Korrekte Software: Grundlagen und Methoden
Vorlesung 2 vom 28.04.20
Operationale Semantik

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2020

Fahrplan

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

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

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** $\mathbf{V}$ (hier ganze Zahlen) aus.
- Adressen **Loc** sind hier Programmvariablen (Namen): $\mathbf{Loc} = \mathbf{Idt}$
- 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*)
- $f[n/x]$ für den Update an der Stelle x mit dem Wert n:

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

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

$$\langle\rangle(x) = \bot$$

Arbeitsblatt 2.1: Jetzt seid ihr dran!

- In euren Gruppen-Arbeitsblättern unter `https://hackmd.informatik.uni-bremen.de/s/SkVLK1Q_I` 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 $\langle x \mapsto y, b \mapsto 6, y \mapsto 2 \rangle$
  - D $\langle x \mapsto 3, b \mapsto 6, y \mapsto 2 \rangle$
- Update von Zuständen:
  - A $\langle x \mapsto 1, a \mapsto 3 \rangle[1/y] :=??$
  - B $\langle x \mapsto 1, a \mapsto 3 \rangle[3/x] :=??$
  - C $\langle x \mapsto 1, a \mapsto 3 \rangle[3/x][y/1][4/x] :=??$

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 $\langle x \mapsto y, b \mapsto 6, y \mapsto 2 \rangle$ −
  - D $\langle x \mapsto 3, b \mapsto 6, y \mapsto 2 \rangle$ +
- Update von Zuständen:
  - A $\langle x \mapsto 1, a \mapsto 3 \rangle[1/y] := \langle x \mapsto 1, a \mapsto 3, y \mapsto 1 \rangle$
  - B $\langle x \mapsto 1, a \mapsto 3 \rangle[3/x] := \langle x \mapsto 3, a \mapsto 3 \rangle$
  - C $\langle x \mapsto 1, a \mapsto 3 \rangle[3/x][y/1][4/x] := \langle x \mapsto 4, y \mapsto 1, a \mapsto 3 \rangle$

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:

$$\frac{}{\langle n, \sigma \rangle \to_{Aexp} n}$$

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

Regelschreibweise vs. Funktionen

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

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

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

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

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

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

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

Regelschreibweise vs. Funktionen

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

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

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

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

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

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

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

Arbeitsblatt 2.2: Jetzt seid ihr dran!

- In euren Gruppen-Arbeitsblättern unter `https://hackmd.informatik.uni-bremen.de/s/SkVLK1Q_I` 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 $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

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

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

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= 0 \mid 1 \mid a_1 == a_2 \mid a_1 < a_2 \mid !b \mid b_1 \,\&\&\, b2 \mid b_1 \mid\mid b_2$

$$\langle b,\sigma\rangle \to_{Bexp} true \mid false \mid \bot$$

Regeln:

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

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

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

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

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= 0 \mid 1 \mid a_1 == a_2 \mid a_1 < a_2 \mid !b \mid b_1 \,\&\&\, b2 \mid b_1 \mid\mid b_2$

$$\langle b,\sigma\rangle \to_{Bexp} true \mid false \mid \bot$$

Regeln:

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

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

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

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= 0 \mid 1 \mid a_1 == a_2 \mid a_1 < a_2 \mid !b \mid b_1 \,\&\&\, b2 \mid b_1 \mid\mid b_2$

$$\langle b,\sigma\rangle \to_{Bexp} true \mid false \mid \bot$$

Regeln:

$$\dfrac{\langle b_1,\sigma\rangle \to_{Bexp} true}{\langle b_1 \,\&\&\, b_2,\sigma\rangle \to_{Bexp} true} \qquad \dfrac{\langle b_1,\sigma\rangle \to_{Bexp} \bot}{\langle b_1 \,\&\&\, b_2,\sigma\rangle \to_{Bexp} \bot}$$

$$\dfrac{\langle b_1,\sigma\rangle \to_{Bexp} false \quad \langle b_2,\sigma\rangle \to_{Bexp} t}{\langle b_1 \mid\mid b_2,\sigma\rangle \to_{Bexp} t}$$

Operationale Semantik: Anweisungen

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

Beispiel:

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

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

wobei $\sigma'(x) = 5$ und $\sigma'(y) = \sigma(y)$ für alle $y \neq x$

Operationale Semantik: Anweisungen

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

Regeln:

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

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

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

$$\dfrac{\langle c_1,\sigma\rangle \to_{Stmt} \bot}{\langle c_1; c_2,\sigma\rangle \to_{Stmt} \bot}$$

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

Operationale Semantik: Anweisungen

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

Regeln:

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

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

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

Operationale Semantik: Anweisungen

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

Regeln:

$$\dfrac{\langle b,\sigma\rangle \to_{Bexp} false}{\langle \textbf{while } (b) \; c,\sigma\rangle \to_{Stmt} \sigma}$$

$$\dfrac{\langle b,\sigma\rangle \to_{Bexp} true \quad \langle c,\sigma\rangle \to_{Stmt} \sigma' \quad \langle \textbf{while } (b) \; c,\sigma'\rangle \to_{Stmt} \sigma''}{\langle \textbf{while } (b) \; c,\sigma\rangle \to_{Stmt} \sigma''}$$

$$\dfrac{\langle b,\sigma\rangle \to_{Bexp} true \quad \langle c,\sigma\rangle \to_{Stmt} \bot}{\langle \textbf{while } (b) \; c,\sigma\rangle \to_{Stmt} \bot} \qquad \dfrac{\langle b,\sigma\rangle \to_{Bexp} \bot}{\langle \textbf{while } (b) \; c,\sigma\rangle \to_{Stmt} \bot}$$

Beispiel

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

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

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

(A)

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

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

(B)

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

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

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

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

(1)

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

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(w)  // ⟨y! = 0, ⟨y ↦ 2, x ↦ 1⟩⟩ →Bexp 1
|           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(w)  // ⟨y! = 0, ⟨y ↦ 2, x ↦ 1⟩⟩ →Bexp 1
|           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(w)  // ⟨y! = 0, ⟨y ↦ 1, x ↦ 2⟩⟩ →Bexp 1
|           y = y - 1;
|         // ⟨y ↦ 0, x ↦ 1⟩
|           x = 2 * x;
|         // ⟨y ↦ 0, x ↦ 4⟩
while(w)  // ⟨y! = 0, ⟨y ↦ 0, x ↦ 2⟩⟩ →Bexp 0
// ⟨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 Programm 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(w)  // ⟨x ↦ 5, y ↦ 2⟩                          σ1
|         // ⟨y! = 0, ⟨x ↦ 5, y ↦ 2⟩⟩ →Bexp 1
|           x = x * x;
|         // ⟨x ↦ 25, y ↦ 2⟩
|           y = y - 1;
|         // ⟨x ↦ 25, y ↦ 1⟩
while(w)  // ⟨y! = 0, ⟨x ↦ 25, y ↦ 1⟩⟩ →Bexp 1
|           x = x * x;
|         // ⟨x ↦ 625, y ↦ 1⟩
|           y = y - 1;
|         // ⟨x ↦ 625, y ↦ 0⟩                        σ5
while(w)  // ⟨y! = 0, ⟨x ↦ 625, y ↦ 0⟩⟩ →Bexp 0
// ⟨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?

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

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

- Wann sind sie gleich?

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

```
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?

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

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


Beweisen

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

Definition

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

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. $\langle b, \sigma\rangle \to_{Bexp} \bot$:

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

2. $\langle b, \sigma\rangle \to_{Bexp} \mathit{false}$:

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

Beweis II

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

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

   2. $\langle c, \sigma\rangle \to_{Stmt} \bot$

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

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