Korrekte Software: Grundlagen und Methoden

Vorlesung 1 vom 03.04.24

Einführung

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2024

Organisatorisches

- Veranstalter:

Serge Autexier
serge.autexier@dfki.de
Cartesium 1.49[1], Tel. 59834

Christoph Lüth
christoph.lueth@dfki.de
MZH 4186, Tel. 59830

- Termine:
  - Mittwoch, 10 – 12, MZH 5600
  - Donnerstag, 10 – 12, MZH 5600
- Webseite: https://www.informatik.uni-bremen.de/~cxl/lehre/ksgm.ss24

Veranstaltungskonzept

- Aus den letzten Jahren: **integrierte Veranstaltung** statt **langer Vorlesung**.
- Kürzere **Vortragseinheiten**, dazwischen **Arbeitsfragen** (Kurzübungen)
- Wöchentliche **Übungsaufgaben** zur Vertiefung
- Technisch:
  - Fragen/Kurzübungen in **HedgeDoc**: `http://hackmd.informatik.uni-bremen.de/`
  - Übungsblätter als **Markdown**, Abgabe über `gitlab`.

Prüfungsform

- 10 Übungsblätter (geplant)
- **Bewertung**:
  - A (sehr gut, 1.3) — nichts zu meckern, keine/kaum Fehler
  - B (gut, 2.3) — kleine Fehler, sonst gut
  - C (befriedigend, 3.3) — größere Fehler oder Mängel
  - Nicht bearbeitet — oder mehr Fehler als Bearbeitung
- **Prüfungsleistung**:
  - **Mündliche Prüfung**: Einzelprüfung ca. 20– 30 Minuten
  - **Übungsbetrieb** (bis zu 15% Bonuspunkte, keine Voraussetzung)

Übungsbetrieb

- Abgabe und Korrektur des Übungsbetriebs erfolgt über **gitlab**.
- Dazu legt **pro Gruppe** ein Repository an.
- Ladet uns (`clueth`, `autexier`) als Developer ein.
- Für jedes Übungsblatt:

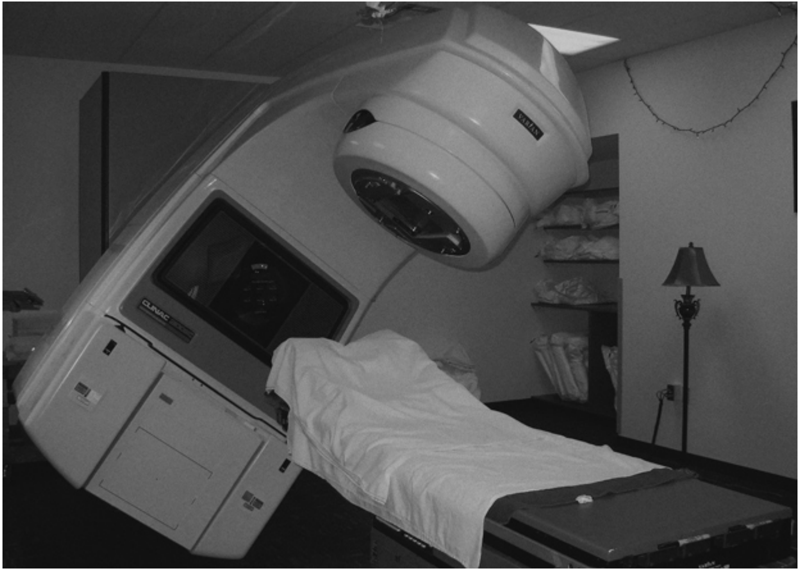
1. Das Übungsblatt ladet ihr von der Webseite herunter und bearbeitet es **elektronisch**.
2. Die Lösung wird als Markdown abgelegt (bitte Namen `uebung-XX.md` nicht verändern; Zusatzmaterial als `uebung-XX-`...wenn nötig), und ladet es **vor** dem Abgabezeitpunkt hoch (`push`).
3. Nach dem Abgabezeitpunkt laden wir die Änderungen herunter (`pull`), korrigieren direkt im Markdown, fügen die Bewertung hinzu, und laden die Korrektur wieder hoch (`push`)

Arbeitsblatt 1.1: Jetzt seid ihr dran!

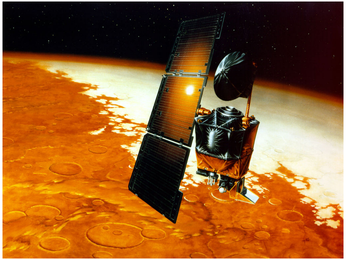
- Gruppiert euch in Gruppen zu drei Teilnehmenden!
- Tragt eure Namen in der Übersicht ein
  `https://hackmd.informatik.uni-bremen.de/s/iwDedtWRO#`
- Und kreiert eine eigene Hackmd Arbeitsblatt Seite pro Gruppe und verlinkt sie auf obiger Übersichtsseite.
- Auf diesem Arbeitsblatt bearbeitet ihr die Arbeitsfragen im Laufe des Kurses.
- Bitte nur in "eurem" Arbeitsblatt arbeiten
- Die Arbeitsblätter sind nicht notenrelevant.

I. Warum Korrekte Software?

Software-Disaster I: Therac-25



Software-Disasters II: Space



Software-Disaster III: AT&T (15.01.1990)

```
while (! empty(ring_rcv_buffer)
      && ! empty(side_buffer empty)) {
 initialize pointer to first message buffer;
 get copy of buffer;
 switch (message) {
   case (incoming_message):
     if (sender is out_of_service) {
       if (empty(ring_wrt_buffer)) {
         send "in service" to status map;
       } else {
         break;
       }
       process incoming message, set up pointers;
       break;
       }
   }
 do optional parameter work;
 }
```

Software-Disaster IV: Ungeplantes Übergewicht

- „A software mistake caused a Tui flight to take off heavier than expected as female passengers using the title “Miss” were classified as children [...]“
- 38 erwachsene Passagiere als Kinder (35kg) statt als Erwachsene (69kg) klassifiziert.

$$38 \cdot (69\,kg - 35\,kg) = 1292\,kg$$

- Software „was programmed in an unnamed foreign country where the title “Miss” is used for a child and “Ms” for an adult female.“

Quelle: *Guardian, 09.04.2021.*

`https://www.theguardian.com/world/2021/apr/09/tui-plane-serious-incident-every-miss-on-board-child-weight-birmingham-majorca`

Software-Disaster V: Der Horizon-Skandal

- 1999 wurde für die lokalen Postämter in Großbritannien das System *Horizon* der Firma Fujitsu für Buchhaltung und Lagerhaltung eingeführt.
- Das System war fehlerhaft, so dass gelegentlich nicht-existente Fehlbestände angezeigt wurden.
- Das Post Office hat trotzdem die Fehlbeständen von den lokalen Postbeamten (subpostmaster) eingetrieben; einige wurden angeklagt und verurteilt, andere privatinsolvent oder schieden aus.
- Erste Berichte über die Fehler tauchten 2005 auf, und wurden 2009 in der Presse publik.
- Erst 2019 nach einer Sammelklage wurden die Fehler amtlich vom High Court festgestellt, und die bis dahin ergangenen Urteile für unrechtmäßig erklärt.

Software-Disaster V: Der Horizon-Skandal

- 1999 wurde für die lokalen Postämter in Großbritannien das System *Horizon* der Firma Fujitsu für Buchhaltung und Lagerhaltung eingeführt.
- Das System war fehlerhaft, so dass gelegentlich nicht-existente Fehlbestände angezeigt wurden.
- Das Post Office hat trotzdem die Fehlbeständen von den lokalen Postbeamten (subpostmaster) eingetrieben; einige wurden angeklagt und verurteilt, andere privatinsolvent oder schieden aus.
- Erste Berichte über die Fehler tauchten 2005 auf, und wurden 2009 in der Presse publik.
- Erst 2019 nach einer Sammelklage wurden die Fehler amtlich vom High Court festgestellt, und die bis dahin ergangenen Urteile für unrechtmäßig erklärt.
- *Horizon* läuft immer noch, Fujitsu hat einen Vertrag über 2.4 Mrd Pfund.

Quellen: `https://www.bbc.com/news/business-56718036`,

`https://www.theguardian.com/uk-news/2024/feb/02/post-office-scandal-key-takeaways-latest-court-hearings`

Arbeitsblatt 1.2: Jetzt seid ihr dran!

- Sucht im Netz nach weiteren Software-Disastern:

  1. Was ist passiert?
  2. Wie ist es passiert?
  3. Was war der Softwarefehler?

- Quellen: Suchmaschine nach Wahl ("software disasters"), The Risks Digest, `https://catless.ncl.ac.uk/Risks/`

II. Inhalt der Vorlesung

Themen

Korrekte Software im Lehrbuch:

- Spielzeugsprache
- Wenig Konstrukte
- Kleine Beispiele

Korrekte Software im Einsatz:

- Richtige Programmiersprache
- Mehr als nur ganze Zahlen
- Skalierbarkeit — wie können große Programme verifiziert werden?

Inhalt

- Grundlagen:
  - Beweis der **Korrektheit** von Programmen: der **Floyd-Hoare-Kalkül**
  - **Bedeutung** von Programmen: **Semantik**
- Betrachtete Programmiersprache: “C0” (erweiterte Untermenge von C)
- Erweiterung der Programmkonstrukte und des Hoare-Kalküls:
  1. Referenzen (Zeiger)
  2. Funktion und Prozeduren (Modularität)
  3. Reiche **Datenstrukturen** (Felder, `struct`)

Fahrplan

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

III. Warum Semantik?

Idee

- Was wird hier berechnet?

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

Idee

- Was wird hier berechnet? $p = n!$
- Warum? Wie können wir das **beweisen**?

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

Idee

- Was wird hier berechnet? $p = n!$
- Warum? Wie können wir das **beweisen**?
- Wir berechnen symbolisch, welche Werte Variablen über den Programmverlauf annehmen.

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

Semantik von Programmiersprachen

Drei wesentliche Möglichkeiten:

- **Operationale Semantik**: Ausführung auf einer **abstrakten** Maschine
- **Denotationale Semantik**: Abbildung in ein **mathematisches Objekt**
- **Axiomatische Semantik**: Beschreibung anhand der **Eigenschaften**

Arbeitsblatt 1.3: Maschinen und Funktionen

Was genau kann man sich unter “abstrakten Maschine” vorstellen?

Betrachtet als Beispiele:

- Eine Taschenlampe
- Eine Waschmaschine
- Einen Taschenrechner

Was ist hier die Abstraktion?

Unsere Sprache C0

- C0 ist eine **Untermenge** der Sprache C
- C0-Programme sind **ausführbare** C-Programme
- Grundausbaustufe:
  - Zuweisungen, Fallunterscheidungen, Schleifen
  - Datentypen: ganze Zahlen mit Arithmetik
  - Relationen: Vergleich ($=$, $\leq$)
  - Boolsche Operatoren: Konjunktion, Disjunktion, Negation
- 1. Ausbaustufe: Felder und Strukturen
- 2. Ausbaustufe: Fehler und Ausnahmen
- 3. Ausbaustufe: Funktionen und Prozeduren (nur Ausblick)
- 4. Ausbaustufe: Referenzen (nur Ausblick)
- Fehlt: **union**, **goto**, ...

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

p	?
c	?
n	3

$\rightsquigarrow$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \rightsquigarrow$$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline \end{array} \leadsto$$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline\end{array} \leadsto$$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline \end{array} \rightsquigarrow$$

$$\begin{array}{|c|c|}\hline p & 1 \\ \hline c & 2 \\ \hline n & 3 \\ \hline \end{array} \rightsquigarrow$$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$
\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \leadsto
\begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \leadsto
\begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline \end{array} \leadsto
\begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline \end{array} \leadsto
$$

$$
\begin{array}{|c|c|}\hline p & 1 \\ \hline c & 2 \\ \hline n & 3 \\ \hline \end{array} \leadsto
\begin{array}{|c|c|}\hline p & 2 \\ \hline c & 2 \\ \hline n & 3 \\ \hline \end{array} \leadsto
$$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline\end{array} \leadsto$$

$$\begin{array}{|c|c|}\hline p & 1 \\ \hline c & 2 \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 2 \\ \hline c & 2 \\ \hline n & 3 \\ \hline\end{array} \leadsto \begin{array}{|c|c|}\hline p & 2 \\ \hline c & 3 \\ \hline n & 3 \\ \hline\end{array} \leadsto$$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline \end{array} \leadsto$$

$$\begin{array}{|c|c|}\hline p & 1 \\ \hline c & 2 \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 2 \\ \hline c & 2 \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 2 \\ \hline c & 3 \\ \hline n & 3 \\ \hline \end{array} \leadsto \begin{array}{|c|c|}\hline p & 6 \\ \hline c & 3 \\ \hline n & 3 \\ \hline \end{array} \leadsto$$

Operationale Semantik

- Kernkonzept: Zustandstandsübergänge einer abstrakten Maschine
- Abstrakte Maschine hat **impliziten Zustand**
- Zustand ordnet **Adressen** veränderliche **Werte** zu
- Konkretes Beispiel: $n \mapsto 3$, p und c undefiniert

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

$$\begin{array}{|c|c|}\hline p & ? \\ \hline c & ? \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 1 \\ \hline c & ? \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 1 \\ \hline c & 1 \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow$$

$$\begin{array}{|c|c|}\hline p & 1 \\ \hline c & 2 \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 2 \\ \hline c & 2 \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 2 \\ \hline c & 3 \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 6 \\ \hline c & 3 \\ \hline n & 3 \\ \hline\end{array} \rightsquigarrow \begin{array}{|c|c|}\hline p & 6 \\ \hline c & 4 \\ \hline n & 3 \\ \hline\end{array}$$

Arbeitsblatt 1.4: Operationale Semantik

Gegeben folgendes C0-Programm:

```
x= 0;
while (n > 0) {
  x= x+ n*n;
  n= n-1;
  }
```

Entwickeln Sie die ersten zehn Schritte der operationalen Semantik wie im Beispiel oben für den initialen Zustand

n	4
x	?

$\rightsquigarrow \ldots$

Denotationale Semantik

- Kernkonzept: Abbildung von Programmen auf mathematisches Gegenstück (**Denotat**)
- **Partielle** Funktionen zwischen Zuständen $[\![c]\!] : \Sigma \rightharpoonup \Sigma$
- Beispiel:

```
p = 1;
c = 1; // p1
while (c <= n) {
  p = p * c;
  c = c + 1; // p2
  }
// p3
```

$$[\![p_1]\!](\sigma) = \sigma[p \mapsto 1][c \mapsto 1]$$
$$[\![p_2]\!](\sigma) = \sigma[p \mapsto \sigma(p) * \sigma(c)][c \mapsto \sigma(c) + 1]$$
$$[\![p_3]\!](\sigma) = ???$$

Denotationale Semantik

- Kernkonzept: Abbildung von Programmen auf mathematisches Gegenstück (**Denotat**)
- **Partielle** Funktionen zwischen Zuständen $[\![c]\!] : \Sigma \rightharpoonup \Sigma$
- Beispiel:

```
p = 1;
c = 1; // p1
while (c <= n) {
  p = p * c;
  c = c + 1; // p2
  }
// p3
```

$$[\![p_1]\!](\sigma) = \sigma[p \mapsto 1][c \mapsto 1]$$
$$[\![p_2]\!](\sigma) = \sigma[p \mapsto \sigma(p) * \sigma(c)][c \mapsto \sigma(c) + 1]$$
$$[\![p_3]\!](\sigma) = ???$$

$$\Gamma([\![c \leq n]\!])([\![p_2]\!])(\varphi)(\sigma) = \begin{cases} \sigma & \text{if } [\![c \leq n]\!](\sigma) = 0 \\ (\varphi \circ [\![p_2]\!])(\sigma) & \text{if } [\![c \leq n]\!](\sigma) = 1 \end{cases}$$

Denotationale Semantik

- Kernkonzept: Abbildung von Programmen auf mathematisches Gegenstück (**Denotat**)
- **Partielle** Funktionen zwischen Zuständen $[\![c]\!] : \Sigma \rightharpoonup \Sigma$
- Beispiel:

```
p = 1;
c = 1; // p1
while (c <= n) {
  p = p * c;
  c = c + 1; // p2
  }
// p3
```

$$[\![p_1]\!](\sigma) = \sigma[p \mapsto 1][c \mapsto 1]$$
$$[\![p_2]\!](\sigma) = \sigma[p \mapsto \sigma(p) * \sigma(c)][c \mapsto \sigma(c) + 1]$$
$$[\![p_3]\!](\sigma) = \mathit{fix}(\Gamma([\![c \leq n]\!])[\![p_2]\!]))([\![p_1]\!](\sigma))$$

$$\Gamma([\![c \leq n]\!])([\![p_2]\!])(\varphi)(\sigma) = \begin{cases} \sigma & \text{if } [\![c \leq n]\!](\sigma) = 0 \\ (\varphi \circ [\![p_2]\!])(\sigma) & \text{if } [\![c \leq n]\!](\sigma) = 1 \end{cases}$$

$$\Gamma(\beta)(\rho)(\varphi)(\sigma) = \begin{cases} \sigma & \text{if } \beta(\sigma) = 0 \\ (\varphi \circ \rho)(\sigma) & \text{if } \beta(\sigma) = 1 \end{cases}$$

Denotationale Semantik

- Kernkonzept: Abbildung von Programmen auf mathematisches Gegenstück (**Denotat**)
- **Partielle** Funktionen zwischen Zuständen $[\![c]\!] : \Sigma \rightharpoonup \Sigma$
- Beispiel:

```
p = 1;
c = 1; // p1
while (c <= n) {
  p = p * c;
  c = c + 1; // p2
  }
// p3
```

$$[\![p_1]\!](\sigma) = \sigma[p \mapsto 1][c \mapsto 1]$$
$$[\![p_2]\!](\sigma) = \sigma[p \mapsto \sigma(p) * \sigma(c)][c \mapsto \sigma(c) + 1]$$
$$[\![p_3]\!] = \mathit{fix}(\Gamma([\![c \leq n]\!])([\![p_2]\!])) \circ [\![p_1]\!]$$

$$\Gamma([\![c \leq n]\!])([\![p_2]\!])(\varphi)(\sigma) = \begin{cases} \sigma & \text{if } [\![c \leq n]\!](\sigma) = 0 \\ (\varphi \circ [\![p_2]\!])(\sigma) & \text{if } [\![c \leq n]\!](\sigma) = 1 \end{cases}$$

$$\Gamma(\beta)(\rho)(\varphi)(\sigma) = \begin{cases} \sigma & \text{if } \beta(\sigma) = 0 \\ (\varphi \circ \rho)(\sigma) & \text{if } \beta(\sigma) = 1 \end{cases}$$

Axiomatische Semantik

- Kernkonzept: Charakterisierung von Programmen durch **Zusicherungen**
- Zusicherungen sind zustandsabhängige Prädikate (Funktionen $\Sigma \to \mathbb{B}$)
- Beispiel (mit $n = 3$)

```
// (1)
p = 1;  // (2)
c = 1;  // (3)
while(c <= n){
   // (4)
   p = p * c;
   c = c + 1;
   // (5)
   }
// (6)
```

(1) $n = 3$

(2) $p = 1 \wedge n = 3$

(3) $p = 1 \wedge c = 1 \wedge n = 3$

(4) ???

(5)

(6) $p = 6 \wedge c = 4 \wedge n = 3$

Axiomatische Semantik

- Kernkonzept: Charakterisierung von Programmen durch **Zusicherungen**
- Zusicherungen sind zustandsabhängige Prädikate (Funktionen $\Sigma \to \mathbb{B}$)
- Beispiel (mit $n = 3$)

```
// (1)
p = 1;  // (2)
c = 1;  // (3)
while (c <= n){
  // (4)
  p = p * c;
  c = c + 1;
  // (5)
  }
// (6)
```

$$
\begin{array}{ll}
(1) & n = 3 \\
(2) & p = 1 \wedge n = 3 \\
(3) & p = 1 \wedge c = 1 \wedge n = 3 \\
\\
(4) & (p = 1 \wedge c = 1 \vee p = 1 \wedge c = 2 \vee p = 2 \wedge c = 3) \\
 & \quad \wedge\, n = 3 \\
(5) & (p = 1 \wedge c = 2 \vee p = 2 \wedge c = 3 \vee p = 6 \wedge c = 4) \\
 & \quad \wedge\, n = 3 \\
(6) & p = 6 \wedge c = 4 \wedge n = 3
\end{array}
$$

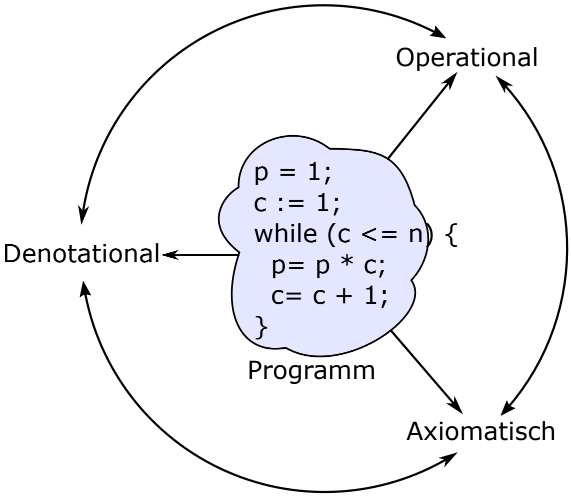
Axiomatische Semantik

- Kernkonzept: Charakterisierung von Programmen durch **Zusicherungen**
- Zusicherungen sind zustandsabhängige Prädikate (Funktionen $\Sigma \to \mathbb{B}$)
- Beispiel (mit $n = 3$)

```
// (1)
p = 1;  // (2)
c = 1;  // (3)
while (c <= n) {
  // (4)
  p = p * c;
  c = c + 1;
  // (5)
  }
// (6)
```

$$
\begin{array}{ll}
(1) & n = 3 \\
(2) & p = 1 \land n = 3 \\
(3) & p = 1 \land c = 1 \land n = 3 \\
\\
(4) & p = (c-1)! \land c \leq n \land n = 3 \\
\\
(5) & p = (c-1)! \land n = 3 \\
\\
(6) & p = 6 \land c = 4 \land n = 3
\end{array}
$$

Drei Semantiken — Eine Sicht



IV. Mengen, Relationen, Regeln

Induktive Definitionen mit Regeln

- Wir nutzen **Regeln**, um induktiv definierte Mengen zu definieren.
  - Konkret: Relationen wie **Zustandsübergänge**.
- Regeln bestehen aus Voraussetzungen $R_1, \ldots, R_n$ und einer Konklusion S:

$$\frac{R_1 \qquad \ldots \qquad R_n}{S}$$

  - R_i und S sind beliebige Relationen.
  - Idee: (Genau dann) wenn $R_1, \ldots, R_n$ wahr sind, dann auch S.

Beispiel Fakultät

- fact(n) ist 1, wenn $n \leq 0$
- fact(n) ist $n \cdot \text{fact}(n-1)$, wenn $n > 0$
- Formalisierung als **Relation**

$$\text{fact}(n, m) \text{ mit } n, m \in N$$

- Können wir auch als rekursive Funktion auffassen, wenn rechtseindeutig

Beispiel Fakultät

- $\mathrm{fact}(n)$ ist 1, wenn $n \leq 0$
- $\mathrm{fact}(n)$ ist $n \cdot \mathrm{fact}(n-1)$, wenn $n > 0$
- Formalisierung als **Relation**

$$\mathrm{fact}(n, m) \text{ mit } n, m \in N$$

$$\frac{n \leq 0}{\mathrm{fact}(n, 1)}$$

- Können wir auch als rekursive Funktion auffassen, wenn rechtseindeutig

Beispiel Fakultät

- $\mathsf{fact}(n)$ ist 1, wenn $n \leq 0$
- $\mathsf{fact}(n)$ ist $n \cdot \mathsf{fact}(n-1)$, wenn $n > 0$
- Formalisierung als **Relation**

$$\mathsf{fact}(n, m) \text{ mit } n, m \in N$$

- Können wir auch als rekursive Funktion auffassen, wenn rechtseindeutig

$$\frac{n \leq 0}{\mathsf{fact}(n, 1)}$$

$$\frac{n > 0 \qquad \mathsf{fact}(n-1, m)}{\mathsf{fact}(n, n \cdot m)}$$

Beispiel Fakultät

- $\mathrm{fact}(n)$ ist 1, wenn $n \leq 0$
- $\mathrm{fact}(n)$ ist $n \cdot \mathrm{fact}(n-1)$, wenn $n > 0$
- Formalisierung als **Relation**

$$\mathrm{fact}(n, m) \text{ mit } n, m \in N$$

$$\frac{n \leq 0}{\mathrm{fact}(n, 1)}$$

$$\frac{n > 0 \qquad \mathrm{fact}(n-1, m)}{\mathrm{fact}(n, n \cdot m)}$$

- Können wir auch als rekursive Funktion auffassen, wenn rechtseindeutig
- Berechnung von $\mathrm{fact}(4, ?)$

Arbeitsblatt 1.5: Beispiel GGT

- Der ggT von n, m ist m wenn $n = 0$, oder n wenn $m = 0$.
- Ansonsten ist der ggT von n, m der ggT des kleineren von n und m und der Differenz der beiden.

1. Beschreibt den Algorithmus in Regelschreibweise
2. Berechnet damit $\mathrm{ggT}(24, 18, ?)$

Arbeitsblatt 1.5: Beispiel GGT

- Der ggT von n, m ist m wenn $n = 0$, oder n wenn $m = 0$.
- Ansonsten ist der ggT von n, m der ggT des kleineren von n und m und der Differenz der beiden.

1. Beschreibt den Algorithmus in Regelschreibweise
2. Berechnet damit $\text{ggT}(24, 18, ?)$

Formalisierung: $\text{ggT}(n, m, p)$ mit $n, m, p \in \mathbb{N}$

$$\frac{}{\text{ggT}(n, 0, n)}$$

Arbeitsblatt 1.5: Beispiel GGT

- Der ggT von n, m ist m wenn $n = 0$, oder n wenn $m = 0$.
- Ansonsten ist der ggT von n, m der ggT des kleineren von n und m und der Differenz der beiden.

1. Beschreibt den Algorithmus in Regelschreibweise
2. Berechnet damit $\text{ggT}(24, 18, ?)$

Formalisierung: $\text{ggT}(n, m, p)$ mit $n, m, p \in \mathbb{N}$

$$\frac{}{\text{ggT}(n, 0, n)}$$

$$\frac{}{\text{ggT}(0, m, m)}$$

Arbeitsblatt 1.5: Beispiel GGT

- Der ggT von n, m ist m wenn $n = 0$, oder n wenn $m = 0$.
- Ansonsten ist der ggT von n, m der ggT des kleineren von n und m und der Differenz der beiden.

1. Beschreibt den Algorithmus in Regelschreibweise
2. Berechnet damit $\mathrm{ggT}(24, 18, ?)$

Formalisierung: $\mathrm{ggT}(n, m, p)$ mit $n, m, p \in \mathbb{N}$

$$\frac{}{\mathrm{ggT}(n, 0, n)}$$

$$\frac{}{\mathrm{ggT}(0, m, m)}$$

$$\frac{n \leq m \qquad \mathrm{ggT}(m - n, n, p)}{\mathrm{ggT}(n, m, p)}$$

Arbeitsblatt 1.5: Beispiel GGT

- Der ggT von n, m ist m wenn $n = 0$, oder n wenn $m = 0$.
- Ansonsten ist der ggT von n, m der ggT des kleineren von n und m und der Differenz der beiden.

1. Beschreibt den Algorithmus in Regelschreibweise
2. Berechnet damit $\mathrm{ggT}(24, 18, ?)$

Formalisierung: $\mathrm{ggT}(n, m, p)$ mit $n, m, p \in \mathbb{N}$

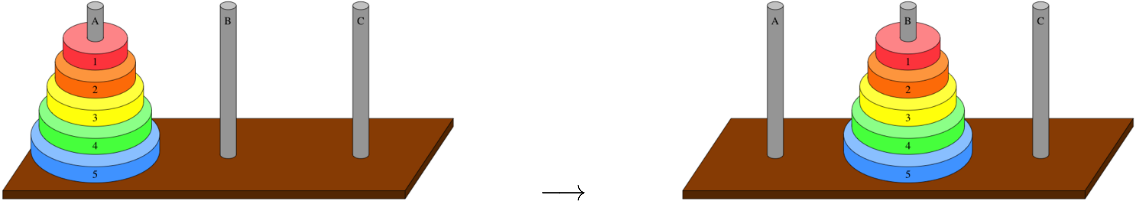
$$\frac{}{\mathrm{ggT}(n, 0, n)}$$

$$\frac{}{\mathrm{ggT}(0, m, m)}$$

$$\frac{n \leq m \qquad \mathrm{ggT}(m - n, n, p)}{\mathrm{ggT}(n, m, p)}$$

$$\frac{m < n \qquad \mathrm{ggT}(n - m, m, p)}{\mathrm{ggT}(n, m, p)}$$

Türme von Hanoi

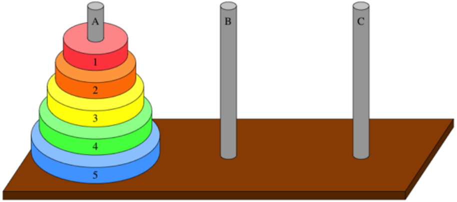


Quelle: https://www.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi

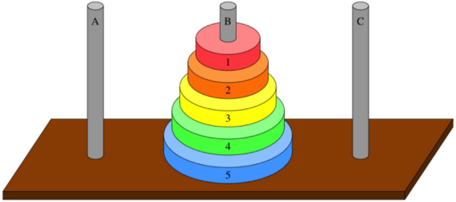
- Umstapeln zwischen den Stäben
- Jede Scheibe darf entweder auf einen leeren Stab oder eine größere Scheibe gelegt werden
  - Dies kann repräsentiert werden bei 9 Scheiben, dass
    1. "ein Stab ist leer" mit der Sequenz $\langle\,\rangle$ der Länge 0
    2. "Ein Stab enthält Scheiben $n_1, \ldots, n_k$" durch die Sequenz $\langle n_1, \ldots, n_k \rangle$ der Länge k, wobei gelten muss $n_i < n_{i+1}$.

    Damit lassen sich Spielzustände repräsentieren als $\langle f_A, f_B, f_C \rangle$ wobei f_A, f_B, f_C die Sequenzen der Nummern der Scheiben auf den entsprechenden Stäben.

Arbeitsblatt 1.6: Türme von Hanoi



$\longrightarrow$



- Seien die Züge beschrieben durch $\rightarrow_{AB}$ als Zug der obersten Scheibe auf A auf den Stapel B. Entsprechend $\rightarrow_{BA}, \rightarrow_{AC}, \rightarrow_{CA}, \rightarrow_{BC}$, und $\rightarrow_{CB}$.
- Beschreibt mittels Regeln die zulässigen Bewegungen zwischen den Stäben:

$$\frac{?}{\langle f_A, f_B, f_C\rangle \rightarrow_{AB} ?} \qquad \frac{?}{\langle f_A, f_B, f_C\rangle \rightarrow_{BA} ?} \qquad \frac{?}{\langle f_A, f_B, f_C\rangle \rightarrow_{AC} ?}$$

$$\frac{?}{\langle f_A, f_B, f_C\rangle \rightarrow_{CA} ?} \qquad \frac{?}{\langle f_A, f_B, f_C\rangle \rightarrow_{BC} ?} \qquad \frac{?}{\langle f_A, f_B, f_C\rangle \rightarrow_{CA} ?}$$

Zusammenfassung

- Wir wollen die **Bedeutung** (Semantik) von Programmen beschreiben, um ihre Korrektheit beweisen zu können.
- Dazu gibt es verschiedene Ansätze, die wir betrachten werden.
- Nächste Woche geht es mit dem ersten los: **operationale** Semantik

Korrekte Software: Grundlagen und Methoden
Vorlesung 2 vom 10.04.24
Operationale Semantik

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2024

Fahrplan

- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten im Floyd-Hoare-Kalkül
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare
- Funktionen und Prozeduren I
- Funktionen und Prozeduren II
- 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

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

NB: Nicht die **konkrete** Syntax.

Was braucht die Semantik?

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

- Ein Programm besteht aus **Anweisungen** und **Ausdrücken**.
- Ausdrücke werden **zustandsabhängig** ausgewertet.
- Anweisungen **überführen** Zustände.

Was braucht die Semantik?

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

- Ein Programm besteht aus **Anweisungen** und **Ausdrücken**.
- Ausdrücke werden **zustandsabhängig** ausgewertet.
- Anweisungen **überführen** Zustände.

Woraus besteht die Semantik?

Was braucht die Semantik?

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

- Ein Programm besteht aus **Anweisungen** und **Ausdrücken**.
- Ausdrücke werden **zustandsabhängig** ausgewertet.
- Anweisungen **überführen** Zustände.

Woraus besteht die Semantik?

1. Mathematische Modellierung des **Zustands**

Was braucht die Semantik?

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

- Ein Programm besteht aus **Anweisungen** und **Ausdrücken**.
- Ausdrücke werden **zustandsabhängig** ausgewertet.
- Anweisungen **überführen** Zustände.

Woraus besteht die Semantik?

1. Mathematische Modellierung des **Zustands**
2. Auswertung von (arithmetischen und boolschen) Ausdrücken

Was braucht die Semantik?

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

- Ein Programm besteht aus **Anweisungen** und **Ausdrücken**.
- Ausdrücke werden **zustandsabhängig** ausgewertet.
- Anweisungen **überführen** Zustände.

Woraus besteht die Semantik?

1. Mathematische Modellierung des **Zustands**
2. Auswertung von (arithmetischen und boolschen) Ausdrücken
3. Auswertung von Anweisungen: Zustandsübergänge

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 $\mathbf{Loc}$ 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 I

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:

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

- Updates sind “linksassoziativ”:

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

Arbeitsblatt 2.1: Zustände!

- 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) $\langle x \mapsto 1, a \mapsto 3 \rangle [x \mapsto 3][y \mapsto 1][x \mapsto 4] = ??$

Operationale Semantik: Arithmetische Ausdrücke

Ein arithmetischer Ausdruck a wertet unter Zustand σ zu einer ganzen Zahl n (Wert) aus.

$$\textbf{Aexp}\ a ::= \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1\ /\ a_2 \qquad \langle a, \sigma \rangle \to_{Aexp} n$$

Operationale Semantik: Arithmetische Ausdrücke

Ein arithmetischer Ausdruck a wertet unter Zustand σ zu einer ganzen Zahl n (Wert) aus.

$$\mathbf{Aexp}\ a ::= \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1\ /\ a_2 \qquad \langle a, \sigma \rangle \rightarrow_{Aexp} n$$

Regeln:

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

Operationale Semantik: Arithmetische Ausdrücke

Ein arithmetischer Ausdruck a wertet unter Zustand σ zu einer ganzen Zahl n (Wert) aus.

$$\mathbf{Aexp}\ a ::= \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1\ /\ a_2 \qquad \langle a, \sigma \rangle \rightarrow_{Aexp} n$$

Regeln:

$$\frac{n \in \mathbf{Z}}{\langle n, \sigma \rangle \rightarrow_{Aexp} [\![n]\!]} \qquad \frac{x \in \mathbf{Idt}, x \in Dom(\sigma), \sigma(x) = v}{\langle x, \sigma \rangle \rightarrow_{Aexp} v}$$

Operationale Semantik: Arithmetische Ausdrücke

$$\textbf{Aexp}\ a ::= \mathbf{Z} \mid \mathbf{Idt} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 \ / \ a_2 \qquad \langle a, \sigma \rangle \rightarrow_{Aexp} n$$

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

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

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

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

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen
- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von x+3 mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

$$\frac{\langle x, \sigma \rangle \rightarrow_{Aexp} \textcolor{red}{?} \qquad \langle 3, \sigma \rangle \rightarrow_{Aexp} \textcolor{blue}{?}}{\langle x + 3, \sigma \rangle \rightarrow_{Aexp} \textcolor{red}{?} + \textcolor{blue}{?}}$$

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen
- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von x+3 mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

$$\frac{}{\langle 3, \sigma \rangle \rightarrow_{Aexp} [\![3]\!]}$$

$$\frac{\langle x, \sigma \rangle \rightarrow_{Aexp} ? \qquad \langle 3, \sigma \rangle \rightarrow_{Aexp} ?}{\langle x + 3, \sigma \rangle \rightarrow_{Aexp} ? + ?}$$

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen
- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von `x+3` mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

$$\frac{}{\langle 3, \sigma \rangle \rightarrow_{Aexp} 3}$$

$$\frac{\langle x, \sigma \rangle \rightarrow_{Aexp} ? \qquad \langle 3, \sigma \rangle \rightarrow_{Aexp} ?}{\langle x + 3, \sigma \rangle \rightarrow_{Aexp} ? + ?}$$

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen
- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von x+3 mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

$$\frac{x \in \mathrm{dom}(\sigma), \sigma(x) = ?}{\langle x, \sigma \rangle \to_{Aexp} ?} \qquad \frac{}{\langle 3, \sigma \rangle \to_{Aexp} 3}$$

$$\frac{\langle x, \sigma \rangle \to_{Aexp} ? \qquad \langle 3, \sigma \rangle \to_{Aexp} ?}{\langle x + 3, \sigma \rangle \to_{Aexp} ? + ?}$$

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen
- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von x+3 mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

$$\frac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \frac{}{\langle 3, \sigma \rangle \rightarrow_{Aexp} 3}$$

$$\frac{\langle x, \sigma \rangle \rightarrow_{Aexp} ? \qquad \langle 3, \sigma \rangle \rightarrow_{Aexp} ?}{\langle x + 3, \sigma \rangle \rightarrow_{Aexp} ? + ?}$$

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen
- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von `x+3` mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5\rangle$.

$$\frac{}{\langle x, \sigma\rangle \rightarrow_{Aexp} 6} \qquad \frac{}{\langle 3, \sigma\rangle \rightarrow_{Aexp} 3}$$

$$\frac{\langle x, \sigma\rangle \rightarrow_{Aexp} ? \qquad \langle 3, \sigma\rangle \rightarrow_{Aexp} ?}{\langle x + 3, \sigma\rangle \rightarrow_{Aexp} ? + ?}$$

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen

- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von x+3 mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

$$\frac{}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \frac{}{\langle 3, \sigma \rangle \rightarrow_{Aexp} 3}$$

$$\frac{\langle x, \sigma \rangle \rightarrow_{Aexp} 6 \qquad \langle 3, \sigma \rangle \rightarrow_{Aexp} 3}{\langle x + 3, \sigma \rangle \rightarrow_{Aexp} 6 + 3}$$

Ableitungen

- Regeln werden von **unten** nach **oben** gelesen
- Regeln werden **komponiert** — es entsteht ein **Ableitungsbaum**

Beispiel: Auswertung von x+3 mit $\sigma \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 5 \rangle$.

$$\dfrac{\dfrac{}{\langle x, \sigma\rangle \rightarrow_{Aexp} 6} \qquad \dfrac{}{\langle 3, \sigma\rangle \rightarrow_{Aexp} 3}}{}$$

$$\frac{\langle x, \sigma\rangle \rightarrow_{Aexp} 6 \qquad \langle 3, \sigma\rangle \rightarrow_{Aexp} 3}{\langle x + 3, \sigma\rangle \rightarrow_{Aexp} 9}$$

Längere Beispiel-Ableitungen

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

$$\frac{}{\langle (x + y) * (x - y), \sigma \rangle \rightarrow_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$\frac{\dfrac{}{\langle x + y, \sigma \rangle \rightarrow_{Aexp}} \qquad \dfrac{}{\langle x - y, \sigma \rangle \rightarrow_{Aexp}}}{\langle (x + y) * (x - y), \sigma \rangle \rightarrow_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$\dfrac{\dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \to_{Aexp} 6}}{\langle x + y, \sigma \rangle \to_{Aexp}} \qquad \dfrac{}{\langle x - y, \sigma \rangle \to_{Aexp}}}{\langle (x + y) * (x - y), \sigma \rangle \to_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$
\dfrac{
  \dfrac{
    \dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \to_{Aexp} 6}
    \qquad
    \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \to_{Aexp} 5}
  }{\langle x + y, \sigma \rangle \to_{Aexp}}
  \qquad
  \dfrac{}{\langle x - y, \sigma \rangle \to_{Aexp}}
}{\langle (x + y) * (x - y), \sigma \rangle \to_{Aexp}}
$$

Längere Beispiel-Ableitungen

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

$$\dfrac{\dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \to_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \to_{Aexp} 5}}{\langle x + y, \sigma \rangle \to_{Aexp} 11} \qquad \dfrac{}{\langle x - y, \sigma \rangle \to_{Aexp}}}{\langle (x + y) * (x - y), \sigma \rangle \to_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$\dfrac{\dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}}{\langle x + y, \sigma \rangle \rightarrow_{Aexp} 11} \qquad \dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}}{\langle x - y, \sigma \rangle \rightarrow_{Aexp}}}{\langle (x + y) * (x - y), \sigma \rangle \rightarrow_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$\dfrac{\dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}}{\langle x + y, \sigma \rangle \rightarrow_{Aexp} 11} \qquad \dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}}{\langle x - y, \sigma \rangle \rightarrow_{Aexp} 1}}{\langle (x + y) * (x - y), \sigma \rangle \rightarrow_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$
\dfrac{
\dfrac{
\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6}
\quad
\dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}
}{\langle x + y, \sigma \rangle \rightarrow_{Aexp} 11}
\quad
\dfrac{
\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6}
\quad
\dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}
}{\langle x - y, \sigma \rangle \rightarrow_{Aexp} 1}
}{\langle (x + y) * (x - y), \sigma \rangle \rightarrow_{Aexp} 11}
$$

Längere Beispiel-Ableitungen

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

$$\dfrac{\dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}}{\langle x + y, \sigma \rangle \rightarrow_{Aexp} 11} \qquad \dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \rightarrow_{Aexp} 5}}{\langle x - y, \sigma \rangle \rightarrow_{Aexp} 1}}{\langle (x + y) * (x - y), \sigma \rangle \rightarrow_{Aexp} 11}$$

$$\dfrac{}{\langle (x * x) - (y * y), \sigma \rangle \rightarrow_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$\dfrac{\dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma\rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma\rangle \rightarrow_{Aexp} 5}}{\langle x + y, \sigma\rangle \rightarrow_{Aexp} 11} \qquad \dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma\rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma\rangle \rightarrow_{Aexp} 5}}{\langle x - y, \sigma\rangle \rightarrow_{Aexp} 1}}{\langle (x + y) * (x - y), \sigma\rangle \rightarrow_{Aexp} 11}$$

$$\dfrac{\dfrac{\langle x, \sigma\rangle \rightarrow_{Aexp} 6 \qquad \langle x, \sigma\rangle \rightarrow_{Aexp} 6}{\langle x * x, \sigma\rangle \rightarrow_{Aexp} 36}}{\langle (x * x) - (y * y), \sigma\rangle \rightarrow_{Aexp}}$$

Längere Beispiel-Ableitungen

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

$$
\dfrac{
\dfrac{
\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \to_{Aexp} 6}
\qquad
\dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \to_{Aexp} 5}
}{\langle x + y, \sigma \rangle \to_{Aexp} 11}
\qquad
\dfrac{
\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma \rangle \to_{Aexp} 6}
\qquad
\dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma \rangle \to_{Aexp} 5}
}{\langle x - y, \sigma \rangle \to_{Aexp} 1}
}{\langle (x + y) * (x - y), \sigma \rangle \to_{Aexp} 11}
$$

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

Längere Beispiel-Ableitungen

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

$$\dfrac{\dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma\rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma\rangle \rightarrow_{Aexp} 5}}{\langle x + y, \sigma\rangle \rightarrow_{Aexp} 11} \qquad \dfrac{\dfrac{x \in \mathrm{dom}(\sigma), \sigma(x) = 6}{\langle x, \sigma\rangle \rightarrow_{Aexp} 6} \qquad \dfrac{y \in \mathrm{dom}(\sigma), \sigma(y) = 5}{\langle y, \sigma\rangle \rightarrow_{Aexp} 5}}{\langle x - y, \sigma\rangle \rightarrow_{Aexp} 1}}{\langle (x + y) * (x - y), \sigma\rangle \rightarrow_{Aexp} 11}$$

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

Arbeitsblatt 2.2: Auswertung

Konstruiert wie oben die Ableitung für den Ausdruck (3*a)/b mit $\sigma \stackrel{def}{=} \langle a \mapsto 8, b \mapsto 7\rangle$.

Hinweis: wahrscheinlich einfacher auf Papier...

Eigenschaften der Semantik

- **Frage:** Gegeben einen Ausdruck a, leitet **jeder** Zustand σ zu einem Wert n ab?

Eigenschaften der Semantik

- **Frage:** Gegeben einen Ausdruck a, leitet **jeder** Zustand σ zu einem Wert n ab?
- **Antwort:** Nein.
- Betrachte folgende Beispiele für $a \stackrel{def}{=}$y+3/x

$$\langle a, \langle y \mapsto 5 \rangle\rangle \rightarrow_{Aexp} ??? \tag{1}$$

$$\langle a, \langle y \mapsto 5, x \mapsto 0 \rangle\rangle \rightarrow_{Aexp} ??? \tag{2}$$

- In diesen Beispielen läßt sich kein **vollständiger** Ableitungsbaum konstruieren.
- Die Auswertung ist **undefiniert** — die Semantik ist **partiell**.

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 \; < a_2 \mid ! b \mid b_1 \;\&\&\; b2 \mid b_1 \;\; || \;\; b_2$

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

Regeln:

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

$$\frac{\langle a_1, \sigma \rangle \rightarrow_{Aexp} n_1 \qquad \langle a_2, \sigma \rangle \rightarrow_{Aexp} n_2 \qquad n_1 \text{ und } n_2 \text{ gleich}}{\langle a_1 == a_2, \sigma \rangle \rightarrow_{Bexp} true}$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow_{Aexp} n_1 \qquad \langle a_2, \sigma \rangle \rightarrow_{Aexp} n_2 \qquad n_1 \text{ und } n_2 \text{ ungleich}}{\langle a_1 == a_2, \sigma \rangle \rightarrow_{Bexp} false}$$

Operationale Semantik: Boolesche Ausdrücke

- **Bexp** $b ::= \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 \; < a_2 \mid !b \mid b_1 \; \&\& \; b2 \mid b_1 \; || \; b_2$

$$\langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{true} \mid \mathit{false}$$

Regeln:

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

$$\frac{\langle b_1, \sigma \rangle \rightarrow_{Bexp} \mathit{false}}{\langle b_1 \; \&\& \; b_2, \sigma \rangle \rightarrow_{Bexp} \mathit{false}} \qquad \frac{\langle b_1, \sigma \rangle \rightarrow_{Bexp} \mathit{true} \qquad \langle b_2, \sigma \rangle \rightarrow_{Bexp} t}{\langle b_1 \; \&\& \; b_2, \sigma \rangle \rightarrow_{Bexp} t}$$

$$\frac{\langle b_1, \sigma \rangle \rightarrow_{Bexp} \mathit{true}}{\langle b_1 \; || \; b_2, \sigma \rangle \rightarrow_{Bexp} \mathit{true}} \qquad \frac{\langle b_1, \sigma \rangle \rightarrow_{Bexp} \mathit{false} \qquad \langle b_2, \sigma \rangle \rightarrow_{Bexp} t}{\langle b_1 \; || \; b_2, \sigma \rangle \rightarrow_{Bexp} t}$$

Arbeitsblatt 2.3: Boolsche Ausdrücke

Konstruiert die Auswertung des Ausdrucks $b =$ `x == 7 && y == 3` unter folgenden Zuständen:

1. $\sigma_1 \stackrel{def}{=} \langle x \mapsto 7, y \mapsto 3 \rangle$
2. $\sigma_2 \stackrel{def}{=} \langle x \mapsto 6, y \mapsto 3 \rangle$
3. $\sigma_3 \stackrel{def}{=} \langle y \mapsto 6 \rangle$
4. $\sigma_4 \stackrel{def}{=} \langle x \mapsto 7 \rangle$
5. $\sigma_5 \stackrel{def}{=} \langle x \mapsto 2 \rangle$

Striktheit

- Eine partielle Funktion f ist **strikt** wenn $f(x)$ undefiniert ist, sobald x undefiniert ist.
- In unserer Semantik sind alle Operatoren (arithmetisch und boolesch) strikt, **bis auf** && und || im **ersten** Argument.
  - Operational nennt man das auch abgekürzte Auswertung (*short-circuit evaluation*)
  - Das erlaubt Idiome wie **if** (x != 0 && 3/x > 1) { ... }
- Wie erkennt man Striktheit an den **Regeln**?

Striktheit

- Eine partielle Funktion f ist **strikt** wenn $f(x)$ undefiniert ist, sobald x undefiniert ist.
- In unserer Semantik sind alle Operatoren (arithmetisch und boolesch) strikt, **bis auf** && und || im **ersten** Argument.
  - Operational nennt man das auch abgekürzte Auswertung (*short-circuit evaluation*)
  - Das erlaubt Idiome wie **if** (x != 0 && 3/x > 1) { ... }
- Wie erkennt man Striktheit an den **Regeln**?
  Alle Variablen der Konklusion kommen in den Bedingungen vor.

Operationale Semantik: Anweisungen

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

Beispiel:

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$$

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

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

Operationale Semantik: Anweisungen

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

Beispiel:

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$$

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

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 | $c_1; c_2$ | { }

Beispiel:

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$$

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

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** $\mid$ **if** $(b)\ c_1$ **else** $c_2 \mid$ **while** $(b)\ c \mid c_1; c_2 \mid \{\,\}$

Regeln:

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

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

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

Operationale Semantik: Anweisungen

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

Regeln:

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

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

Beispiel

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

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

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

(A)

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

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

(B)

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

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

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

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

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

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

Lineare, abgekürzte Schreibweise

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

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⟩⟩ →Stmt ⟨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 ↦ 2, x ↦ 1⟩⟩ →Stmt ⟨y ↦
1, x ↦ 1⟩
|        // ⟨y ↦ 1, x ↦ 1⟩
|        x = 2 * x;     // Ableitung für ⟨x = 2 ∗ x, ⟨y ↦ 1, x ↦ 1⟩⟩ →Stmt . . .
|        // ⟨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⟩                    σ1
x = 1;
// ⟨y ↦ 2, x ↦ 1⟩
while (y != 0) {
 y = y − 1;
 x = 2 * x;
}
// ⟨y ↦ 0, x ↦ 4⟩             σE
```

- Für **einen festen Anfangszustand** $\sigma_1 = \langle y \mapsto 2\rangle$ gilt am Ende $\sigma_E(x) = 4 = 2^2 = 2^{\sigma_1(y)}$.

Was haben wir gezeigt?

```
// ⟨y ↦ 2⟩                σ1
x = 1;
// ⟨y ↦ 2, x ↦ 1⟩
while (y != 0) {
  y = y - 1;
  x = 2 * x;
}
// ⟨y ↦ 0, x ↦ 4⟩         σE
```

- Für **einen festen Anfangszustand** $\sigma_1 = \langle y \mapsto 2\rangle$ gilt am Ende $\sigma_E(x) = 4 = 2^2 = 2^{\sigma_1(y)}$.
- Gilt das für alle?

Was haben wir gezeigt?

```
// ⟨y ↦ 2⟩                          σ1
x = 1;
// ⟨y ↦ 2, x ↦ 1⟩
while (y != 0) {
  y = y - 1;
  x = 2 * x;
}
// ⟨y ↦ 0, x ↦ 4⟩                    σE
```

- Für **einen festen Anfangszustand** $\sigma_1 = \langle y \mapsto 2\rangle$ gilt am Ende $\sigma_E(x) = 4 = 2^2 = 2^{\sigma_1(y)}$.
- Gilt das für alle?
- Für welche nicht?

Was haben wir gezeigt?

```
// ⟨y ↦ 2⟩                σ1
x = 1;
// ⟨y ↦ 2, x ↦ 1⟩
while (y != 0) {
 y = y - 1;
 x = 2 * x;
}
// ⟨y ↦ 0, x ↦ 4⟩          σE
```

- Für **einen festen Anfangszustand** $\sigma_1 = \langle y \mapsto 2\rangle$ gilt am Ende $\sigma_E(x) = 4 = 2^2 = 2^{\sigma_1(y)}$.
- 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;
}
```

Was passiert hier?

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

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

Arbeitsblatt 2.4: Programme!

- 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 Programms zwischen den Werten von x und y im Endzustand und im Anfangszustand?

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

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**: sowohl w also auch **if** (b) $\{c; w\}$ **else** $\{\,\}$ werten zum gleichen Programmzustand aus (wenn sie auswerten).
- Der Beweis geht per Fallunterscheidung über die Auswertung von Teilausdrücken bzw. Teilprogrammen.

Beweis

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

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

Beweis

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

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

2. $\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{true}$: Sei $\langle c, \sigma\rangle \rightarrow_{Stmt} \sigma'$, dann:

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

Beweis

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

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

2. $\langle b, \sigma\rangle \rightarrow_{Bexp}$ *true*: Sei $\langle c, \sigma\rangle \rightarrow_{Stmt} \sigma'$, dann:

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

3. $\langle b, \sigma\rangle$ wertet gar nicht aus — dann werten weder w noch **if** $(b)\ \{c; w\}$ **else** $\{\,\}$ aus.

Zusammenfassung

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

Korrekte Software: Grundlagen und Methoden
Vorlesung 3 vom 17.04.24
Denotationale Semantik

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2024

Fahrplan

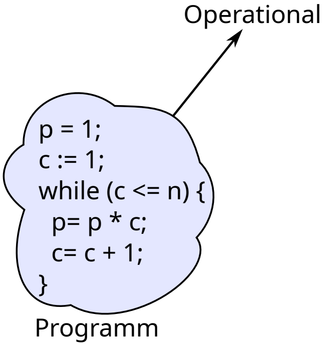
- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten im Floyd-Hoare-Kalkül
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare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

Überblick

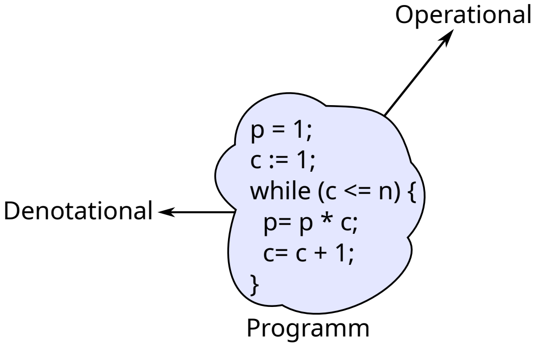
```
p = 1;
c := 1;
while (c <= n) {
  p= p * c;
  c= c + 1;
}
```

Programm

Überblick



Überblick



- Denotationale Semantik für C0
- Fixpunkte

Denotationale Semantik — Motivation

- **Operationale Semantik:**

  Eine Menge von Regeln, die einen Zustand und ein Programm in einen neuen Zustand überführen:

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$$

- **Denotationale Semantik:**

  Eine Menge von Regeln, die ein Programm in eine $\underbrace{\textbf{partielle Funktion}}_{\text{Denotat}}$ von Zustand nach Zustand überführen

$$[\![c]\!]_{\mathcal{C}} : \Sigma \rightharpoonup \Sigma$$

Denotationale Semantik — Kompositionalität

- Semantik von zusammengesetzten Ausdrücken durch Kombination der Semantiken der Teilausdrücke
  - Bsp: Semantik einer Sequenz von Anweisungen durch Verknüpfung der Semantik der einzelnen Anweisungen
- Operationale Semantik ist **nicht** kompositional:

```
x= 3;
y= x+ 7; // (*)
z= x+ y;
```

  - Semantik von Zeile (*) ergibt sich aus der Ableitung davor
  - Kann nicht unabhängig abgeleitet werden
- Denotationale Semantik ist kompositional.
  - Wesentlicher Baustein: **partielle Funktionen**

Partielle Funktionen und ihre Graphen

- Der **Graph** einer partiellen Funktion $f : X \rightharpoonup Y$ ist eine Relation

$$grph(f) \subseteq X \times Y \stackrel{def}{=} \{(x, f(x)) \mid x \in \mathrm{dom}(f)\}$$

- Wir können eine partielle Funktion durch ihren Graph definieren:

Definition (Partielle Funktion)

Eine **partielle Funktion** $f : X \rightharpoonup Y$ ist eine Relation $f \subseteq X \times Y$ so dass wenn $(x, y_1) \in f$ und $(x, y_2) \in f$ dann $y_1 = y_2$ (**Rechtseindeutigkeit**)

- Wir benutzen beide Notationen, aber für die denotationale Semantik die Graph-Notation.
- **Systemzustände** sind partielle Abbildungen $\Sigma \stackrel{def}{=} \mathbf{Loc} \rightharpoonup \mathbf{V}$ ($\longrightarrow$ letzte VL)

Beispiel

Als Beispiel betrachten wir die partielle Funktion $\mathit{div3} : \{0 \ldots 10\} \to \mathbb{N}$

$$\mathit{div3}(x) = y \quad \text{g.d.w.} \quad 3 \cdot y = x$$

- Zuordnung:

$$\begin{array}{ccc} 0 & \mapsto & 0 \\ 1 & & \\ 2 & & \\ 3 & \mapsto & 1 \\ 4 & & \\ 5 & & \\ 6 & \mapsto & 2 \\ 7 & & \\ 8 & & \\ 9 & \mapsto & 3 \\ 10 & & \end{array}$$

- Notation als Relation (**Graph**):

$$\mathit{div3} \stackrel{\mathit{def}}{=} \{(0,0),(3,1),(6,2),(9,3)\}$$

- Wir schreiben

$$\begin{array}{ll} \mathit{div3}(3) = 1 & \text{für } (3,1) \in \mathit{div3} \\ \mathit{div3}(5) = \bot & \text{für es gibt kein } y \text{ mit } (5,y) \in \mathit{div3} \\ \mathit{div3}(5) = \bot & \text{für } \forall y.(5,y) \notin \mathit{div3} \end{array}$$

- Achtung, Partialität!

Achtung, Partialität!

- Beim Rechnen mit partiellen Funktionen muss die **Definiertheit** beachtet werden.
- Insbesondere darf nicht mit undefinierten Ausdrücken gerechnet werden.
- Bspw. gilt oben **nicht** im allgemeinen:

$$3 \cdot \mathit{div3}(x) = x \quad \times$$

  oder

$$\mathit{div3}(1) = \perp = \mathit{div3}(2) \Longrightarrow \mathit{div3}(1) = \mathit{div3}(2) \quad \times$$

- Warum? Dann gälte

Achtung, Partialität!

- Beim Rechnen mit partiellen Funktionen muss die **Definiertheit** beachtet werden.
- Insbesondere darf nicht mit undefinierten Ausdrücken gerechnet werden.
- Bspw. gilt oben **nicht** im allgemeinen:

$$3 \cdot \mathit{div3}(x) = x \quad \times$$

oder

$$\mathit{div3}(1) = \bot = \mathit{div3}(2) \Longrightarrow \mathit{div3}(1) = \mathit{div3}(2) \quad \times$$

- Warum? Dann gälte

$$\mathit{div3}(1) = \mathit{div3}(2)$$

Achtung, Partialität!

- Beim Rechnen mit partiellen Funktionen muss die **Definiertheit** beachtet werden.
- Insbesondere darf nicht mit undefinierten Ausdrücken gerechnet werden.
- Bspw. gilt oben **nicht** im allgemeinen:

$$3 \cdot \mathit{div3}(x) = x \quad \times$$

oder

$$\mathit{div3}(1) = \perp = \mathit{div3}(2) \Longrightarrow \mathit{div3}(1) = \mathit{div3}(2) \quad \times$$

- Warum? Dann gälte

$$\mathit{div3}(1) = \mathit{div3}(2)$$
$$3 \cdot \mathit{div3}(1) = 3 \cdot \mathit{div3}(2)$$

Achtung, Partialität!

- Beim Rechnen mit partiellen Funktionen muss die **Definiertheit** beachtet werden.
- Insbesondere darf nicht mit undefinierten Ausdrücken gerechnet werden.
- Bspw. gilt oben **nicht** im allgemeinen:

$$3 \cdot \mathit{div3}(x) = x \quad \times$$

oder

$$\mathit{div3}(1) = \perp = \mathit{div3}(2) \Longrightarrow \mathit{div3}(1) = \mathit{div3}(2) \quad \times$$

- Warum? Dann gälte

$$\begin{aligned} \mathit{div3}(1) &= \mathit{div3}(2) \\ 3 \cdot \mathit{div3}(1) &= 3 \cdot \mathit{div3}(2) \\ 1 &= 2 \quad \lightning \end{aligned}$$

Achtung, Partialität!

- Beim Rechnen mit partiellen Funktionen muss die **Definiertheit** beachtet werden.
- Insbesondere darf nicht mit undefinierten Ausdrücken gerechnet werden.
- Bspw. gilt oben **nicht** im allgemeinen:

$$3 \cdot \mathit{div3}(x) = x \quad \times$$

oder

$$\mathit{div3}(1) = \bot = \mathit{div3}(2) \Longrightarrow \mathit{div3}(1) = \mathit{div3}(2) \quad \times$$

- Warum? Dann gälte

$$\begin{aligned} \mathit{div3}(1) &= \mathit{div3}(2) \\ 3 \cdot \mathit{div3}(1) &= 3 \cdot \mathit{div3}(2) \\ 1 &= 2 \quad \lightning \end{aligned}$$

- Vgl. `https://de.wikipedia.org/wiki/Trugschluss_(Mathematik)#Division_durch_0`

Arbeitsblatt 3.1: Relationen als Funktionen

Definiert wie im Beispiel eben die Funktion $sqrt : \{0, \ldots, 100\} \to \mathbb{N}$ mit

$$sqrt(x) = y \quad \text{g.d.w.} \quad y^2 = x$$

Was ist der Wert folgender Ausdrücke:

$$t_1 = 5 - sqrt(32) \quad t_2 = sqrt(49) + sqrt(0) \quad t_3 = \sqrt{3} \cdot sqrt(3) \quad t_4 = \frac{sqrt(64)}{0}$$

Arbeitsblatt 3.1: Relationen als Funktionen

Definiert wie im Beispiel eben die Funktion $\mathit{sqrt} : \{0, \ldots, 100\} \to \mathbb{N}$ mit

$$\mathit{sqrt}(x) = y \quad \text{g.d.w.} \quad y^2 = x$$

Was ist der Wert folgender Ausdrücke:

$$t_1 = 5 - \mathit{sqrt}(32) \quad t_2 = \mathit{sqrt}(49) + \mathit{sqrt}(0) \quad t_3 = \sqrt{3} \cdot \mathit{sqrt}(3) \quad t_4 = \frac{\mathit{sqrt}(64)}{0}$$

Denotierende Funktionen (Denotate)

- Arithmetische Ausdrücke: $a \in \mathbf{Aexp}$ denotieren eine partielle Funktion $\Sigma \rightharpoonup \mathbb{Z}$
- Boolsche Ausdrücke: $b \in \mathbf{Bexp}$ denotieren eine partielle Funktion $\Sigma \rightharpoonup \mathbb{B}$
- Anweisungen: $c \in \mathbf{Stmt}$ denotieren eine partielle Funktion $\Sigma \rightharpoonup \Sigma$

Denotat von Aexp

$$[\![a]\!]_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$$

$$\begin{aligned}
[\![n]\!]_{\mathcal{A}} =& \{(\sigma, [\![n]\!]) \mid \sigma \in \Sigma\} \\
[\![x]\!]_{\mathcal{A}} =& \{(\sigma, \sigma(x)) \mid \sigma \in \Sigma, x \in Dom(\sigma)\} \\
[\![a_0 + a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \land (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}\} \\
[\![a_0 - a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \land (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}\} \\
[\![a_0 * a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 \cdot n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \land (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}\} \\
[\![a_0 / a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 \div n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \land (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}} \land n_1 \neq 0\}
\end{aligned}$$

Rechtseindeutigkeit

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{A}}$ *ist rechtseindeutig und damit eine* ***partielle Funktion***.

Beweis.

z.z.: wenn $(\sigma, v_1) \in [\![a]\!]_{\mathcal{A}}, (\sigma, v_2) \in [\![a]\!]_{\mathcal{A}}$ dann $v_1 = v_2$.
Strukturelle Induktion über **Aexp**:

Rechtseindeutigkeit

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{A}}$ *ist rechtseindeutig und damit eine* ***partielle Funktion***.

Beweis.

z.z.: wenn $(\sigma, v_1) \in [\![a]\!]_{\mathcal{A}}, (\sigma, v_2) \in [\![a]\!]_{\mathcal{A}}$ dann $v_1 = v_2$.
Strukturelle Induktion über **Aexp**:

- Induktionsbasis sind $n \in \mathbf{Z}$ und $x \in \mathbf{Idt}$.
  Sei $a \equiv x$, dann $v_1 = \sigma(x) = v_2$.

Rechtseindeutigkeit

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{A}}$ *ist rechtseindeutig und damit eine* ***partielle Funktion***.

Beweis.

z.z.: wenn $(\sigma, v_1) \in [\![a]\!]_{\mathcal{A}}, (\sigma, v_2) \in [\![a]\!]_{\mathcal{A}}$ dann $v_1 = v_2$.
Strukturelle Induktion über **Aexp**:

- Induktionsbasis sind $n \in \mathbf{Z}$ und $x \in \mathbf{Idt}$.
  Sei $a \equiv x$, dann $v_1 = \sigma(x) = v_2$.
- Induktionsschritt sind die anderen Klauseln.
  Sei $a \equiv a_1 + a_2$.
  Induktionsannahme ist: wenn $(\sigma, n_i) \in [\![a_i]\!]_{\mathcal{A}}, (\sigma, m_i) \in [\![a_i]\!]_{\mathcal{A}}$ dann $n_i = m_i$.
  Sei $v_1 = n_1 + n_2$ mit $(\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, (\sigma, n_2) \in [\![a_2]\!]_{\mathcal{A}}$, und $v_2 = m_1 + m_2$ mit $(\sigma, m_1) \in [\![a_1]\!]_{\mathcal{A}}, (\sigma, m_2) \in [\![a_2]\!]_{\mathcal{A}}$.

Rechtseindeutigkeit

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{A}}$ *ist rechtseindeutig und damit eine* ***partielle Funktion****.*

Beweis.

z.z.: wenn $(\sigma, v_1) \in [\![a]\!]_{\mathcal{A}}, (\sigma, v_2) \in [\![a]\!]_{\mathcal{A}}$ dann $v_1 = v_2$.
Strukturelle Induktion über **Aexp**:

- Induktionsbasis sind $n \in \mathbf{Z}$ und $x \in$ **Idt**.
  Sei $a \equiv x$, dann $v_1 = \sigma(x) = v_2$.
- Induktionsschritt sind die anderen Klauseln.
  Sei $a \equiv a_1 + a_2$.
  Induktionsannahme ist: wenn $(\sigma, n_i) \in [\![a_i]\!]_{\mathcal{A}}, (\sigma, m_i) \in [\![a_i]\!]_{\mathcal{A}}$ dann $n_i = m_i$.
  Sei $v_1 = n_1 + n_2$ mit $(\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, (\sigma, n_2) \in [\![a_2]\!]_{\mathcal{A}}$, und $v_2 = m_1 + m_2$ mit $(\sigma, m_1) \in [\![a_1]\!]_{\mathcal{A}}, (\sigma, m_2) \in [\![a_2]\!]_{\mathcal{A}}$.
  Aus der Annahme folgt $n_1 = m_1$ und $n_2 = m_2$, deshalb $v_1 = v_2$.

$\square$

Kompositionalität und Striktheit

- Die Rechtseindeutigkeit erlaubt die Notation als partielle Funktion:

$$
\begin{aligned}
[\![3 * (x+y)]\!]_{\mathcal{A}}(\sigma) &= [\![3]\!]_{\mathcal{A}}(\sigma) \cdot ([\![x]\!]_{\mathcal{A}}(\sigma) + [\![y]\!]_{\mathcal{A}}(\sigma)) \\
&= 3 \cdot ([\![x]\!]_{\mathcal{A}}(\sigma) + [\![y]\!]_{\mathcal{A}}(\sigma)) \\
&= 3 \cdot (\sigma(x) + \sigma(y))
\end{aligned}
$$

- Diese Notation versteckt die **Partialität**:

$$
[\![1 + x/0]\!]_{\mathcal{A}}(\sigma) = 1 + \sigma(x)/0 = 1 + \perp = \perp
$$

- Wenn ein Teilausdruck undefiniert ist, wird der gesamte Ausdruck undefiniert: $[\![-]\!]_{\mathcal{A}}$ ist **strikt** für alle arithmetischen Operatoren.

Arbeitsblatt 3.2: Semantik I

Hier üben wir noch einmal den Zusammenhang zwischen den beiden Notationen. Gegeben sei der Zustand $s = \langle x \mapsto 3, y \mapsto 4\rangle$ und der Ausdruck $a = 7 * x + y$.
Berechnen Sie die Semantik zum einen als Relation (füllen Sie die Fragezeichen aus):

```
(s, ?) : [[7]]
(s, ?) : [[x]]
(s, ?) : [[7*x]]
(s, ?) : [[y]]
(s, ?) : [[7*x+ y]]
```

Berechnen Sie zum anderen die Semantik in der Funktionsnotation:

```
[[7*x+y]](s) = [[7*x]](s)+[[y]](s) = ... = ?
```

Ist das Ergebnis am Ende gleich?

Lösung

Denotat von Bexp

$$[\![a]\!]_{\mathcal{B}} : \textbf{Bexp} \to (\Sigma \rightharpoonup \mathbb{B})$$

$$\begin{aligned}
[\![\mathbf{1}]\!]_{\mathcal{B}} =& \{(\sigma, \mathit{true}) \mid \sigma \in \Sigma\} \\
[\![\mathbf{0}]\!]_{\mathcal{B}} =& \{(\sigma, \mathit{false}) \mid \sigma \in \Sigma\} \\
[\![a_0 == a_1]\!]_{\mathcal{B}} =& \{(\sigma, \mathit{true}) \mid \sigma \in \Sigma, (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}}, (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, n_0 = n_1\} \\
& \cup \{(\sigma, \mathit{false}) \mid \sigma \in \Sigma, (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}}, (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, n_0 \neq n_1\} \\
[\![a_0 < a_1]\!]_{\mathcal{B}} =& \{(\sigma, \mathit{true}) \mid \sigma \in \Sigma, (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}}, (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, n_0 < n_1\} \\
& \cup \{(\sigma, \mathit{false}) \mid \sigma \in \Sigma, (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}}, (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, n_0 \geq n_1\}
\end{aligned}$$

Denotat von Bexp

$$[\![a]\!]_{\mathcal{B}} : \textbf{Bexp} \rightarrow (\Sigma \rightharpoonup \mathbb{B})$$

$$\begin{array}{rcl}
[\![!b]\!]_{\mathcal{B}} & = & \{(\sigma, \mathit{true}) \mid \sigma \in \Sigma, (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\} \\
& & \cup\{(\sigma, \mathit{false}) \mid \sigma \in \Sigma, (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}\} \\
[\![b_1 \;\&\&\; b2]\!]_{\mathcal{B}} & = & \{(\sigma, \mathit{false}) \mid \sigma \in \Sigma, (\sigma, \mathit{false}) \in [\![b_1]\!]_{\mathcal{B}}\} \\
& & \cup\{(\sigma, t_2) \mid \sigma \in \Sigma, (\sigma, \mathit{true}) \in [\![b_1]\!]_{\mathcal{B}}, (\sigma, t_2) \in [\![b_2]\!]_{\mathcal{B}}\} \\
[\![b_1 \;||\; b_2]\!]_{\mathcal{B}} & = & \{(\sigma, \mathit{true}) \mid \sigma \in \Sigma, (\sigma, \mathit{true}) \in [\![b_1]\!]_{\mathcal{B}}\} \\
& & \cup\{(\sigma, t_2) \mid \sigma \in \Sigma, (\sigma, \mathit{false}) \in [\![b_1]\!]_{\mathcal{B}}, (\sigma, t_2) \in [\![b_2]\!]_{\mathcal{B}}\}
\end{array}$$

Kompositionalität und Striktheit

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{B}}$ *ist rechtseindeutig und damit eine* ***partielle Funktion****.*

- Beweis analog zu $[\![-]\!]_{\mathcal{A}}$.
- Ist $[\![-]\!]_{\mathcal{B}}$ strikt?

Kompositionalität und Striktheit

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{B}}$ *ist rechtseindeutig und damit eine* ***partielle Funktion***.

- Beweis analog zu $[\![-]\!]_{\mathcal{A}}$.
- Ist $[\![-]\!]_{\mathcal{B}}$ strikt? Natürlich nicht:
- Sei $[\![b_1]\!]_{\mathcal{B}}(\sigma) = \mathit{false}$, dann $[\![b_1 \;\&\&\; b_2]\!]_{\mathcal{B}}(\sigma) = [\![b_1]\!]_{\mathcal{B}}(\sigma) = \mathit{false}$

Kompositionalität und Striktheit

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{B}}$ *ist rechtseindeutig und damit eine* ***partielle Funktion****.*

- Beweis analog zu $[\![-]\!]_{\mathcal{A}}$.
- Ist $[\![-]\!]_{\mathcal{B}}$ strikt? Natürlich nicht:
- Sei $[\![b_1]\!]_{\mathcal{B}}(\sigma) = \mathit{false}$, dann $[\![b_1 \;\&\&\; b_2]\!]_{\mathcal{B}}(\sigma) = [\![b_1]\!]_{\mathcal{B}}(\sigma) = \mathit{false}$
- Wir können deshalb nicht so einfach schreiben $[\![b_1 \;\&\&\; b_2]\!]_{\mathcal{B}}(\sigma) = [\![b_1]\!]_{\mathcal{B}}(\sigma) \wedge [\![b_2]\!]_{\mathcal{B}}(\sigma)$
- Die normale zweiwertige Logik behandelt Definiertheit gar nicht. Bei uns müssen die logischen Operatoren links-strikt sein:

$$\begin{array}{ccc} \bot \wedge a = \bot & \mathit{false} \wedge a = \mathit{false} & \mathit{true} \wedge a = a \\ \bot \vee a = \bot & \mathit{true} \vee a = \mathit{true} & \mathit{false} \vee a = a \end{array}$$

Arbeitsblatt 3.3: Semantik II

Wir üben noch einmal die Nichtstrikheit. Gegeben $s = \langle x \mapsto 7 \rangle$ und
$b \equiv (7 == x) \,||\, (x/0 == 1)$
Berechnenen Sie die Semantik in den Notationen von oben:

```
(s, ?) : [[ (7 == x) || (x/0 == 1) ]]
...

[[ (7 == x) || (x/0 == 1) ]](s) = ... ?
```

Hilfreiche Notation: $a \wedge b =$ `a /\ b`, $a \vee b =$ `a \/ b`

Lösung

Denotationale Semantik von Anweisungen

- Zuweisung: punktweise Änderung des Zustands σ zu $\sigma[x \mapsto n]$
- Sequenz: Komposition von Relationen

Definition (Komposition von Relationen)

Für zwei Relationen $R \subseteq X \times Y, S \subseteq Y \times Z$ ist ihre **Komposition**

$$R \circ S \stackrel{def}{=} \{(x, z) \mid \exists y \in Y. (x, y) \in R \wedge (y, z) \in S\}$$

Wenn R, S zwei partielle Funktionen sind, ist $R \circ S$ ihre Funktionskomposition.

- Leere Sequenz: Leere Funktion?

Denotationale Semantik von Anweisungen

- Zuweisung: punktweise Änderung des Zustands σ zu $\sigma[x \mapsto n]$
- Sequenz: Komposition von Relationen

Definition (Komposition von Relationen)

Für zwei Relationen $R \subseteq X \times Y, S \subseteq Y \times Z$ ist ihre **Komposition**

$$R \circ S \stackrel{def}{=} \{(x, z) \mid \exists y \in Y. (x, y) \in R \land (y, z) \in S\}$$

Wenn R, S zwei partielle Funktionen sind, ist $R \circ S$ ihre Funktionskomposition.

- Leere Sequenz: Leere Funktion? Nein, Identität. Für Menge X,

$$\mathbf{Id}_X \stackrel{def}{=} X \times X = \{(x, x) \mid x \in X\}$$

ist die **Identitätsfunktion** ($\mathbf{Id}_X(x) = x$).

Arbeitsblatt 3.4: Komposition von Relationen

Zur Übung: betrachten Sie folgende Relationen:

$$R = \{(1,7), (2,3), (3,9), (4,3)\}$$
$$S = \{(1,0), (2,0), (3,1), (4,7), (5,9), (7,3), (8,15)\}$$

Berechnen Sie $R \circ S = \{(1,?), \ldots\}$

Arbeitsblatt 3.4: Komposition von Relationen

Zur Übung: betrachten Sie folgende Relationen:

$$R = \{(1,7),\ (2,3),\ (3,9),\ (4,3)\}$$
$$S = \{(1,0),\ (2,0),\ (3,1),\ (4,7),\ (5,9),\ (7,3),\ (8,15)\}$$

Berechnen Sie $R \circ S = \{(1, ?), \ldots\}$

Arbeitsblatt 3.4: Komposition von Relationen

Zur Übung: betrachten Sie folgende Relationen:

$$R = \{(1,7),\ (2,3),\ (3,9),\ (4,3)\}$$
$$S = \{(1,0),\ (2,0),\ (3,1),\ (3,5),\ (4,7),\ (5,9),\ (7,3),\ (8,15)\}$$

Berechnen Sie $R \circ S = \{(1,?), \ldots\}$

Denotat von Stmt

$$[\![.]\!]_{\mathcal{C}} : \textbf{Stmt} \to (\Sigma \rightharpoonup \Sigma)$$

$$\begin{aligned}
[\![x = a]\!]_{\mathcal{C}} =& \{(\sigma, \sigma[x \mapsto n]) \mid \sigma \in \Sigma \wedge (\sigma, n) \in [\![a]\!]_{\mathcal{A}}\} \\
[\![c_1; c_2]\!]_{\mathcal{C}} =& [\![c_1]\!]_{\mathcal{C}} \circ [\![c_2]\!]_{\mathcal{C}} \\
[\![\{\,\}]\!]_{\mathcal{C}} =& \textbf{Id}_{\Sigma} \\
[\![\textbf{if } (b)\ c_0\ \textbf{ else }\ c_1]\!]_{\mathcal{C}} =& \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c_0]\!]_{\mathcal{C}}\} \\
& \cup \{(\sigma, \sigma') \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}\}
\end{aligned}$$

Denotat von Stmt

$$[\![.]\!]_{\mathcal{C}} : \textbf{Stmt} \to (\Sigma \rightharpoonup \Sigma)$$

$$\begin{aligned}
[\![x = a]\!]_{\mathcal{C}} =& \{(\sigma, \sigma[x \mapsto n]) \mid \sigma \in \Sigma \land (\sigma, n) \in [\![a]\!]_{\mathcal{A}}\} \\
[\![c_1; c_2]\!]_{\mathcal{C}} =& [\![c_1]\!]_{\mathcal{C}} \circ [\![c_2]\!]_{\mathcal{C}} \\
[\![\{\ \}]\!]_{\mathcal{C}} =& \mathbf{Id}_{\Sigma} \\
[\![\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1]\!]_{\mathcal{C}} =& \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c_0]\!]_{\mathcal{C}}\} \\
& \cup \{(\sigma, \sigma') \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}\}
\end{aligned}$$

Aber was ist

$$[\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} = ??$$

Denotationale Semantik von while

- Sei $w \equiv \textbf{while}\ (b)\ c$ (und $\sigma \in \Sigma$). Operational gilt:

$$w \sim \textbf{if}\ (b)\ \{c; w\}\ \textbf{else}\ \{\,\}$$

- Dann sollte auch gelten

$$\begin{aligned}
[\![w]\!]_{\mathcal{C}} &\stackrel{?}{=} [\![\textbf{if}\ (b)\ \{c; w\}\ \textbf{else}\ \{\,\}]\!]_{\mathcal{C}} \\
&= \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ [\![w]\!]_{\mathcal{C}}\} \\
&\quad \cup \{(\sigma, \sigma') \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![\{\,\}]\!]_{\mathcal{C}}\}
\end{aligned}$$

- Das ist eine **rekursive** Definition von $[\![w]\!]_{\mathcal{C}}$:

$$x = F(x)$$

- Das ist ein **Fixpunkt**:

$$x = \mathit{fix}(F)$$

- Was ist das?

Denotationale Semantik von while

- Sei $w \equiv \textbf{while}\ (b)\ c$ (und $\sigma \in \Sigma$). Operational gilt:

$$w \sim \textbf{if}\ (b)\ \{c; w\}\ \textbf{else}\ \{\,\}$$

- Dann sollte auch gelten

$$\begin{aligned}
[\![w]\!]_{\mathcal{C}} &\stackrel{?}{=} [\![\textbf{if}\ (b)\ \{c; w\}\ \textbf{else}\ \{\,\}]\!]_{\mathcal{C}} \\
&= \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ [\![w]\!]_{\mathcal{C}}\} \\
&\quad \cup \{(\sigma, \sigma') \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![\{\,\}]\!]_{\mathcal{C}}\}
\end{aligned}$$

- Das ist eine **rekursive** Definition von $[\![w]\!]_{\mathcal{C}}$:

$$x = F(x)$$

- Das ist ein **Fixpunkt**:

$$x = \mathit{fix}(F)$$

- Was ist das?

Fixpunkte

Definition (Fixpunkt)

Für $f : X \rightharpoonup X$ ist ein **Fixpunkt** ein $x \in X$ so dass $f(x) = x$.

- Hat jede Funktion $f : X \rightharpoonup X$ einen Fixpunkt?

Fixpunkte

Definition (Fixpunkt)

Für $f : X \rightharpoonup X$ ist ein **Fixpunkt** ein $x \in X$ so dass $f(x) = x$.

- Hat jede Funktion $f : X \rightharpoonup X$ einen Fixpunkt? Nein
- Kann eine Funktion mehrere Fixpunkte haben?

Fixpunkte

Definition (Fixpunkt)

Für $f : X \rightharpoonup X$ ist ein **Fixpunkt** ein $x \in X$ so dass $f(x) = x$.

- Hat jede Funktion $f : X \rightharpoonup X$ einen Fixpunkt? Nein
- Kann eine Funktion mehrere Fixpunkte haben? Ja — aber nur einen kleinsten.
- Beispiele
  - Fixpunkte von $f(x) = \sqrt{x}$ sind 0 und 1; ebenfalls für $f(x) = x^2$.
  - Für die Sortierfunktion sind alle sortierten Listen Fixpunkte
  - Die Funktion $f(x) = x + 1$ hat keinen Fixpunkt in $\mathbb{Z}$
  - Die Funktion $f(X) = \mathbb{P}(X)$ hat überhaupt keinen Fixpunkt
- $\mathit{fix}(f)$ ist also der **kleinste Fixpunkt** von f.

Denotationale Semantik für die Iteration

- Sei $w \equiv \textbf{while}\ (b)\ c$
- Konstruktion: "Auffalten" der Schleife (f ist ein Denotat):

$$\begin{aligned}\Gamma(f) = & \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ f\} \\ & \cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}\end{aligned}$$

- b und c sind Parameter von Γ
- Dann ist

$$[\![w]\!]_{\mathcal{C}} = \mathit{fix}(\Gamma)$$

Konstruktion des kleinsten Fixpunktes (Kurzversion)

- Gegeben Funktion Γ auf Denotaten $\Gamma : (\Sigma \rightharpoonup \Sigma) \rightharpoonup (\Sigma \rightharpoonup \Sigma)$
- Wir konstruieren eine Sequenz $\Gamma^i : \Sigma \rightharpoonup \Sigma$ (mit $i \in \mathbb{N}$) von Funktionen:

$$\Gamma^0(s) \stackrel{def}{=} \emptyset$$
$$\Gamma^{i+1}(s) \stackrel{def}{=} \Gamma(\Gamma^i)(s)$$

- Dann ist

$$fix(\Gamma) \stackrel{def}{=} \bigcup_{i \in \mathbb{N}} \Gamma^i$$

- Verkürzte Version — der Fixpunkt muss so nicht existieren (er tut es aber für alle Programme)

Denotation für Stmt

$$[\![.]\!]_{\mathcal{C}} : \{Stmt \to (\Sigma \rightharpoonup \Sigma)\}$$

$$\begin{aligned}
[\![x = a]\!]_{\mathcal{C}} =& \{(\sigma, \sigma[x \mapsto n]) \mid \sigma \in \Sigma \land (\sigma, n) \in [\![a]\!]_{\mathcal{A}}\} \\
[\![c_1; c_2]\!]_{\mathcal{C}} =& [\![c_1]\!]_{\mathcal{C}} \circ [\![c_2]\!]_{\mathcal{C}} \\
[\![\{\,\}]\!]_{\mathcal{C}} =& \mathbf{Id}_{\Sigma} \\
[\![\textbf{if}\ (b)\ c_0\ \ \textbf{else}\ \ c_1]\!]_{\mathcal{C}} =& \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c_0]\!]_{\mathcal{C}}\} \\
& \cup\ \{(\sigma, \sigma') \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}\} \\
[\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} =& \mathit{fix}(\Gamma)
\end{aligned}$$

$$\begin{aligned}
\Gamma(s) =& \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ s\} \\
& \cup\ \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}$$

Der Fixpunkt bei der Arbeit (I)

```
while (x < 0) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(x) \geq 0 \\ f(\sigma[x \mapsto \sigma(x) + 1]) & \sigma(x) < 0 \end{cases}$$

Wir betrachten den Zustand $s = \langle x \mapsto ? \rangle$ (nur eine Variable):

Der Fixpunkt bei der Arbeit (I)

```
while (x < 0) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(x) \geq 0 \\ f(\sigma[x \mapsto \sigma(x)+1]) & \sigma(x) < 0 \end{cases}$$

Wir betrachten den Zustand $s = \langle x \mapsto ? \rangle$ (nur eine Variable):

s
-2
-1
0
1

Der Fixpunkt bei der Arbeit (I)

```
while (x < 0) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(x) \geq 0 \\ f(\sigma[x \mapsto \sigma(x) + 1]) & \sigma(x) < 0 \end{cases}$$

Wir betrachten den Zustand $s = \langle x \mapsto ? \rangle$ (nur eine Variable):

s	$\Gamma^0(s)$
-2	$\perp$
-1	$\perp$
0	$\perp$
1	$\perp$

Der Fixpunkt bei der Arbeit (I)

```
while (x < 0) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(x) \geq 0 \\ f(\sigma[x \mapsto \sigma(x) + 1]) & \sigma(x) < 0 \end{cases}$$

Wir betrachten den Zustand $s = \langle x \mapsto ? \rangle$ (nur eine Variable):

s	$\Gamma^0(s)$	$\Gamma^1(s)$
-2	$\perp$	$\Gamma^0(s[x \mapsto -1]) = \perp$
-1	$\perp$	$\Gamma^0(s[x \mapsto 0]) = \perp$
0	$\perp$	0
1	$\perp$	1

Der Fixpunkt bei der Arbeit (I)

```
while (x < 0) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(x) \geq 0 \\ f(\sigma[x \mapsto \sigma(x)+1]) & \sigma(x) < 0 \end{cases}$$

Wir betrachten den Zustand $s = \langle x \mapsto ? \rangle$ (nur eine Variable):

s	$\Gamma^0(s)$	$\Gamma^1(s)$	$\Gamma^2(s)$
-2	$\perp$	$\Gamma^0(s[x \mapsto -1]) = \perp$	$\Gamma^1(s[x \mapsto -1]) = \perp$
-1	$\perp$	$\Gamma^0(s[x \mapsto 0]) = \perp$	$\Gamma^1(s[x \mapsto 0]) = 0$
0	$\perp$	0	0
1	$\perp$	1	1

Der Fixpunkt bei der Arbeit (I)

```
while (x < 0) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(x) \geq 0 \\ f(\sigma[x \mapsto \sigma(x)+1]) & \sigma(x) < 0 \end{cases}$$

Wir betrachten den Zustand $s = \langle x \mapsto ? \rangle$ (nur eine Variable):

s	$\Gamma^0(s)$	$\Gamma^1(s)$	$\Gamma^2(s)$	$\Gamma^3(s)$
-2	$\perp$	$\Gamma^0(s[x \mapsto -1]) = \perp$	$\Gamma^1(s[x \mapsto -1]) = \perp$	$\Gamma^2(s[x \mapsto -1]) = 0$
-1	$\perp$	$\Gamma^0(s[x \mapsto 0]) = \perp$	$\Gamma^1(s[x \mapsto 0]) = 0$	$\Gamma^2(s[x \mapsto 0]) = 0$
0	$\perp$	0	0	0
1	$\perp$	1	1	1

Der Fixpunkt bei der Arbeit (II)

```
x= 0;
while (n > 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) \leq 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \sigma(n) > 0 \end{cases}$$

Wir betrachten Zustände $s = \langle x \mapsto ?, n \mapsto ? \rangle$ (zwei Variablen).
Der Wert von x im Initialzustand ist dabei unerheblich:

s
n
-1
0
1
2
3
4

Der Fixpunkt bei der Arbeit (II)

```
x= 0;
while (n > 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) \leq 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \sigma(n) > 0 \end{cases}$$

Wir betrachten Zustände $s = \langle x \mapsto ?, n \mapsto ? \rangle$ (zwei Variablen).
Der Wert von x im Initialzustand ist dabei unerheblich:

s	$\Gamma^0(s)$	
n	x	n
-1	$\perp$	$\perp$
0	$\perp$	$\perp$
1	$\perp$	$\perp$
2	$\perp$	$\perp$
3	$\perp$	$\perp$
4	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (II)

```
x= 0;
while (n > 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) \leq 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \sigma(n) > 0 \end{cases}$$

Wir betrachten Zustände $s = \langle x \mapsto ?, n \mapsto ? \rangle$ (zwei Variablen).
Der Wert von x im Initialzustand ist dabei unerheblich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$	
n	x	n	x	n
-1	$\perp$	$\perp$	0	-1
0	$\perp$	$\perp$	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$
2	$\perp$	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$	$\perp$
4	$\perp$	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (II)

```
x= 0;
while (n > 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) \leq 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \sigma(n) > 0 \end{cases}$$

Wir betrachten Zustände $s = \langle x \mapsto ?, n \mapsto ? \rangle$ (zwei Variablen).
Der Wert von x im Initialzustand ist dabei unerheblich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$		$\Gamma^2(s)$	
n	x	n	x	n	x	n
-1	$\perp$	$\perp$	0	-1	0	-1
0	$\perp$	$\perp$	0	0	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$	1	0
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
4	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (II)

```
x= 0;
while (n > 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) \leq 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \sigma(n) > 0 \end{cases}$$

Wir betrachten Zustände $s = \langle x \mapsto ?, n \mapsto ? \rangle$ (zwei Variablen).
Der Wert von x im Initialzustand ist dabei unerheblich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$		$\Gamma^2(s)$		$\Gamma^3(s)$	
n	x	n	x	n	x	n	x	n
-1	$\perp$	$\perp$	0	-1	0	-1	0	-1
0	$\perp$	$\perp$	0	0	0	0	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$	1	0	1	0
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	3	0
3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
4	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (II)

```
x= 0;
while (n > 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) \leq 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \sigma(n) > 0 \end{cases}$$

Wir betrachten Zustände $s = \langle x \mapsto ?, n \mapsto ? \rangle$ (zwei Variablen).
Der Wert von x im Initialzustand ist dabei unerheblich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$		$\Gamma^2(s)$		$\Gamma^3(s)$		$\Gamma^4(s)$	
n	x	n	x	n	x	n	x	n	x	n
-1	$\perp$	$\perp$	0	-1	0	-1	0	-1	0	-1
0	$\perp$	$\perp$	0	0	0	0	0	0	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$	1	0	1	0	1	0
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	3	0	3	0
3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	6	0
4	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (II)

```
x= 0;
while (n > 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) \leq 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \sigma(n) > 0 \end{cases}$$

Wir betrachten Zustände $s = \langle x \mapsto ?, n \mapsto ? \rangle$ (zwei Variablen).
Der Wert von x im Initialzustand ist dabei unerheblich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$		$\Gamma^2(s)$		$\Gamma^3(s)$		$\Gamma^4(s)$		$\Gamma^5(s)$	
n	x	n	x	n	x	n	x	n	x	n	x	n
-1	$\perp$	$\perp$	0	-1	0	-1	0	-1	0	-1	0	-1
0	$\perp$	$\perp$	0	0	0	0	0	0	0	0	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$	1	0	1	0	1	0	1	0
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	3	0	3	0	3	0
3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	6	0	6	0
4	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	10	0

Der Fixpunkt bei der Arbeit (III)

Kleine Änderung im Beispielprogramm:

```
x= 0;
while (n != 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) = 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \text{sonst} \end{cases}$$

Jetzt ergibt sich:

$$\begin{array}{r} s \\ n \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \\ 3 \end{array}$$

Der Fixpunkt bei der Arbeit (III)

Kleine Änderung im Beispielprogramm:

```
x= 0;
while (n != 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) = 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \text{sonst} \end{cases}$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$	
n	x	n
-2	$\perp$	$\perp$
-1	$\perp$	$\perp$
0	$\perp$	$\perp$
1	$\perp$	$\perp$
2	$\perp$	$\perp$
3	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (III)

Kleine Änderung im Beispielprogramm:

```
x= 0;
while (n != 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) = 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \text{sonst} \end{cases}$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$	
n	x	n	x	n
-2	$\perp$	$\perp$	$\perp$	$\perp$
-1	$\perp$	$\perp$	$\perp$	$\perp$
0	$\perp$	$\perp$	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$
2	$\perp$	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (III)

Kleine Änderung im Beispielprogramm:

```
x= 0;
while (n != 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) = 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \text{sonst} \end{cases}$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$		$\Gamma^2(s)$	
n	x	n	x	n	x	n
-2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
-1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
0	$\perp$	$\perp$	0	0	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$	1	0
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (III)

Kleine Änderung im Beispielprogramm:

```
x= 0;
while (n != 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) = 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \text{sonst} \end{cases}$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$		$\Gamma^2(s)$		$\Gamma^3(s)$	
n	x	n	x	n	x	n	x	n
-2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
-1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
0	$\perp$	$\perp$	0	0	0	0	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$	1	0	1	0
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	3	0
3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (III)

Kleine Änderung im Beispielprogramm:

```
x= 0;
while (n != 0) {
  x= x+n;
  n= n-1;
  }
```

$$\Gamma(f)(\sigma) = \begin{cases} \sigma & \sigma(n) = 0 \\ f(\sigma[x \mapsto \sigma(x) + \sigma(n)][n \mapsto \sigma(n) - 1]) & \text{sonst} \end{cases}$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$		$\Gamma^1(s)$		$\Gamma^2(s)$		$\Gamma^3(s)$		$\Gamma^4(s)$	
n	x	n	x	n	x	n	x	n	x	n
-2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
-1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
0	$\perp$	$\perp$	0	0	0	0	0	0	0	0
1	$\perp$	$\perp$	$\perp$	$\perp$	1	0	1	0	1	0
2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	3	0	3	0
3	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	6	0

Der Fixpunkt bei der Arbeit (IV)

```
while (1) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} f(\sigma[x \mapsto \sigma(x) + 1])$$

Jetzt ergibt sich:

s

-2

-1

0

1

2

3

Der Fixpunkt bei der Arbeit (IV)

```
while (1) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} f(\sigma[x \mapsto \sigma(x) + 1])$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$
-2	$\perp$
-1	$\perp$
0	$\perp$
1	$\perp$
2	$\perp$
3	$\perp$

Der Fixpunkt bei der Arbeit (IV)

```
while (1) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} f(\sigma[x \mapsto \sigma(x) + 1])$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$	$\Gamma^1(s)$
-2	$\perp$	$\perp$
-1	$\perp$	$\perp$
0	$\perp$	$\perp$
1	$\perp$	$\perp$
2	$\perp$	$\perp$
3	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (IV)

```
while (1) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} f(\sigma[x \mapsto \sigma(x) + 1])$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$	$\Gamma^1(s)$	$\Gamma^2(s)$
-2	$\perp$	$\perp$	$\perp$
-1	$\perp$	$\perp$	$\perp$
0	$\perp$	$\perp$	$\perp$
1	$\perp$	$\perp$	$\perp$
2	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (IV)

```
while (1) {
  x= x+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} f(\sigma[x \mapsto \sigma(x) + 1])$$

Jetzt ergibt sich:

s	$\Gamma^0(s)$	$\Gamma^1(s)$	$\Gamma^2(s)$	$\Gamma^3(s)$
-2	$\perp$	$\perp$	$\perp$	$\perp$
-1	$\perp$	$\perp$	$\perp$	$\perp$
0	$\perp$	$\perp$	$\perp$	$\perp$
1	$\perp$	$\perp$	$\perp$	$\perp$
2	$\perp$	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$	$\perp$

Arbeitsblatt 3.5: Semantik III

Wir betrachten das Beispielprogramm:

```
x= 1;
while (n > 0) {
  x= x*n;
  n= n-1;
  }
```

Berechnen Sie wie oben den Fixpunkt:

```
s     G^0          G^1           G^2        G^3          G^4
n   x    n      x    n        x    n     x    n       x    n
0
1
2
3
```

Arbeitsblatt 3.5: Semantik III

Wir betrachten das Beispielprogramm:

```
x= 1;
while (n > 0) {
  x= x*n;
  n= n-1;
  }
```

Der Fixpunkt bei der Arbeit (V)

```
x= 0;
i= 0;
while (i<=n) {
  x= x+i;
  i= i+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(i) > \sigma(n) \\ f(\sigma[x \mapsto \sigma(x) + \sigma(i)][i \mapsto \sigma(i) + 1]) & \text{sonst} \end{cases}$$

Wir betrachten nur die **while**-Schleife mit $s = \langle n \mapsto ?, i \mapsto ?, x \mapsto ? \rangle$.

s		$\Gamma^0(s)$		
n	i	n	i	x
0	0	$\perp$	$\perp$	$\perp$
0	1	$\perp$	$\perp$	$\perp$
1	0	$\perp$	$\perp$	$\perp$
1	1	$\perp$	$\perp$	$\perp$
1	2	$\perp$	$\perp$	$\perp$
2	0	$\perp$	$\perp$	$\perp$
2	1	$\perp$	$\perp$	$\perp$
2	2	$\perp$	$\perp$	$\perp$
2	3	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (V)

```
x= 0;
i= 0;
while (i<=n) {
  x= x+i;
  i= i+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(i) > \sigma(n) \\ f(\sigma[x \mapsto \sigma(x) + \sigma(i)][i \mapsto \sigma(i) + 1]) & \text{sonst} \end{cases}$$

Wir betrachten nur die **while**-Schleife mit $s = \langle n \mapsto ?, i \mapsto ?, x \mapsto ? \rangle$.

s		$\Gamma^0(s)$		
n	i	n	i	x
0	0	$\perp$	$\perp$	$\perp$
0	1	$\perp$	$\perp$	$\perp$
1	0	$\perp$	$\perp$	$\perp$
1	1	$\perp$	$\perp$	$\perp$
1	2	$\perp$	$\perp$	$\perp$
2	0	$\perp$	$\perp$	$\perp$
2	1	$\perp$	$\perp$	$\perp$
2	2	$\perp$	$\perp$	$\perp$
2	3	$\perp$	$\perp$	$\perp$

Der Fixpunkt bei der Arbeit (V)

```
x= 0;
i= 0;
while (i<=n) {
  x= x+i ;
  i= i+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(i) > \sigma(n) \\ f(\sigma[x \mapsto \sigma(x) + \sigma(i)][i \mapsto \sigma(i) + 1]) & \text{sonst} \end{cases}$$

Wir betrachten nur die **while**-Schleife
mit $s = \langle n \mapsto ?, i \mapsto ?, x \mapsto ? \rangle$.

s		$\Gamma^0(s)$			$\Gamma^1(s)$		
n	i	n	i	x	n	i	x
0	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
0	1	$\perp$	$\perp$	$\perp$	0	1	x
1	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
1	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
1	2	$\perp$	$\perp$	$\perp$	1	2	x
2	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
2	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
2	2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
2	3	$\perp$	$\perp$	$\perp$	2	3	x

Der Fixpunkt bei der Arbeit (V)

```
x= 0;
i= 0;
while (i<=n) {
  x= x+i;
  i= i+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(i) > \sigma(n) \\ f(\sigma[x \mapsto \sigma(x) + \sigma(i)][i \mapsto \sigma(i) + 1]) & \text{sonst} \end{cases}$$

Wir betrachten nur die **while**-Schleife mit $s = \langle n \mapsto ?, i \mapsto ?, x \mapsto ? \rangle$.

s		$\Gamma^0(s)$			$\Gamma^1(s)$			$\Gamma^2(s)$		
n	i	n	i	x	n	i	x	n	i	x
0	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0	1	x
0	1	$\perp$	$\perp$	$\perp$	0	1	x	0	1	x
1	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
1	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	1	2	$x+1$
1	2	$\perp$	$\perp$	$\perp$	1	2	x	1	2	x
2	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
2	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
2	2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	2	3	$x+2$
2	3	$\perp$	$\perp$	$\perp$	2	3	x	2	3	x

Der Fixpunkt bei der Arbeit (V)

```
x= 0;
i= 0;
while (i<=n) {
  x= x+i;
  i= i+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(i) > \sigma(n) \\ f(\sigma[x \mapsto \sigma(x) + \sigma(i)][i \mapsto \sigma(i) + 1]) & \text{sonst} \end{cases}$$

Wir betrachten nur die **while**-Schleife
mit $s = \langle n \mapsto ?, i \mapsto ?, x \mapsto ? \rangle$.

s		$\Gamma^0(s)$			$\Gamma^1(s)$			$\Gamma^2(s)$			$\Gamma^3(s)$		
n	i	n	i	x	n	i	x	n	i	x	n	i	x
0	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0	1	x	0	1	x
0	1	$\perp$	$\perp$	$\perp$	0	1	x	0	1	x	0	1	x
1	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	1	2	$x+1$
1	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	1	2	$x+1$	1	2	$x+1$
1	2	$\perp$	$\perp$	$\perp$	1	2	x	1	2	x	1	2	x
2	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
2	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	2	3	$x+3$
2	2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	2	3	$x+2$	2	3	$x+2$
2	3	$\perp$	$\perp$	$\perp$	2	3	x	2	3	x	2	3	x

Der Fixpunkt bei der Arbeit (V)

```
x= 0;
i= 0;
while (i<=n) {
  x= x+i;
  i= i+1;
  }
```

$$\Gamma(f)(\sigma) \stackrel{def}{=} \begin{cases} \sigma & \sigma(i) > \sigma(n) \\ f(\sigma[x \mapsto \sigma(x) + \sigma(i)][i \mapsto \sigma(i) + 1]) & \text{sonst} \end{cases}$$

Wir betrachten nur die **while**-Schleife
mit $s = \langle n \mapsto ?, i \mapsto ?, x \mapsto ? \rangle$.

s		$\Gamma^0(s)$			$\Gamma^1(s)$			$\Gamma^2(s)$			$\Gamma^3(s)$			$\Gamma^4(s)$		
n	i	n	i	x	n	i	x	n	i	x	n	i	x	n	i	x
0	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	0	1	x	0	1	x	0	1	x
0	1	$\perp$	$\perp$	$\perp$	0	1	x	0	1	x	0	1	x	0	1	x
1	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	1	2	$x+1$	1	2	$x+1$
1	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	1	2	$x+1$	1	2	$x+1$	1	2	$x+1$
1	2	$\perp$	$\perp$	$\perp$	1	2	x	1	2	x	1	2	x	1	2	x
2	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	2	3	$x+3$
2	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	2	3	$x+3$	2	3	$x+3$
2	2	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	2	3	$x+2$	2	3	$x+2$	2	3	$x+2$
2	3	$\perp$	$\perp$	$\perp$	2	3	x	2	3	x	2	3	x	2	3	x

Weitere Eigenschaften der denotationalen Semantik

Lemma (Partielle Funktion)

$[\![-]\!]_{\mathcal{C}}$ *ist rechtseindeutig und damit eine **partielle Funktion**.*

- Beweis über strukturelle Induktion über $c \in$ **Stmt** und über **Fixpunktinduktion**:
  - Zu zeigen: wenn s rechtseindeutig, dann ist $\Gamma(s)$ rechtseindeutig
  - Dann ist $fix(\Gamma)$ rechtseindeutig.
- Eigenschaften der Iteration:
  - Sei $w \equiv \textbf{while}\ (b)\ c$
  - Dann

$$[\![w]\!]_{\mathcal{C}} = [\![\textbf{if}\ (b)\ \{c; w\}\ \textbf{else}\ \{\ \}]\!]_{\mathcal{C}} \tag{1}$$

$$(\sigma, \sigma') \in [\![w]\!]_{\mathcal{C}} \Longrightarrow (\sigma', \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \tag{2}$$

Beweis (1)

$$[\![w]\!]_{\mathcal{C}} = \mathit{fix}(\Gamma)$$

Note

$$\mathit{fix}(\Gamma) = \Gamma(\mathit{fix}(\Gamma))$$

Beweis (1)

$$
\begin{aligned}
[\![w]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\
&= \Gamma(\mathit{fix}(\Gamma))
\end{aligned}
$$

Note

$$\mathit{fix}(\Gamma) = \Gamma(\mathit{fix}(\Gamma))$$

Beweis (1)

$$\begin{aligned} [\![w]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\ &= \Gamma(\mathit{fix}(\Gamma)) \\ &= \Gamma([\![w]\!]_{\mathcal{C}}) \end{aligned}$$

Note

$$\begin{aligned} \Gamma(s) = \; & \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ s\} \\ & \cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\} \end{aligned}$$

Beweis (1)

$$
\begin{aligned}
[\![w]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\
&= \Gamma(\mathit{fix}(\Gamma)) \\
&= \Gamma([\![w]\!]_{\mathcal{C}}) \\
&= \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ [\![w]\!]_{\mathcal{C}}\} \\
&\qquad \cup \ \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

Note

$$
\begin{aligned}
\Gamma(s) = \ &\{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \land (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ s\} \\
&\cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

Beweis (1)

$$
\begin{aligned}
[\![w]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\
&= \Gamma(\mathit{fix}(\Gamma)) \\
&= \Gamma([\![w]\!]_{\mathcal{C}}) \\
&= \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ [\![w]\!]_{\mathcal{C}}\} \\
&\quad \cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

Note

Beweis (1)

$$
\begin{aligned}
[\![w]\!]_{\mathcal{C}} &= fix(\Gamma) \\
&= \Gamma(fix(\Gamma)) \\
&= \Gamma([\![w]\!]_{\mathcal{C}}) \\
&= \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ [\![w]\!]_{\mathcal{C}}\} \\
&\quad \cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\} \\
&= \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c;\ w]\!]_{\mathcal{C}}\} \\
&\quad \cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma) \in [\![\{\ \}]\!]_{\mathcal{C}}\}
\end{aligned}
$$

Note

Beweis (1)

$$
\begin{aligned}
[\![w]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\
&= \Gamma(\mathit{fix}(\Gamma)) \\
&= \Gamma([\![w]\!]_{\mathcal{C}}) \\
&= \{(\sigma,\sigma') \mid (\sigma,\mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma,\sigma') \in [\![c]\!]_{\mathcal{C}} \circ [\![w]\!]_{\mathcal{C}}\} \\
&\quad\ \cup\ \{(\sigma,\sigma) \mid (\sigma,\mathit{false}) \in [\![b]\!]_{\mathcal{B}}\} \\
&= \{(\sigma,\sigma') \mid (\sigma,\mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma,\sigma') \in [\![c;\ \ w]\!]_{\mathcal{C}}\} \\
&\quad\ \cup\ \{(\sigma,\sigma) \mid (\sigma,\mathit{false}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma,\sigma) \in [\![\{\ \}]\!]_{\mathcal{C}}\}
\end{aligned}
$$

Note

$$
\begin{aligned}
[\![\textbf{if}\ (b)\ c_0\ \ \textbf{else}\ \ c_1]\!]_{\mathcal{C}} = \ &\{(\sigma,\sigma') \mid (\sigma,\mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma,\sigma') \in [\![c_0]\!]_{\mathcal{C}}\} \\
&\cup \{(\sigma,\sigma') \mid (\sigma,\mathit{false}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma,\sigma') \in [\![c_1]\!]_{\mathcal{C}}\}
\end{aligned}
$$

Beweis (1)

$$
\begin{aligned}
[\![w]\!]_{\mathcal{C}} &= fix(\Gamma) \\
&= \Gamma(fix(\Gamma)) \\
&= \Gamma([\![w]\!]_{\mathcal{C}}) \\
&= \{(\sigma, \sigma') \mid (\sigma, true) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ [\![w]\!]_{\mathcal{C}}\} \\
&\quad \cup \{(\sigma, \sigma) \mid (\sigma, false) \in [\![b]\!]_{\mathcal{B}}\} \\
&= \{(\sigma, \sigma') \mid (\sigma, true) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c;\; w]\!]_{\mathcal{C}}\} \\
&\quad \cup \{(\sigma, \sigma) \mid (\sigma, false) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma) \in [\![\{\,\}]\!]_{\mathcal{C}}\} \\
&= [\![\textbf{if}\ (b)\ \{c; w\}\ \textbf{else}\ \{\,\}]\!]_{\mathcal{C}}
\end{aligned}
$$

Note

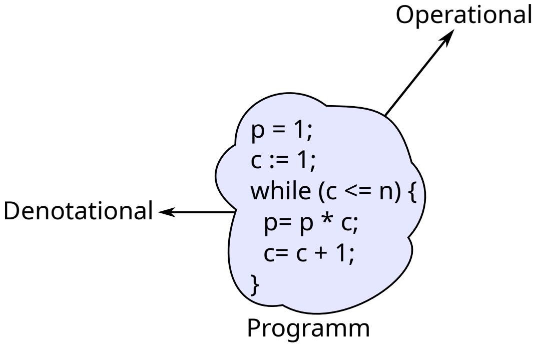
$$
\begin{aligned}
[\![\textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1]\!]_{\mathcal{C}} = &\{(\sigma, \sigma') \mid (\sigma, true) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c_0]\!]_{\mathcal{C}}\} \\
&\cup \{(\sigma, \sigma') \mid (\sigma, false) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}\}
\end{aligned}
$$

Zusammenfassung

- Die denotationale Semantik bildet Programme (Ausdrücke) auf **partielle Funktionen** $\Sigma \rightharpoonup \Sigma$ ab.
- Zentral ist der Begriff des **kleinsten Fixpunktes**, der die Semantik der while-Schleife bildet.
- Undefiniertheit wird **implizit** behandelt (durch die Partialität von $\Sigma \rightharpoonup \Sigma$).
  - Nicht-Termination und Undefiniertheit sind semantisch äquivalent.
- Genaues Verhältnis zur **operationalen Semantik?**

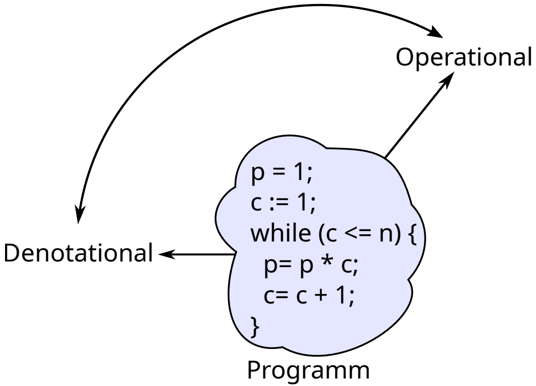
Zusammenfassung

- Die denotationale Semantik bildet Programme (Ausdrücke) auf **partielle Funktionen** $\Sigma \rightharpoonup \Sigma$ ab.
- Zentral ist der Begriff des **kleinsten Fixpunktes**, der die Semantik der while-Schleife bildet.
- Undefiniertheit wird **implizit** behandelt (durch die Partialität von $\Sigma \rightharpoonup \Sigma$).
  - Nicht-Termination und Undefiniertheit sind semantisch äquivalent.
- Genaues Verhältnis zur **operationalen Semantik?** Nächste Vorlesung



Zusammenfassung

- Die denotationale Semantik bildet Programme (Ausdrücke) auf **partielle Funktionen** $\Sigma \rightharpoonup \Sigma$ ab.
- Zentral ist der Begriff des **kleinsten Fixpunktes**, der die Semantik der while-Schleife bildet.
- Undefiniertheit wird **implizit** behandelt (durch die Partialität von $\Sigma \rightharpoonup \Sigma$).
  - Nicht-Termination und Undefiniertheit sind semantisch äquivalent.
- Genaues Verhältnis zur **operationalen Semantik?** Nächste Vorlesung



Korrekte Software: Grundlagen und Methoden

Vorlesung 4 vom 24.04.24

Äquivalenz der Operationalen und Denotationalen Semantik

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2024

Fahrplan

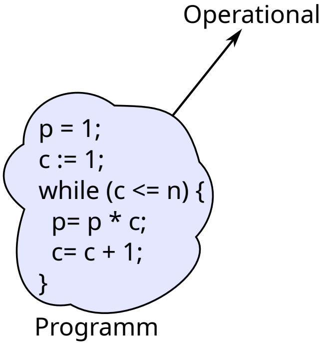
- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten im Floyd-Hoare-Kalkül
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare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

Operationale und Denotationale Semantik

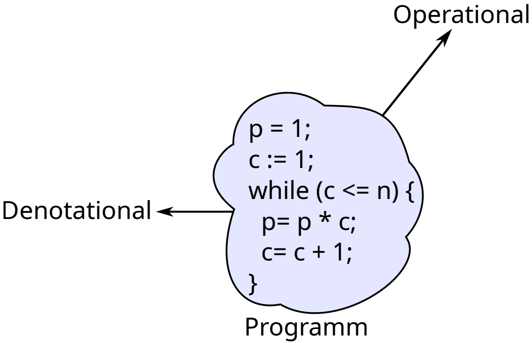
```
p = 1;
c := 1;
while (c <= n) {
  p= p * c;
  c= c + 1;
}
```

Programm

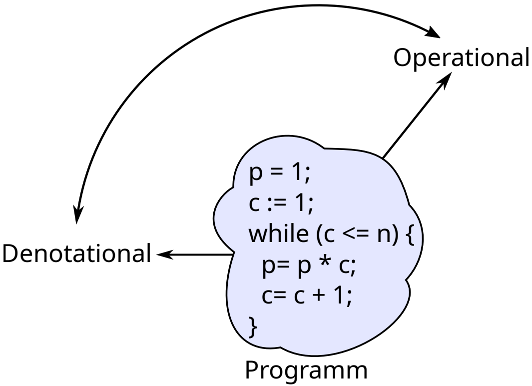
Operationale und Denotationale Semantik



Operationale und Denotationale Semantik



Operationale und Denotationale Semantik



Äquivalenz der Operationalen und Denotationalen Semantik

- Was müssen wir zeigen?

Äquivalenz der Operationalen und Denotationalen Semantik

- Was müssen wir zeigen?
- Auf oberster Ebene: für alle $c \in \textbf{Stmt}, \sigma, \sigma' \in \Sigma$:

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longleftrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \tag{1}$$

- Semantik von Anweisungen ist über Semantik von Ausdrücken definiert, deshalb benötigen wir Hilfsaussagen

$$\langle b, \sigma \rangle \rightarrow_{Bexp} t \Longleftrightarrow (\sigma, t) \in [\![b]\!]_{\mathcal{B}} \tag{2}$$

$$\langle a, \sigma \rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a]\!]_{\mathcal{A}} \tag{3}$$

- Wie zeigen wir das?

Operationale vs. denotationale Semantik

	Operational $\langle a, \sigma \rangle \rightarrow_{Aexp} n$	**Denotational** $[\![a]\!]_{\mathcal{A}}$
m$\in$ **Z**	$\langle m, \sigma \rangle \rightarrow_{Aexp} m$	$\{(\sigma, m) \mid \sigma \in \Sigma\}$
x$\in$ **Loc**	$\dfrac{x \in Dom(\sigma)}{\langle x, \sigma \rangle \rightarrow_{Aexp} \sigma(x)}$	$\{(\sigma, \sigma(x)) \mid \sigma \in \Sigma, x \in Dom(\sigma)\}$

Operationale vs. denotationale Semantik

	Operational $\langle a, \sigma \rangle \rightarrow_{Aexp} n$	**Denotational** $[\![a]\!]_{\mathcal{A}}$
$a_1 \otimes a_2$	$\dfrac{\langle a_1, \sigma \rangle \rightarrow_{Aexp} n \quad \langle a_2, \sigma \rangle \rightarrow_{Aexp} m}{\langle a_1 \otimes a_2, \sigma \rangle \rightarrow_{Aexp} n \otimes^I m}$	$\{(\sigma, n \otimes^I m) \mid \sigma \in \Sigma, (\sigma, n) \in [\![a_1]\!]_{\mathcal{A}}, (\sigma, m) \in [\![a_2]\!]_{\mathcal{A}}\}$
	$\otimes \in \{+, *, -\}$	
a_1 / a_2	$\dfrac{\langle a_1, \sigma \rangle \rightarrow_{Aexp} n \quad \langle a_2, \sigma \rangle \rightarrow_{Aexp} m \quad m \neq 0}{\langle a_1 / a_2, \sigma \rangle \rightarrow_{Aexp} n \div m}$	$\{(\sigma, n \div m) \mid \sigma \in \Sigma, (\sigma, n) \in [\![a_1]\!]_{\mathcal{A}}, (\sigma, m) \in [\![a_2]\!]_{\mathcal{A}}, m \neq 0\}$

Äquivalenz operationale und denotationale Semantik

- Zu zeigen Gleichung (3) von Folie 4:

- Für alle $a \in \textbf{Aexp}$, für alle $n \in \mathbb{Z}$, für alle Zustände σ:

$$\langle a, \sigma \rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a]\!]_{\mathcal{A}}$$

- Beweis Prinzip?

Exkurs: Beweisprinzipien

- Induktion über $\mathbb{N}$ ($\mathrm{nf}(n)$ ist der **Nachfolger** von n):

$$\frac{P(0) \wedge \forall n \in \mathbb{N}.\, P(n) \Longrightarrow P(\mathrm{nf}(n))}{\forall x \in \mathbb{N}.\, P(x)}$$

- Beispiel: Addition ist definiert durch

$$x + 0 = x$$
$$x + \mathrm{nf}(y) = \mathrm{nf}(x + y)$$

- Zeige $x + y = y + x$ durch Induktion über y:
  1. Basis: $x + 0 = 0 + x$
  2. Induktionsschritt: Annahme $x + y = y + x$, dann zeige $x + \mathrm{nf}(y) = \mathrm{nf}(y) + x$.

  - Benötigt Hilfsbeweise $0 + x = x$ und $\mathrm{nf}(x + y) = \mathrm{nf}(x) + y$

Arbeitsblatt 4.1: Natürliche Induktion

- Zeigt durch natürliche Induktion:

$$0 + x = x \qquad \mathrm{nf}(x + y) = \mathrm{nf}(x) + y$$

- Welche Variable benutzt ihr für die Induktion? Was ist der Unterschied?

Wohlfundiertheit

Wohlfundiertheit

Eine binäre Relation $\prec \subseteq S \times S$ ist **wohlfundiert**, wenn es keine unendlich **absteigenden** Ketten gibt

$$\cdots \prec a_3 \prec a_2 \prec a_1$$

Beispiele:

- $(N, \leq)$?

Wohlfundiertheit

Wohlfundiertheit

Eine binäre Relation $\prec \subseteq S \times S$ ist **wohlfundiert**, wenn es keine unendlich **absteigenden** Ketten gibt

$$\cdots \prec a_3 \prec a_2 \prec a_1$$

Beispiele:

- $(N, \leq)$? Nein: $\cdots \leq 1 \leq 1 \leq 1$
- $(\mathbb{N}, <)$?

Wohlfundiertheit

Wohlfundiertheit

Eine binäre Relation $\prec \subseteq S \times S$ ist **wohlfundiert**, wenn es keine unendlich **absteigenden** Ketten gibt

$$\cdots \prec a_3 \prec a_2 \prec a_1$$

Beispiele:

- $(N, \leq)$? Nein: $\cdots \leq 1 \leq 1 \leq 1$
- $(\mathbb{N}, <)$? Ja.
- $(\mathbb{Z}, <)$?

Wohlfundiertheit

Wohlfundiertheit

Eine binäre Relation $\prec \subseteq S \times S$ ist **wohlfundiert**, wenn es keine unendlich **absteigenden** Ketten gibt

$$\cdots \prec a_3 \prec a_2 \prec a_1$$

Beispiele:

- $(N, \leq)$? Nein: $\cdots \leq 1 \leq 1 \leq 1$
- $(\mathbb{N}, <)$? Ja.
- $(\mathbb{Z}, <)$? Nein: $\cdots < -3 < -2 < -1 < 0$
- $(\mathbb{Q}^+, <)$?

Wohlfundiertheit

Wohlfundiertheit

Eine binäre Relation $\prec \subseteq S \times S$ ist **wohlfundiert**, wenn es keine unendlich **absteigenden** Ketten gibt

$$\cdots \prec a_3 \prec a_2 \prec a_1$$

Beispiele:

- $(N, \leq)$? Nein: $\cdots \leq 1 \leq 1 \leq 1$
- $(\mathbb{N}, <)$? Ja.
- $(\mathbb{Z}, <)$? Nein: $\cdots < -3 < -2 < -1 < 0$
- $(\mathbb{Q}^+, <)$? Nein: $\ldots < \frac{1}{n} \ldots < \frac{1}{4} < \frac{1}{3} < \frac{1}{2} < 1$

Eigenschaften wohlfundierter Relationen

- Eine wohlfundierte Relation ist **irreflexiv:** $\forall x \in S. x \nprec x$

Eigenschaften wohlfundierter Relationen

- Eine wohlfundierte Relation ist **irreflexiv:** $\forall x \in S.\, x \not\prec x$
  - Ansonsten gäbe es $\cdots \prec x \prec x \prec x$
- **Lemma**: $\prec$ ist wohlfundiert gdw. jede nicht-leere Untermenge $Q \subseteq S$ ein minimales Element $\min Q$ hat:

$$\min Q \in Q \wedge \forall b.\, b \prec \min Q \Longrightarrow b \notin Q$$

Wohlfundierte Induktion

Noethersche Induktion (Wohlfundierte Induktion)

Sei $\prec \subseteq R \times R$ **wohlfundiert** und P eine Aussage über Elemente von R. Dann gilt

$$\frac{\forall v \in R.\, (\forall u \in R.\, u \prec v \Longrightarrow P(u)) \Longrightarrow P(v)}{\forall x \in R.\, P(x)}$$

Beispiele:

- Mit $S = \mathbb{N}$, $a \prec a + 1$: natürliche Induktion.
- Warum?

Wohlfundierte Induktion

Noethersche Induktion (Wohlfundierte Induktion)

Sei $\prec \subseteq R \times R$ **wohlfundiert** und P eine Aussage über Elemente von R. Dann gilt

$$\frac{\forall v \in R.\, (\forall u \in R.\, u \prec v \Longrightarrow P(u)) \Longrightarrow P(v)}{\forall x \in R.\, P(x)}$$

Beispiele:

- Mit $S = \mathbb{N}$, $a \prec a + 1$: natürliche Induktion.
- Warum? Fallunterscheidung über v: entweder $v = 0$, dann gibt es kein u so dass $u \prec 0$ und die Voraussetzung ist $P(0)$; oder $v = w + 1$, dann $w \prec w + 1$, und die Voraussetzung ist $P(w) \Longrightarrow P(w + 1)$

Strukturelle Ordnung

Strukturelle Ordnung

Die strukturelle Ordnung auf arithmetischen Ausdrücken ist definiert als:

$$\forall a, a' \in \mathbf{Aexp}., a' \prec a \Longleftrightarrow a' \text{ ist Teilausdruck von } a$$

Dabei ist "Teilausdruck" formalisiert als $\otimes \in \{+, *, -, /\}$:

$$a \;\text{ Teilausdruck-von } (a_1 \otimes a_2) \Longleftrightarrow \begin{pmatrix} a = a_1 \vee a \,\text{Teilausdruck-von } a_1 \vee \\ a = a_2 \vee a \,\text{Teilausdruck-von } a_2 \end{pmatrix}$$

- Beispiel für strukturelle Induktion: Rechtseindeutigkeit von $[\![-]\!]_{\mathcal{A}}$ ($\longrightarrow$ Vorlesung 3)

Arbeitsblatt 4.2: Strukturelle Induktion

- **Beweist**, dass die Relation “Teilausdruck-von” wohlfundiert ist.

Äquivalenz operationale und denotationale Semantik

- Für alle $a \in \mathbf{Aexp}$, für alle $n \in \mathbb{Z}$, für alle Zustände σ:

$$\langle a, \sigma \rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a]\!]_{\mathcal{A}}$$

- Beweis Prinzip?

Äquivalenz operationale und denotationale Semantik

- Für alle $a \in \mathbf{Aexp}$, für alle $n \in \mathbb{Z}$, für alle Zustände σ:

$$\langle a, \sigma \rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a]\!]_{\mathcal{A}}$$

- Beweis per struktureller Induktion über a. (Warum?)

Beweis: $\forall a \in \mathbf{Aexp}. \forall n \in \mathbb{Z}. \forall \sigma. \langle a, \sigma \rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a]\!]_{\mathcal{A}}$

Induktionsanfänge

- $a \equiv m \in \mathbf{Z}$:

$$\left.\begin{array}{l} \langle m, \sigma \rangle \rightarrow_{Aexp} [\![m]\!] \\ [\![m]\!]_{\mathcal{A}} = \{(\sigma', [\![m]\!]) \mid \sigma' \in \Sigma\} \Rightarrow (\sigma, [\![m]\!]) \in [\![m]\!]_{\mathcal{A}} \end{array}\right] \Longleftrightarrow$$

- $a \equiv X \in \mathbf{Loc}$:

1. $X \in Dom(\sigma)$:

$$\left.\begin{array}{l} \langle X, \sigma \rangle \rightarrow_{Aexp} \sigma(X) \\ [\![X]\!]_{\mathcal{A}} = \{(\sigma', \sigma'(X)) | \sigma' \in \Sigma, X \in Dom(\sigma')\} \Rightarrow (\sigma, \sigma(X)) \in [\![X]\!]_{\mathcal{A}} \end{array}\right] \Longleftrightarrow$$

2. $X \notin Dom(\sigma)$:

$$\left.\begin{array}{l} \langle X, \sigma \rangle \rightarrow_{Aexp} \perp \\ [\![X]\!]_{\mathcal{A}} = \{(\sigma', \sigma'(X)) | \sigma' \in \Sigma, X \in Dom(\sigma')\} \Rightarrow \sigma \notin Dom([\![X]\!]_{\mathcal{A}}) \end{array}\right] \Longleftrightarrow$$

Beweis: $\forall a \in \textbf{Aexp}.\, \forall n \in \mathbb{Z}.\, \forall \sigma.\, \langle a, \sigma \rangle \to_{Aexp} n \iff (\sigma, n) \in [\![a]\!]_{\mathcal{A}}$

Induktionsschritte

- $a \equiv a_1 + a_2$ — Induktionsannahme: für alle m, n

$$\langle a_1, \sigma \rangle \to_{Aexp} m \iff (\sigma, m) \in [\![a_1]\!]_{\mathcal{A}}$$
$$\langle a_2, \sigma \rangle \to_{Aexp} n \iff (\sigma, n) \in [\![a_2]\!]_{\mathcal{A}}$$

Dann;

$$\langle a_1 + a_2, \sigma \rangle \to_{Aexp} m + n \overset{(\text{Def. } \langle .,. \rangle \to_{Aexp}.)}{\iff} \langle a_1, \sigma \rangle \to_{Aexp} m \overset{\text{IA für } a_1}{\iff} (\sigma, m) \in [\![a_1]\!]_{\mathcal{A}}$$

$$\&\qquad\qquad\qquad\qquad \&$$

$$\langle a_2, \sigma \rangle \to_{Aexp} n \overset{\text{IA für } a_2}{\iff} (\sigma, n) \in [\![a_2]\!]_{\mathcal{A}}$$

$$\Big\Updownarrow (Def.[\![.]\!]_{\mathcal{A}})$$

$$(\sigma, m + n) \in [\![a_1 + a_2]\!]_{\mathcal{A}}$$

Beweis: $\forall a \in \mathbf{Aexp}. \forall n \in \mathbb{Z}. \forall \sigma. \langle a, \sigma \rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a]\!]_{\mathcal{A}}$

Induktionsschritte

- $a \equiv a_1 / a_2$ — Induktionsannahme:

$$\langle a_1, \sigma \rangle \rightarrow_{Aexp} m \Longleftrightarrow (\sigma, m) \in [\![a_1]\!]_{\mathcal{A}}$$
$$\langle a_2, \sigma \rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a_2]\!]_{\mathcal{A}}$$

1. Fall: $n \neq 0$

$$\langle a_1/a_2, \sigma \rangle \rightarrow_{Aexp} m/n \overset{(\text{Def. } \langle .,. \rangle \rightarrow_{Aexp})}{\Longleftrightarrow} \langle a_1, \sigma \rangle \rightarrow_{Aexp} m \overset{\text{IA für } a_1}{\Longleftrightarrow} (\sigma, m) \in [\![a_1]\!]_{\mathcal{A}}$$

$$\&\qquad\qquad\qquad\qquad \&$$

$$\langle a_2, \sigma \rangle \rightarrow_{Aexp} n \overset{\text{IA für } a_2}{\Longleftrightarrow} (\sigma, n) \in [\![a_2]\!]_{\mathcal{A}}$$

$$\Big\Updownarrow (Def. [\![.]\!]_{\mathcal{A}})$$

$$(\sigma, m/n) \in [\![a_1/a_2]\!]_{\mathcal{A}}$$

Induktionsschritte

- $a \equiv a_1 / a_2$ — Induktionsannahme:

$$\langle a_1, \sigma\rangle \rightarrow_{Aexp} m \Longleftrightarrow (\sigma, m) \in [\![a_1]\!]_{\mathcal{A}}$$
$$\langle a_2, \sigma\rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a_2]\!]_{\mathcal{A}}$$

1. Fall: $n = 0$
   Dann gibt es kein v so dass $\langle a_1 / a_2, \sigma\rangle \rightarrow_{Aexp} v$, aber auch $\sigma \notin \mathrm{dom}\, [\![a_1 / a_2]\!]_{\mathcal{A}}$.

q.e.d.

Operationale vs. denotationale Semantik

	Operational $\langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{false} \mid \mathit{true}$	**Denotational** $[\![b]\!]_{\mathcal{B}}$
$\mathbf{1}$	$\langle \mathbf{1}, \sigma \rangle \rightarrow_{Bexp} \mathit{true}$	$\{(\sigma, \mathit{true}) \mid \sigma \in \Sigma\}$
$\mathbf{0}$	$\langle \mathbf{0}, \sigma \rangle \rightarrow_{Bexp} \mathit{false}$	$\{(\sigma, \mathit{false}) \mid \sigma \in \Sigma\}$

Operationale vs. denotationale Semantik

	Operat. $\langle b, \sigma \rangle \to_{Bexp} t$	**Denotational** $[\![b]\!]_{\mathcal{B}}$
$a_0 == a_1$	$$\frac{\langle a_0, \sigma \rangle \to_{Aexp} n \quad \langle a_1, \sigma \rangle \to_{Aexp} m \quad n = m}{\langle a_0 == a_1, \sigma \rangle \to_{Bexp} true}$$ $$\frac{\langle a_0, \sigma \rangle \to_{Aexp} n \quad \langle a_1, \sigma \rangle \to_{Aexp} m \quad n \neq m}{\langle a_0 == a_1, \sigma \rangle \to_{Bexp} false}$$	$\{(\sigma, true) \mid \sigma \in \Sigma, (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}}, (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, n_0 = n_1\}$ $\cup$ $\{(\sigma, false) \mid \sigma \in \Sigma, (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}}, (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}, n_0 \neq n_1\}$
$a1 < a2$		analog

Operationale vs. denotationale Semantik

	Operational $\langle a, \sigma \rangle \rightarrow_{Bexp} b$	**Denotational** $[\![b]\!]_{\mathcal{B}}$
b_1 && b_2	$\dfrac{\langle b_1, \sigma \rangle \rightarrow_{Bexp} \mathit{false}}{\langle b_1 \;\&\&\; b_2, \sigma \rangle \rightarrow \mathit{false}}$	$\{(\sigma, \mathit{false}) \vert (\sigma, \mathit{false}) \in [\![b_1]\!]_{\mathcal{B}}\}$
	$\dfrac{\langle b_1, \sigma \rangle \rightarrow_{Bexp} \mathit{true} \quad \langle b_2, \sigma \rangle \rightarrow_{Bexp} t}{\langle b_1 \;\&\&\; b_2, \sigma \rangle \rightarrow t}$	$\{(\sigma, t) \vert (\sigma, \mathit{true}) \in [\![b_1]\!]_{\mathcal{B}}, (\sigma, t) \in [\![b_2]\!]_{\mathcal{B}}\}$
$b_1 \;\vert\vert\; b_2$		analog
$!n$	$\ldots$	

Äquivalenz operationale und denotationale Semantik

- Zu zeigen Gleichung (2) von Folie 4:
- Für alle $b \in \mathbf{Bexp}$, für alle $t \in \mathbb{B}$, for alle Zustände σ:

$$\langle b, \sigma \rangle \to_{Bexp} t \iff (\sigma, t) \in [\![b]\!]_{\mathcal{B}}$$

- Beweis Prinzip?

Äquivalenz operationale und denotationale Semantik

- Zu zeigen Gleichung (2) von Folie 4:
- Für alle $b \in \mathbf{Bexp}$, für alle $t \in \mathbb{B}$, for alle Zustände σ:

$$\langle b, \sigma \rangle \rightarrow_{Bexp} t \Longleftrightarrow (\sigma, t) \in [\![b]\!]_{\mathcal{B}}$$

- Beweis per struktureller Induktion über b (unter Verwendung der Äquivalenz für AExp). (Warum?)

Beweis $\langle b, \sigma \rangle \rightarrow_{Bexp} t \Longleftrightarrow (\sigma, t) \in [\![b]\!]_{\mathcal{B}}$

Induktionsanfänge

- $b \equiv \mathbf{0}$:

$$\left.\begin{array}{l} \langle \mathbf{0}, \sigma \rangle \rightarrow_{Bexp} \mathit{false} \\ [\![\mathbf{0}]\!]_{\mathcal{B}} = \{(\sigma', \mathit{false}) | \sigma' \in \Sigma\} \Rightarrow (\sigma, \mathit{false}) \in [\![\mathbf{0}]\!]_{\mathcal{B}} \end{array}\right] \Longleftrightarrow$$

- $b \equiv \mathbf{1}$:

$$\left.\begin{array}{l} \langle \mathbf{1}, \sigma \rangle \rightarrow_{Bexp} \mathit{true} \\ [\![\mathbf{1}]\!]_{\mathcal{B}} = \{(\sigma', \mathit{true}) | \sigma' \in \Sigma\} \Rightarrow (\sigma, \mathit{true}) \in [\![\mathbf{1}]\!]_{\mathcal{B}} \end{array}\right] \Longleftrightarrow$$

Beweis $\langle b, \sigma\rangle \to_{Bexp} t \Longleftrightarrow (\sigma, t) \in [\![b]\!]_{\mathcal{B}}$

Induktionsschritte

- $b \equiv b_1 \&\& b_2$ — Induktionsannahme:

$$\langle b_1, \sigma\rangle \to_{Bexp} v \Longleftrightarrow (\sigma, v) \in [\![b_1]\!]_{\mathcal{B}}$$
$$\langle b_2, \sigma\rangle \to_{Bexp} w \Longleftrightarrow (\sigma, w) \in [\![b_2]\!]_{\mathcal{B}}$$

1. Fall $v = false$

$$\langle b_1 \&\& b_2, \sigma\rangle \to_{Bexp} false \overset{(\text{Def. } \langle .,.\rangle \to_{Bexp})}{\Longleftrightarrow} \langle b_1, \sigma\rangle \to_{Bexp} false \overset{\text{IA für } b_1}{\Longleftrightarrow} (\sigma, false) \in [\![b_1]\!]_{\mathcal{B}}$$
$$\Updownarrow \text{Def. } [\![.]\!]_{\mathcal{B}}$$
$$(\sigma, false) \in [\![b_1 \&\& b_2]\!]_{\mathcal{B}}$$

Beweis $\langle b, \sigma \rangle \rightarrow_{Bexp} t \Longleftrightarrow (\sigma, t) \in [\![b]\!]_{\mathcal{B}}$

Induktionsschritte

- $b \equiv b_1 \&\& b_2$ — Induktionsannahme:

$$\langle b_1, \sigma \rangle \rightarrow_{Bexp} v \Longleftrightarrow (\sigma, v) \in [\![b_1]\!]_{\mathcal{B}}$$
$$\langle b_2, \sigma \rangle \rightarrow_{Bexp} w \Longleftrightarrow (\sigma, w) \in [\![b_2]\!]_{\mathcal{B}}$$

1. Fall $v = true$

$$\langle b_1 \&\& b_2, \sigma \rangle \rightarrow_{Bexp} w \overset{(\text{Def. } \langle .,. \rangle \rightarrow_{Bexp})}{\Longleftrightarrow} \langle b_1, \sigma \rangle \rightarrow_{Bexp} true \overset{\text{IA für } b_1}{\Longleftrightarrow} (\sigma, true) \in [\![b_1]\!]_{\mathcal{B}}$$

$$\&\qquad\qquad\qquad\&$$

$$\langle b_2, \sigma \rangle \rightarrow_{Bexp} w \overset{\text{IA für } b_2}{\Longleftrightarrow} (\sigma, w) \in [\![b_2]\!]_{\mathcal{B}}$$

$$\Big\Updownarrow \text{ Def. } [\![.]\!]_{\mathcal{B}}$$

$$(\sigma, w) \in [\![b_1 \&\& b_2]\!]_{\mathcal{B}}$$

q.e.d.

Arbeitsblatt 4.3: Beweis Induktionsanfang

$\langle a_1 == a_2, \sigma \rangle \to_{Bexp} v \Longleftrightarrow (\sigma, v) \in [\![a_1 == a_2]\!]_{\mathcal{B}}$

Beweist obige Aussage unter Verwendung des für arithmetische Ausdrücke geltenden Lemmas

$$\forall a \in \mathbf{Aexp}. \forall n \in \mathbb{Z}. \forall \sigma. \langle a, \sigma \rangle \to_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a]\!]_{\mathcal{A}}$$

1. Was sind die Annahmen?

2. Welche Fälle unterscheiden wir?

Beweis $\langle a_1 == a_2, \sigma\rangle \rightarrow_{Bexp} v \Longleftrightarrow (\sigma, v) \in [\![a_1 == a_2]\!]_{\mathcal{B}}$

- Annahmen: für $n, m \in \mathbb{B}$:

$$\langle a_1, \sigma\rangle \rightarrow_{Aexp} m \Longleftrightarrow (\sigma, m) \in [\![a_1]\!]_{\mathcal{B}}$$
$$\langle a_2, \sigma\rangle \rightarrow_{Aexp} n \Longleftrightarrow (\sigma, n) \in [\![a_2]\!]_{\mathcal{B}}$$

- 1. Fall: $v = true\ (m = n)$

$$\begin{array}{ccccc}
\langle a_1 == a_2, \sigma\rangle \rightarrow_{Bexp} true & \overset{(\text{Def. } \langle .,. \rangle \rightarrow_{Bexp} .)}{\Longleftrightarrow} & \langle a_1, \sigma\rangle \rightarrow_{Bexp} m & \overset{\text{Annahme für } a_1}{\Longleftrightarrow} & (\sigma, m) \in [\![a_1]\!]_{\mathcal{A}} \\
 & & \& & & \& \\
 & & \langle a_2, \sigma\rangle \rightarrow_{Bexp} m & \overset{\text{Annahme für } a_2}{\Longleftrightarrow} & (\sigma, m) \in [\![a_2]\!]_{\mathcal{A}} \\
 & & & & \text{Def. } [\![.]\!]_{\mathcal{B}} \Big\Updownarrow \\
 & & & & (\sigma, true)[\![a_1 == a_2]\!]_{\mathcal{B}}
\end{array}$$

Beweis $\langle a_1 == a_2, \sigma\rangle \rightarrow_{Bexp} v \iff (\sigma, v) \in [\![a_1 == a_2]\!]_{\mathcal{B}}$

- Annahmen: für $m, n \in \mathbb{B}$:

$$\langle a_1, \sigma\rangle \rightarrow_{Aexp} m \iff (\sigma, m) \in [\![a_1]\!]_{\mathcal{B}}$$
$$\langle a_2, \sigma\rangle \rightarrow_{Aexp} n \iff (\sigma, n) \in [\![a_2]\!]_{\mathcal{B}}$$

- 2. Fall: $v = \mathit{false}\ (m \neq n)$

$$\langle a_1 == a_2, \sigma\rangle \rightarrow_{Bexp} \mathit{false} \overset{(\text{Def. } \langle .,. \rangle \rightarrow_{Bexp}.)}{\iff} \langle a_1, \sigma\rangle \rightarrow_{Aexp} m \overset{\text{Annahme für } a_1}{\iff} (\sigma, m) \in [\![a_1]\!]_{\mathcal{A}}$$

$$\&\qquad\qquad\qquad\qquad\&$$

$$\langle a_2, \sigma\rangle \rightarrow_{Aexp} n \overset{\text{Annahme für } a_2}{\iff} (\sigma, n) \in [\![a_2]\!]_{\mathcal{A}}$$

$$\Big\Updownarrow \text{Def. } [\![.]\!]_{\mathcal{B}}$$

$$(\sigma, \mathit{false})[\![a_1 == a_2]\!]_{\mathcal{B}}$$

Operationale vs. denotationale Semantik

	Operational $\langle c, \sigma\rangle \to_{Stmt} \sigma'$	**Denotational** $[\![c]\!]_{\mathcal{C}}$
$\{\,\}$	$\dfrac{}{\langle \{\}, \sigma\rangle \to_{Stmt} \sigma}$	$[\![\{\,\}]\!]_{\mathcal{C}} = Id$
$c_1; c_2$	$\dfrac{\langle c_1, \sigma\rangle \to_{Stmt} \sigma' \qquad \langle c_2, \sigma'\rangle \to_{Stmt} \sigma''}{\langle c_1; c_2, \sigma\rangle \to_{Stmt} \sigma''}$	$[\![c_1]\!]_{\mathcal{C}} \circ [\![c_2]\!]_{\mathcal{C}}$
$x = a$	$\dfrac{\langle a, \sigma\rangle \to_{Aexp} n}{\langle x = a, \sigma\rangle \to_{Stmt} \sigma[x \mapsto n]}$	$\{(\sigma, \sigma[x \mapsto n]) \mid (\sigma, n) \in [\![a]\!]_{\mathcal{A}}\}$

Operationale vs. denotationale Semantik

	Operational $\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$	**Denotational** $[\![c]\!]_{\mathcal{C}}$
if (b) c_0	$\dfrac{\langle b, \sigma \rangle \rightarrow_{Bexp} true \quad \langle c_0, \sigma \rangle \rightarrow_{Stmt} \sigma'}{\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'}$	$\{(\sigma, \sigma') \mid (\sigma, true) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c_0]\!]_{\mathcal{C}}\}$
else c_1	$\dfrac{\langle b, \sigma \rangle \rightarrow_{Bexp} false \quad \langle c_1, \sigma \rangle \rightarrow_{Stmt} \sigma'}{\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'}$	$\{(\sigma, \sigma') \mid (\sigma, false) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}\}$

Operationale vs. denotationale Semantik

Operational $\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

Denotational $[\![c]\!]_{\mathcal{C}}$

$\underbrace{\textbf{while}\ (b)\ c}_{w}$

$$\frac{\langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{false}}{\langle w, \sigma \rangle \rightarrow_{Stmt} \sigma}$$

$$\mathit{fix}(\Gamma)$$

$$\frac{\langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{true} \quad \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \quad \langle w, \sigma' \rangle \rightarrow_{Stmt} \sigma''}{\langle w, \sigma \rangle \rightarrow_{Stmt} \sigma''}$$

mit

$$\begin{aligned} \Gamma(\varphi) \quad = \quad & \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ \varphi\} \\ & \cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\} \end{aligned}$$

Äquivalenz operationale und denotationale Semantik

- Zu zeigen Gleichung (1) von Folie 4:
- Für alle $c \in \textbf{Stmt}$, für alle Zustände σ, σ':

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longleftrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$$

- $\Longrightarrow$ Beweis Prinzip?
- $\Longleftarrow$ Beweis Prinzip?

Operationale Semantik: C0 Programme

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

Regeln:

$$\frac{}{\langle \{\,\}, \sigma\rangle \rightarrow_{Stmt} \sigma} \qquad \frac{\langle a, \sigma\rangle \rightarrow_{Aexp} n \in \mathbb{Z}}{\langle x = a, \sigma\rangle \rightarrow_{Stmt} \sigma[x \mapsto n]} \qquad \frac{\langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma' \qquad \langle c_2, \sigma'\rangle \rightarrow_{Stmt} \sigma''}{\langle c_1; c_2, \sigma\rangle \rightarrow_{Stmt} \sigma''}$$

$$\frac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{true} \qquad \langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma'}{\langle \mathbf{if}\ (b)\ c_1\ \mathbf{else}\ c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'} \qquad \frac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{false} \qquad \langle c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'}{\langle \mathbf{if}\ (b)\ c_1\ \mathbf{else}\ c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'}$$

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

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

Operationale Semantik: C0 Programme

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

Regeln: **Programmstruktur**

$$\frac{\langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma' \qquad \langle c_2, \sigma'\rangle \rightarrow_{Stmt} \sigma''}{\langle c_1; c_2, \sigma\rangle \rightarrow_{Stmt} \sigma''} \qquad \searrow$$

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

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

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

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

Operationale Semantik: C0 Programme

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

Regeln: **Programmstruktur**

$$\frac{\langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma' \qquad \langle c_2, \sigma'\rangle \rightarrow_{Stmt} \sigma''}{\langle c_1; c_2, \sigma\rangle \rightarrow_{Stmt} \sigma''} \qquad \searrow$$

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

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

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

Strukturelle Induktion über c **nicht** möglich.

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

Ableitungstiefe für Programme

- Die Ableitungstiefe einer Programmauswertung mittels Regeln der operationaler Semantik ist die **Anzahl der Regelanwendungen** mit Conclusion der Form $\langle ., . \rangle \rightarrow_{Stmt} .$.

$$\frac{\begin{array}{c}\vdots \\ \mathit{Prämisse}_1\end{array} \quad \cdots \quad \begin{array}{c}\vdots \\ \mathit{Prämisse}_n\end{array}}{\mathit{Conclusion}}$$

Operationale Semantik: C0 Programme

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

Regeln: **Programmstruktur**

$$\frac{\langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma' \quad \langle c_2, \sigma'\rangle \rightarrow_{Stmt} \sigma''}{\langle c_1; c_2, \sigma\rangle \rightarrow_{Stmt} \sigma''} \qquad \searrow$$

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

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

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

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

Operationale Semantik: C0 Programme

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

Regeln:	Programmstruktur	Ableitungstiefe
$\dfrac{\langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma' \quad \langle c_2, \sigma'\rangle \rightarrow_{Stmt} \sigma''}{\langle c_1; c_2, \sigma\rangle \rightarrow_{Stmt} \sigma''}$	$\searrow$	$\searrow$
$\dfrac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{true} \qquad \langle c_1, \sigma\rangle \rightarrow_{Stmt} \sigma'}{\langle \textbf{if } (b)\ c_1 \textbf{ else } c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'}$	$\searrow$	$\searrow$
$\dfrac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{false} \qquad \langle c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'}{\langle \textbf{if } (b)\ c_1 \textbf{ else } c_2, \sigma\rangle \rightarrow_{Stmt} \sigma'}$	$\searrow$	$\searrow$
$\dfrac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{true} \qquad \langle c, \sigma\rangle \rightarrow_{Stmt} \sigma' \quad \langle \textbf{while } (b)\ c, \sigma'\rangle \rightarrow_{Stmt} \sigma''}{\langle \textbf{while } (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma''}$	$\rightarrow$	$\searrow$
$\dfrac{\langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{false}}{\langle \textbf{while } (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma}$	$\searrow$	$\searrow$

Äquivalenz operationale und denotationale Semantik

- Für alle $c \in \mathbf{Stmt}$, für alle Zustände σ, σ':

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longleftrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$$

- $\Longrightarrow$ Beweis Prinzip?
- $\Longleftarrow$ Beweis Prinzip?

Äquivalenz operationale und denotationale Semantik

- Für alle $c \in \textbf{Stmt}$, für alle Zustände σ, σ':

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longleftrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$$

- $\Longrightarrow$ Beweis per Induktion über **die (Tiefe der) Ableitung** in der operationalen Semantik (Warum?)

- $\Longleftarrow$ Beweis Prinzip?

Beweis: $\forall c \in \textbf{Stmt}. \forall \sigma, \sigma'. \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$

Induktionsanfang — Ableitungstiefe 1

- Fall $c \equiv x = a$:

$$[\![x = a]\!]_{\mathcal{C}} = \{(\sigma, \sigma[x \mapsto m]) | (\sigma, m) \in [\![a]\!]_{\mathcal{A}}\}$$

Sei $\langle a, \sigma \rangle \rightarrow_{Aexp} m \in \mathbb{Z}$:

$$\begin{tikzcd}
\langle x = a, \sigma \rangle \rightarrow_{Stmt} \sigma[x \mapsto m] \arrow[d, Leftrightarrow, "\text{(Def. } \langle .,. \rangle \rightarrow_{Stmt} .\text{)}"] & \\
\langle a, \sigma \rangle \rightarrow_{Aexp} m \in \mathbb{Z} \arrow[r, Leftrightarrow, "\text{Lemma für } a"] & (\sigma, m) \in [\![a]\!]_{\mathcal{A}} \arrow[d, Rightarrow, "\text{Def. } [\![.]\!]_{\mathcal{C}}"'] \\
& (\sigma, \sigma[x \mapsto m]) \in [\![x = a]\!]_{\mathcal{C}}
\end{tikzcd}$$

Beweis: $\forall c \in \textbf{Stmt}.\, \forall \sigma, \sigma'.\, \langle c, \sigma\rangle \rightarrow_{Stmt} \sigma' \Longrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$

Induktionsanfang — Ableitungstiefe 1

- Fall $c \equiv x = a$:

$$[\![x = a]\!]_{\mathcal{C}} = \{(\sigma, \sigma[x \mapsto m]) | (\sigma, m) \in [\![a]\!]_{\mathcal{A}}\}$$

Sei $\langle a, \sigma\rangle \rightarrow_{Aexp} m \in \mathbb{Z}$:

$$\begin{tikzcd}
\langle x = a, \sigma\rangle \rightarrow_{Stmt} \sigma[x \mapsto m] \arrow[d, Leftrightarrow, "(\text{Def. } \langle .,. \rangle \rightarrow_{Stmt} .)"] & \\
\langle a, \sigma\rangle \rightarrow_{Aexp} m \in \mathbb{Z} \arrow[r, Leftrightarrow, "\text{Lemma für } a"] & (\sigma, m) \in [\![a]\!]_{\mathcal{A}} \arrow[d, Rightarrow, "\text{Def. } [\![.]\!]_{\mathcal{C}}"'] \\
& (\sigma, \sigma[x \mapsto m]) \in [\![x = a]\!]_{\mathcal{C}}
\end{tikzcd}$$

- Fall $c \equiv \{\}$: ...

Beweis: $\forall c \in \mathbf{Stmt}.\, \forall \sigma, \sigma'.\, \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$

Induktionsschritt:

- Fall $c \equiv \mathbf{if}(b)\ c_1\ \mathbf{else}\ c_2$:

$$[\![\mathbf{if}(b)\ c_1\ \mathbf{else}\ c_2]\!]_{\mathcal{C}} = \{(\sigma, \sigma') | (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}, (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}\}$$
$$\cup \{(\sigma, \sigma') | (\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}}, (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}$$

  - Fall $\langle \sigma, b \rangle \rightarrow_{Bexp} \mathit{true}$ mit $\langle c_1, \sigma \rangle \rightarrow_{Stmt} \sigma'$:

$$\langle \mathbf{if}(b)\ c_1\ \mathbf{else}\ c_2, \sigma \rangle \rightarrow_{Stmt} \sigma' \overset{(\text{Def. } \langle .,. \rangle \rightarrow_{Stmt} .)}{\Longleftrightarrow} \langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{true} \overset{\text{Lemma für } b}{\Longleftrightarrow} (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}$$

$$\&\qquad\qquad\qquad\qquad\&$$

$$\langle c_1, \sigma \rangle \rightarrow_{Stmt} \sigma' \xRightarrow{\text{IH für } c_1} (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}$$

$$\Big\Downarrow\ \text{Def. } [\![.]\!]_{\mathcal{C}}$$

$$(\sigma, \sigma') \in [\![\mathbf{if}(b)\ c_1\ \mathbf{else}\ c_2]\!]_{\mathcal{C}}$$

Beweis: $\forall c \in \textbf{Stmt}.\, \forall \sigma, \sigma'.\, \langle c, \sigma \rangle \to_{Stmt} \sigma' \Longrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$

Induktionsschritt:

- Fall $c \equiv \textbf{if}(b)\ c_1\ \textbf{else}\ c_2$:

$$
\begin{aligned}
[\![\textbf{if}(b)\ c_1\ \textbf{else}\ c_2]\!]_{\mathcal{C}} = & \{(\sigma, \sigma') | (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}, (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}\} \\
& \cup \{(\sigma, \sigma') | (\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}}, (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

  - Fall $\langle \sigma, b \rangle \to_{Bexp} \mathit{false}$ mit $\langle c_2, \sigma \rangle \to_{Stmt} \sigma'$:

$$
\begin{array}{ccccc}
\langle \textbf{if}(b)\ c_1\ \textbf{else}\ c_2, \sigma \rangle \to_{Stmt} \sigma' & \overset{\text{(Def. } \langle .,. \rangle \to_{Stmt} .)}{\Longleftrightarrow} & \langle b, \sigma \rangle \to_{Bexp} \mathit{false} & \overset{\text{Lemma für } b}{\Longleftrightarrow} & (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \\
 & & \& & & \& \\
 & & \langle c_2, \sigma \rangle \to_{Stmt} \sigma' & \xrightarrow{\text{IH für } c_2} & (\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}} \\
 & & & & \Big\Downarrow \text{Def. } [\![.]\!]_{\mathcal{C}} \\
 & & & & (\sigma, \sigma') \in [\![\textbf{if}(b)\ c_1\ \textbf{else}\ c_2]\!]_{\mathcal{C}}
\end{array}
$$

Beweis: $\forall c \in \mathbf{Stmt}. \forall \sigma, \sigma'. \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$

Induktionsschritt:

- Fall $c \equiv \mathbf{while}(b)\ c$: $[\![\mathbf{while}(b)\ c]\!]_{\mathcal{C}} = \mathit{fix}(\Gamma)$
  - Fall $\langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{true}$ mit $\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$, $\langle \mathbf{while}(b)\ c, \sigma' \rangle \rightarrow_{Stmt} \sigma''$

$$\langle \mathbf{while}(b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma'' \xLeftrightarrow{(\text{Def. } \langle .,. \rangle \rightarrow_{Stmt} .)} \langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{true} \xLeftrightarrow{\text{Lemma für } b} (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}$$

$$\&\qquad\qquad\&$$

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \xRightarrow{\text{IH für } \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'} (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$$

$$\&\qquad\qquad\&$$

$$\langle \mathbf{while}(b)\ c, \sigma' \rangle \rightarrow_{Stmt} \sigma'' \xRightarrow{\text{IH für } \langle \mathbf{while}(b)\ c, \sigma' \rangle \rightarrow_{Stmt} \sigma''} (\sigma', \sigma'') \in [\![\mathbf{while}(b)\ c]\!]_{\mathcal{C}}$$

$$\Big\Downarrow \text{Def. } [\![.]\!]_{\mathcal{C}} \text{ \& Fixpunkt Eigenschaft}$$

$$(\sigma, \sigma'') \in [\![\mathbf{while}(b)\ c]\!]_{\mathcal{C}}$$

Beweis: $\forall c \in \mathbf{Stmt}.\, \forall \sigma, \sigma'.\, \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$

Induktionsschritt:

- Fall $c \equiv \mathbf{while}(b)\ c$: $\qquad [\![\mathbf{while}(b)\ c]\!]_{\mathcal{C}} = \mathit{fix}(\Gamma)$
  - Fall $\langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{false}$, $\langle \mathbf{while}(b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma$

$$\langle \mathbf{while}(b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma \xLeftrightarrow{(\text{Def. } \langle .,. \rangle \rightarrow_{Stmt})} \langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{false} \xLeftrightarrow{\text{Lemma für } b} (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}$$

$$\Big\Downarrow \text{Def. } [\![.]\!]_{\mathcal{C}}$$

$$(\sigma, \sigma) \in [\![\mathbf{while}(b)\ c]\!]_{\mathcal{C}}$$

Äquivalenz operationale und denotationale Semantik

- Für alle $c \in \mathbf{Stmt}$, für alle Zustände σ, σ':

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \Longleftrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$$

- $\Longrightarrow$ Beweis per Induktion über **die (Tiefe der) Ableitung** in der operationalen Semantik (Warum?)

- $\Longleftarrow$ Beweis Prinzip?

Äquivalenz operationale und denotationale Semantik

- Für alle $c \in \mathbf{Stmt}$, für alle Zustände σ, σ':

$$\langle c, \sigma \rangle \rightarrow_{Stmt} \sigma' \iff (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$$

- $\Longrightarrow$ Beweis per Induktion über **die (Tiefe der) Ableitung** in der operationalen Semantik (Warum?)
- $\Longleftarrow$ Beweis per struktureller Induktion über c (Verwendung der Äquivalenz für arithmetische und boolsche Ausdrücke). Für die While-Schleife Rückgriff auf Definition des Fixpunkts und Induktion über die Teilmengen $\Gamma^i(\emptyset)$ des Fixpunkts. (Warum?)

Beweis: $\forall c \in \textbf{Stmt}. \forall \sigma, \sigma'. (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

Induktionsanfang:

- Fall $c \equiv x = a$:

$$[\![x = a]\!]_{\mathcal{C}} = \{(\sigma, \sigma[x \mapsto t]) | (\sigma, t) \in [\![a]\!]_{\mathcal{A}}\}$$

$$(\sigma, \sigma[x \mapsto t]) \in [\![x = a]\!]_{\mathcal{C}} \wedge \underbrace{(\sigma, t) \in [\![a]\!]_{\mathcal{A}}}$$

$$\stackrel{\text{Lemma } \textbf{Aexp}}{\Longrightarrow} \qquad \langle a, \sigma \rangle \rightarrow_{Aexp} t$$

$$\stackrel{\text{Def. } \langle .,. \rangle \rightarrow_{Stmt} \cdot}{\Longrightarrow} \langle x = a, \sigma \rangle \rightarrow_{Stmt} \sigma[x \mapsto t]$$

- Fall $c \equiv \{\,\}$

$$[\![\{\,\}]\!]_{\mathcal{C}} = \{(\sigma, \sigma) | \sigma \in \Sigma\}$$

$$(\sigma, \sigma) \in [\![\{\,\}]\!]_{\mathcal{C}}$$

$$\stackrel{\text{Def. } \langle .,. \rangle \rightarrow_{Stmt} \cdot}{\Longrightarrow} \langle \{\,\}, \sigma \rangle \rightarrow_{Stmt} \sigma$$

Beweis: $\forall c \in \textbf{Stmt}. \forall \sigma, \sigma'. (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

Induktionsschritt:

- Fall **if** (b) c_1 **else** c_2:

$$[\![\textbf{if } (b)\ c_1 \textbf{ else } c_2]\!]_{\mathcal{C}} = \begin{array}{l} \{(\sigma, \sigma') | (\sigma, true) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}\} \\ \cup \{(\sigma, \sigma') | (\sigma, false) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}}\} \end{array}$$

Induktionsannahme gilt für c_1 und c_2

- Fall: $(\sigma, true) \in [\![b]\!]_{\mathcal{B}}$ mit $(\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}$

$$\begin{array}{rl} & (\sigma, true) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}} \\ \overset{\text{Lemma } \textbf{Bexp}}{\Longrightarrow} & \langle b, \sigma \rangle \rightarrow_{Bexp} true \wedge (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}} \\ \overset{\text{IA für } c_1}{\Longrightarrow} & \langle b, \sigma \rangle \rightarrow_{Bexp} true \wedge \langle c_1, \sigma \rangle \rightarrow_{Stmt} \sigma' \\ \overset{\text{Def. } \langle .,. \rangle \rightarrow_{Stmt} .}{\Longrightarrow} & \langle \textbf{if } (b)\ c_1 \textbf{ else } c_2, \sigma \rangle \rightarrow_{Stmt} \sigma' \end{array}$$

Beweis: $\forall c \in \textbf{Stmt}. \forall \sigma, \sigma'. (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

Induktionsschritt:

- Fall **if** (b) c_1 **else** c_2:

$$\begin{aligned}[\![\textbf{if } (b)\ c_1 \textbf{ else } c_2]\!]_{\mathcal{C}} = \; & \{(\sigma, \sigma') | (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c_1]\!]_{\mathcal{C}}\} \\ & \cup \{(\sigma, \sigma') | (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}}\}\end{aligned}$$

Induktionsannahme gilt für c_1 und c_2

- Fall: $(\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}$ mit $(\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}}$

$$\begin{aligned} & (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}} \\ \overset{\text{Lemma } \textbf{Bexp}}{\Longrightarrow} \quad & \langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{false} \wedge (\sigma, \sigma') \in [\![c_2]\!]_{\mathcal{C}} \\ \overset{\text{IA für } c_2}{\Longrightarrow} \quad & \langle b, \sigma \rangle \rightarrow_{Bexp} \mathit{false} \wedge \langle c_2, \sigma \rangle \rightarrow_{Stmt} \sigma' \\ \overset{\text{Def. } \langle .,. \rangle \rightarrow_{Stmt} .}{\Longrightarrow} \quad & \langle \textbf{if } (b)\ c_1 \textbf{ else } c_2, \sigma \rangle \rightarrow_{Stmt} \sigma' \end{aligned}$$

Beweis: $\forall c \in \textbf{Stmt}.\, \forall \sigma, \sigma'.\, (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \langle c, \sigma\rangle \rightarrow_{Stmt} \sigma'$

Induktionsschritt:

- Fall **while** (b) c

$$
\begin{aligned}
[\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\
\text{mit } \Gamma(s) =& \{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ s\} \\
&\cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

Induktionsannahme gilt für c

$$
\begin{aligned}
(\sigma, \sigma') \in [\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &\Longrightarrow (\sigma, \sigma') \in \mathit{fix}(\Gamma) && \text{nach Def. } [\![.]\!]_{\mathcal{C}} \\
&\Longrightarrow (\sigma, \sigma') \in \bigcup_{i \in \mathbb{N}} \Gamma^i(\emptyset) && \text{nach Def. } \mathit{fix}(\Gamma) \\
&\Longrightarrow (\sigma, \sigma') \in \Gamma^i(\emptyset) \text{ für ein } i \in \mathbb{N}
\end{aligned}
$$

Beweis: $\forall c \in \textbf{Stmt}. \forall \sigma, \sigma'. (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \langle c, \sigma \rangle \to_{Stmt} \sigma'$

Induktionsschritt:

- Fall **while** (b) c

$$
\begin{aligned}
[\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &= fix(\Gamma) \\
\text{mit } \Gamma(s) &= \{(\sigma, \sigma') \mid (\sigma, true) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ s\} \\
&\quad \cup \{(\sigma, \sigma) \mid (\sigma, false) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

Induktionsannahme gilt für c

$$
\begin{aligned}
(\sigma, \sigma') \in [\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &\Longrightarrow (\sigma, \sigma') \in fix(\Gamma) && \text{nach Def. } [\![.]\!]_{\mathcal{C}} \\
&\Longrightarrow (\sigma, \sigma') \in \bigcup_{i \in \mathbb{N}} \Gamma^i(\emptyset) && \text{nach Def. } fix(\Gamma) \\
&\Longrightarrow (\sigma, \sigma') \in \Gamma^i(\emptyset) \text{ für ein } i \in \mathbb{N}
\end{aligned}
$$

Unterbeweis: $\forall i \in \mathbb{N}. (\sigma, \sigma') \in \Gamma^i(\emptyset) \Rightarrow \langle \textbf{while}\ (b)\ c, \sigma \rangle \to_{Stmt} \sigma' \quad \text{(UB)}$

Beweis: $\forall c \in \textbf{Stmt}.\, \forall \sigma, \sigma'.\, (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

Induktionsschritt:

- Fall **while** (b) c

$$
\begin{aligned}
[\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\
\text{mit } \Gamma(s) = &\{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ s\} \\
&\cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

Induktionsannahme gilt für c

$$
\begin{aligned}
(\sigma, \sigma') \in [\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &\Longrightarrow (\sigma, \sigma') \in \mathit{fix}(\Gamma) && \text{nach Def. } [\![.]\!]_{\mathcal{C}} \\
&\Longrightarrow (\sigma, \sigma') \in \bigcup_{i \in \mathbb{N}} \Gamma^i(\emptyset) && \text{nach Def. } \mathit{fix}(\Gamma) \\
&\Longrightarrow (\sigma, \sigma') \in \Gamma^i(\emptyset) \text{ für ein } i \in \mathbb{N} \\
&\Longrightarrow \langle \textbf{while}\ (b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma' && \text{nach (UB)}
\end{aligned}
$$

Unterbeweis: $\forall i \in \mathbb{N}. (\sigma, \sigma') \in \Gamma^i(\emptyset) \Rightarrow \langle \textbf{while}\ (b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma' \quad \text{(UB)}$

Unterbeweis: $\forall i \in \mathbb{N}.(\sigma, \sigma') \in \Gamma^i(\emptyset) \Longrightarrow \langle \mathbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma'$

Es gilt die Induktionsannahme für c:

$$\forall \rho, \rho'.(\rho, \rho') \in [\![c]\!]_{\mathcal{C}} \Rightarrow \langle c, \rho\rangle \rightarrow_{Stmt} \rho' \qquad (*)$$

Beweis per Induktion über i:

- Induktionsanfang $i = 0$:

$$(\sigma, \sigma') \in \underbrace{\Gamma^0(\emptyset)}_{\emptyset} \Longrightarrow (\sigma, \sigma') \in \emptyset \Longrightarrow \mathit{false}$$

Implikation trivialerweise erfüllt da $\mathit{false} \Longrightarrow P$ immer wahr

Unterbeweis: $\forall i \in \mathbb{N}.(\sigma, \sigma') \in \Gamma^i(\emptyset) \Longrightarrow \langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma'$

Es gilt die Induktionsannahme für c:

$$\forall \rho, \rho'.(\rho, \rho') \in [\![c]\!]_{\mathcal{C}} \Rightarrow \langle c, \rho\rangle \rightarrow_{Stmt} \rho' \qquad (*)$$

Beweis per Induktion über i:

- Induktionsschritt $i \rightarrow i + 1$:
- Induktionsannahme (UB) gilt für i

$$\begin{aligned}(\sigma, \sigma') \in \Gamma^{i+1}(\emptyset) &\Longrightarrow (\sigma, \sigma') \in \Gamma(\Gamma^i(\emptyset))\\ &\overset{\text{Def. }\Gamma}{\Longrightarrow} (\sigma, \sigma') \in \ \{(\sigma, \sigma'') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}}, (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}, (\sigma', \sigma'') \in \Gamma^i(\emptyset)\}\\ &\qquad\qquad \cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}\end{aligned}$$

- Fallunterscheidung über Zugehörigkeit zur Teilmenge

Unterbeweis: $\forall i \in \mathbb{N}.(\sigma, \sigma') \in \Gamma^i(\emptyset) \Longrightarrow \langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma'$

Es gilt die Induktionsannahme für c:

$$\forall \rho, \rho'.(\rho, \rho') \in [\![c]\!]_{\mathcal{C}} \Rightarrow \langle c, \rho\rangle \rightarrow_{Stmt} \rho' \qquad (*)$$

Beweis per Induktion über i:

- Induktionsschritt $i \rightarrow i+1$:
- Induktionsannahme (UB) gilt für i
- Fall $(\sigma, true) \in [\![b]\!]_{\mathcal{B}}$ mit $(\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}, (\sigma', \sigma'') \in \Gamma^i(\emptyset)$

$$\begin{aligned}(\sigma, \sigma'') \in \Gamma(\Gamma^i(\emptyset)) &\Longrightarrow \underbrace{(\sigma, true) \in [\![b]\!]_{\mathcal{B}}}_{\text{Lemma } \textbf{Bexp}} \wedge \underbrace{(\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}}_{\text{IA } (*)} \wedge \underbrace{(\sigma', \sigma'') \in \Gamma^i(\emptyset)}_{\text{IA (UB) für } i} \\ &\Longrightarrow \langle b, \sigma\rangle \rightarrow_{Bexp} true \wedge \langle c, \sigma\rangle \rightarrow_{Stmt} \sigma' \wedge \langle \textbf{while}\ (b)\ c, \sigma'\rangle \rightarrow_{Stmt} \sigma'' \\ &\Longrightarrow \langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma''\end{aligned}$$

Unterbeweis: $\forall i \in \mathbb{N}.(\sigma, \sigma') \in \Gamma^i(\emptyset) \Longrightarrow \langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma'$

Es gilt die Induktionsannahme für c:

$$\forall \rho, \rho'.(\rho, \rho') \in [\![c]\!]_{\mathcal{C}} \Rightarrow \langle c, \rho\rangle \rightarrow_{Stmt} \rho' \qquad (*)$$

Beweis per Induktion über i:

- Induktionsschritt $i \rightarrow i+1$:
- Induktionsannahme (UB) gilt für i
- Fall $(\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}$

$$\begin{aligned}
(\sigma, \sigma') \in \Gamma(\Gamma^i(\emptyset)) &\Longrightarrow (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}} \wedge \sigma' = \sigma \\
&\Longrightarrow \langle b, \sigma\rangle \rightarrow_{Bexp} \mathit{false} \wedge \sigma' = \sigma && \text{Lemma für } \mathbf{Bexp} \\
&\Longrightarrow \langle \textbf{while}\ (b)\ c, \sigma\rangle \rightarrow_{Stmt} \sigma (= \sigma') && \square
\end{aligned}$$

Beweis: $\forall c \in \textbf{Stmt}. \forall \sigma, \sigma'. (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \langle c, \sigma \rangle \rightarrow_{Stmt} \sigma'$

Induktionsschritt:

- Fall **while** $(b)\ c$

$$
\begin{aligned}
[\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &= \mathit{fix}(\Gamma) \\
\text{mit } \Gamma(s) = &\{(\sigma, \sigma') \mid (\sigma, \mathit{true}) \in [\![b]\!]_{\mathcal{B}} \wedge (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}} \circ s\} \\
&\cup \{(\sigma, \sigma) \mid (\sigma, \mathit{false}) \in [\![b]\!]_{\mathcal{B}}\}
\end{aligned}
$$

Induktionsannahme gilt für c

$$
\begin{aligned}
(\sigma, \sigma') \in [\![\textbf{while}\ (b)\ c]\!]_{\mathcal{C}} &\Longrightarrow (\sigma, \sigma') \in \mathit{fix}(\Gamma) && \text{nach Def. } [\![.]\!]_{\mathcal{C}} \\
&\Longrightarrow (\sigma, \sigma') \in \bigcup_{i \in \mathbb{N}} \Gamma^i(\emptyset) && \text{nach Def. } \mathit{fix}(\Gamma) \\
&\Longrightarrow (\sigma, \sigma') \in \Gamma^i(\emptyset) \text{ für ein } i \in \mathbb{N} \\
&\Longrightarrow \langle \textbf{while}\ (b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma' && \text{nach (UB)}
\end{aligned}
$$

Unterbeweis: $\forall i \in \mathbb{N}. (\sigma, \sigma') \in \Gamma^i(\emptyset) \Rightarrow \langle \textbf{while}\ (b)\ c, \sigma \rangle \rightarrow_{Stmt} \sigma'$ (UB)

Zusammenfassung: Äquivalenz der Semantiken

- Wir haben gezeigt: für alle $c \in \mathbf{Stmt}$, für alle Zustände σ, σ'

$$\langle c, \sigma \rangle \to_{Stmt} \sigma' \Longleftrightarrow (\sigma, \sigma') \in [\![c]\!]_{\mathcal{C}}$$

- Das ist äquivalent zu (für alle $c \in \mathbf{Stmt}$, für alle Zustände σ, σ'):

$$[\![c]\!]_{\mathcal{C}} = \{(\sigma, \sigma') \mid \langle c, \sigma \rangle \to_{Stmt} \sigma'\}$$

- Insbesondere ist die Undefiniertheit gleich:
  wenn es keine Ableitung für c, σ gibt, dann ist auch $\sigma \notin Dom([\![c]\!]_{\mathcal{C}})$.

Fahrplan

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

Korrekte Software: Grundlagen und Methoden
Vorlesung 5 vom 02.05.24
Die Floyd-Hoare-Logik

Serge Autexier, Christoph Lüth

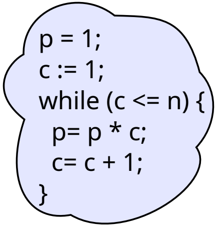
Universität Bremen

Sommersemester 2024

Fahrplan

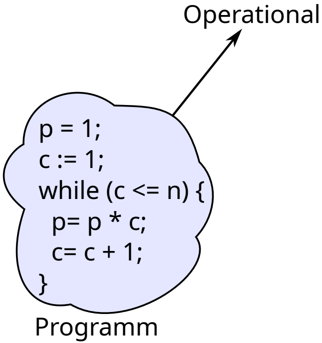
- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten im Floyd-Hoare-Kalkül
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare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

Drei Semantiken — Eine Sicht

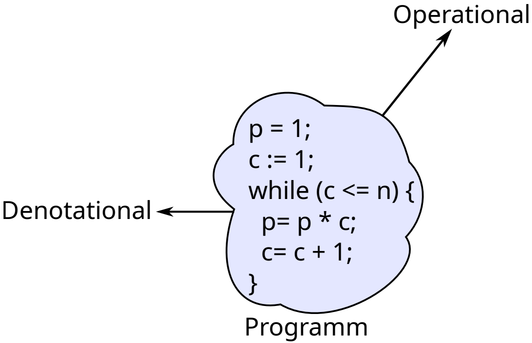


Programm

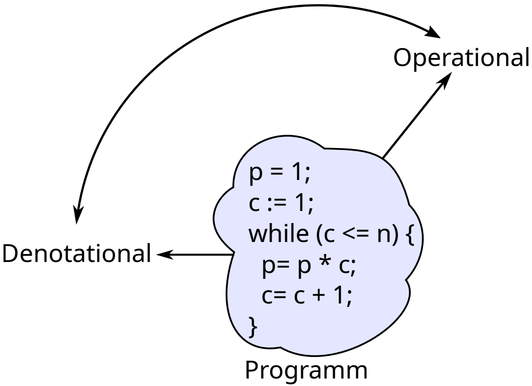
Drei Semantiken — Eine Sicht



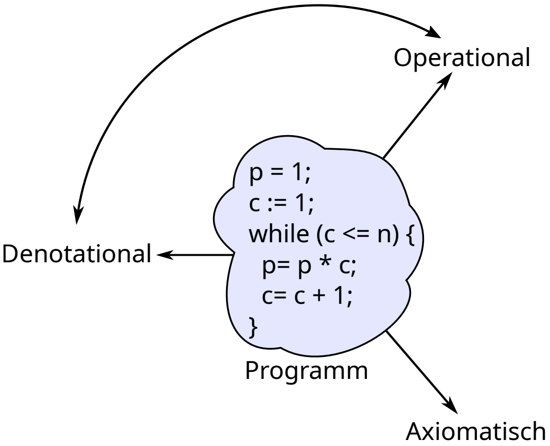
Drei Semantiken — Eine Sicht



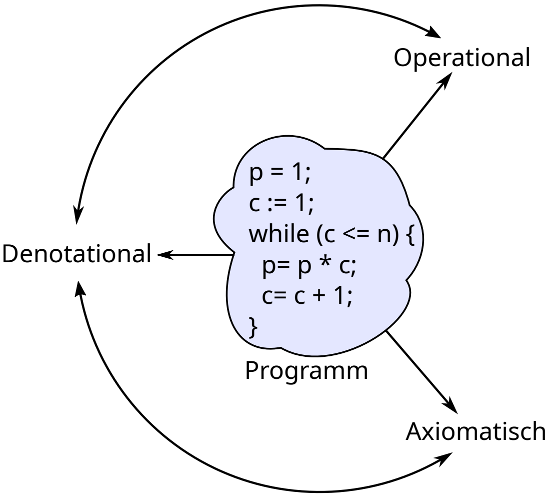
Drei Semantiken — Eine Sicht



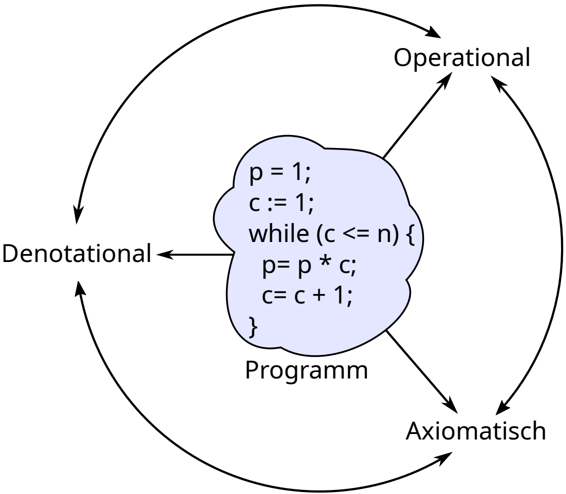
Drei Semantiken — Eine Sicht



Drei Semantiken — Eine Sicht



Drei Semantiken — Eine Sicht



Floyd-Hoare-Logik: Idee

- Was wird hier berechnet?

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

Floyd-Hoare-Logik: Idee

- Was wird hier berechnet? $p = n!$
- Warum? Wie können wir das **beweisen**?

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

Floyd-Hoare-Logik: Idee

- Was wird hier berechnet? $p = n!$
- Warum? Wie können wir das **beweisen**?
- Wir berechnen symbolisch, welche Werte Variablen über den Programmverlauf annehmen.

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

Floyd-Hoare-Logik: Idee

- Was wird hier berechnet? $p = n!$
- Warum? Wie können wir das **beweisen**?
- Wir berechnen symbolisch, welche Werte Variablen über den Programmverlauf annehmen.

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

- Operationale/denotionale Semantik nicht für **Korrektheitsbeweise** geeignet: Ausdrücke werden zu groß, skaliert nicht — **Abstraktion** nötig.

Floyd-Hoare-Logik: Idee

- Was wird hier berechnet? $p = n!$
- Warum? Wie können wir das **beweisen**?
- Wir berechnen symbolisch, welche Werte Variablen über den Programmverlauf annehmen.

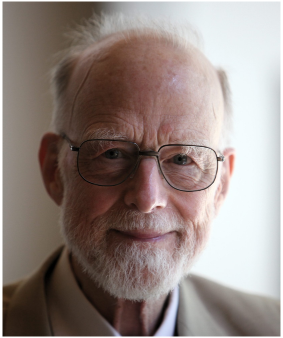
```
p= 1;
c= 1;
while (c <= n) {
  p = p * c;
  c = c + 1;
}
```

- Operationale/denotionale Semantik nicht für **Korrektheitsbeweise** geeignet: Ausdrücke werden zu groß, skaliert nicht — **Abstraktion** nötig.
- Grundprinzip:
  1. Zustandsabhängige **Zusicherungen** für bestimmte Punkte im Programmablauf.
  2. Berechnung der Gültigkeit dieser Zusicherungen durch **zustandsfreie Regeln**.

Bob Floyd und Tony Hoare

Bildquelle: Stanford University

Robert Floyd
1936 – 2001



Bildquelle: Wikipedia

Sir Anthony Charles Richard Hoare
* 1934

Grundbausteine der Floyd-Hoare-Logik

```
// (A)
p= 1;
c= 1;
// (B)
while (c <= n) {
  // (C)
  p= p * c;
  c= c + 1;
  // (D)
}
// (E)
```

- **Zusicherungen** über den Zustand
- Beispiele:
  - (B): Hier gilt $p = c = 1$
  - (D): Hier ist c ist um eines größer als der Wert von c an Punkt (C)
- Gesamtaussage: Wenn bei (A) der Wert von $n \geq 0$ ist, dann ist bei (E) $p = n!$

Grundbausteine der Floyd-Hoare-Logik

```
// (A)
p= 1;
c= 1;
// (B)
while (c <= n) {
  // (C)
  p= p * c;
  c= c + 1;
  // (D)
}
// (E)
```

- **Zusicherungen** über den Zustand
- Beispiele:
  - (B): Hier gilt $p = c = 1$
  - (D): Hier ist c ist um eines größer als der Wert von c an Punkt (C)
- Gesamtaussage: Wenn bei (A) der Wert von $n \geq 0$ ist, dann ist bei (E) $p = n!$
- Beobachtung:
  - n ist eine „Eingabevariable“, der Wert am Anfang des Programmes (A) ist relevant;

Grundbausteine der Floyd-Hoare-Logik

```
// (A)
p= 1;
c= 1;
// (B)
while (c <= n) {
  // (C)
  p= p * c;
  c= c + 1;
  // (D)
}
// (E)
```

- **Zusicherungen** über den Zustand
- Beispiele:
  - (B): Hier gilt $p = c = 1$
  - (D): Hier ist c ist um eines größer als der Wert von c an Punkt (C)
- Gesamtaussage: Wenn bei (A) der Wert von $n \geq 0$ ist, dann ist bei (E) $p = n!$
- Beobachtung:
  - n ist eine „Eingabevariable“, der Wert am Anfang des Programmes (A) ist relevant;
  - p ist eine „Ausgabevariable“, der Wert am Ende des Programmes (E) ist relevant;

Grundbausteine der Floyd-Hoare-Logik

```
// (A)
p= 1;
c= 1;
// (B)
while (c <= n) {
  // (C)
  p= p * c;
  c= c + 1;
  // (D)
}
// (E)
```

- **Zusicherungen** über den Zustand
- Beispiele:
  - (B): Hier gilt $p = c = 1$
  - (D): Hier ist c ist um eines größer als der Wert von c an Punkt (C)
- Gesamtaussage: Wenn bei (A) der Wert von $n \geq 0$ ist, dann ist bei (E) $p = n!$
- Beobachtung:
  - n ist eine „Eingabevariable“, der Wert am Anfang des Programmes (A) ist relevant;
  - p ist eine „Ausgabevariable“, der Wert am Ende des Programmes (E) ist relevant;
  - c ist eine „Arbeitsvariable“, der Wert am Anfang und Ende ist irrelevant

Arbeitsblatt 5.1: Was berechnet dieses Programm?

```
// (A)
x= 1;
c= 1;
// (B)
while (c <= y) {
  // (C)
  x= 2*x;
  c= c+1;
  // (D)
 }
// (E)
```

Betrachtet nebenstehendes Programm.
Analog zu dem Beispiel auf der vorherigen Folie:

1. Was berechnet das Programm?
2. Welches sind „Eingabevariablen", welches „Ausgabevariablen", welches sind „Arbeitsvariablen"?
3. Welche Zusicherungen und Zusammenhänge gelten zwischen den Variablen an den Punkten (A) bis (E)?

Auf dem Weg zur Floyd-Hoare-Logik

- Kern der Floyd-Hore-Logik sind **zustandsabhängige Aussagen**
- Aber: wie können wir Aussagen **jenseits** des Zustandes treffen?
- Einfaches Beispiel:

```
x = x+ 1;
```

- Der Wert von x wird um 1 erhöht
- Der Wert von x ist hinterher größer als vorher

Auf dem Weg zur Floyd-Hoare-Logik

- Kern der Floyd-Hore-Logik sind **zustandsabhängige Aussagen**
- Aber: wie können wir Aussagen **jenseits** des Zustandes treffen?
- Einfaches Beispiel:

```
x = x+ 1;
```

- Der Wert von x wird um 1 erhöht
- Der Wert von x ist hinterher größer als vorher

- Wir benötigen **zustandsfreie** Aussagen, um von Zuständen unabhängig **vergleichen** zu können.
- Die Logik **abstrahiert** den Effekt von Programmen.

Grundbausteine der Floyd-Hoare-Logik

- **Logische Variablen** (zustandsfrei) und **Programmvariablen** (zustandsabhängig)
- **Zusicherungen** mit logischen und Programmvariablen
- **Floyd-Hoare-Tripel** $\{P\}\, c\, \{Q\}$
  - Vorbedingung P (Zusicherung)
  - Programm c
  - Nachbedingung Q (Zusicherung)
- Floyd-Hoare-Logik abstrahiert von Programmen zu logischen Formeln.

Zusicherungen (Assertions)

- Erweiterung von **Aexp** and **Bexp** durch
  - **Logische** Variablen **Var** $v := N, M, L, U, V, X, Y, Z$
  - Definierte Funktionen und Prädikate über **Aexp** $n!, x^y, \ldots$
  - Implikation und Quantoren $b_1 \longrightarrow b_2, \forall v.\, b, \exists v.\, b$
- Formal:

$$\begin{array}{rrl}
\textbf{Aexpv} & a ::= & \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 / a_2 \\
 & & \mid f(e_1, \ldots, e_n) \\
\textbf{Assn} & b ::= & \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 < a_2 \\
 & & \mid \,!\, b \mid b_1 \,\&\&\, b2 \mid b_1 \mid\mid b_2 \\
 & & \mid b_1 -- > b_2 \mid p(e_1, \ldots, e_n) \mid \backslash\textbf{forall}\ v.\, b \mid \backslash\textbf{exists}\ v.\, b
\end{array}$$

Zusicherungen (Assertions)

- Erweiterung von **Aexp** and **Bexp** durch
  - **Logische** Variablen **Var** $v := N, M, L, U, V, X, Y, Z$
  - Definierte Funktionen und Prädikate über **Aexp** $n!, x^y, \ldots$
  - Implikation und Quantoren $b_1 \longrightarrow b_2, \forall v.\, b, \exists v.\, b$
- Formal:

$$
\begin{array}{lll}
\textbf{Aexpv} & a ::= & \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 / a_2 \\
 & & \mid f(e_1, \ldots, e_n) \\
\textbf{Assn} & b ::= & \mathit{true} \mid \mathit{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \\
 & & \mid \neg b \mid b_1 \wedge b2 \mid b_1 \vee b_2 \\
 & & \mid b_1 \longrightarrow b_2 \mid p(e_1, \ldots, e_n) \mid \forall v.\, b \mid \exists v.\, b
\end{array}
$$

Denotationale Semantik von Zusicherungen

- Erste Näherung: Funktion

$$
\begin{aligned}
[\![a]\!]_{\mathcal{A}} &: \textbf{Aexpv} \to (\Sigma \rightharpoonup \mathbb{Z}) \\
[\![b]\!]_{\mathcal{B}} &: \textbf{Assn} \to (\Sigma \rightharpoonup \mathbb{B})
\end{aligned}
$$

  - **Konservative** Erweiterung von $[\![a]\!]_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$
- Aber: was ist mit den logischen Variablen?

Denotationale Semantik von Zusicherungen

- Erste Näherung: Funktion

$$
\begin{aligned}
[\![a]\!]_{\mathcal{A}} &: \textbf{Aexpv} \to (\Sigma \rightharpoonup \mathbb{Z}) \\
[\![b]\!]_{\mathcal{B}} &: \textbf{Assn} \to (\Sigma \rightharpoonup \mathbb{B})
\end{aligned}
$$

  - **Konservative** Erweiterung von $[\![a]\!]_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$
- Aber: was ist mit den logischen Variablen?
- Zusätzlicher Parameter **Belegung** der logischen Variablen $I : \textbf{Var} \to \mathbb{Z}$

$$
\begin{aligned}
[\![a]\!]_{\mathcal{A}} &: \textbf{Aexpv} \to (\textbf{Var} \to \mathbb{Z}) \to (\Sigma \rightharpoonup \mathbb{Z}) \\
[\![b]\!]_{\mathcal{B}} &: \textbf{Assn} \to (\textbf{Var} \to \mathbb{Z}) \to (\Sigma \rightharpoonup \mathcal{B})
\end{aligned}
$$

- Bemerkung: $I : \textbf{Var} \to \mathbb{Z}$ ist immer eine **totale Funktion** im Gegensatz zu einem Zustand.

Denotat von Aexp

$$[\![a]\!]_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$$

$$\begin{aligned}
[\![n]\!]_{\mathcal{A}} =& \{(\sigma, [\![n]\!]) \mid \sigma \in \Sigma\} \\
[\![x]\!]_{\mathcal{A}} =& \{(\sigma, \sigma(x)) \mid \sigma \in \Sigma, x \in Dom(\sigma)\} \\
[\![a_0 + a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}\} \\
[\![a_0 - a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}\} \\
[\![a_0 * a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}}\} \\
[\![a_0 / a_1]\!]_{\mathcal{A}} =& \{(\sigma, n_0 \div n_1) \mid (\sigma, n_0) \in [\![a_0]\!]_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]_{\mathcal{A}} \wedge n_1 \neq 0\}
\end{aligned}$$

Denotat von Aexpv

$$[\![a]\!]_{\mathcal{A}} : \textbf{Aexpv} \to (\textbf{Var} \to \mathbb{Z}) \to\to (\Sigma \rightharpoonup \mathbb{Z})$$

Sei I : **Var** $\to \mathbb{Z}$ eine beliebige Belegung

$$\begin{aligned}
[\![n]\!]^I_{\mathcal{A}} =& \{(\sigma, [\![n]\!]) \mid \sigma \in \Sigma\} \\
[\![x]\!]^I_{\mathcal{A}} =& \{(\sigma, \sigma(x)) \mid \sigma \in \Sigma, x \in Dom(\sigma)\} \\
[\![a_0 + a_1]\!]^I_{\mathcal{A}} =& \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^I_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^I_{\mathcal{A}}\} \\
[\![a_0 - a_1]\!]^I_{\mathcal{A}} =& \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^I_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^I_{\mathcal{A}}\} \\
[\![a_0 * a_1]\!]^I_{\mathcal{A}} =& \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^I_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^I_{\mathcal{A}}\} \\
[\![a_0 / a_1]\!]^I_{\mathcal{A}} =& \{(\sigma, n_0 \div n_1) \mid (\sigma, n_0) \in [\![a_0]\!]^I_{\mathcal{A}} \wedge (\sigma, n_1) \in [\![a_1]\!]^I_{\mathcal{A}} \wedge n_1 \neq 0\} \\
[\![X]\!]^I_{\mathcal{A}} =& \{(\sigma, I(X)) \mid \sigma \in \Sigma, X \in V\}
\end{aligned}$$

Erfüllung von Zusicherungen

- Wann gilt eine Zusicherung $b \in \mathbf{Assn}$ in einem Zustand σ?
  - Auswertung (denotationale Semantik) ergibt *true*
  - Belegung ist zusätzlicher Parameter

Erfülltheit von Zusicherungen

$b \in \mathbf{Assn}$ ist in Zustand σ mit Belegung I erfüllt ($\sigma \models^I b$), gdw

$$[\![b]\!]^I_{\mathcal{B}}(\sigma) = \mathit{true}$$

Arbeitsblatt 5.2: Zusicherungen

Betrachte folgende Zusicherung:

$$a \quad \equiv \quad \underbrace{2 \cdot x = X}_{p} \longrightarrow \underbrace{x < X}_{q}$$

Gegeben folgende Belegungen $I_1, \ldots, I_3$ und Zustände $s_1, \ldots, s_3$:

$$s_1 = \langle x \mapsto 0 \rangle, s_2 = \langle x \mapsto 1 \rangle, s_3 = \langle x \mapsto 5 \rangle$$
$$I_1 = \langle X \mapsto 0 \rangle, I_2 = \langle X \mapsto 2 \rangle, I_3 = \langle X \mapsto 10 \rangle$$

Unter welchen Belegungen und Zuständen ist a wahr?

	I_1			I_2			I_3		
	p	q	a	p	q	a	p	q	a
s_1									
s_2									
s_3									

Wie kann man a so ändern, dass a für **alle** Belegungen und Zustände wahr ist?

Floyd-Hoare-Tripel

Partielle Korrektheit $\models \{P\}\, c\, \{Q\}$

$\{P\}\, c\, \{Q\}$ ist **partiell korrekt**, wenn für all Belegungen I und alle Zustände σ, die P erfüllen, gilt: **wenn** die Ausführung von c mit σ in einem Zustand τ terminiert, **dann** erfüllt τ mit Belegung I Q.

$$\models \{P\}\, c\, \{Q\} \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^{I} P \wedge \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \tau \models^{I} Q$$

- Gleiche Belegung der logischen Variablen in P und Q erlaubt **Vergleich** zwischen Zuständen

Totale Korrektheit $\models [P]\, c\, [Q]$

$[P]\, c\, [Q]$ ist **total korrekt**, wenn für all Belegungen I und alle Zustande σ, die P erfüllen, die Ausführung von c mit σ in einem Zustand τ terminiert, und τ mit der Belegung I erfüllt Q.

$$\models [P]\, c\, [Q] \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^{I} P \Longrightarrow \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \wedge \tau \models^{I} Q$$

Arbeitsblatt 5.3: Gültigkeit

Welche dieser Hoare-Tripel ist semantisch gültig?

```
// {x = X ∧ x ≥ 3}
x= x−3;
if (x < 0) x= 0;
x= x+3;
// {x = X}
```

```
// {b = B}
b= b−a;
x= a+b;
// {x = a + B}
```

```
// {x = X ∧ y = Y}
x= x+y;
y= x−y;
x= x−y;
// {x = Y ∧ y = X}
```

Arbeitsblatt 5.3: Gültigkeit

Welche dieser Hoare-Tripel ist semantisch gültig?

```
// {x = X ∧ x ≥ 3}
x= x−3;
if (x < 0) x= 0;
x= x+3;
// {x = X}
```

```
// {b = B}
b= b−a;
x= a+b;
// {x = a + B}
```

```
// {x = X ∧ y = Y}
x= x+y;
y= x−y;
x= x−y;
// {x = Y ∧ y = X}
```

Arbeitsblatt 5.3: Gültigkeit

Welche dieser Hoare-Tripel ist semantisch gültig?

```
// {x = X ∧ x ≥ 3}
x= x−3;
if (x < 0) x= 0;
x= x+3;
// {x = X}
```

```
// {b = B}
b= b−a;
x= a+b;
// {x = a + B}
```

```
// {x = X ∧ y = Y}
x= x+y;
y= x−y;
x= x−y;
// {x = Y ∧ y = X}
```

Arbeitsblatt 5.3: Gültigkeit

Welche dieser Hoare-Tripel ist semantisch gültig?

```
// {x = X ∧ x ≥ 3}
x= x−3;
if (x < 0) x= 0;
x= x+3;
// {x = X}
```

```
// {b = B}
b= b−a;
x= a+b;
// {x = a + B}
```

```
// {x = X ∧ y = Y}
x= x+y;
y= x−y;
x= x−y;
// {x = Y ∧ y = X}
```

Weitere Beispiele

- Folgendes **gilt**:

$$\models \{true\}\ \textbf{while}(1)\{\ \}\ \{true\}$$

Weitere Beispiele

- Folgendes **gilt**:

$$\models \{true\}\ \textbf{while}(1)\{\ \}\ \{true\}$$

- Folgendes gilt **nicht**:

$$\models [true]\ \textbf{while}(1)\{\ \}\ [true]$$

Weitere Beispiele

- Folgendes **gilt**:

$$\models \{true\}\ \mathbf{while}(1)\{\ \}\ \{true\}$$

- Folgendes gilt **nicht**:

$$\models [true]\ \mathbf{while}(1)\{\ \}\ [true]$$

- Folgende **gelten**:

$$\models \{false\}\ \mathbf{while}\ (\mathbf{1})\ \{\ \}\ \{true\}$$
$$\models [false]\ \mathbf{while}\ (\mathbf{1})\ \{\ \}\ [true]$$

Wegen *ex falso quodlibet*: $false \Longrightarrow \phi$

Gültigkeit und Herleitbarkeit

- **Semantische Gültigkeit:** $\models \{P\}\, c\, \{Q\}$
  - Definiert durch denotationale Semantik:

$$\models \{P\}\, c\, \{Q\} \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^{I} P \wedge \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \tau \models^{I} Q$$

  - Problem: müssten Semantik von c ausrechnen

Gültigkeit und Herleitbarkeit

- **Semantische Gültigkeit:** $\models \{P\}\, c\, \{Q\}$
  - Definiert durch denotationale Semantik:
    $$\models \{P\}\, c\, \{Q\} \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^I P \wedge \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \tau \models^I Q$$
  - Problem: müssten Semantik von c ausrechnen
- **Syntaktische Herleitbarkeit:** $\vdash \{P\}\, c\, \{Q\}$
  - Durch **Regeln** definiert
  - Kann **hergeleitet** werden
  - Muss **korrekt** bezüglich semantischer Gültigkeit gezeigt werden
- Generelles Vorgehen in der Logik

Regeln des Floyd-Hoare-Kalküls

- Der Floyd-Hoare-Kalkül erlaubt es, Zusicherungen der Form $\vdash \{P\}\, c\, \{Q\}$ syntaktisch **herzuleiten**.
- Der **Kalkül** der Logik besteht aus sechs Regeln der Form

$$\frac{\vdash \{P_1\}\, c_1\, \{Q_1\} \ldots \vdash \{P_n\}\, c_n\, \{Q_n\}}{\vdash \{P\}\, c\, \{Q\}}$$

- Für jedes Konstrukt der Programmiersprache gibt es eine Regel.

Regeln des Floyd-Hoare-Kalküls: Zuweisung

$$\frac{}{\vdash \{P[e/x]\}\, x = e\, \{P\}}$$

- Eine Zuweisung `x=e` ändert den Zustand so dass an der Stelle x jetzt der Wert von e steht. Damit **nachher** das Prädikat P gilt, muss also **vorher** das Prädikat gelten, wenn wir x durch e ersetzen.
- Es ist völlig normal (aber dennoch falsch) zu denken, die Substitution gehöre eigentlich in die Nachbedingung.
- Beispiele:

```
// {?}
x = 5
// {x < 10}
```

Regeln des Floyd-Hoare-Kalküls: Zuweisung

$$\frac{}{\vdash \{P[e/x]\}\, x = e\, \{P\}}$$

- Eine Zuweisung x=e ändert den Zustand so dass an der Stelle x jetzt der Wert von e steht. Damit **nachher** das Prädikat P gilt, muss also **vorher** das Prädikat gelten, wenn wir x durch e ersetzen.
- Es ist völlig normal (aber dennoch falsch) zu denken, die Substitution gehöre eigentlich in die Nachbedingung.
- Beispiele:

```
// {(x < 10)[5/x]}
x = 5
// {x < 10}
```

Regeln des Floyd-Hoare-Kalküls: Zuweisung

$$\frac{}{\vdash \{P[e/x]\}\, x = e\, \{P\}}$$

- Eine Zuweisung `x=e` ändert den Zustand so dass an der Stelle x jetzt der Wert von e steht. Damit **nachher** das Prädikat P gilt, muss also **vorher** das Prädikat gelten, wenn wir x durch e ersetzen.
- Es ist völlig normal (aber dennoch falsch) zu denken, die Substitution gehöre eigentlich in die Nachbedingung.
- Beispiele:

```
// {(x < 10)[5/x] ⟺ 5 < 10}
x = 5
// {x < 10}
```

Regeln des Floyd-Hoare-Kalküls: Zuweisung

$$\frac{}{\vdash \{P[e/x]\}\, x = e\, \{P\}}$$

- Eine Zuweisung `x=e` ändert den Zustand so dass an der Stelle x jetzt der Wert von e steht. Damit **nachher** das Prädikat P gilt, muss also **vorher** das Prädikat gelten, wenn wir x durch e ersetzen.
- Es ist völlig normal (aber dennoch falsch) zu denken, die Substitution gehöre eigentlich in die Nachbedingung.
- Beispiele:

```
// {(x < 10)[5/x] ⟺ 5 < 10}
x = 5
// {x < 10}
```

```
//{x + 1 < 10}
x = x+ 1
//{x < 10}
```

Regeln des Floyd-Hoare-Kalküls: Zuweisung

$$\frac{}{\vdash \{P[e/x]\}\, x = e\, \{P\}}$$

- Eine Zuweisung `x=e` ändert den Zustand so dass an der Stelle x jetzt der Wert von e steht. Damit **nachher** das Prädikat P gilt, muss also **vorher** das Prädikat gelten, wenn wir x durch e ersetzen.
- Es ist völlig normal (aber dennoch falsch) zu denken, die Substitution gehöre eigentlich in die Nachbedingung.
- Beispiele:

```
// {(x < 10)[5/x] <==> 5 < 10}
x = 5
// {x < 10}
```

```
//{x + 1 < 10 <==> x < 9}
x = x+ 1
//{x < 10}
```

Regeln des Floyd-Hoare-Kalküls: Sequenzierung

$$\frac{\vdash \{A\}\, c_1\, \{B\} \qquad \vdash \{B\}\, c_2\, \{C\}}{\vdash \{A\}\, c_1; c_2\, \{C\}}$$

- Hier wird eine Zwischenzusicherung B benötigt.

$$\frac{}{\vdash \{A\}\, \{\,\}\, \{A\}}$$

- Trivial.

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?
- $\vdash \{x = X \wedge y = Y\}\, p\, \{y = X \wedge x = Y\}$

Herleitung:

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?
- $\vdash \{x = X \wedge y = Y\}\, p\, \{y = X \wedge x = Y\}$

Herleitung:

$$\frac{}{\vdash \{x = X \wedge y = Y\} \\ z = x; x = y; y = z; \\ \{y = X \wedge x = Y\}}$$

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?
- $\vdash \{x = X \land y = Y\}\, p\, \{y = X \land x = Y\}$

Herleitung:

$$\dfrac{\dfrac{}{\begin{array}{l}\vdash \{x = X \land y = Y\} \\ \quad z = x; x = y; \\ \quad \{?\}\end{array}} \qquad \dfrac{}{\begin{array}{l}\vdash \{?\} \\ \quad y = z; \\ \quad \{y = X \land x = Y\}\end{array}}}{\begin{array}{l}\vdash \{x = X \land y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \land x = Y\}\end{array}}$$

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?
- $\vdash \{x = X \land y = Y\}\, p\, \{y = X \land x = Y\}$

Herleitung:

$$
\dfrac{\dfrac{}{\vdash \{x = X \land y = Y\}\ z = x; x = y;\ \{z = X \land x = Y\}} \qquad \dfrac{}{\vdash \{z = X \land x = Y\}\ y = z;\ \{y = X \land x = Y\}}}{\vdash \{x = X \land y = Y\}\ z = x; x = y; y = z;\ \{y = X \land x = Y\}}
$$

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?
- $\vdash \{x = X \wedge y = Y\}\, p\, \{y = X \wedge x = Y\}$

Herleitung:

$$
\dfrac{
  \dfrac{
    \dfrac{}{\vdash \{x = X \wedge y = Y\}\; z = x;\; \{?\}}
    \qquad
    \dfrac{}{\vdash \{?\}\; x = y;\; \{z = X \wedge x = Y\}}
  }{\vdash \{x = X \wedge y = Y\}\; z = x; x = y;\; \{z = X \wedge x = Y\}}
  \qquad
  \dfrac{}{\vdash \{z = X \wedge x = Y\}\; y = z;\; \{y = X \wedge x = Y\}}
}{\vdash \{x = X \wedge y = Y\}\; z = x; x = y; y = z;\; \{y = X \wedge x = Y\}}
$$

Ein allererstes Beispiel

```
z= x;
x= y;
y= z;
```

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?
- $\vdash \{x = X \wedge y = Y\}\, p\, \{y = X \wedge x = Y\}$

Herleitung:

$$
\dfrac{
  \dfrac{
    \dfrac{}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; \\ \quad \{z = X \wedge y = Y\}\end{array}}
    \qquad
    \dfrac{}{\begin{array}{l}\vdash \{z = X \wedge y = Y\} \\ \quad x = y; \\ \quad \{z = X \wedge x = Y\}\end{array}}
  }{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; \\ \quad \{z = X \wedge x = Y\}\end{array}}
  \qquad
  \dfrac{}{\begin{array}{l}\vdash \{z = X \wedge x = Y\} \\ \quad y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
$$

Vereinfachte Notation für Sequenzen

```
// {y = Y ∧ x = X}
z= x;
//
x= y;
//
y= z;
// {x = Y ∧ y = X}
```

- Die **gleiche** Information wie der Herleitungsbaum
- aber **kompakt** dargestellt
- Beweis erfolgt **rückwärts** (von der letzten Zuweisung ausgehend)

Vereinfachte Notation für Sequenzen

```
// {y = Y ∧ x = X}
z= x;
//
x= y;
// {x = Y ∧ z = X}
y= z;
// {x = Y ∧ y = X}
```

- Die **gleiche** Information wie der Herleitungsbaum
- aber **kompakt** dargestellt
- Beweis erfolgt **rückwärts** (von der letzten Zuweisung ausgehend)

Vereinfachte Notation für Sequenzen

```
// {y = Y ∧ x = X}
z= x;
// {y = Y ∧ z = X}
x= y;
// {x = Y ∧ z = X}
y= z;
// {x = Y ∧ y = X}
```

- Die **gleiche** Information wie der Herleitungsbaum
- aber **kompakt** dargestellt
- Beweis erfolgt **rückwärts** (von der letzten Zuweisung ausgehend)

Arbeitsblatt 5.4: Ein erster Beweis

Betrachte den Rumpf des Fakultätsprogramms:

```
// (B)
p= p* c;
// (A)
c= c+ 1;
// {p = (c − 1)!}
```

- Welche Zusicherungen gelten
  1. an der Stelle (A)?
  2. an der Stelle (B)?

Arbeitsblatt 5.4: Ein erster Beweis

Betrachte den Rumpf des Fakultätsprogramms:

```
// (B)
p= p* c;
// (A)
c= c+ 1;
// {p = (c − 1)!}
```

- Welche Zusicherungen gelten

  1. an der Stelle (A)?

  2. an der Stelle (B)?

Arbeitsblatt 5.4: Ein erster Beweis

Betrachte den Rumpf des Fakultätsprogramms:

```
// (B)
p= p* c;
// (A)
c= c+ 1;
// {p = (c − 1)!}
```

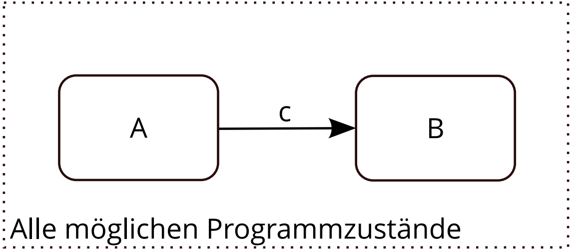
- Welche Zusicherungen gelten

  1. an der Stelle (A)?

  2. an der Stelle (B)?

Regeln des Floyd-Hoare-Kalküls: Weakening

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\,c\,\{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\,c\,\{B'\}}$$

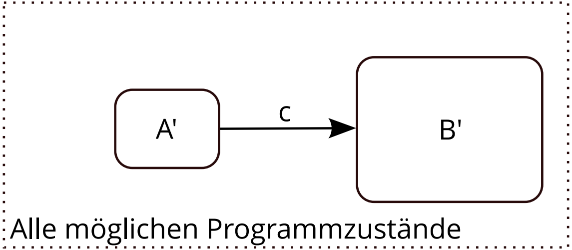


- $\models \{A\}\,c\,\{B\}$: Ausführung von c startet in Zustand, in dem A gilt, und endet (ggf) in Zustand, in dem B gilt.
- Zustandsprädikate beschreiben Mengen von Zuständen:

$$\{\sigma \in \Sigma | \sigma \models^I P\} \subseteq \{\sigma \in \Sigma | \sigma \models^I Q\} \text{ gdw. } P \Longrightarrow Q$$

Regeln des Floyd-Hoare-Kalküls: Weakening

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$



- $\models \{A\}\, c\, \{B\}$: Ausführung von c startet in Zustand, in dem A gilt, und endet (ggf) in Zustand, in dem B gilt.
- Zustandsprädikate beschreiben Mengen von Zuständen:

$$\{\sigma \in \Sigma | \sigma \models^I P\} \subseteq \{\sigma \in \Sigma | \sigma \models^I Q\} \text{ gdw. } P \Longrightarrow Q$$

Arbeitsblatt 5.5: Ein zweiter Beweis

Wir betrachten noch einmal das Vertauschen, diesmal ohne Hilfsvariable:

```
// {x = X ∧ y = Y}
// (A)
x= x+y;
// (B)
y= x-y;
// (C)
x= x-y;
// {y = X ∧ x = Y}
```

- Welche Zusicherungen gelten an den Stellen (A), (B), (C) und wie werden sie so vereinfacht, dass die Vorbedingung entsteht?

1. (C)?
2. (B)?
3. (A)?

Arbeitsblatt 5.5: Ein zweiter Beweis

Wir betrachten noch einmal das Vertauschen, diesmal ohne Hilfsvariable:

```
// {x = X ∧ y = Y}
// (A)
x= x+y;
// (B)
y= x−y;
// (C)
x= x−y;
// {y = X ∧ x = Y}
```

- Welche Zusicherungen gelten an den Stellen (A), (B), (C) und wie werden sie so vereinfacht, dass die Vorbedingung entsteht?

1. (C)?
2. (B)?
3. (A)?

Arbeitsblatt 5.5: Ein zweiter Beweis

Wir betrachten noch einmal das Vertauschen, diesmal ohne Hilfsvariable:

```
// {x = X ∧ y = Y}
// (A)
x= x+y;
// (B)
y= x-y;
// (C)
x= x-y;
// {y = X ∧ x = Y}
```

- Welche Zusicherungen gelten an den Stellen (A), (B), (C) und wie werden sie so vereinfacht, dass die Vorbedingung entsteht?

1. (C)?
2. (B)?
3. (A)?

Arbeitsblatt 5.5: Ein zweiter Beweis

Wir betrachten noch einmal das Vertauschen, diesmal ohne Hilfsvariable:

```
// {x = X ∧ y = Y}
// (A)
x= x+y;
// (B)
y= x−y;
// (C)
x= x−y;
// {y = X ∧ x = Y}
```

- Welche Zusicherungen gelten an den Stellen (A), (B), (C) und wie werden sie so vereinfacht, dass die Vorbedingung entsteht?

  1. (C)?
  2. (B)?
  3. (A)?

Regeln des Floyd-Hoare-Kalküls: Fallunterscheidung

$$\frac{\vdash \{A \land b\}\, c_0\, \{B\} \qquad \vdash \{A \land \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

- In der Vorbedingung des **if**-Zweiges gilt die Bedingung b, und im **else**-Zweig gilt die Negation $\neg b$.
- Beide Zweige müssen mit derselben Nachbedingung enden.

Arbeitsblatt 5.6: Dreimal ist Bremer Recht

Betrachte folgendes Programm:

```
// (F)
if (x < y) {
  // (E)
  // ...
  z = x;
  // (C)
} else {
  // (D)
  // ...
  z= y;
  // (B)
}
// (A)
```

- Was berechnet dieses Programm?
- Wie spezifizieren wir das?
- Welche Zusicherungen müssen an den Stellen (A) – (F) gelten?
- Wo müssen wir welche logische Umformungen nutzen?

Regeln des Floyd-Hoare-Kalküls: Iteration

$$\frac{\vdash \{A \wedge b\}\, c\, \{A\}}{\vdash \{A\}\ \textbf{while}(b)\ c\, \{A \wedge \neg b\}}$$

- Iteration korrespondiert zu **Induktion**.
- Bei wohlfundierter Induktion zeigen wir, dass die **gleiche** Eigenschaft für alle x gilt, $P(x)$, wenn sie für alle kleineren y gilt — d.h. wenn y größer wird muss die Eigenschaft weiterhin gelten.
- Analog dazu benötigen wir hier eine **Invariante** A, die sowohl **vor** als auch **nach** dem Schleifenrumpf gilt.
- In der Vorbedingung des Schleifenrumpfes können wir die Schleifenbedingung b annehmen.
- Die **Vorbedingung** der **Schleife** ist die Invariante A, und die **Nachbedingung** der **Schleife** ist A und die Negation der Schleifenbedingung b.

Wie wir Floyd-Hoare-Beweise aufschreiben

```
// {P}
// {P2[e/x]}
x= e;
// {P3}
while (x< n) {
    // {P3 ∧ x < n}
    // {P3[a/z]}
    z= a;
    // {P3}
    }
// {P3 ∧ ¬(x < n)}
// {Q}
```

- Beispiel zeigt: $\vdash \{P\}\, c\, \{Q\}$
- Programm wird mit gültigen Zusicherungen annotiert.
- Vor einer Zeile steht die Vorbedingung, danach die Nachbedingung.
  - Muss genau auf Anweisung passen.
- Implizite Anwendung der Sequenzenregel.
- Weakening wird notiert durch mehrere Zusicherungen, und muss **bewiesen** werden.
  - Im Beispiel: $P \Longrightarrow P_2[e/x]$, $P_2 \Longrightarrow P_3$, $P_3 \wedge x < n \Longrightarrow P_4$, $P_3 \wedge \neg(x < n) \Longrightarrow Q$.

Überblick: die Regeln des Floyd-Hoare-Kalküls

$$\frac{}{\vdash \{P[e/x]\}\, x = e\, \{P\}}$$

$$\frac{\vdash \{A \wedge b\}\, c_0\, \{B\} \qquad \vdash \{A \wedge \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

$$\frac{\vdash \{A \wedge b\}\, c\, \{A\}}{\vdash \{A\}\ \textbf{while}(b)\ c\, \{A \wedge \neg b\}}$$

$$\frac{}{\vdash \{A\}\, \{\,\}\, \{A\}} \qquad \frac{\vdash \{A\}\, c_1\, \{B\} \qquad \vdash \{B\}\, c_2\, \{C\}}{\vdash \{A\}\, c_1; c_2\, \{C\}}$$

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$

Zusammenfassung Floyd-Hoare-Logik

- Die Logik abstrahiert über konkrete Systemzustände durch **Zusicherungen**
- Zusicherungen sind boolsche Ausdrücke, angereichert durch logische Variablen.
- **Hoare-Tripel** $\{P\}\, c\, \{Q\}$ abstrahieren die Semantik von c
  - Semantische **Gültigkeit** von Hoare-Tripeln: $\models \{P\}\, c\, \{Q\}$.
  - Syntaktische **Herleitbarkeit** von Hoare-Tripeln: $\vdash \{P\}\, c\, \{Q\}$
- **Zuweisungen** werden durch **Substitution** modelliert, d.h. die Menge der gültigen Aussagen ändert sich.
- Für Iterationen wird eine **Invariante** benötigt (die **nicht** hergeleitet werden kann).