

# Korrekte Software: Grundlagen und Methoden

Vorlesung 5 vom 11/18.05.21

Die Floyd-Hoare-Logik I

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2021

# Fahrplan

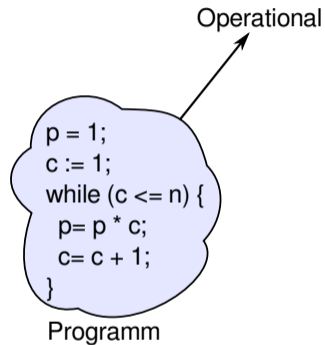
- ▶ Einführung
- ▶ Operationale Semantik
- ▶ Denotationale Semantik
- ▶ Äquivalenz der Operationalen und Denotationalen Semantik
- ▶ Der Floyd-Hoare-Kalkül I
- ▶ Der Floyd-Hoare-Kalkül II: Invarianten
- ▶ Korrektheit des Floyd-Hoare-Kalküls
- ▶ Strukturierte Datentypen
- ▶ Verifikationsbedingungen
-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

# Drei Semantiken — Eine Sicht

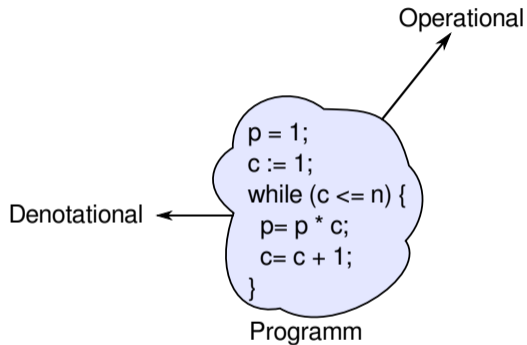
```
p = 1;  
c := 1;  
while (c <= n) {  
  p = p * c;  
  c = c + 1;  
}
```

Programm

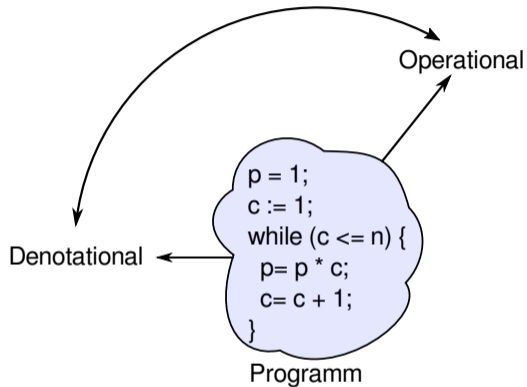
# Drei Semantiken — Eine Sicht



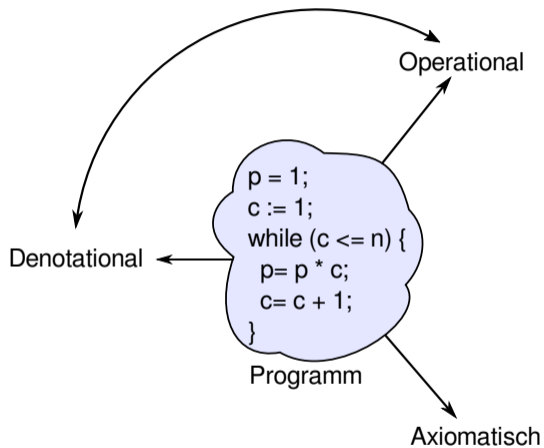
# Drei Semantiken — Eine Sicht



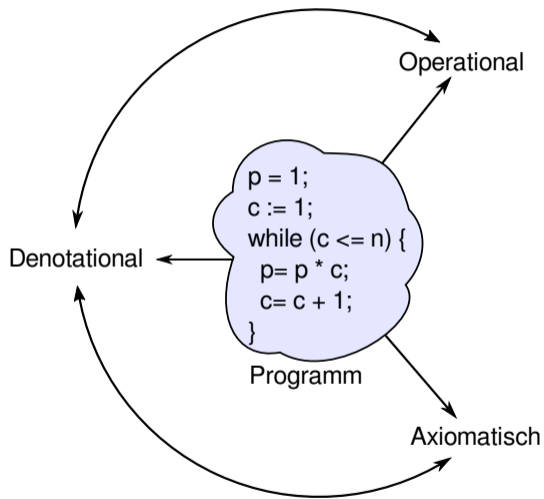
# Drei Semantiken — Eine Sicht



# Drei Semantiken — Eine Sicht

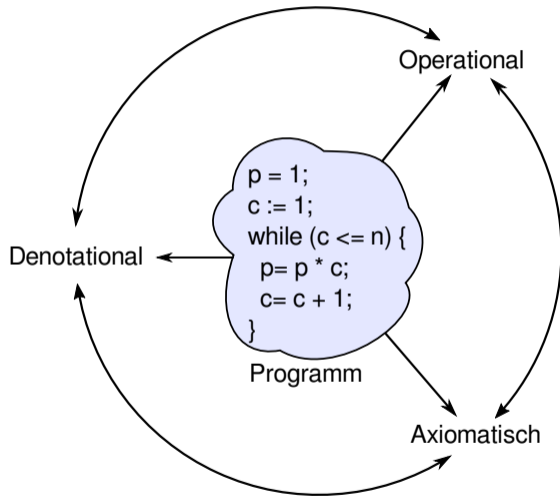


# Drei Semantiken — Eine Sicht





# 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.
- ▶ Operationale/denotationale Semantik nicht für **Korrektheitsbeweise** geeignet: Ausdrücke werden zu groß, skaliert nicht — **Abstraktion** nötig.

```
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.
- ▶ 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;  
}
```

# 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

- ▶ **Zusicherungen** über den Zustand
- ▶ Beispiele:
  - ▶ (B): Hier gilt  $p = c = 1$
  - ▶ (D): Hier ist  $c$  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

- ▶ **Zusicherungen** über den Zustand
- ▶ Beispiele:
  - ▶ (B): Hier gilt  $p = c = 1$
  - ▶ (D): Hier ist  $c$  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 ein „Eingabevariable“, der Wert am Anfang des Programmes ist relevant;
  - ▶  $p$  ist eine „Ausgabevariable“, der Wert am Ende des Programmes 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)
```

## 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-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

# 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 **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**
  - ▶ Definierte Funktionen und Prädikate über **Aexp**
  - ▶ Implikation und Quantoren
- ▶ Formal:

**Aexpv**  $a ::= \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 / a_2$   
 $\mid f(e_1, \dots, e_n)$

**Assn**  $b ::= \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 < a_2$   
 $\mid !b \mid b_1 \&\& b_2 \mid b_1 \parallel b_2$   
 $\mid b_1 \longrightarrow b_2 \mid p(e_1, \dots, e_n) \mid \backslash \mathbf{forall} \ v. \ b \mid \backslash \mathbf{exists} \ v. \ b$

$v ::= N, M, L, U, V, X, Y, Z$

$n!, x^y, \dots$

$b_1 \longrightarrow b_2, \forall v.. b, \exists v.. b$

# Zusicherungen (Assertions)

- ▶ Erweiterung von **Aexp** and **Bexp** durch
  - ▶ **Logische** Variablen **Var**
  - ▶ Definierte Funktionen und Prädikate über **Aexp**
  - ▶ Implikation und Quantoren
- ▶ Formal:

**Aexpv**  $a ::= \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \mid a_1 / a_2$   
 $\mid f(e_1, \dots, e_n)$

**Assn**  $b ::= true \mid false \mid a_1 = a_2 \mid a_1 \leq a_2$   
 $\mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2$   
 $\mid b_1 \longrightarrow b_2 \mid p(e_1, \dots, e_n) \mid \forall v. b \mid \exists v. b$

$v ::= N, M, L, U, V, X, Y, Z$

$n!, x^y, \dots$

$b_1 \longrightarrow b_2, \forall v. b, \exists v. b$

# Denotationale Semantik von Zusicherungen

- ▶ Erste Näherung: Funktion

$$\llbracket a \rrbracket_{\mathcal{A}} : \mathbf{Aexpv} \rightarrow (\Sigma \rightarrow \mathbb{Z})$$

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

- ▶ **Konservative** Erweiterung von  $\llbracket a \rrbracket_{\mathcal{A}} : \mathbf{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{Z})$

- ▶ Aber: was ist mit den logischen Variablen?



# Denotationale Semantik von Zusicherungen

- ▶ Erste Näherung: Funktion

$$\llbracket a \rrbracket_{\mathcal{A}} : \mathbf{Aexpv} \rightarrow (\Sigma \rightarrow \mathbb{Z})$$

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

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

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

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

# Erfüllung von Zusicherungen

- ▶ Wann gilt eine Zusicherung  $b \in \mathbf{Assn}$  in einem Zustand  $\sigma$ ?
  - ▶ Auswertung (denotationale Semantik) ergibt *true*
  - ▶ Belegung ist zusätzlicher Parameter

## Erfülltheit von Zusicherungen

$b \in \mathbf{Assn}$  ist in Zustand  $\sigma$  mit Belegung  $l$  erfüllt ( $\sigma \models^l b$ ), gdw

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

## Arbeitsblatt 5.2: Zusicherungen

Betrachte folgende Zusicherung:

$$a \equiv \underbrace{2 \cdot x = X}_p \longrightarrow \underbrace{x < X}_q$$

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?

	$l_1$			$l_2$			$l_3$		
	$p$	$q$	$a$	$p$	$q$	$q$	$p$	$q$	$a$
$s_1$									
$s_2$									
$s_3$									

Wie kann man  $a$  so ändern, dass  $a$  für **alle** Belegungen und Zustände wahr ist?

# Floyd-Hoare-Tripel

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

$c$  ist **partiell korrekt**, wenn für alle Zustände  $\sigma$ , die  $P$  erfüllen, gilt:  
**wenn** die Ausführung von  $c$  mit  $\sigma$  in  $\tau$  terminiert, **dann** erfüllt  $\tau$   $Q$ .

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

- ▶ Gleiche Belegung der logischen Variablen in  $P$  und  $Q$  erlaubt **Vergleich** zwischen Zuständen

## Totale Korrektheit ( $\models [P] c [Q]$ )

$c$  ist **total korrekt**, wenn für alle Zustände  $\sigma$ , die  $P$  erfüllen, die Ausführung von  $c$  mit  $\sigma$  in  $\tau$  terminiert, und  $\tau$  erfüllt  $Q$ .

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

# Beispiele

- ▶ Folgendes **gilt**:

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

# Beispiele

- ▶ Folgendes **gilt**:

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

- ▶ Folgendes gilt **nicht**:

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

# 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}
```



## 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}
```

# Gültigkeit und Herleitbarkeit

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

► Definiert durch denotationale Semantik:

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

► Problem: müssten Semantik von  $c$  ausrechnen

# Gültigkeit und Herleitbarkeit

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

- ▶ Definiert durch denotationale Semantik:

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

- ▶ 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

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

- ▶ 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

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

- ▶ 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

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

- ▶ 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

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

- ▶ 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

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

- ▶ 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.

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

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

Herleitung:

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

## 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:

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



# 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:

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

## 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:

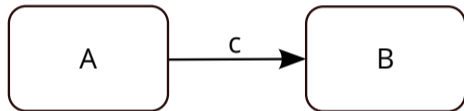
```
// (B)  
p= p* c;  
// (A)  
c= c+ 1;  
// {p = (c - 1)!}
```

► Welche Zusicherungen gelten

- 1 an der Stelle (A)?
- 2 an der Stelle (B)?

## Regeln des Floyd-Hoare-Kalküls: Weakening

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

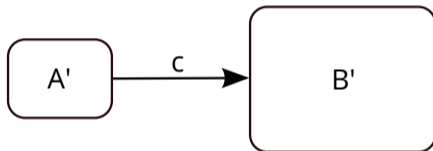


Alle möglichen Programmzustände

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

## Regeln des Floyd-Hoare-Kalküls: Weakening

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



Alle möglichen Programmzustände

- ▶  $\models \{A\} c \{B\}$ : Ausführung von  $c$  startet in Zustand, in dem  $A$  gilt, und endet (ggf) in Zustand, in dem  $B$  gilt.
- ▶ Zustandsprädikate beschreiben Mengen von Zuständen:  $P \subseteq Q$  gdw.  $P \implies Q$ .
- ▶ Wir können  $A$  zu  $A'$  einschränken ( $A' \subseteq A$  oder  $A' \implies A$ ), oder  $B$  zu  $B'$  vergrößern ( $B \subseteq B'$  oder  $B \implies B'$ ), und erhalten  $\models \{A'\} c \{B'\}$ .

## Arbeitsblatt 5.5: Ein zweiter Beweis

Wir betrachten noch einmal das Vertauschen 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 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 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 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

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

- ▶ 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

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

- ▶ Iteration korrespondiert zu **Induktion**.
- ▶ Bei (natürlicher) Induktion zeigen wir, dass die **gleiche** Eigenschaft  $P$  für 0 gilt, und dass wenn sie für  $P(n)$  gilt, daraus folgt, dass sie für  $P(n+1)$  gilt.
- ▶ 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:  $P \implies P_2[e/x]$ ,  $P_2 \implies P_3$ ,  $P_3 \wedge x < n \implies P_4$ ,  $P_3 \wedge \neg(x < n) \implies Q$ .

# Überblick: die Regeln des Floyd-Hoare-Kalküls

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

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

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

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

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

# Zusammenfassung Floyd-Hoare-Logik

- ▶ Die Logik abstrahiert über konkrete Systemzustände durch **Zusicherungen**
- ▶ Zusicherungen sind boolesche 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).