

Semantics and Verification of Software

Lecture 4: Operational and Denotational Semantics

Thomas Noll

Lehrstuhl für Informatik 2
RWTH Aachen University
noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/svsw/>

Summer semester 2007

- 1 Repetition: Execution of Statements
- 2 Functional of the Operational Semantics
- 3 Summary: Operational Semantics
- 4 The Denotational Approach
- 5 Denotational Semantics of Expressions
- 6 Denotational Semantics of Statements
- 7 Fixpoint Semantics of `while` Loop

Execution of Statements

Remember:

$c ::= \text{skip} \mid x := a \mid c_1; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \in \text{Cmd}$

Definition (Execution relation for statements)

For $c \in \text{Cmd}$ and $\sigma, \sigma' \in \Sigma$, the **execution relation** $\langle c, \sigma \rangle \rightarrow \sigma'$ is defined by the following rules:

$$\begin{array}{c} \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma} \text{ (skip)} \qquad \frac{\langle a, \sigma \rangle \rightarrow z}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \mapsto z]} \text{ (asgn)} \\[10pt] \frac{\langle c_1, \sigma \rangle \rightarrow \sigma' \quad \langle c_2, \sigma' \rangle \rightarrow \sigma''}{\langle c_1; c_2, \sigma \rangle \rightarrow \sigma''} \text{ (seq)} \qquad \frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma'} \text{ (if-t)} \\[10pt] \frac{\langle b, \sigma \rangle \rightarrow \text{false} \quad \langle c_2, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \sigma'} \text{ (if-f)} \qquad \frac{\langle b, \sigma \rangle \rightarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma} \text{ (wh-f)} \\[10pt] \frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''} \text{ (wh-t)} \end{array}$$

Determinism of Execution Relation

This operational semantics is well defined in the following sense:

Theorem

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

Proof.

by structural induction on derivation trees



- 1 Repetition: Execution of Statements
- 2 Functional of the Operational Semantics
- 3 Summary: Operational Semantics
- 4 The Denotational Approach
- 5 Denotational Semantics of Expressions
- 6 Denotational Semantics of Statements
- 7 Fixpoint Semantics of `while` Loop

Functional of the Operational Semantics

The determinism of the execution relation (Theorem 3.5) justifies the following definition:

Definition 4.3 (Operational functional)

The **functional of the operational semantics**,

$$\mathfrak{D}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \multimap \Sigma),$$

assigns to every statement $c \in Cmd$ a partial state transformation $\mathfrak{D}[\![c]\!] : \Sigma \multimap \Sigma$, which is defined as follows:

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

Remark: $\mathfrak{D}[\![c]\!]\sigma$ can indeed be undefined
(consider e.g. $c = \text{while true do skip}$; see Corollary 2.9)

Functional of the Operational Semantics

The determinism of the execution relation (Theorem 3.5) justifies the following definition:

Definition 4.3 (Operational functional)

The **functional of the operational semantics**,

$$\mathfrak{D}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \multimap \Sigma),$$

assigns to every statement $c \in Cmd$ a partial state transformation $\mathfrak{D}[\![c]\!] : \Sigma \multimap \Sigma$, which is defined as follows:

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

Remark: $\mathfrak{D}[\![c]\!]\sigma$ can indeed be undefined
(consider e.g. $c = \text{while true do skip}$; see Corollary 2.9)

Definition 4.4 (Operational equivalence)

Two statements $c_1, c_2 \in Cmd$ are called **(operationally) equivalent** (notation: $c_1 \sim c_2$) if

$$\mathfrak{D}[[c_1]] = \mathfrak{D}[[c_2]].$$

Thus:

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

Definition 4.4 (Operational equivalence)

Two statements $c_1, c_2 \in Cmd$ are called **(operationally) equivalent** (notation: $c_1 \sim c_2$) if

$$\mathcal{V}[[c_1]] = \mathcal{V}[[c_2]].$$

Thus:

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

“Unwinding” of Loops

Simple application of statement equivalence: test of execution condition in a `while` loop can be represented by an `if` statement

Lemma 4.5

For every $b \in BExp$ and $c \in Cmd$,

$\text{while } b \text{ do } c \sim \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip.}$

Proof.

on the board



“Unwinding” of Loops

Simple application of statement equivalence: test of execution condition in a `while` loop can be represented by an `if` statement

Lemma 4.5

For every $b \in BExp$ and $c \in Cmd$,

$\text{while } b \text{ do } c \sim \text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip.}$

Proof.

on the board



- 1 Repetition: Execution of Statements
- 2 Functional of the Operational Semantics
- 3 Summary: Operational Semantics**
- 4 The Denotational Approach
- 5 Denotational Semantics of Expressions
- 6 Denotational Semantics of Statements
- 7 Fixpoint Semantics of `while` Loop

Summary: Operational Semantics

- Formalized by **evaluation/execution relations**
- Inductively defined by **derivation trees** using **operational rules**
- Enables proofs about operational behaviour of programs using **structural induction**
- **Semantic functional** characterizes complete input/output behaviour of programs

Summary: Operational Semantics

- Formalized by **evaluation/execution relations**
- Inductively defined by **derivation trees** using **operational rules**
- Enables proofs about operational behaviour of programs using **structural induction**
- **Semantic functional** characterizes complete input/output behaviour of programs

Summary: Operational Semantics

- Formalized by **evaluation/execution relations**
- Inductively defined by **derivation trees** using **operational rules**
- Enables proofs about operational behaviour of programs using **structural induction**
- **Semantic functional** characterizes complete input/output behaviour of programs

Summary: Operational Semantics

- Formalized by **evaluation/execution relations**
- Inductively defined by **derivation trees** using **operational rules**
- Enables proofs about operational behaviour of programs using **structural induction**
- **Semantic functional** characterizes complete input/output behaviour of programs

- 1 Repetition: Execution of Statements
- 2 Functional of the Operational Semantics
- 3 Summary: Operational Semantics
- 4 The Denotational Approach**
- 5 Denotational Semantics of Expressions
- 6 Denotational Semantics of Statements
- 7 Fixpoint Semantics of `while` Loop

- Primary aspect of a program: its “effect”, i.e., **input/output behaviour**
- In operational semantics: **indirect** definition of semantic functional $\mathcal{D}[\![\cdot]\!]$ by execution relation
- Now: **abstract** from operational details
- **Denotational semantics**: direct definition of program effect by induction on its syntactic structure

- Primary aspect of a program: its “effect”, i.e., **input/output behaviour**
- In operational semantics: **indirect** definition of semantic functional $\mathcal{D}[\![\cdot]\!]$ by execution relation
- Now: **abstract** from operational details
- **Denotational semantics**: direct definition of program effect by induction on its syntactic structure

- Primary aspect of a program: its “effect”, i.e., **input/output behaviour**
- In operational semantics: **indirect** definition of semantic functional $\mathcal{D}[\![\cdot]\!]$ by execution relation
- Now: **abstract** from operational details
- **Denotational semantics**: direct definition of program effect by induction on its syntactic structure

- Primary aspect of a program: its “effect”, i.e., **input/output behaviour**
- In operational semantics: **indirect** definition of semantic functional $\mathcal{D}[\![\cdot]\!]$ by execution relation
- Now: **abstract** from operational details
- **Denotational semantics**: direct definition of program effect by induction on its syntactic structure

- 1 Repetition: Execution of Statements
- 2 Functional of the Operational Semantics
- 3 Summary: Operational Semantics
- 4 The Denotational Approach
- 5 Denotational Semantics of Expressions**
- 6 Denotational Semantics of Statements
- 7 Fixpoint Semantics of `while` Loop

Again: value of an expression determined by current state

Definition 4.6 (Denotational semantics of arithmetic expressions)

The (denotational) semantic functional for arithmetic expressions,

$$\mathcal{A}[\![\cdot]\!] : AExp \rightarrow (\Sigma \rightarrow \mathbb{Z}),$$

is given by:

$$\begin{array}{ll} \mathcal{A}[\![z]\!]\sigma := z & \mathcal{A}[\![a_1 + a_2]\!]\sigma := \mathcal{A}[\![a_1]\!]\sigma + \mathcal{A}[\![a_2]\!]\sigma \\ \mathcal{A}[\![x]\!]\sigma := \sigma(x) & \mathcal{A}[\![a_1 - a_2]\!]\sigma := \mathcal{A}[\![a_1]\!]\sigma - \mathcal{A}[\![a_2]\!]\sigma \\ & \mathcal{A}[\![a_1 * a_2]\!]\sigma := \mathcal{A}[\![a_1]\!]\sigma * \mathcal{A}[\![a_2]\!]\sigma \end{array}$$

Definition 4.7 (Denotational semantics of Boolean expressions)

The (denotational) semantic functional for Boolean expressions,

$$\mathfrak{B}[\cdot] : BExp \rightarrow (\Sigma \rightarrow \mathbb{B}),$$

is given by:

$$\begin{aligned}\mathfrak{B}[\![t]\!]\sigma &:= t \\ \mathfrak{B}[\![a_1 = a_2]\!]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma = \mathfrak{A}[\![a_2]\!]\sigma \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[\![a_1 > a_2]\!]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{A}[\![a_1]\!]\sigma > \mathfrak{A}[\![a_2]\!]\sigma \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[\![\neg b]\!]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{B}[\![b]\!]\sigma = \text{false} \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[\![b_1 \wedge b_2]\!]\sigma &:= \begin{cases} \text{true} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \\ \mathfrak{B}[\![b_1 \vee b_2]\!]\sigma &:= \begin{cases} \text{false} & \text{if } \mathfrak{B}[\![b_1]\!]\sigma = \mathfrak{B}[\![b_2]\!]\sigma = \text{false} \\ \text{true} & \text{otherwise} \end{cases}\end{aligned}$$

- 1 Repetition: Execution of Statements
- 2 Functional of the Operational Semantics
- 3 Summary: Operational Semantics
- 4 The Denotational Approach
- 5 Denotational Semantics of Expressions
- 6 Denotational Semantics of Statements**
- 7 Fixpoint Semantics of `while` Loop

Semantics of Statements I

- Now: semantic functional

$$\mathfrak{C}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \rightarrow \Sigma)$$

- Same type as operational functional

$$\mathfrak{O}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \rightarrow \Sigma)$$

(in fact, both will turn out to be the same)

- Inductive definition employs auxiliary functions:

- identity on states: $\text{id}_\Sigma : \Sigma \rightarrow \Sigma : \sigma \mapsto \sigma$

- (strict) composition of partial state transformations:

$$\circ : (\Sigma \rightarrow \Sigma) \times (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$$

where, for every $f, g : \Sigma \rightarrow \Sigma$ 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:

$$\text{cond} : (\Sigma \rightarrow \mathbb{B}) \times (\Sigma \rightarrow \Sigma) \times (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$$

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

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

Semantics of Statements I

- Now: semantic functional

$$\mathfrak{C}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \rightarrow \Sigma)$$

- Same type as operational functional

$$\mathfrak{O}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \rightarrow \Sigma)$$

(in fact, both will turn out to be the **same**)

- Inductive definition employs auxiliary functions:

- identity** on states: $\text{id}_\Sigma : \Sigma \rightarrow \Sigma : \sigma \mapsto \sigma$

- (strict) composition** of partial state transformations:

$$\circ : (\Sigma \rightarrow \Sigma) \times (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$$

where, for every $f, g : \Sigma \rightarrow \Sigma$ 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:**

$$\text{cond} : (\Sigma \rightarrow \mathbb{B}) \times (\Sigma \rightarrow \Sigma) \times (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$$

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

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

Semantics of Statements I

- Now: semantic functional

$$\mathfrak{C}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \rightarrow \Sigma)$$

- Same type as operational functional

$$\mathfrak{O}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \rightarrow \Sigma)$$

(in fact, both will turn out to be the **same**)

- Inductive definition employs auxiliary functions:

- identity** on states: $\text{id}_\Sigma : \Sigma \rightarrow \Sigma : \sigma \mapsto \sigma$

- (strict) composition** of partial state transformations:

$$\circ : (\Sigma \rightarrow \Sigma) \times (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$$

where, for every $f, g : \Sigma \rightarrow \Sigma$ 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:**

$$\text{cond} : (\Sigma \rightarrow \mathbb{B}) \times (\Sigma \rightarrow \Sigma) \times (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$$

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

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

Definition 4.8 (Denotational semantics of statements)

The (denotational) semantic functional for statements,

$$\mathcal{C}[\![\cdot]\!] : Cmd \rightarrow (\Sigma \rightarrow \Sigma),$$

is given by:

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

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

Remarks:

- Definition of $\mathcal{C}[[c]]$ given by **induction on syntactic structure** of $c \in Cmd$
 - in particular, $\mathcal{C}[[\text{while } b \text{ do } c]]$ only refers to $\mathcal{B}[[b]]$ and $\mathcal{C}[[c]]$ (and not to $\mathcal{C}[[\text{while } b \text{ do } c]]$ again)
 - note difference to $\mathcal{D}[[c]]$:

$$\frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''} \text{ (wh-t)}$$

- In $\mathcal{C}[[c_1; c_2]] := \mathcal{C}[[c_2]] \circ \mathcal{C}[[c_1]]$, function composition $\circ$ has to be **strict** since non-termination of c_1 implies non-termination of $c_1; c_2$
- In $\mathcal{C}[[\text{while } b \text{ do } c]] := \text{fix}(\Phi)$, fix denotes a fixpoint operator (which remains to be defined)
 $\implies$ “**fixpoint semantics**”

But: why **fixpoints**?

Remarks:

- Definition of $\mathcal{C}[[c]]$ given by **induction on syntactic structure** of $c \in Cmd$
 - in particular, $\mathcal{C}[[\text{while } b \text{ do } c]]$ only refers to $\mathcal{B}[[b]]$ and $\mathcal{C}[[c]]$ (and not to $\mathcal{C}[[\text{while } b \text{ do } c]]$ again)
 - note difference to $\mathcal{D}[[c]]$:

$$\frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''} \text{ (wh-t)}$$

- In $\mathcal{C}[[c_1; c_2]] := \mathcal{C}[[c_2]] \circ \mathcal{C}[[c_1]]$, function composition $\circ$ has to be **strict** since non-termination of c_1 implies non-termination of $c_1; c_2$
- In $\mathcal{C}[[\text{while } b \text{ do } c]] := \text{fix}(\Phi)$, fix denotes a fixpoint operator (which remains to be defined)
 $\implies$ “fixpoint semantics”

But: why **fixpoints**?

Remarks:

- Definition of $\mathcal{C}[[c]]$ given by **induction on syntactic structure** of $c \in Cmd$
 - in particular, $\mathcal{C}[[\text{while } b \text{ do } c]]$ only refers to $\mathcal{B}[[b]]$ and $\mathcal{C}[[c]]$ (and not to $\mathcal{C}[[\text{while } b \text{ do } c]]$ again)
 - note difference to $\mathcal{D}[[c]]$:

$$\frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''} \text{ (wh-t)}$$

- In $\mathcal{C}[[c_1; c_2]] := \mathcal{C}[[c_2]] \circ \mathcal{C}[[c_1]]$, function composition $\circ$ has to be **strict** since non-termination of c_1 implies non-termination of $c_1; c_2$
- In $\mathcal{C}[[\text{while } b \text{ do } c]] := \text{fix}(\Phi)$, fix denotes a fixpoint operator (which remains to be defined)
 $\implies$ **“fixpoint semantics”**

But: why **fixpoints**?

Remarks:

- Definition of $\mathfrak{C}[[c]]$ given by **induction on syntactic structure** of $c \in Cmd$
 - in particular, $\mathfrak{C}[[\text{while } b \text{ do } c]]$ only refers to $\mathfrak{B}[[b]]$ and $\mathfrak{C}[[c]]$ (and not to $\mathfrak{C}[[\text{while } b \text{ do } c]]$ again)
 - note difference to $\mathfrak{D}[[c]]$:

$$\frac{\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''} \text{ (wh-t)}$$

- In $\mathfrak{C}[[c_1; c_2]] := \mathfrak{C}[[c_2]] \circ \mathfrak{C}[[c_1]]$, function composition $\circ$ has to be **strict** since non-termination of c_1 implies non-termination of $c_1; c_2$
- In $\mathfrak{C}[[\text{while } b \text{ do } c]] := \text{fix}(\Phi)$, fix denotes a fixpoint operator (which remains to be defined)
 $\implies$ **“fixpoint semantics”**

But: why **fixpoints**?

- 1 Repetition: Execution of Statements
- 2 Functional of the Operational Semantics
- 3 Summary: Operational Semantics
- 4 The Denotational Approach
- 5 Denotational Semantics of Expressions
- 6 Denotational Semantics of Statements
- 7 Fixpoint Semantics of **while** Loop

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation
- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation
- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation
- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation
- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation
- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation
- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation

- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

Fixpoint Semantics of while Loop I

- Goal: preserve **validity of equivalence**

$$\mathcal{C}[\text{while } b \text{ do } c] = \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}]$$

(cf. Lemma 4.5)

- Using the known parts of Def. 4.8, we obtain:

$$\begin{aligned}\mathcal{C}[\text{while } b \text{ do } c] &= \mathcal{C}[\text{if } b \text{ then } (c; \text{while } b \text{ do } c) \text{ else skip}] \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[c; \text{while } b \text{ do } c], \mathcal{C}[\text{skip}]) \\ &= \text{cond}(\mathcal{B}[b], \mathcal{C}[\text{while } b \text{ do } c] \circ \mathcal{C}[c], \text{id}_{\Sigma})\end{aligned}$$

- Abbreviating $f := \mathcal{C}[\text{while } b \text{ do } c]$ this yields:

$$f = \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

- Hence f must be a **solution** of this recursive equation
- Or: f must be a **fixpoint** of the mapping

$$\Phi : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \text{cond}(\mathcal{B}[b], f \circ \mathcal{C}[c], \text{id}_{\Sigma})$$

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

But: fixpoint property not sufficient to obtain a well-defined semantics

Existence: there does not need to exist any fixpoint. Examples:

- ❶ $\phi_1 : \mathbb{N} \rightarrow \mathbb{N} : n \mapsto n + 1$ has no fixpoint
- ❷ $\Phi_1 : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \begin{cases} g_1 & \text{if } f = g_2 \\ g_2 & \text{otherwise} \end{cases}$
(where $g_1 \neq g_2$) has no fixpoint

Uniqueness: there might exist several fixpoints. Examples:

- ❶ $\phi_2 : \mathbb{N} \rightarrow \mathbb{N} : n \mapsto n^3$ has fixpoints $\{0, 1\}$
- ❷ every state transformation f is a fixpoint of
 $\Phi_2 : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto f$

But: fixpoint property not sufficient to obtain a well-defined semantics

Existence: there does not need to exist any fixpoint. Examples:

- ❶ $\phi_1 : \mathbb{N} \rightarrow \mathbb{N} : n \mapsto n + 1$ has no fixpoint
- ❷ $\Phi_1 : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto \begin{cases} g_1 & \text{if } f = g_2 \\ g_2 & \text{otherwise} \end{cases}$
(where $g_1 \neq g_2$) has no fixpoint

Uniqueness: there might exist several fixpoints. Examples:

- ❶ $\phi_2 : \mathbb{N} \rightarrow \mathbb{N} : n \mapsto n^3$ has fixpoints $\{0, 1\}$
- ❷ every state transformation f is a fixpoint of
 $\Phi_2 : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) : f \mapsto f$

Solution:

Existence: we will show that this cannot happen in our setting, i.e., that **fixpoints always exist**

Uniqueness: will be guaranteed by **choosing a special fixpoint**

Solution:

Existence: we will show that this cannot happen in our setting, i.e., that **fixpoints always exist**

Uniqueness: will be guaranteed by **choosing a special fixpoint**