

Korrekte Software: Grundlagen und Methoden

Vorlesung 5 vom 17.05.22

Die Floyd-Hoare-Logik I

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2022

10:20:57 2022-06-28

1 [33]



Fahrplan

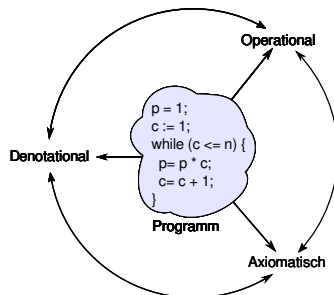
- ▶ Einführung
- ▶ Operationale Semantik
- ▶ Denotationale Semantik
- ▶ Äquivalenz der Operationalen und Denotationalen Semantik
- ▶ **Der Floyd-Hoare-Kalkül I**
- ▶ Der Floyd-Hoare-Kalkül II: Invarianten
- ▶ Korrektheit des Floyd-Hoare-Kalküls
- ▶ Strukturierte Datentypen
- ▶ Verifikationsbedingungen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

Korrekte Software

2 [33]



Drei Semantiken — Eine Sicht



Korrekte Software

3 [33]



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.
- ▶ Operationale/denotationale 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**.

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

Korrekte Software

4 [33]



Bob Floyd und Tony Hoare



Bildquelle: Stanford University

Robert Floyd
1936 – 2001



Bildquelle: Wikipedia

Sir Anthony Charles Richard Hoare
* 1934

Korrekte Software

5 [33]



Grundbausteine der Floyd-Hoare-Logik

- ▶ **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

```
// (A)
p = 1;
c = 1;
// (B)
while (c <= n) {
  // (C)
  p = p * c;
  c = c + 1;
  // (D)
}
// (E)
```

Korrekte Software

6 [33]



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)?

Korrekte Software

7 [33]



Auf dem Weg zur Floyd-Hoare-Logik

- ▶ Kern der Floyd-Hoare-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.

Korrekte Software

8 [33]



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.

Korrekte Software

9 [33]



Zusicherungen (Assertions)

- Erweiterung von **Aexp** und **Bexp** durch
 - **Logische** Variablen **Var** $v := N, M, L, U, V, X, Y, Z$
 - Definierte Funktionen und Prädikate über **Aexp** $n!, x^y, \dots$
 - Implikation und Quantoren $b_1 \longrightarrow b_2, \forall v. b, \exists v. b$
- Formal:
 - Aexp** $a ::= \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 / a_2 \mid f(e_1, \dots, e_n)$
 - Assn** $b ::= \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 < a_2 \mid ! b \mid b_1 \&\& b_2 \mid b_1 \parallel b_2 \mid b_1 \longrightarrow b_2 \mid p(e_1, \dots, e_n) \mid \backslash \text{forall } v. b \mid \backslash \text{exists } v. b$
 - Assn** $b ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid b_1 \longrightarrow b_2 \mid p(e_1, \dots, e_n) \mid \forall v. b \mid \exists v. b$

Korrekte Software

10 [33]



Denotationale Semantik von Zusicherungen

- Erste Näherung: Funktion

$$\llbracket a \rrbracket_A : \mathbf{Aexpv} \rightarrow (\Sigma \rightarrow \mathbb{Z})$$

$$\llbracket b \rrbracket_B : \mathbf{Assn} \rightarrow (\Sigma \rightarrow \mathbb{B})$$

- **Konservative** Erweiterung von $\llbracket a \rrbracket_A : \mathbf{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{Z})$
- Aber: was ist mit den logischen Variablen?
- Zusätzlicher Parameter **Belegung** der logischen Variablen $l : \mathbf{Var} \rightarrow \mathbb{Z}$

$$\llbracket a \rrbracket_A : \mathbf{Aexpv} \rightarrow (\mathbf{Var} \rightarrow \mathbb{Z}) \rightarrow (\Sigma \rightarrow \mathbb{Z})$$

$$\llbracket b \rrbracket_B : \mathbf{Assn} \rightarrow (\mathbf{Var} \rightarrow \mathbb{Z}) \rightarrow (\Sigma \rightarrow \mathbb{B})$$

- Bemerkung: $l : \mathbf{Var} \rightarrow \mathbb{Z}$ ist immer eine **totale Funktion** im Gegensatz zu einem Zustand.

Korrekte Software

11 [33]



Denotat von Aexp

$$\llbracket a \rrbracket_A : \mathbf{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{Z})$$

$$\llbracket n \rrbracket_A = \{(\sigma, \llbracket n \rrbracket) \mid \sigma \in \Sigma\}$$

$$\llbracket x \rrbracket_A = \{(\sigma, \sigma(x)) \mid \sigma \in \Sigma, x \in \text{Dom}(\sigma)\}$$

$$\llbracket a_0 + a_1 \rrbracket_A = \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A\}$$

$$\llbracket a_0 - a_1 \rrbracket_A = \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A\}$$

$$\llbracket a_0 * a_1 \rrbracket_A = \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A\}$$

$$\llbracket a_0 / a_1 \rrbracket_A = \{(\sigma, n_0 \div n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A \wedge n_1 \neq 0\}$$

Korrekte Software

12 [33]



Denotat von Aexpv

$$\llbracket a \rrbracket_A : \mathbf{Aexpv} \rightarrow (\mathbf{Var} \rightarrow \mathbb{Z}) \rightarrow (\Sigma \rightarrow \mathbb{Z})$$

Sei $l : \mathbf{Var} \rightarrow \mathbb{Z}$ eine beliebige Belegung

$$\llbracket n \rrbracket_A^l = \{(\sigma, \llbracket n \rrbracket) \mid \sigma \in \Sigma\}$$

$$\llbracket x \rrbracket_A^l = \{(\sigma, \sigma(x)) \mid \sigma \in \Sigma, x \in \text{Dom}(\sigma)\}$$

$$\llbracket a_0 + a_1 \rrbracket_A^l = \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A^l \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A^l\}$$

$$\llbracket a_0 - a_1 \rrbracket_A^l = \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A^l \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A^l\}$$

$$\llbracket a_0 * a_1 \rrbracket_A^l = \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A^l \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A^l\}$$

$$\llbracket a_0 / a_1 \rrbracket_A^l = \{(\sigma, n_0 \div n_1) \mid (\sigma, n_0) \in \llbracket a_0 \rrbracket_A^l \wedge (\sigma, n_1) \in \llbracket a_1 \rrbracket_A^l \wedge n_1 \neq 0\}$$

$$\llbracket X \rrbracket_A^l = \{(\sigma, l(X)) \mid \sigma \in \Sigma, X \in V\}$$

Korrekte Software

13 [33]



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 l erfüllt ($\sigma \models^l b$), gdw

$$\llbracket b \rrbracket_B^l(\sigma) = \text{true}$$

Korrekte Software

14 [33]



Arbeitsblatt 5.2: Zusicherungen

Betrachte folgende Zusicherung:

$$a \equiv \frac{2 \cdot x = X}{p} \longrightarrow \frac{x \leq X}{q}$$

Gegeben folgende Belegungen $l_1, \dots, l_3$ und Zustände $s_1, \dots, s_3$:

$$s_1 = \langle x \mapsto 0 \rangle, s_2 = \langle x \mapsto 1 \rangle, s_3 = \langle x \mapsto 5 \rangle$$

$$l_1 = \langle X \mapsto 0 \rangle, l_2 = \langle X \mapsto 2 \rangle, l_3 = \langle X \mapsto 10 \rangle$$

Unter welchen Belegungen und Zuständen ist a wahr?

	l_1			l_2			l_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?

Korrekte Software

15 [33]



Floyd-Hoare-Tripel

Partielle Korrektheit ($\models \{P\} c \{Q\}$)

$\{P\} c \{Q\}$ ist **partiell korrekt**, wenn für alle Belegungen l 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 l Q .

$$\models \{P\} c \{Q\} \iff \forall l. \forall \sigma. \sigma \models^l P \wedge \exists \tau. (\sigma, \tau) \in \llbracket c \rrbracket_c \implies \tau \models^l Q$$

- 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 alle Belegungen l und alle Zustände σ , die P erfüllen, die Ausführung von c mit σ in einem Zustand τ terminiert, und τ mit der Belegung l erfüllt Q .

$$\models [P] c [Q] \iff \forall l. \forall \sigma. \sigma \models^l P \implies \exists \tau. (\sigma, \tau) \in \llbracket c \rrbracket_c \wedge \tau \models^l Q$$

Korrekte Software

16 [33]



Beispiele

- Folgendes **gilt**:

$$\models \{true\} \text{ while}(1) \{ \} \{true\}$$

- Folgendes gilt **nicht**:

$$\models [true] \text{ while}(1) \{ \} [true]$$

- Folgende **gelten**:

$$\models \{false\} \text{ while } (1) \{ \} \{true\}$$

$$\models [false] \text{ while } (1) \{ \} [true]$$

Wegen *ex falso quodlibet*: $false \implies \phi$

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}
```

Gültigkeit und Herleitbarkeit

- Semantische Gültigkeit**: $\models \{P\} c \{Q\}$

- Definiert durch denotationale Semantik:

$$\models \{P\} c \{Q\} \iff \forall I. \forall \sigma. \sigma \models^I P \wedge \exists \tau. (\sigma, \tau) \in \llbracket c \rrbracket \implies \tau \models^I Q$$

- 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

$$\frac{\vdash \{P_1\} c_1 \{Q_1\} \dots \vdash \{P_n\} c_n \{Q_n\}}{\vdash \{P\} c \{Q\}}$$

- Für jedes Konstrukt der Programmiersprache gibt es eine Regel.

Regeln des Floyd-Hoare-Kalküls: Zuweisung

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

- 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

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

- Hier wird eine Zwischenzusicherung B benötigt.

$$\vdash \{A\} \{ \} \{A\}$$

- Trivial.

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:

$$\frac{\frac{\vdash \{x = X \wedge y = Y\} \quad \vdash \{z = X \wedge y = Y\}}{\vdash \{x = X; z = x; x = y; \{z = X \wedge x = Y\}\}} \quad \frac{\vdash \{z = X \wedge y = Y\} \quad \vdash \{z = X \wedge x = Y\}}{\vdash \{z = X; y = z; \{y = X \wedge x = Y\}\}}}{\vdash \{x = X \wedge y = Y\} \quad \vdash \{z = X \wedge x = Y\} \quad \vdash \{y = X \wedge x = Y\}} \quad \vdash \{x = X \wedge y = Y\} \quad \vdash \{y = X \wedge x = Y\}$$

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:

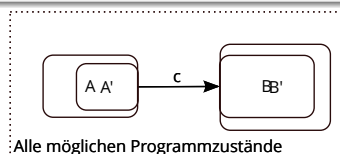
```
// (B)
p = p * c;
// (A)
c = c + 1;
// {p = (c - 1)!}
```

► Welche Zusicherungen gelten

- ❶ an der Stelle (A)?
- ❷ an der Stelle (B)?

Regeln des Floyd-Hoare-Kalküls: Weakening

$$\frac{A' \Rightarrow A \quad \vdash \{A\} c \{B\} \quad B \Rightarrow B'}{\vdash \{A'\} c \{B'\}}$$



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

$$\{\sigma \in \Sigma \mid \sigma \models P\} \subseteq \{\sigma \in \Sigma \mid \sigma \models Q\} \text{ gdw. } P \Rightarrow Q$$

- Wir können A zu A' einschränken ($A' \subseteq A$ oder $A' \Rightarrow A$), oder B zu B' vergrößern ($B \subseteq B'$ oder $B \Rightarrow B'$), und erhalten $\vdash \{A'\} c \{B'\}$.

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?

- ❶ (C)?
- ❷ (B)?
- ❸ (A)?

Regeln des Floyd-Hoare-Kalküls: Fallunterscheidung

$$\frac{\vdash \{A \wedge b\} c_0 \{B\} \quad \vdash \{A \wedge \neg b\} c_1 \{B\}}{\vdash \{A\} \text{if}(b) c_0 \text{else } c_1 \{B\}}$$

- 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

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{while}(b) c \{A \wedge \neg b\}}$$

- 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: $P \Rightarrow P_2[e/x]$, $P_2 \Rightarrow P_3$, $P_3 \wedge x < n \Rightarrow P_4$, $P_3 \wedge \neg(x < n) \Rightarrow Q$.

Überblick: die Regeln des Floyd-Hoare-Kalküls

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

$$\frac{\vdash \{A \wedge b\} c_0 \{B\} \quad \vdash \{A \wedge \neg b\} c_1 \{B\}}{\vdash \{A\} \text{if}(b) c_0 \text{else } c_1 \{B\}}$$

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{while}(b) c \{A \wedge \neg b\}}$$

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

$$\frac{A' \Rightarrow A \quad \vdash \{A\} c \{B\} \quad B \Rightarrow B'}{\vdash \{A'\} c \{B'\}}$$

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).