underbrace{\mathsf{while}\ x \neq 0 \wedge x \neq 1\ \mathsf{do}\ \overbrace{x := x - 2}^{c_0}\ \mathsf{end}}_{c}\{x = 1\}$$

i.e. $\mathsf{wlp}[\![c]\!](S)$ for $S := [\![x = 1]\!] = \{\sigma \in \Sigma \mid \sigma(x) = 1\}$.

- $\mathsf{wlp}[\![c]\!](S) = \mathsf{FIX}(\Psi)$ for $\Psi(T) = ([\![x \notin \{0,1\}]\!] \cap \mathsf{wlp}[\![c_0]\!](T)) \cup \underbrace{([\![x \in \{0,1\}]\!] \cap S)}_{=S}$
- $\mathsf{wlp}[\![c_0]\!](T) = \{\sigma \in \Sigma \mid \sigma[x \mapsto \sigma(x) - 2] \in T\}$
- Fixpoint iteration (with initial value $\bigsqcap \varnothing = \Sigma$):

$$\begin{aligned}
\Psi(\Sigma) &= ([\![x \notin \{0,1\}]\!] \cap \mathsf{wlp}[\![c_0]\!](\Sigma)) \cup S = [\![x \neq 0]\!] \\
\Psi^2(\Sigma) &= ([\![x \notin \{0,1\}]\!] \cap \mathsf{wlp}[\![c_0]\!]([\![x \neq 0]\!])) \cup S = [\![x \neq 0 \wedge x \neq 2]\!] \\
\Psi^3(\Sigma) &= ([\![x \notin \{0,1\}]\!] \cap \mathsf{wlp}[\![c_0]\!]([\![x \neq 0 \wedge x \neq 2]\!])) \cup S = [\![x \neq 0 \wedge x \neq 2 \wedge x \neq 4]\!] \\
&\vdots
\end{aligned}$$

$$\implies \mathsf{FIX}(\Psi) = \bigcap_{n \in \mathbb{N}} \Psi^n(\Sigma) = \{\sigma \in \Sigma \mid \sigma(x) \in \mathbb{Z}_{<0} \cup \{1, 3, 5, ...\}\}$$

The following Lemma shows that syntactic weakest preconditions are "provable":

Lemma 17.8

For every $c \in \mathsf{Cmd}$ and $B \in \mathsf{Assn}$:

$$\vdash \{A_{c,B}\}c\{B\}$$

The proof of Lemma 17.8 is done by structural induction over c (omitted).

5.7 Expressivity

Definition 17.6 (Expressivity of assertion languages)

An assertion language Assn is called **expressive** if it allows to "syntactify" weakest precondition, that is, for every $c \in \mathsf{Cmd}$ and $B \in \mathsf{Assn}$, there exists $A_{c,B} \in \mathsf{Assn}$ such that $[\![A_{c,B}]\!] = \mathsf{wlp}[\![c]\!]([\![B]\!])$.

Theorem 17.7 (Expressivity of Assn)

Assn is expressive.

Proof of Theorem 17.7:

Given $c \in \mathsf{Cmd}$ and $B \in \mathsf{Assn}$, construct $A_{c,B} \in \mathsf{Assn}$ with
$\sigma \models A_{c,B} \iff \mathfrak{C}[\![c]\!]\sigma \models B$ (for every $\sigma \in \Sigma$). For example:

$$\begin{aligned} A_{\mathsf{skip},B} &:= B \\ A_{c_1;c_2,B} &:= A_{c_1,A_{c_2,B}} \\ A_{x:=a,B} &:= B[x \mapsto a] \\ &\vdots \end{aligned}$$

(for while : "Gödelisation" of sequences of intermediate states)

Lemma 17.9 (Unexpressiveness of BExp)

BExp (i.e. Assn without quantification over variables) is **not expressive**.

Proof of Lemma 17.9:

Let us assume that BExp is expressive. According to Definition 17.6, for every $c \in \mathsf{Cmd}$ there exists $b_c \in \mathsf{BExp}$ such that $[\![b_c]\!] = \mathsf{wlp}[\![c]\!]([\![\mathsf{false}]\!]) = \mathsf{wlp}[\![c]\!](\varnothing)$.

But: for every $\sigma \in \Sigma$, $\sigma \models b_c$ iff $\mathfrak{C}[\![c]\!]\sigma = \bot$

- $\sigma \models b_c$ easily checkable (by evaluation $\mathfrak{B}[\![b_c]\!]$)
- $\mathfrak{C}[\![c]\!]\sigma = \bot$ undecidable (halting problem)

which is clearly a contradiction.

5.8 Total Correctness

5.8.1 Semantics of total correctness properties

Definition 18.1 (Semantics of total correctness properties)

Let $A, B \in \mathsf{Assn}$ and $c \in \mathsf{Cmd}$.

- $\{A\}c\{\downarrow B\}$ is called **valid in** $\sigma \in \Sigma$ (notation: $\sigma \models \{a\}c\{\downarrow B\}$) if $\sigma \models A$ implies that $\mathfrak{C}[\![c]\!]\sigma \models B$.
- $\{A\}c\{\downarrow B\}$ is called **valid** (notation: $\models \{A\}c\{\downarrow B\}$) if $\sigma \models \{A\}c\{\downarrow B\}$ for every $\sigma \in \Sigma$.

Obviously, total implies partial correctness (but not vice versa):

Corollary 18.2

For all $A, B \in \mathsf{Assn}$ and $c \in \mathsf{Cmd}$,

$$\models \{A\}c\{\downarrow B\} \implies \models \{A\}c\{B\}$$

5.8.2 Hoare Logic for Total Correctness

Definition 18.3 (Hoare Logic for total correctness)

The **Hoare rules for total correctness** are given by (where $i \in \mathsf{Var}$)

$$\text{(skip)}\ \frac{}{\{A\}\mathsf{skip}\{\downarrow A\}} \qquad \text{(seq)}\ \frac{\{A\}c_1\{\downarrow C\} \qquad \{C\}c_2\{\downarrow B\}}{\{A\}c_1; c_2\{\downarrow B\}}$$

$$\text{(asgn)}\ \frac{}{\{A[x \mapsto a]\}x := a\{\downarrow A\}} \qquad \text{(if)}\ \frac{\{A \wedge b\}c_1\{\downarrow B\} \qquad \{A \wedge \neg b\}c_2\{\downarrow B\}}{\{A\}\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}\{\downarrow B\}}$$

$$\text{(while)}\ \frac{\models (i \geqslant 0 \wedge A(i+1) \implies b) \qquad \{i \geqslant 0 \wedge A(i+1)\}c\{\downarrow A(i)\} \qquad \models (A(0) \implies \neg b)}{\{\exists i. i \geqslant 0 \wedge A(i)\}\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\{\downarrow A(0)\}}$$

$$\text{(cons)}\ \frac{\models (A \implies A') \qquad \{A'\}c\{\downarrow B'\} \qquad \models (B' \implies B)}{\{A\}c\{\downarrow B\}}$$

A total correctness property is **provable** (notation: $\vdash \{A\}c\{\downarrow B\}$) if it is derivable by the Hoare rules. In case of (while), $A(i)$ is called a **(loop) invariant**.

5.8.3 Proving Total Correctness

- In rule

$$(\text{while})\ \frac{\models (i \geqslant 0 \wedge A(i+1) \implies b) \qquad \{i \geqslant 0 \wedge A(i+1)\}c\{\downarrow A(i)\} \qquad \models (A(0) \implies \neg b)}{\{\exists i. i \geqslant 0 \wedge A(i)\}\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\{\downarrow A(0)\}}$$

  the notation $A(i)$ indicates that assertion A **parametrically depends** on the value of variable $i \in \mathsf{Var}$.

- Idea: i represents the **remaining number of loop iterations**

- Loop to be traversed $i+1$ times ($i \geqslant 0$)
  $\implies A(i+1)$ holds
  $\implies$ execution condition b satisfied
  Thus: $\models (i \geqslant 0 \wedge A(i+1) \implies b)$, and $i+1$ decreased to i by execution of c

- Execution terminated
  $\implies A(0)$ holds
  $\implies$ execution condition b violated
  Thus: $\models (A(0) \implies \neg b)$

5.8.4 Example: Total Correctness of Factorial Program

Example 18.4 (Total Correctness of factorial program)

Proof of $\{A\}c\{\downarrow B\}$ where $A := (x > 0 \wedge x = i)$, $B := (y = i!)$ and c given below (with loop invariant $C(j) := (x > 0 \wedge y * x! = i! \wedge j = x - 1)$; all correctness properties total):

$\{x > 0 \wedge x = i\} \implies$

$\{\exists j.j \geqslant 0 \wedge C(j)[y \mapsto 1]\}$

$y := 1;$

$\{\exists j.j \geqslant 0 \wedge C(j)\}$

while $\neg(x = 1)$ do

$\{j \geqslant 0 \wedge C(j + 1)\} \implies$

$\{j \geqslant 0 \wedge C(j)[x \mapsto x - 1, y \mapsto y * x]\}$

$y := y * x;$

$\{j \geqslant 0 \wedge C(j)[x \mapsto x - 1]\}$

$x := x - 1$

$\{j \geqslant 0 \wedge C(j)\} \implies \{C(j)\}$

end

$\{C(0)\} \implies \{y = i!\}$

5.8.5 Soundness of Hoare Logic for TCP

Theorem 18.5 (Soundness of Hoare Logic for TCP)

For every total correctness property $\{A\}c\{\downarrow B\}$,

$$\vdash \{A\}c\{\downarrow B\} \implies \models \{A\}c\{\downarrow B\}$$

Proof by structural induction over the derivation $\vdash \{A\}c\{\downarrow B\}$ (only (while) case): TODO add the proof (L18 P11)

5.8.6 Relative Completeness of Hoare Logic for TCP

Theorem 18.5 (Relative Completeness of Hoare Logic for TCP)

The Hoare Logic for total correctness properties is **relatively complete**, i.e. for every $\{A\}c\{\Downarrow B\}$:

$$\models \{A\}c\{\Downarrow B\} \implies \vdash \{A\}c\{\Downarrow B\}$$

5.9 Weakest total precondition

Definition 18.6 (Weakest (total) precondition)

Given $c \in \mathsf{Cmd}$ and $S \subseteq \Sigma$, the **weakest (total) precondition** of S with respect to c collects all states σ such that executing c in σ terminates and yields a state in S:

$$\mathsf{wp}[\![c]\!](S) := \{\sigma \in \Sigma \mid \mathfrak{C}[\![c]\!]\sigma \in S\}$$

Lemma 18.7

For every $c \in \mathsf{Cmd}$ and $A, B \in \mathsf{Assn}$:

1. $\models \{A\}c\{\downarrow B\} \iff [\![A]\!] \subseteq \mathsf{wp}[\![c]\!]([\![B]\!])$
2. If $A_0 \in \mathsf{Assn}$ such that $[\![A_0]\!] = \mathsf{wp}[\![c]\!]([\![B]\!])$, then $\models \{A\}c\{\downarrow B\} \iff \models (A \implies A_0)$.
3. Assn is expressive also w.r.t. weakest total preconditions, that is, there exists $A_{c,B} \in \mathsf{Assn}$ such that $[\![A_{c,B}]\!] = \mathsf{wp}[\![c]\!]([\![B]\!])$.

Lemma 18.8 (Weakest precondition transformer)

Weakest preconditions $\mathsf{wp}[\![.]\!](.) : \mathsf{Cmd} \to (2^\Sigma \to 2^\Sigma)$ can be computed as follows:

$$\begin{aligned}
\mathsf{wp}[\![\mathsf{skip}]\!](S) &= S \\
\mathsf{wp}[\![x := a]\!](S) &= \{\sigma \in \Sigma \mid \sigma[x \mapsto \mathfrak{A}[\![a]\!]\sigma] \in \Sigma\} \\
\mathsf{wp}[\![c_1; c_2]\!](S) &= \mathsf{wp}[\![c_1]\!](\mathsf{wp}[\![c_2]\!](S)) \\
\mathsf{wp}[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}]\!](S) &= ([\![b]\!] \cap \mathsf{wp}[\![c_1]\!](S)) \cup ([\![\neg b]\!] \cap \mathsf{wp}[\![c_2]\!](S)) \\
\mathsf{wp}[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!](S) &= \mathsf{fix}(\Psi)
\end{aligned}$$

where $\mathsf{fix}(\Psi)$ denotes the least fixpoint (w.r.t. $(2^\Sigma, \subseteq)$) of

$$\Psi : 2^\Sigma \to 2^\Sigma : T \mapsto ([\![b]\!] \cap \mathsf{wp}[\![c]\!](T)) \cup ([\![\neg b]\!] \cap S)$$

Example 18.9

Using Lemma 18.8, we want to determine the weakest precondition for

$$\{?\}\underbrace{\mathsf{while}\ x \neq 0 \wedge x \neq 1\ \mathsf{do}\ \overbrace{x := x - 2}^{c_0}\ \mathsf{end}}_{c}\{x = 1\}$$

i.e. $\mathsf{wp}[\![c]\!](S)$ for $S := [\![x = 1]\!] = \{\sigma \in \Sigma \mid \sigma(x) = 1\}$.

- $\mathsf{wp}[\![c]\!](S) = \mathsf{fix}(\Psi)$ for $\Psi(T) = ([\![x \notin \{0,1\}]\!] \cap \mathsf{wp}[\![c_0]\!](T)) \cup \underbrace{([\![x \in \{0,1\}]\!] \cap S)}_{=S}$
- $\mathsf{wp}[\![c_0]\!](T) = \{\sigma \in \Sigma \mid \sigma[x \mapsto \sigma(x) - 2] \in T\}$
- Fixpoint iteration (with initial value $\bigsqcup \varnothing = \varnothing$):

$$\begin{aligned}
\Psi(\varnothing) &= ([\![x \notin \{0,1\}]\!] \cap \mathsf{wp}[\![c_0]\!](\varnothing)) \cup S = [\![x = 0]\!] \\
\Psi^2(\varnothing) &= ([\![x \notin \{0,1\}]\!] \cap \mathsf{wp}[\![c_0]\!]([\![x = 1]\!])) \cup S = [\![x \in \{1,3\}]\!] \\
\Psi^3(\varnothing) &= ([\![x \notin \{0,1\}]\!] \cap \mathsf{wp}[\![c_0]\!]([\![x \in \{1,3\}]\!])) \cup S = [\![x \in \{1,3,5\}]\!] \\
&\vdots
\end{aligned}$$

$\Longrightarrow \mathsf{fix}(\Psi) = \bigcup_{n \in \mathbb{N}} \Psi^n(\varnothing) = \{\sigma \in \Sigma \mid \sigma(x) \in \{1, 3, 5, ...\}\}$

5.10 Axiomatic Equivalence

In the axiomatic semantics, two statements have to be considered equivalent if they are **inditinguishable** w.r.t. (partial correctness properties:

Definition 19.1 (Axiomatic equivalence)

Two statements $c_1, c_2 \in \mathsf{Cmd}$ are called **axiomatically equivalent** (notation: $c_1 \approx c_2$) if, for all assertions $A, B \in \mathsf{Assn}$,

$$\models \{A\}c_1\{B\} \iff \models \{A\}c_2\{B\}$$

Total correctness yields same notion of equivalence (see Theorem 19.8, p. 79).

Example 19.2 (Axiomatic equivalence)

We show that $\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end} \approx \mathsf{if}\ b\ \mathsf{then}\ c; \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{end}$.
Let $A, B \in \mathsf{Assn}$:

$$\begin{aligned}
&\models \{A\}\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\{B\}\\
\overset{\text{(Theorem 16.2, 17.1)}}{\iff} &\vdash \{A\}\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\{B\}\\
\overset{\text{(rule (while))}}{\iff} &\text{ex. } C \in \mathsf{Assn} \text{ such that } \models (A \implies C), \models (C \wedge \neg b \implies B),\\
&\vdash \{C \wedge b\}c\{C\}\\
\overset{\text{(rule (seq),(skip))}}{\iff} &\text{ex. } C \in \mathsf{Assn} \text{ such that } \models (A \implies C), \models (C \wedge \neg b \implies B),\\
&\vdash \{C \wedge b\}c; \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\{C \wedge \neg b\}\\
&\vdash \{C \wedge \neg b\}\mathsf{skip}\{C \wedge \neg b\}\\
\overset{\text{(rule (if))}}{\iff} &\text{ex. } C \in \mathsf{Assn} \text{ such that } \models (A \implies C), \models (C \wedge \neg b \implies B),\\
&\vdash \{C\}\mathsf{if}\ b\ \mathsf{then}\ c; \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{end}\{C \wedge \neg b\}\\
\overset{\text{(rule (cons))}}{\iff} &\vdash \{A\}\mathsf{if}\ b\ \mathsf{then}\ c; \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{end}\{B\}\\
\overset{\text{(Theorem 16.2, 17.1)}}{\iff} &\models \{A\}\mathsf{if}\ b\ \mathsf{then}\ c; \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{end}\{B\}
\end{aligned}$$

5.11 Characteristic Assertions

To relate axiomatic and operational/denotional equivalence, we have to **encode states by assertions**:

Definition 19.2 (Characteristic assertion)

Given a state $\sigma \in \Sigma$ and a finite subset of program variables $X \subseteq \mathsf{Var}$, the **characteristic assertion of** σ **w.r.t.** X is given by

$$\mathsf{state}(\sigma, X) := \bigwedge_{x \in X} (x = \underbrace{\sigma(x)}_{\in \mathbb{Z}}) \in \mathsf{Assn}$$

(where $\mathsf{state}(\sigma, \varnothing) := \mathsf{true}$). Moreover, we let $\mathsf{state}(\bot, X) := \mathsf{false}$.

Corollary 19.4

For all finite $X \subseteq \mathsf{Var}$ and $\sigma \in \Sigma$,

$$\sigma \models \mathsf{state}(\sigma, X)$$

Programs and characteristic state assertions are obviously related as follows:

Corollary 19.5

Let $c \in \mathsf{Cmd}$, and let $\mathsf{FV}(c) \subseteq \mathsf{Var}$ denote the set of all variables occuring in c.
Then, for every finite $X \supseteq \mathsf{FV}(c)$ and $\sigma \in \Sigma$,

$$\models \{\mathsf{state}(\sigma, X)\} c \{\mathsf{state}(\mathfrak{C}[\![c]\!]\sigma, X)\}$$

If moreover $\mathfrak{C}[\![c]\!]\sigma \neq \bot$, then $\models \{\mathsf{state}(\sigma, X)\} c \{\downarrow \mathsf{state}(\mathfrak{C}[\![c]\!]\sigma, X)\}$.

Example 19.6 (Characteristic Assertions of factorial program)

- For $c := (y := 1; \textsf{while } \neg(x = 1) \textsf{ do } y := y * x; x := x - 1 \textsf{ end})$,
  $X = \{x, y, z\} \supseteq \textsf{FV}(c) = \{x, y\}$, $\sigma(x) = 3, \sigma(y) = 0$, and $\sigma(z) = 1$, we obtain

  $$state(\sigma, X) = (x = 3 \wedge y = 0 \wedge z = 1)$$
  $$state(\mathfrak{C}[\![c]\!]\sigma, X) = (x = 1 \wedge y = 6 \wedge z = 1)$$

  and thus $\models \{\textsf{state}(\sigma, X)\}c\{\downarrow \textsf{state}(\mathfrak{C}[\![c]\!]\sigma, X)\}$.

- If $X \not\supseteq \textsf{FV}(c)$, then the claim generally does not hold: e.g. $\not\models \{y = 0\}c\{y = 6\}$!

5.12 Axiomatic vs. Operational/Denotional Equivalence

Theorem 19.7

Axiomatic and operational/denotional equivalence coincide, i.e. for all $c_1, c_2 \in \mathsf{Cmd}$,

$$c_1 \approx c_2 \iff c_1 \sim c_2$$

Proof of Theorem 19.7

We have to show:
$\forall A, B \in \mathsf{Assn} : \models \{A\}c_1\{B\} \iff \models \{A\}c_2\{B\}$
iff $\forall \sigma \in \Sigma : \mathfrak{C}[\![c_1]\!]\sigma = \mathfrak{C}[\![c_2]\!]\sigma$

- " $\implies$ ":
  Let $c_1 \approx c_2$ and $X := \mathsf{FV}(c_1) \cup \mathsf{FV}(c_2)$.
  Assume ex. $\sigma \in \Sigma$ such that $\mathfrak{C}[\![c_1]\!]\sigma \neq \mathfrak{C}[\![c_2]\!]\sigma$.
  Two cases are possible:
  - $\mathfrak{C}[\![c_1]\!]\sigma = \bot \neq \mathfrak{C}[\![c_2]\!]\sigma$ (or vice versa): Here

    $$\models \{\mathsf{state}(\sigma, X)\}c_1\{\mathsf{false}\} \text{ but } \not\models \{\mathsf{state}(\sigma, X)\}c_2\{\mathsf{false}\}$$

    which contradicts $c_1 \approx c_2$.
  - $\sigma_1 := \mathfrak{C}[\![c_1]\!]\sigma \neq \bot \neq \mathfrak{C}[\![c_2]\!]\sigma =: \sigma_2$:
    Here ex. $x \in X$ with $\sigma_1(x) \neq \sigma_2(x)$, such that (using Corollary 19.5, p. 76)

    $$\models \{\mathsf{state}(\sigma, X)\}c_1\{\mathsf{state}(\sigma_1, X)\} \text{ but } \not\models \{\mathsf{state}(\sigma, X)\}c_2\{\mathsf{state}(\sigma_1, X)\}$$

    which again contradicts $c_1 \approx c_2$.
- " $\impliedby$ ":
  Let $c_1 \sim c_2$.
  Assume ex. $A, B \in \mathsf{Assn}$ with $\models \{A\}c_1\{B\}$ but $\not\models \{A\}c_2\{B\}$ (or vice versa).
  Thus ex. $\sigma \in [\![A]\!]$ with $\bot \neq \mathfrak{C}[\![c_2]\!]\sigma \not\models B$. Again two cases are possible:
  - $\mathfrak{C}[\![c_1]\!]\sigma = \bot \neq \mathfrak{C}[\![c_2]\!]\sigma$:
    This contradicts $c_1 \sim c_2$.
  - $\bot \neq \sigma_1 := \mathfrak{C}[\![c_1]\!]\sigma \models B$ (and $\sigma_2 := \mathfrak{C}[\![c_2]\!]\sigma \not\models B$):
    Here ex. $x \in \mathsf{FV}(B)$ with $\sigma_1(x) \neq \sigma_2(x)$, which contradicts $c_1 \sim c_2$.

5.12.1 Partial vs. Total Equivalence

Using characteristic state assertions, we can show that considering **total** rather than partial correctness properties yields the same notion of equivalence:

Theorem 19.8

Let $c_1, c_2 \in \mathsf{Cmd}$. The following propositions are equivalent:

- For all $A, B \in \mathsf{Assn} : \models \{A\}c_1\{B\} \iff \models \{A\}c_2\{B\}$
- For all $A, B \in \mathsf{Assn} : \models \{A\}c_1\{\downarrow B\} \iff \models \{A\}c_2\{\downarrow B\}$

TODO: proof of Theorem 19.8 (L19 Page 14)

6 Extension by Blocks and Procedures

- Extension of WHILE by nested **blocks** with local **variables** and recursive **procedures**
- Simple memory model ($\Sigma := \{\sigma \mid \sigma : \mathsf{Var} to \mathbb{Z}\}$) not sufficient any more as variables can occur in several **instances**
- Involves new semantic concepts:
  - variable and procedure **environments**
  - **locations** (memory addresses) and **stores** (memory states)
- Important: **scope** of variable and procedure identifiers
  - **static scoping:** scope of identifier = **declaration environment**
    (also: "lexical" scoping; used here)
  - **dynamic scoping:** scope of identifier = **calling environment**
    (old Algo/Lisp dialects)

6.1 Extending the syntax

6.1.1 Syntactic categories

Category	Domain	Meta variable
Procedure identifiers	$\mathsf{Pid} = \{P, Q, ...\}$	P
Procedure declarations	PDec	p
Variable declarations	VDec	v
Commands (statements)	Cmd	c

6.1.2 Syntax of extended WHILE

Definition L20P6 (Syntax of extended WHILE)

The **syntax of extended WHILE Programs** is defined by the following context-free grammar:

$$\begin{aligned} p &::= \mathsf{proc}\ P\ \mathsf{is}\ c\ \mathsf{end}; p \mid \epsilon & \in \mathsf{PDec} \\ v &::= \mathsf{var}\ x; v \mid \epsilon & \in \mathsf{BExp} \\ c &::= \mathsf{skip} \mid x := a \mid c_1; c_2 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end} \mid \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end} \mid \\ &\quad\ \mathsf{call}\ P \mid \mathsf{begin}\ v\ p\ c\ \mathsf{end} & \in \mathsf{Cmd} \end{aligned}$$

- All used variable/procedure identifiers have to be declared
- Identifiers declared within a block must be distinct

6.2 Locations and Stores

- So far: **states** $\Sigma = \{\sigma \mid \sigma : \mathsf{Var} \to \mathbb{Z}\}$
- Now: explicit control over all (nested) **instances** of a variable:

Definition L20P8 (Variable Environments, Locations and Stores)

- **variable environments:**

$$\mathsf{VEnv} := \{\rho \mid \rho : \mathsf{Var} \dashrightarrow \mathsf{Loc}\}$$

(Partial function to maintain **declaredness** information)

- **locations:**

$$\mathsf{Loc} := \mathbb{N}$$

- **stores:**

$$\mathsf{Sto} := \{\sigma \mid \sigma : \mathsf{Loc} \dashrightarrow \mathbb{Z}\}$$

(partial function to maintain **allocation** information)

$\Longrightarrow$ **Two-level access** to a variable $x \in \mathsf{Var}$:

1. determine current memory location of x:

$$l := \rho(x)$$

2. reading/writing access to σ at location l

Thus: previous **state** information represented as $\sigma \circ \rho : \mathsf{Var} \dashrightarrow \mathbb{Z}$

Definition L20P9.2.1 (Update Relation of Variable Declaration)

Effects of declaration: update of variable environment and store

$$\mathsf{upd}_v[\![.]\!] : \mathsf{VDec} \times \mathsf{VEnv} \times \mathsf{Sto} \to \mathsf{VEnv} \times \mathsf{Sto}$$

$$\mathsf{upd}_v[\![\mathsf{var}\ x; v]\!](\rho, \sigma) := \mathsf{upd}_v[\![v]\!](\rho[x \mapsto l_x], \sigma[l_x \mapsto 0])$$
$$\mathsf{upd}_v[\![\epsilon]\!](\rho, \sigma) := (\rho, \sigma)$$

where $l_x := \mathsf{min}\{l \in \mathsf{Loc} \mid \sigma(l) = \bot\}$

6.3 Procedure Environments and Declarations

Definition Procedure Environment (L20P9.1)

The **Effect of a procedure call** is determined by its body and variable and procedure environment of its declaration:

$$\mathsf{PEnv} := \{\pi \mid \pi : \mathsf{Pid} \to \mathsf{Cmd} \times \mathsf{VEnv} \times \mathsf{PEnv}\}$$

denotes the set of **procedure environments**.

Definition L20P9.2.2 (Update Relation of Procedure Declaration)

Effects of procedure declaration: update of procedure environment

$$\mathsf{upd}_p[\![.]\!] : \mathsf{PDec} \times \mathsf{VEnv} \times \mathsf{PEnv} \to \mathsf{PEnv}$$

$$\mathsf{upd}_p[\![\mathsf{proc}\ P\ \mathsf{is}\ c\ \mathsf{end}; p]\!](\rho, \pi) := \mathsf{upd}_p[\![p]\!](\rho, \pi[P \mapsto (c, \rho, \pi)])$$
$$\mathsf{upd}_p[\![\epsilon]\!](\rho, \pi) := \pi$$

6.4 Execution Relation

Definition 20.2 (Execution relation of extended WHILE)

For $c \in \mathsf{Cmd}$, $\sigma, \sigma' \in \mathsf{Sto}$, $\rho \in \mathsf{VEnv}$, and $\pi \in \mathsf{PEnv}$, the **execution relation** $(\rho, \pi) \vdash \langle c, \sigma \rangle \to \sigma'$ ("in environment (ρ, π), statement c transforms store σ into σ'") is defined by the following rules:

$$\text{(skip)}\ \frac{}{(\rho,\pi) \vdash \langle \mathsf{skip}, \sigma \rangle \to \sigma}$$

$$\text{(asgn)}\ \frac{\langle a, \sigma \circ \rho \rangle \to z}{(\rho,\pi) \vdash \langle x := a, \sigma \rangle \to \sigma[\rho(x) \mapsto z]}$$

$$\text{(seq)}\ \frac{(\rho,\pi) \vdash \langle c_1, \sigma \rangle \to \sigma' \qquad (\rho,\pi) \vdash \langle c_2, \sigma' \rangle \to \sigma''}{(\rho,\pi) \vdash \langle c_1; c_2, \sigma \rangle \to \sigma''}$$

$$\text{(if-t)}\ \frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{true} \qquad (\rho,\pi) \vdash \langle c_1, \sigma \rangle \to \sigma'}{(\rho,\pi) \vdash \langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}$$

$$\text{(if-f)}\ \frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{false} \qquad (\rho,\pi) \vdash \langle c_2, \sigma \rangle \to \sigma'}{(\rho,\pi) \vdash \langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}, \sigma \rangle \to \sigma'}$$

$$\text{(wh-f)}\ \frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{false}}{(\rho,\pi) \vdash \langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma \rangle \to \sigma}$$

$$\text{(wh-t)}\ \frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{true} \qquad (\rho,\pi) \vdash \langle c, \sigma \rangle \to \sigma' \qquad (\rho,\pi) \vdash \langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma' \rangle \to \sigma''}{(\rho,\pi) \vdash \langle \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}, \sigma \rangle \to \sigma''}$$

$$\text{(call)}\ \frac{\pi(P) = (c, \rho', \pi') \qquad (\rho', \pi'[P \mapsto (c, \rho', \pi')]) \vdash \langle c, \sigma \rangle \to \sigma'}{(\rho,\pi) \vdash \langle \mathsf{call}\ P, \sigma \rangle \to \sigma'}$$

$$\text{(block)}\ \frac{\mathsf{upd}_v[\![v]\!](\rho, \sigma) = (\rho', \sigma') \qquad \mathsf{upd}_p[\![p]\!](\rho', \pi) = \pi' \qquad (\rho', \pi') \vdash \langle c, \sigma' \rangle \to \sigma''}{(\rho,\pi) \vdash \langle \mathsf{begin}\ v\ p\ c\ \mathsf{end}, \sigma \rangle \to \sigma''}$$

The **initial environment** $(\rho_\varnothing, \pi_\varnothing)$ is given by $\rho_\varnothing(x) = \pi_\varnothing(P) = \bot$ ($x \in \mathsf{Var}$, $P \in \mathsf{Pid}$).

Remarks:

- Evaluation of (arithmetic and Boolean) expressions can now **fail** due to undeclared variables.
- In rules for composite statements, the exceution of sub-statements can have an effect on the environments (due to nested blocks), but this effect is **transient**.
- Rule $\text{(call)}\ \frac{\pi(P) = (c, \rho', \pi') \qquad (\rho', \pi'[P \mapsto (c, \rho', \pi')]) \vdash \langle c, \sigma \rangle \to \sigma'}{(\rho,\pi) \vdash \langle \mathsf{call}\ P, \sigma \rangle \to \sigma'}$:
  - **Static scoping** is modelled by using the environments ρ' and π' (as determined in (block)) from the **declaration** site of procedure P (and not ρ and π from the **calling** site).
  - For executing the procedure call, the procedure environment associated with P (π') is extended by a P-entry to handle **recursive calls** of P:

    $\pi'[P \mapsto (c, \rho', \pi')]$

6.5 Command Semantics using Variable Environments

- **First step:** reformulation of Definition 8.3 (p. 34) using **variable environments and locations** (initially disregarding procedures)
- **So far:** $\mathfrak{C}[\![.]\!] : \mathsf{Cmd} \to (\Sigma \to \Sigma)$

Definition 21.1 (Denotational semantics using locations)

The **(denotational) semantic functional dor commands**,

$$\mathfrak{C}'[\![.]\!] : \mathsf{Cmd} \to \mathsf{VEnv} \to (\mathsf{Sto} \to \mathsf{Sto})$$

is given by:

$$\begin{aligned}
\mathfrak{C}'[\![\mathsf{skip}]\!]\rho &:= \mathsf{id}_{\mathsf{Sto}} \\
\mathfrak{C}'[\![x := a]\!]\rho &:= \lambda\sigma.\sigma[\rho(x) \mapsto \mathfrak{A}[\![a]\!](\mathsf{lookup}\ \rho\ \sigma)] \\
\mathfrak{C}'[\![c_1; c_2]\!]\rho &:= (\mathfrak{C}'[\![c_2]\!]\rho) \circ (\mathfrak{C}'[\![c_1]\!]\rho) \\
\mathfrak{C}'[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{end}]\!]\rho &:= \mathsf{cond}(\mathfrak{B}[\![b]\!] \circ (\mathsf{lookup}\ \rho), \mathfrak{C}'[\![c_1]\!]\rho) \\
\mathfrak{C}'[\![\mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{end}]\!]\rho &:= \mathsf{fix}(\Phi)
\end{aligned}$$

where $\mathsf{lookup} : \mathsf{VEnv} \to \mathsf{Sto} \to (\mathsf{Var} \to \mathbb{Z})$ with $\mathsf{lookup}\ \rho\ \sigma := \sigma \circ \rho$ and

$$\Phi : (\mathsf{Sto} \to \mathsf{Sto}) \to (\mathsf{Sto} \to \mathsf{Sto}) : f \mapsto \mathsf{cond}(\mathfrak{B}[\![b]\!] \circ (\mathsf{lookup}\ \rho), f \circ \mathfrak{C}'[\![c]\!]\rho, \mathsf{id}_{\mathsf{Sto}})$$

Index