Korrekte Software: Grundlagen und Methoden
Vorlesung 5 vom 02.05.24
Die Floyd-Hoare-Logik

Serge Autexier, Christoph Lüth

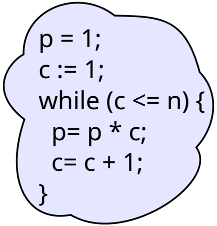
Universität Bremen

Sommersemester 2024

Fahrplan

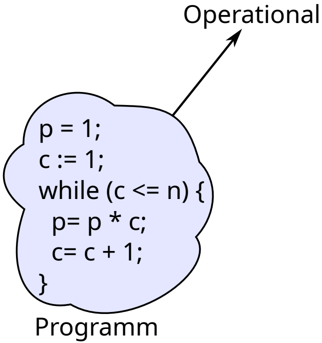
- Einführung
- Operationale Semantik
- Denotationale Semantik
- Äquivalenz der Operationalen und Denotationalen Semantik
- Der Floyd-Hoare-Kalkül
- Invarianten im Floyd-Hoare-Kalkül
- Korrektheit des Floyd-Hoare-Kalküls
- Strukturierte Datentypen
- Verifikationsbedingungen
- Vorwärts mit Floyd und Hoare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

Drei Semantiken — Eine Sicht

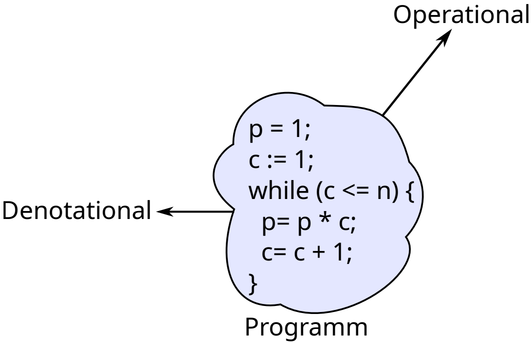


Programm

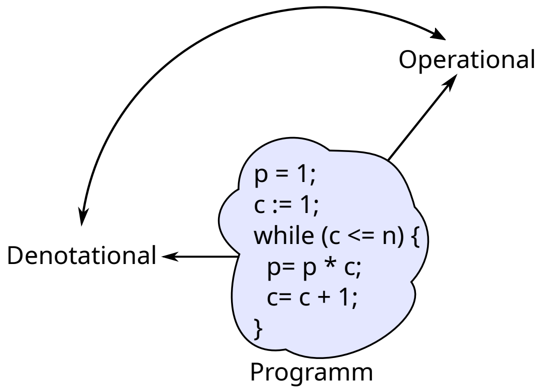
Drei Semantiken — Eine Sicht



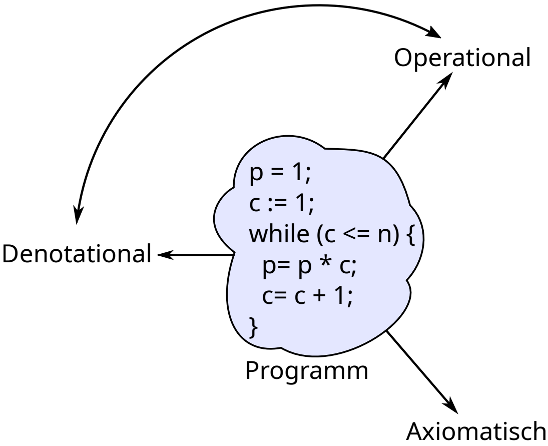
Drei Semantiken — Eine Sicht



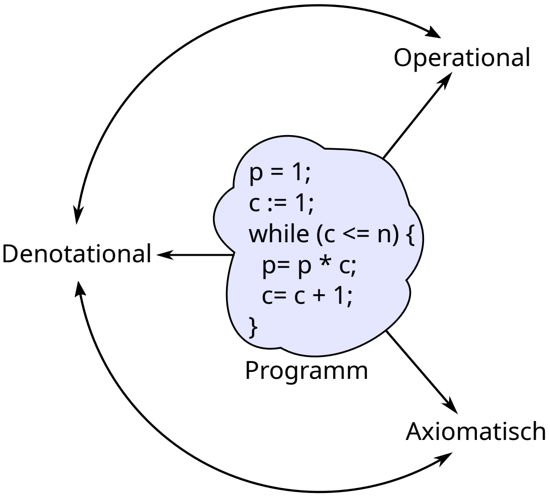
Drei Semantiken — Eine Sicht



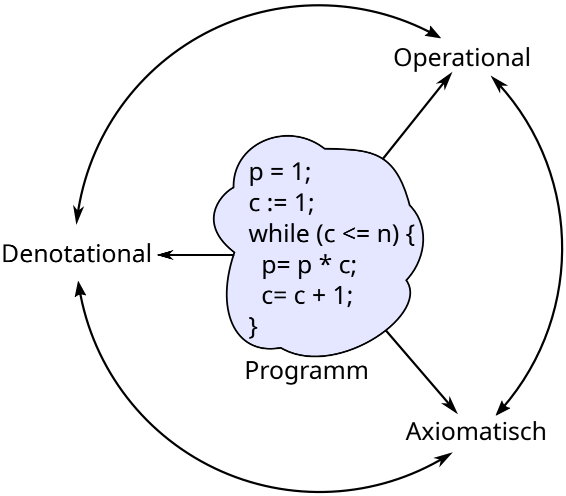
Drei Semantiken — Eine Sicht



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.

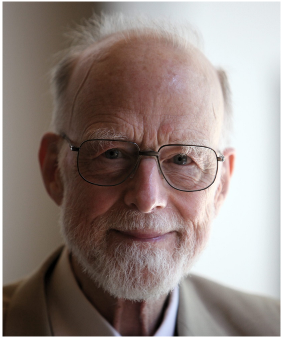
```
p= 1;
c= 1;
while (c <= n) {
  p = p * c;
  c = c + 1;
}
```

- 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:

$$
\begin{aligned}
\textbf{Aexpv} \quad a ::= \; & \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 / a_2 \\
& \mid f(e_1, \ldots, e_n) \\
\textbf{Assn} \quad b ::= \; & \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 < a_2 \\
& \mid \,!\, b \mid b_1 \,\&\&\, b2 \mid b_1 \,||\, b_2 \\
& \mid b_1 -- > b_2 \mid p(e_1, \ldots, e_n) \mid \backslash\textbf{forall}\; v.\, b \mid \backslash\textbf{exists}\; v.\, b
\end{aligned}
$$

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:

$$\begin{array}{rrl} \textbf{Aexpv} & a ::= & \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 / a_2 \\ & & \mid f(e_1, \ldots, e_n) \\ \textbf{Assn} & b ::= & \mathit{true} \mid \mathit{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \\ & & \mid \neg b \mid b_1 \wedge b2 \mid b_1 \vee b_2 \\ & & \mid b_1 \longrightarrow b_2 \mid p(e_1, \ldots, e_n) \mid \forall v.\, b \mid \exists v.\, b \end{array}$$

Denotationale Semantik von Zusicherungen

- Erste Näherung: Funktion

$$\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexpv} \to (\Sigma \rightharpoonup \mathbb{Z})$$
$$\llbracket b \rrbracket_{\mathcal{B}} : \textbf{Assn} \to (\Sigma \rightharpoonup \mathbb{B})$$

  - **Konservative** Erweiterung von $\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$
- Aber: was ist mit den logischen Variablen?

Denotationale Semantik von Zusicherungen

- Erste Näherung: Funktion

$$\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexpv} \to (\Sigma \rightharpoonup \mathbb{Z})$$
$$\llbracket b \rrbracket_{\mathcal{B}} : \textbf{Assn} \to (\Sigma \rightharpoonup \mathbb{B})$$

  - **Konservative** Erweiterung von $\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$
- Aber: was ist mit den logischen Variablen?
- Zusätzlicher Parameter **Belegung** der logischen Variablen $I : \textbf{Var} \to \mathbb{Z}$

$$\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexpv} \to (\textbf{Var} \to \mathbb{Z}) \to (\Sigma \rightharpoonup \mathbb{Z})$$
$$\llbracket b \rrbracket_{\mathcal{B}} : \textbf{Assn} \to (\textbf{Var} \to \mathbb{Z}) \to (\Sigma \rightharpoonup \mathcal{B})$$

- Bemerkung: $I : \textbf{Var} \to \mathbb{Z}$ ist immer eine **totale Funktion** im Gegensatz zu einem Zustand.

Denotat von Aexp

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

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

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;
```

Herleitung:

- Was berechnet dieses Programm?
- Die Werte von x und y werden vertauscht.
- Wie spezifizieren wir das?
- $\vdash \{x = X \land y = Y\}\, p\, \{y = X \land x = Y\}$

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?
- $\vdash \{x = X \wedge y = Y\}\, p\, \{y = X \wedge x = Y\}$

Herleitung:

$$\dfrac{\dfrac{}{\vdash \{x = X \wedge y = Y\}\; z = x; x = y; \; \{?\}} \qquad \dfrac{}{\vdash \{?\}\; y = z; \;\{y = X \wedge x = Y\}}}{\vdash \{x = X \wedge y = Y\}\; z = x; x = y; y = z; \;\{y = X \wedge x = Y\}}$$

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:

$$
\dfrac{
\dfrac{}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; \\ \quad \{z = X \wedge x = Y\}\end{array}}
\qquad
\dfrac{}{\begin{array}{l}\vdash \{z = X \wedge x = Y\} \\ \quad y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
}{
\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}
}
$$

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:

$$
\dfrac{
  \dfrac{}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; \\ \quad \{z = X \wedge y = Y\}\end{array}}
  \quad
  \dfrac{}{\begin{array}{l}\vdash \{z = X \wedge y = Y\} \\ \quad x = y; \\ \quad \{z = X \wedge x = Y\}\end{array}}
}{
  \begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; \\ \quad \{z = X \wedge x = Y\}\end{array}
}
\quad
\dfrac{}{\begin{array}{l}\vdash \{z = X \wedge x = Y\} \\ \quad y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
$$

$$
\dfrac{\cdots}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
$$

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)!}
```

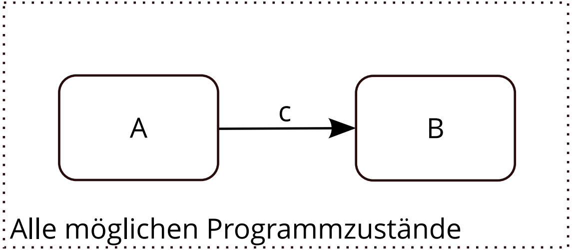
- Welche Zusicherungen gelten

  1. an der Stelle (A)?

  2. an der Stelle (B)?

Regeln des Floyd-Hoare-Kalküls: Weakening

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$

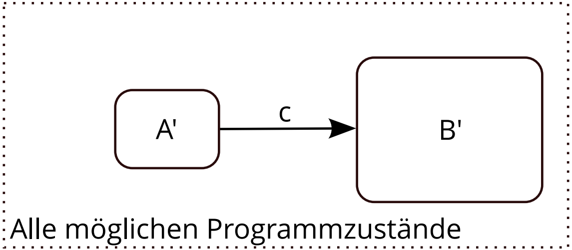


- $\models \{A\}\, c\, \{B\}$: Ausführung von c startet in Zustand, in dem A gilt, und endet (ggf) in Zustand, in dem B gilt.
- Zustandsprädikate beschreiben Mengen von Zuständen:

$$\{\sigma \in \Sigma | \sigma \models^{I} P\} \subseteq \{\sigma \in \Sigma | \sigma \models^{I} Q\} \text{ gdw. } P \Longrightarrow Q$$

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

$$\frac{\vdash \{A \land b\}\, c\, \{A\}}{\vdash \{A\}\ \textbf{while}(b)\ c\, \{A \land \neg b\}}$$

- 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

$$\frac{}{\vdash \{P[e/x]\}\, x = e\, \{P\}}$$

$$\frac{\vdash \{A \land b\}\, c_0\, \{B\} \qquad \vdash \{A \land \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

$$\frac{\vdash \{A \land b\}\, c\, \{A\}}{\vdash \{A\}\ \textbf{while}(b)\ c\, \{A \land \neg b\}}$$

$$\frac{}{\vdash \{A\}\, \{\,\}\, \{A\}} \qquad \frac{\vdash \{A\}\, c_1\, \{B\} \qquad \vdash \{B\}\, c_2\, \{C\}}{\vdash \{A\}\, c_1; c_2\, \{C\}}$$

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$

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).