Korrekte Software: Grundlagen und Methoden

Vorlesung 2 vom 20.04.21

Operationale Semantik

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2021

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül I
- Der Floyd-Hoare-Kalkül II: Invarianten
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Modellierung
- Spezifikation von Funktionen
- Referenzen und Speichermodelle
- Ausblick und Rückblick

Zutaten

```
// GGT(A,B)
if (a == 0) r = b;
else {
  while (b != 0) {
    if (a <= b)
       b = b - a;
    else a = a - b;
  }
  r = a;
}
```

- Programme berechnen **Werte**
- Basierend auf
  - Werte sind **Variablen** zugewiesen
  - Evaluation von **Ausdrücken**
- Folgt dem Programmablauf

Unsere Programmiersprache

Wir betrachten einen Ausschnitt der Programmiersprache **C** (**C0**).

Ausbaustufe 1 kennt folgende Konstrukte:

- Typen: **int**;
- Ausdrücke: Variablen, Literale (für ganze Zahlen), arithmetische Operatoren (für ganze Zahlen), Relationen (==, <, ...), boolsche Operatoren (&&, ||);
- Anweisungen:
  - Fallunterscheidung (**if**...**else**...), Iteration (**while**), Zuweisung, Blöcke;
  - Sequenzierung und leere Anweisung sind implizit

C0: Ausdrücke und Anweisungen

$$
\begin{array}{lrcl}
\textbf{Aexp} & a & ::= & \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 / a_2 \\
\textbf{Bexp} & b & ::= & \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 < a_2 \mid \,!\,b \mid b_1 \,\&\&\, b_2 \mid b_1 \,||\, b_2 \\
\textbf{Exp} & e & := & a \mid b \\
\textbf{Stmt} & c & ::= & \mathbf{Idt} = \mathbf{Exp} \\
 & & \mid & \textbf{if}\ (b)\ c_1\ \textbf{else}\ c_2 \\
 & & \mid & \textbf{while}\ (b)\ c \\
 & & \mid & c_1; c_2 \\
 & & \mid & \{\,\}
\end{array}
$$

NB: Nicht die **konkrete** Syntax.

Eine Handvoll Beispiele

```
a = (3+y)*x+5*b;
a = ((3+y)*x)+(5*b);
a = 3+y*x+5*b;
```

```
p = 1;
c = 1;
while (c <= n) {
  p= p * c;
  c= c + 1;
  }
```


Semantik von C0

- Die (operationale) Semantik einer imperativen Sprache wie C0 ist ein **Zustandsübergang**: das System hat einen impliziten Zustand, der durch Zuweisung von **Werten** an **Adressen** geändert werden kann.

Systemzustände

- Ausdrücke werten zu **Werten** **V** (hier ganze Zahlen) aus.
- Adressen **Loc** sind hier Programmvariablen (Namen): **Loc** = **Idt**
- Ein **Systemzustand** bildet Adressen auf Werte ab: $\Sigma = \mathbf{Loc} \rightharpoonup \mathbf{V}$
- Ein Programm bildet einen Anfangszustand **möglicherweise** auf einen Endzustand ab (wenn es terminiert).

Partielle, endliche Abbildungen

Zustände sind **partielle, endliche Abbildungen** (finite partial maps)

$$f : X \rightharpoonup A$$

Notation:

- $f(x)$ für den Wert von x in f (*lookup*)
- $f(x) = \bot$ wenn x nicht in f (*undefined*)
- $f[x \mapsto n]$ für den Update an der Stelle x mit dem Wert n:

$$f[x \mapsto n](y) \stackrel{def}{=} \begin{cases} n & \text{if } x = y \\ f(y) & \text{otherwise} \end{cases}$$

Partielle, endliche Abbildungen II

Zustände sind **partielle, endliche Abbildungen** (finite partial maps)

$$f : X \rightharpoonup A$$

Notation:

- $\langle x \mapsto n, y \mapsto m\rangle$ u.ä. für konkrete Abbildungen.
- $\langle\rangle$ ist die leere (überall undefinierte Abbildung):

$$\text{für alle } x \in X \text{ gilt: } \langle\rangle(x) = \bot$$

- Die Domäne eines Zustands sind alle Stellen, an denen er definiert ist:

$$Dom(f) \stackrel{def}{=} \{x \in X | f(x) \neq \bot\}$$

- Updates sind "linksassoziativ":

$$f[x \mapsto n][y \mapsto m] = (f[x \mapsto n])[y \mapsto m]$$

Arbeitsblatt 2.1: Jetzt seid ihr dran!

- In euren Gruppen-Arbeitsblättern unter `https://hackmd.informatik.uni-bremen.de/rfF0alFiS8y6nUtspD4YgA#` gebt folgendes an
- Wie sieht ein Zustand aus, der *a* den Wert 6 und c den Wert 2 zuweist.
- Welches sind Zustände, und welche nicht:
  - A $\langle x \mapsto 1, a \mapsto 3\rangle$
  - B $\langle x \mapsto y, b \mapsto 6\rangle$
  - C $\langle x \mapsto 2, b \mapsto 6, x \mapsto 5\rangle$
  - D $\langle x \mapsto 3, b \mapsto 6, y \mapsto 5\rangle$
- Update von Zuständen:
  - A $\langle x \mapsto 1, a \mapsto 3\rangle[y \mapsto 1] = ??$
  - B $\langle x \mapsto 1, a \mapsto 3\rangle[x \mapsto 3] = ??$
  - C $\langle x \mapsto 1, a \mapsto 3\rangle[x \mapsto 3][y \mapsto 1][x \mapsto 4] = ??$

Besprechung

- Wie sieht ein Zustand aus, der *a* den Wert 6 und c den Wert 2 zuweist: $\langle a \mapsto 6, c \mapsto 2\rangle$
- Welches sind Zustände, und welche nicht:
  - A $\langle x \mapsto 1, a \mapsto 3\rangle$ +
  - B $\langle x \mapsto y, b \mapsto 6\rangle$ −
  - C $\langle x \mapsto 2, b \mapsto 6, x \mapsto 5\rangle$ −
  - D $\langle x \mapsto 3, b \mapsto 6, y \mapsto 5\rangle$ +
- Update von Zuständen:
  - A $\langle x \mapsto 1, a \mapsto 3\rangle[y \mapsto 1] = \langle x \mapsto 1, a \mapsto 3, y \mapsto 1\rangle$
  - B $\langle x \mapsto 1, a \mapsto 3\rangle[x \mapsto 3] = \langle x \mapsto 3, a \mapsto 3\rangle$
  - C $\langle x \mapsto 1, a \mapsto 3\rangle[x \mapsto 3][y \mapsto 1][x \mapsto 4] = \langle x \mapsto 4, y \mapsto 1, a \mapsto 3\rangle$

Operationale Semantik: Arithmetische Ausdrücke

Ein arithmetischer Ausdruck *a* wertet unter gegebenen Zustand σ zu einer ganzen Zahl *n* (Wert) aus oder zu einem Fehler $\bot$.

- **Aexp** $a ::= \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 \,/\, a_2$

$$\langle a, \sigma\rangle \to_{Aexp} n \mid \bot$$

Regeln:

$$\frac{n \in \mathbf{Z}}{\langle n, \sigma\rangle \to_{Aexp} [\![n]\!]}$$

$$\frac{x \in \mathbf{Idt}, x \in Dom(\sigma), \sigma(x) = v}{\langle x, \sigma\rangle \to_{Aexp} v} \qquad \frac{x \in \mathbf{Idt}, x \notin Dom(\sigma)}{\langle x, \sigma\rangle \to_{Aexp} \bot}$$

Regelschreibweise vs. Funktionen

Sei $\mathsf{Int}+ = \mathsf{Int} \cup \{\bot\}$

```
AexpEval :: AExp -> (Zustand -> Int+)
AexpEval n :: Int s -> n
AexpEval x :: Loc s if Dom(s) contains x -> s(x)
AexpEval x :: Loc s if not(Dom(s) contains x) -> ⊥
```


Operationale Semantik: Arithmetische Ausdrücke

- **Aexp** $a ::= \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 \,/\, a_2$

$$\langle a, \sigma\rangle \to_{Aexp} n \mid \bot$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad n_i \in \mathbb{Z}, n \text{ Summe } n_1 \text{ und } n_2}{\langle a_1 + a_2, \sigma\rangle \to_{Aexp} n}$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad \text{falls } n_1 = \bot \text{ oder } n_2 = \bot}{\langle a_1 + a_2, \sigma\rangle \to_{Aexp} \bot}$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad n_i \in \mathbb{Z}, n \text{ Differenz von } n_1 \text{ und } n_2}{\langle a_1 - a_2, \sigma\rangle \to_{Aexp} n}$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad \text{falls } n_1 = \bot \text{ oder } n_2 = \bot}{\langle a_1 - a_2, \sigma\rangle \to_{Aexp} \bot}$$

Regelschreibweise vs. Funktionen

Sei $\mathsf{Int}+ = \mathsf{Int} \cup \{\bot\}$

```
AexpEval :: AExp -> (Zustand -> Int+)
AexpEval n :: Int s -> n
AexpEval x :: Loc s if Dom(s) contains x -> s(x)
AexpEval x :: Loc s if not(Dom(s) contains x) -> ⊥
AExpEval (a1 + a2) s -> let n1 = AExpEval a1 s
                            n2 = AExpEval a2 s
                        in
                        if n1 :: Int and n2 :: Int then n1 + n2
                        if n1 == ⊥ or n2 == ⊥ then ⊥
```


Operationale Semantik: Arithmetische Ausdrücke

- **Aexp** $a ::= \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 \,/\, a_2$

$$\langle a, \sigma\rangle \to_{Aexp} n \mid \bot$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad n_i \in \mathbb{Z}, n \text{ Produkt } n_1 \text{ und } n_2}{\langle a_1 * a_2, \sigma\rangle \to_{Aexp} n}$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad \text{falls } n_1 = \bot \text{ oder } n_2 = \bot}{\langle a_1 * a_2, \sigma\rangle \to_{Aexp} \bot}$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad n_i \in \mathbb{Z}, n_2 \neq 0, n \text{ Quotient } n_1, n_2}{\langle a_1 \div a_2, \sigma\rangle \to_{Aexp} n}$$

$$\frac{\langle a_1, \sigma\rangle \to_{Aexp} n_1 \qquad \langle a_2, \sigma\rangle \to_{Aexp} n_2 \qquad \text{falls } n_1 = \bot, n_2 = \bot \text{ oder } n_2 = 0}{\langle a_1 \div a_2, \sigma\rangle \to_{Aexp} \bot}$$

Arbeitsblatt 2.2: Jetzt seid ihr dran!

- In euren Gruppen-Arbeitsblättern unter `https://hackmd.informatik.uni-bremen.de/rfF0alFiS8y6nUtspD4YgA#` vervollständigt die Funktion

```
AexpEval :: AExp -> (Zustand -> Int+)
AexpEval n :: Int s -> n
AexpEval x :: Loc s if Dom(s) contains x -> s(x)
AexpEval x :: Loc s if not(Dom(s) contains x) -> ⊥
AExpEval (a1 + a2) s -> let n1 = AExpEval a1 s
                           n2 = AExpEval a2 s
                       in
                       if n1 :: Int and n2 :: Int then n1 + n2
                       if n1 == ⊥ or n2 == ⊥ then ⊥
```

- Ergänzt dies für * und für /
- Für $\bot$ könnt ihr einfach \bot schreiben.

Beispiel-Ableitungen

Sei $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5\rangle$.

$$\frac{\dfrac{\langle x,\sigma\rangle \to_{Aexp} 6 \quad \langle y,\sigma\rangle \to_{Aexp} 5}{\langle x+y,\sigma\rangle \to_{Aexp} 11} \quad \dfrac{\langle x,\sigma\rangle \to_{Aexp} 6 \quad \langle y,\sigma\rangle \to_{Aexp} 5}{\langle x-y,\sigma\rangle \to_{Aexp} 1}}{\langle (x+y)*(x-y),\sigma\rangle \to_{Aexp} 11}$$

$$\frac{\dfrac{\langle x,\sigma\rangle \to_{Aexp} 6 \quad \langle x,\sigma\rangle \to_{Aexp} 6}{\langle x*x,\sigma\rangle \to_{Aexp} 36} \quad \dfrac{\langle y,\sigma\rangle \to_{Aexp} 5 \quad \langle y,\sigma\rangle \to_{Aexp} 5}{\langle y*y,\sigma\rangle \to_{Aexp} 25}}{\langle (x*x)-(y*y),\sigma\rangle \to_{Aexp} 11}$$

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= 0 \mid 1 \mid a_1 == a_2 \mid a_1 < a_2 \mid !b \mid b_1 \mathbin{\&\&} b2 \mid b_1 \mathbin{||} b_2$
  $\langle b,\sigma\rangle \to_{Bexp} \mathit{true} \mid \mathit{false} \mid \bot$

Regeln:

$$\frac{}{\langle \mathbf{1},\sigma\rangle \to_{Bexp} \mathit{true}} \qquad \frac{}{\langle \mathbf{0},\sigma\rangle \to_{Bexp} \mathit{false}}$$

$$\frac{\langle a_1,\sigma\rangle \to_{Aexp} n_1 \quad \langle a_2,\sigma\rangle \to_{Aexp} n_2 \quad n_i \neq \bot, n_1 \text{ und } n_2 \text{ gleich}}{\langle a_1 == a_2,\sigma\rangle \to_{Bexp} \mathit{true}}$$

$$\frac{\langle a_1,\sigma\rangle \to_{Aexp} n_1 \quad \langle a_2,\sigma\rangle \to_{Aexp} n_2 \quad n_i \neq \bot, n_1 \text{ und } n_2 \text{ ungleich}}{\langle a_1 == a_2,\sigma\rangle \to_{Bexp} \mathit{false}}$$

$$\frac{\langle a_1,\sigma\rangle \to_{Aexp} n_1 \quad \langle a_2,\sigma\rangle \to_{Aexp} n_2 \quad n_1 = \bot \text{ or } n_2 = \bot}{\langle a_1 == a_2,\sigma\rangle \to_{Bexp} \bot}$$

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= 0 \mid 1 \mid a_1 == a_2 \mid a_1 < a_2 \mid !b \mid b_1 \mathbin{\&\&} b2 \mid b_1 \mathbin{||} b_2$
  $\langle b,\sigma\rangle \to_{Bexp} \mathit{true} \mid \mathit{false} \mid \bot$

Regeln:

$$\frac{\langle b,\sigma\rangle \to_{Bexp} \mathit{true}}{\langle !b,\sigma\rangle \to_{Bexp} \mathit{false}} \qquad \frac{\langle b,\sigma\rangle \to_{Bexp} \mathit{false}}{\langle !b,\sigma\rangle \to_{Bexp} \mathit{true}} \qquad \frac{\langle b,\sigma\rangle \to_{Bexp} \bot}{\langle !b,\sigma\rangle \to_{Bexp} \bot}$$

$$\frac{\langle b_1,\sigma\rangle \to_{Bexp} \mathit{false}}{\langle b_1 \mathbin{\&\&} b_2,\sigma\rangle \to_{Bexp} \mathit{false}} \qquad \frac{\langle b_1,\sigma\rangle \to_{Bexp} \bot}{\langle b_1 \mathbin{\&\&} b_2,\sigma\rangle \to_{Bexp} \bot}$$

$$\frac{\langle b_1,\sigma\rangle \to_{Bexp} \mathit{true} \quad \langle b_2,\sigma\rangle \to_{Bexp} t}{\langle b_1 \mathbin{\&\&} b_2,\sigma\rangle \to_{Bexp} t}$$

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= 0 \mid 1 \mid a_1 == a_2 \mid a_1 < a_2 \mid !b \mid b_1 \mathbin{\&\&} b2 \mid b_1 \mathbin{||} b_2$
  $\langle b,\sigma\rangle \to_{Bexp} \mathit{true} \mid \mathit{false} \mid \bot$

Regeln:

$$\frac{\langle b_1,\sigma\rangle \to_{Bexp} \mathit{true}}{\langle b_1 \mathbin{||} b_2,\sigma\rangle \to_{Bexp} \mathit{true}} \qquad \frac{\langle b_1,\sigma\rangle \to_{Bexp} \bot}{\langle b_1 \mathbin{||} b_2,\sigma\rangle \to_{Bexp} \bot}$$

$$\frac{\langle b_1,\sigma\rangle \to_{Bexp} \mathit{false} \quad \langle b_2,\sigma\rangle \to_{Bexp} t}{\langle b_1 \mathbin{||} b_2,\sigma\rangle \to_{Bexp} t}$$

Operationale Semantik: Anweisungen

- **Stmt** $c ::=$ **Idt** $=$ **Exp** | **if** (b) c_1 **else** c_2 | **while** (b) c | $c_1; c_2$ | $\{\,\}$

Beispiel:

$$\langle c,\sigma\rangle \to_{Stmt} \sigma' \mid \bot$$

$$\langle x = 5,\sigma\rangle \to_{Stmt} \sigma' \sigma[x \mapsto 5]$$

wobei $\sigma'(x) = 5$ und $\sigma'(y) = \sigma(y)$ für alle $y \neq x$
bzw. $\sigma' \stackrel{def}{=} \sigma[x \mapsto 5]$

Operationale Semantik: Anweisungen

- **Stmt** $c ::=$ **Idt** $=$ **Exp** | **if** (b) c_1 **else** c_2 | **while** (b) c | $c_1; c_2$ | $\{\,\}$

Regeln:

$$\frac{}{\langle \{\,\},\sigma\rangle \to_{Stmt} \sigma}$$

$$\frac{\langle a,\sigma\rangle \to_{Aexp} n \in \mathbb{Z}}{\langle x = a,\sigma\rangle \to_{Stmt} \sigma[x \mapsto n]} \qquad \frac{\langle a,\sigma\rangle \to_{Aexp} \bot}{\langle x = a,\sigma\rangle \to_{Stmt} \bot}$$

$$\frac{\langle c_1,\sigma\rangle \to_{Stmt} \sigma' \neq \bot \quad \langle c_2,\sigma'\rangle \to_{Stmt} \sigma'' \neq \bot}{\langle c_1; c_2,\sigma\rangle \to_{Stmt} \sigma''}$$

$$\frac{\langle c_1,\sigma\rangle \to_{Stmt} \bot}{\langle c_1; c_2,\sigma\rangle \to_{Stmt} \bot} \qquad \frac{\langle c_1,\sigma\rangle \to_{Stmt} \sigma' \neq \bot \quad \langle c_2,\sigma'\rangle \to_{Stmt} \bot}{\langle c_1; c_2,\sigma\rangle \to_{Stmt} \bot}$$

Operationale Semantik: Anweisungen

- **Stmt** $c ::=$ **Idt** $=$ **Exp** | **if** (b) c_1 **else** c_2 | **while** (b) c | $c_1; c_2$ | $\{\,\}$

Regeln:

$$\frac{\langle b,\sigma\rangle \to_{Bexp} \mathit{true} \quad \langle c_1,\sigma\rangle \to_{Stmt} \sigma'}{\langle \textbf{if } (b)\ c_1 \textbf{ else } c_2,\sigma\rangle \to_{Stmt} \sigma'}$$

$$\frac{\langle b,\sigma\rangle \to_{Bexp} \mathit{false} \quad \langle c_2,\sigma\rangle \to_{Stmt} \sigma'}{\langle \textbf{if } (b)\ c_1 \textbf{ else } c_2,\sigma\rangle \to_{Stmt} \sigma'}$$

$$\frac{\langle b,\sigma\rangle \to_{Bexp} \bot}{\langle \textbf{if } (b)\ c_1 \textbf{ else } c_2,\sigma\rangle \to_{Stmt} \bot}$$

Operationale Semantik: Anweisungen

- **Stmt** c ::= **Idt** = **Exp** | **if** (b) c_1 **else** c_2 | **while** (b) c | $c_1; c_2$ | { }

Regeln:

$$\frac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{false}}{\langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma}$$

$$\frac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{true} \qquad \langle c, \sigma\rangle \rightarrow_{Stmt} \sigma' \qquad \langle \textbf{while}\ (b)\ c, \sigma'\rangle \rightarrow_{Stmt} \sigma''}{\langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma''}$$

$$\frac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{true} \qquad \langle c, \sigma\rangle \rightarrow_{Stmt} \bot}{\langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \bot} \qquad \frac{\langle b, \sigma\rangle \rightarrow_{Bexp} \bot}{\langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \bot}$$

Beispiel

```
x = 1;
while (y != 0) {
 y = y − 1;
 x = 2 ∗ x;
}
// x = 2^y
```

$\sigma \stackrel{def}{=} \langle y \mapsto 2\rangle$

$$\dfrac{\dfrac{\langle 1, \sigma\rangle \rightarrow_{Aexp} 1}{\langle x = 1, \sigma\rangle \rightarrow_{Stmt} \sigma[x \mapsto 1] := \sigma_1} \qquad \dfrac{\dfrac{\langle y, \sigma_1\rangle \rightarrow_{Aexp} 2}{\langle y! = 0, \sigma_1\rangle \rightarrow_{Bexp} \mathit{true}} \quad \dfrac{(A)}{\langle y = y - 1; x = 2 * x, \sigma_1\rangle \rightarrow_{Stmt} ?} \quad \dfrac{(B)}{\langle w, ?\rangle \rightarrow_{Stmt} ?}}{\langle \textbf{while}\ (y! = 0)\ \{y = y - 1; x = 2 * x\}, \sigma_1\rangle \rightarrow_{Stmt} ?}}{\langle x = 1; \underbrace{\textbf{while}\ (y! = 0)\ \{y = y - 1; x = 2 * x\}}_{w}, \sigma\rangle \rightarrow_{Stmt} ?}$$

(A)

$$\dfrac{\dfrac{\langle y - 1, \sigma_1\rangle \rightarrow_{Aexp} 1}{\langle y = y - 1, \sigma_1\rangle \rightarrow_{Stmt} \sigma_1[y \mapsto 1] := \sigma_2} \quad \dfrac{\langle 2 * x, \sigma_2\rangle \rightarrow_{Aexp} 2}{\langle x = 2 * x, \sigma_2\rangle \rightarrow_{Stmt} \sigma_2[x \mapsto 2] := \sigma_3}}{\langle y = y - 1; x = 2 * x, \sigma_1\rangle \rightarrow_{Stmt} \sigma_3}$$

$$\dfrac{\dfrac{\langle 1, \sigma\rangle \rightarrow_{Aexp} 1}{\langle x = 1, \sigma\rangle \rightarrow_{Stmt} \sigma_1} \qquad \dfrac{\dfrac{\langle y, \sigma_1\rangle \rightarrow_{Aexp} 2}{\langle y! = 0, \sigma_1\rangle \rightarrow_{Bexp} \mathit{true}} \quad \dfrac{(A)}{\langle y = y - 1; x = 2 * x, \sigma_1\rangle \rightarrow_{Stmt} \sigma_3} \quad \dfrac{(B)}{\langle w, \sigma_3\rangle \rightarrow_{Stmt} ?}}{\langle \textbf{while}\ (y! = 0)\ \{y = y - 1; x = 2 * x\}, \sigma_1\rangle \rightarrow_{Stmt} ?}}{\langle x = 1; \underbrace{\textbf{while}\ (y! = 0)\ \{y = y - 1; x = 2 * x\}}_{w}, \sigma\rangle \rightarrow_{Stmt} ?}$$

(B)

$$\dfrac{\dfrac{\langle y, \sigma_3\rangle \rightarrow_{Aexp} 1}{\langle y! = 0, \sigma_3\rangle \rightarrow_{Bexp} \mathit{true}} \quad \dfrac{\dfrac{\langle y - 1, \sigma_3\rangle \rightarrow_{Aexp} 0}{\langle y = y - 1, \sigma_3\rangle \rightarrow_{Stmt} \sigma_3[y \mapsto 0] := \sigma_4} \quad \dfrac{\langle 2 * x, \sigma_4\rangle \rightarrow_{Aexp} 4}{\langle x = 2 * x, \sigma_4\rangle \rightarrow_{Stmt} \sigma_4[x \mapsto 4] := \sigma_5}}{\langle y = y - 1; x = 2 * x, \sigma_3\rangle \rightarrow_{Stmt} \sigma_5} \quad \dfrac{(C)}{\langle w, \sigma_5\rangle \rightarrow_{Stmt} \sigma_5}}{\langle w, \sigma_3\rangle \rightarrow_{Stmt} \sigma_5}$$

$$\left.\dfrac{\dfrac{\langle y, \sigma_5\rangle \rightarrow_{Aexp} 0}{\langle y! = 0, \sigma_3\rangle \rightarrow_{Bexp} \mathit{false}}}{\langle w, \sigma_5\rangle \rightarrow_{Stmt} \sigma_5}\right\}(C)$$

$$\underbrace{\textbf{while}\ (y! = 0)\ \{y = y - 1; x = 2 * x\}}_{w}$$

$$\dfrac{\cdots \qquad \dfrac{\dfrac{\langle y, \sigma_1\rangle \rightarrow_{Aexp} 2}{\langle y! = 0, \sigma_1\rangle \rightarrow_{Bexp} \mathit{true}} \quad \dfrac{(A)}{\langle y = y - 1; x = 2 * x, \sigma_1\rangle \rightarrow_{Stmt} \sigma_3} \quad \dfrac{(B)}{\langle w, \sigma_3\rangle \rightarrow_{Stmt} \sigma_5}}{\langle \textbf{while}\ (y! = 0)\ \{y = y - 1; x = 2 * x\}, \sigma_1\rangle \rightarrow_{Stmt} \sigma_5}}{\langle x = 1; \underbrace{\textbf{while}\ (y! = 0)\ \{y = y - 1; x = 2 * x\}}_{w}, \sigma\rangle \rightarrow_{Stmt} \sigma_5}$$

$$\begin{aligned} \sigma_5 &= \sigma_4[x \mapsto 4] = \sigma_3[y \mapsto 0][x \mapsto 4] = \sigma_2[x \mapsto 2][y \mapsto 0][x \mapsto 4] \\ &= \sigma_1[y \mapsto 1][x \mapsto 2][y \mapsto 0][x \mapsto 4] = \langle y \mapsto 2\rangle[y \mapsto 1][x \mapsto 2][y \mapsto 0][x \mapsto 4] \\ &= \langle y \mapsto 0, x \mapsto 4\rangle \end{aligned}$$

und es gilt $\sigma_5(x) = 4 = 2^2 = 2^{\sigma_1(y)}$

Lineare, abgekürzte Schreibweise

```
// ⟨y ↦ 2⟩
x = 1;
// ⟨y ↦ 2, x ↦ 1⟩
while (y != 0) {
 y = y − 1;
 x = 2 ∗ x;
}
```

Lineare, abgekürzte Schreibweise

```
// ⟨y ↦ 2⟩
x = 1;                                   // Ableitung für x = 1
// ⟨y ↦ 2, x ↦ 1⟩
while(y!=0)  // ⟨y! = 0, ⟨y ↦ 2, x ↦ 1⟩⟩ →Bexp true
|             y = y − 1;                 // Ableitung für y = y − 1
|            // ⟨y ↦ 1, x ↦ 1⟩
|            x = 2 * x;                  // Ableitung für x = 2 * x
|            // ⟨y ↦ 1, x ↦ 2⟩
while (y != 0) {
 y = y − 1;
 x = 2 * x;
}
```


Lineare, abgekürzte Schreibweise

```
// ⟨y ↦ 2⟩
x = 1;
// ⟨y ↦ 2, x ↦ 1⟩
while(y!=0)  // ⟨y! = 0, ⟨y ↦ 2, x ↦ 1⟩⟩ →Bexp true
|             y = y − 1;              // Ableitung für y = y − 1
|            // ⟨y ↦ 1, x ↦ 1⟩
|            x = 2 * x;               // Ableitung für x = 2 * x
|            // ⟨y ↦ 1, x ↦ 2⟩
while(y!=0)  // ⟨y! = 0, ⟨y ↦ 1, x ↦ 2⟩⟩ →Bexp true
|            y = y − 1;
|            // ⟨y ↦ 0, x ↦ 2⟩
|            x = 2 * x;
|            // ⟨y ↦ 0, x ↦ 4⟩
while(y!=0)  // ⟨y! = 0, ⟨y ↦ 0, x ↦ 4⟩⟩ →Bexp false
// ⟨y ↦ 0, x ↦ 4⟩
```


Was haben wir gezeigt?

```
// ⟨y ↦ 2⟩
x = 1;
// ⟨y ↦ 2, x ↦ 1⟩
while (y != 0) {
 y = y − 1;
 x = 2 * x;
}
// ⟨y ↦ 0, x ↦ 4⟩
```

- Für **einen festen Anfangszustand** $\sigma_1 = \langle y \mapsto 2\rangle$ gilt am Ende $x = 4 = 2^2 = 2^{\sigma_1(y)}$.
- Gilt das für alle?
- Für welche nicht?
- Wie kann man das für alle Anfangs-Zustände, für die es gilt, zeigen?

Was passiert hier?

```
// ⟨y ↦ −1⟩
x = 1;
while (y != 0) {
 y = y − 1;
 x = 2 * x;
}
```

- Ableitung terminiert nicht (Ableitungsbaum der Auswerttung der while-Schleife wächst unendlich)
- In linearer Schreibweise geht es immer wieder unten weiter.

Arbeitsblatt 2.3: Jetzt seid ihr dran!

- Werten Sie das nebenstehende Program aus für den Anfangszustand $\langle x \mapsto 5, y \mapsto 2\rangle$
- Geben Sie die Auswertung in abgekürzter Schreibweise an.
- Welche Beziehung gilt am Ende des Programs zwischen den Werten von x und y im Endzustand und im Anfangszustand?

```
while (y != 0) {
 x = x * x;
 y = y − 1;
}
```


Lineare, abgekürzte Schreibweise

```
while(y!=0)  // ⟨x ↦ 5, y ↦ 2⟩                                σ1
|            // ⟨y! = 0, ⟨x ↦ 5, y ↦ 2⟩⟩ →Bexp true
|            x = x * x;
|            // ⟨x ↦ 25, y ↦ 2⟩
|            y = y − 1;
|            // ⟨x ↦ 25, y ↦ 1⟩
while(y!=0)  // ⟨y! = 0, ⟨x ↦ 25, y ↦ 1⟩⟩ →Bexp true
|            x = x * x;
|            // ⟨x ↦ 625, y ↦ 1⟩
|            y = y − 1;
|            // ⟨x ↦ 625, y ↦ 0⟩                               σ5
while(y!=0)  // ⟨y! = 0, ⟨x ↦ 625, y ↦ 0⟩⟩ →Bexp false
// ⟨x ↦ 625, y ↦ 0⟩
```

Und es gilt $625 = 5^4 = 5^{2^2}$ bzw. $\sigma_5(x) = \sigma_1(x)^{2^{\sigma_1(y)}}$

Äquivalenz arithmetischer Ausdrücke

Gegeben zwei Aexp a_1 and a_2

- Sind sie gleich?

$$a_1 \sim_{Aexp} a_2 \text{ gdw } \forall \sigma, n. \langle a_1, \sigma\rangle \rightarrow_{Aexp} n \Leftrightarrow \langle a_2, \sigma\rangle \rightarrow_{Aexp} n$$

```
(x*x) + 2*x*y + (y*y)      und      (x+y) *  (x+y)
```

- Wann sind sie gleich?

$$\forall \sigma, n. \langle a_1, \sigma\rangle \rightarrow_{Aexp} n \Leftrightarrow \langle a_2, \sigma\rangle \rightarrow_{Aexp} n$$

```
x*x      und      8*x+9
x*x      und      x*x+1
```


Äquivalenz Boolscher Ausdrücke

Gegeben zwei Bexp-Ausdrücke b_1 and b_2

- Sind sie gleich?

$$b_1 \sim_{Bexp} b_2 \text{ iff } \forall \sigma, b. \langle b_1, \sigma\rangle \rightarrow_{Bexp} b \Leftrightarrow \langle b_2, \sigma\rangle \rightarrow_{Bexp} b$$

```
A || (A && B)      und      A
```

Beweisen

Zwei Programme c_0, c_1 sind äquivalent gdw. sie die gleichen Zustandsveränderungen bewirken. Formal definieren wir

Definition

$$c_0 \sim c_1 \text{ iff } \forall \sigma, \sigma'. \langle c_0, \sigma\rangle \rightarrow_{Stmt} \sigma' \Leftrightarrow \langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma'$$

Ein einfaches Beispiel:

Lemma

*Sei $w \equiv$ **while** (b) c mit $b \in$ **Bexp**, $c \in$ **Stmt**.*
*Dann gilt: $w \sim$ **if** (b) $\{c; w\}$ **else** $\{\,\}$*

Beweis

Gegeben beliebiger Programmzustand σ. Zu zeigen ist, dass sowohl w also auch **if** (b) $\{c; w\}$ **else** $\{\,\}$ zu dem selben Programmzustand auswerten oder beide zu einem Fehler. Der Beweis geht per Fallunterscheidung über die Auswertung von Teilausdrücken bzw. Teilprogrammen.

1. $\langle b, \sigma\rangle \rightarrow_{Bexp} \perp$:

$$\langle \textbf{while } (b)\ c, \sigma\rangle \rightarrow_{Stmt} \perp$$
$$\langle \textbf{if } (b)\ \{c; w\} \textbf{ else } \{\,\}, \sigma\rangle \rightarrow_{Stmt} \perp$$

2. $\langle b, \sigma\rangle \rightarrow_{Bexp}$ *false*:

$$\langle \textbf{while } (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma$$
$$\langle \textbf{if } (b)\ \{c; w\} \textbf{ else } \{\,\}, \sigma\rangle \rightarrow_{Stmt} \langle\{\,\}, \sigma\rangle \rightarrow_{Stmt} \sigma$$

Beweis II

3. $\langle b, \sigma\rangle \rightarrow_{Bexp}$ *true*:
   1. $\langle c, \sigma\rangle \rightarrow_{Stmt} \sigma'$

$$\langle \overbrace{\textbf{while } (b)\ c}^{w}, \sigma\rangle \rightarrow_{Stmt} \langle c, \sigma\rangle \rightarrow_{Stmt} \sigma'$$
$$\langle w, \sigma'\rangle \rightarrow_{Stmt} \sigma''$$
$$\langle \textbf{if } (b)\ \{c; w\} \textbf{ else } \{\,\}, \sigma\rangle \rightarrow_{Stmt} \langle\{c; w\}, \sigma\rangle \rightarrow_{Stmt} \langle c, \sigma\rangle \rightarrow_{Stmt} \sigma'$$
$$\langle w, \sigma'\rangle \rightarrow_{Stmt} \sigma''$$

   2. $\langle c, \sigma\rangle \rightarrow_{Stmt} \perp$

$$\langle \overbrace{\textbf{while } (b)\ c}^{w}, \sigma\rangle \rightarrow_{Stmt} \langle c, \sigma\rangle \rightarrow_{Stmt} \perp$$
$$\langle \textbf{if } (b)\ \{c; w\} \textbf{ else } \{\,\}, \sigma\rangle \rightarrow_{Stmt} \langle\{c; w\}, \sigma\rangle \rightarrow_{Stmt} \langle c, \sigma\rangle \rightarrow_{Stmt} \perp$$

Zusammenfassung

- Operationale Semantik als ein Mittel zur Beschreibung der Semantik
- Auswertungsregeln arbeiten entlang der syntaktischen Struktur
- Werten Ausdrücke zu Werten aus und Programme zu Zuständen (zu gegebenen Zustand)
- Fragen zu Programmen: Gleichheit