

Korrekte Software: Grundlagen und Methoden
Vorlesung 5 vom 07.05.19
Die Floyd-Hoare-Logik

Serge Autexier, Christoph Lüth

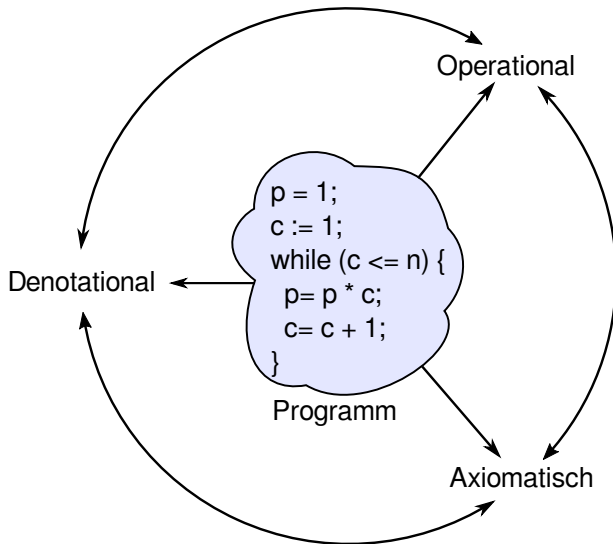
Universität Bremen

Sommersemester 2019

Fahrplan

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

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**?

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

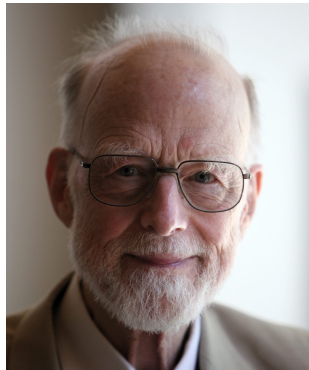
- ▶ Operationale/denotationale Semantik nicht für **Korrektheitsbeweise** geeignet: Ausdrücke werden zu groß, skaliert nicht.
- ▶ **Abstraktion** nötig.
- ▶ Grundidee: **Zusicherungen** über den Zustand an bestimmten Punkten im Programmablauf.

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 am Punkt(A) der Wert von $n \geq 0$, dann ist am Punkt (E) $p = n!$.

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

Grundbausteine der Floyd-Hoare-Logik

- ▶ **Logische Variablen** (zustandsfrei) und **Programmvariablen**
- ▶ **Zusicherungen** mit logischen und Programmvariablen
- ▶ **Floyd-Hoare-Tripel** $\{P\} c \{Q\}$
 - ▶ Vorbedingung P (Zusicherung)
 - ▶ Programm c
 - ▶ Nachbedingung Q (Zusicherung)
- ▶ Floyd-Hoare-Logik abstrahiert Programme durch logische 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**

$n!, \sum_{i=1}^n i, \dots$

- Implikation und Quantoren

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

- Formal:

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

Assn $b ::=$

$\mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 != a_2 \mid a_1 <= a_2$

$\mid !b \mid b_1 \wedge b_2 \mid b_1 \parallel b_2$

$\mid b_1 \longrightarrow b_2 \mid p(e_1, \dots, e_n) \mid \mathbf{forall} \ v; \ b \mid \mathbf{exists} \ v; \ b$

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**

$n!, \sum_{i=1}^n i, \dots$

- Implikation und Quantoren

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

- Formal:

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

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

Erfüllung von Zusicherungen

- ▶ Wann gilt eine Zusicherung $b \in \mathbf{Assn}$ in einem Zustand σ ?
 - ▶ Auswertung (denotationale Semantik) ergibt *true*
 - ▶ **Aber:** was ist mit den logischen Variablen?

Erfüllung von Zusicherungen

- ▶ Wann gilt eine Zusicherung $b \in \mathbf{Assn}$ in einem Zustand σ ?
 - ▶ Auswertung (denotationale Semantik) ergibt *true*
 - ▶ **Aber:** was ist mit den logischen Variablen?
- ▶ **Belegung** der logischen Variablen: $l : \mathbf{Var} \rightarrow \mathbb{Z}$
- ▶ Semantik von b unter der Belegung l : $\mathcal{B}_v[[b]]^l, \mathcal{A}_v[[a]]^l$

Erfülltheit von Zusicherungen

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

$$\mathcal{B}_v[[b]]^l(\sigma) = \text{true}$$

Floyd-Hoare-Tripel

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

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

$$\models \{P\} c \{Q\} \iff \forall I. \forall \sigma. \sigma \models^I P \wedge \exists \sigma'. (\sigma, \sigma') \in \mathcal{C}[\![c]\!] \implies \sigma' \models^I Q$$

- Gleiche Belegung der logischen Variablen für P und Q .

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

c ist **total korrekt**, wenn für alle Zustände σ , die P erfüllen, die Ausführung von c mit σ in σ' terminiert, und σ' erfüllt Q .

$$\models [P] c [Q] \iff \forall I. \forall \sigma. \sigma \models^I P \implies \exists \sigma'. (\sigma, \sigma') \in \mathcal{C}[\![c]\!] \wedge \sigma' \models^I Q$$

Beispiele

► Folgendes **gilt**:

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

Beispiele

- ▶ Folgendes **gilt**:

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

- ▶ Folgendes gilt **nicht**:

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

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*: $false \implies \phi$

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

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

- ▶ 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:

$$\begin{array}{ll} @ \backslash \text{only} <2-> \{ \backslash \text{PCA} \{ 5 < 10 \} \backslash \text{Iff} @ \backslash \text{only} <10-> \{ \backslash \text{PCA} \{ x < 9 \} \backslash \text{Iff} x + \\ x = 5 & x = x + 1 \\ @ \backslash \text{PCA} \{ x < 10 \} @ & @ \backslash \text{only} <3-> \{ \backslash \text{PCA} \{ x < 10 \} \} @ \end{array}$$

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.

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 (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 .

Regeln des Floyd-Hoare-Kalküls: Sequenzierung

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

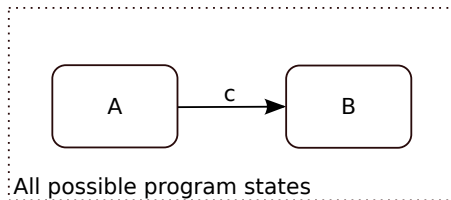
- Hier wird eine Zwischenzusicherung B benötigt.

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

- Trivial.

Regeln des Floyd-Hoare-Kalküls: Weakening

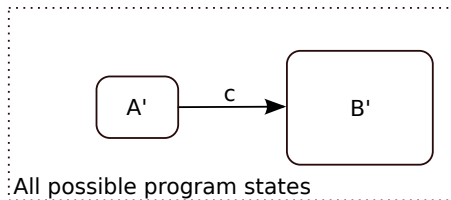
$$\frac{A' \implies A \quad \vdash \{A\} c \{B\} \quad B \implies B'}{\vdash \{A'\} c \{B'\}}$$



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

Regeln des Floyd-Hoare-Kalküls: Weakening

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



- ▶ $\models \{A\} c \{B\}$: 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 $\models \{A'\} c \{B'\}$.

Überblick: die Regeln des Floyd-Hoare-Kalküls

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

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

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

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

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

Einfache Beispiele

Sei p :

```
z = x;  
x = y;  
y = z;
```

Zu zeigen:

► p vertauscht x und y

Sei q :

```
if (x < y) {  
    z = x;  
} else {  
    z = y;  
}
```

Zu zeigen:

Einfache Beispiele

Sei p :

$z = x;$

$x = y;$

$y = z;$

Zu zeigen:

► p vertauscht x und y

► $\vdash \{x = X \wedge y = Y\}$

p
 $\{x = Y \wedge y = X\}$

Sei q :

```
if (x < y) {  
    z = x;  
} else {  
    z = y;  
}
```

Zu zeigen:

Einfache Beispiele

Sei p :

$z = x;$

$x = y;$

$y = z;$

Zu zeigen:

► p vertauscht x und y

► $\vdash \{x = X \wedge y = Y\}$

p

$\{x = Y \wedge y = X\}$

Sei q :

if $(x < y)$ {

$z = x;$

} **else** {

$z = y;$

}

Zu zeigen:

► q berechnet in z das Minimum von x und y

Einfache Beispiele

Sei p :

```
z = x;  
x = y;  
y = z;
```

Zu zeigen:

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

Sei q :

```
if (x < y) {  
    z = x;  
} else {  
    z = y;  
}
```

Zu zeigen:

- ▶ q berechnet in z das Minimum von x und y
- ▶ $\vdash \{true\}$
 q
 $\{z \leq x \wedge z \leq y\}$

Wie wir Floyd-Hoare-Beweise aufschreiben

```
// {P}
// {P1}
x = e;
// {P2}
// {P3}
while (x < n) {
  // {P3 ∧ x < n}
  // {P4}
  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.
- ▶ Implizite Anwendung der Sequenzenregel.
- ▶ Weakening wird notiert durch mehrere Zusicherungen, und muss **bewiesen** werden.
- ▶ Im Beispiel: $P \implies P_1$,
 $P_2 \implies P_3$, $P_3 \wedge x < n \implies P_4$,
 $P_3 \wedge \neg(x < n) \implies Q$.

Das einfache Beispiel in neuer Notation

```
// {x = X ∧ y = Y}  
// {y = Y ∧ x = X}  
z = x;  
// {y = Y ∧ z = X}  
x = y;  
// {x = Y ∧ z = X}  
y = z;  
// {x = Y ∧ y = X}
```

Das Fakultätsbeispiel

```
// {1 = 0! ∧ 0 ≤ n}
// {1 = (1 - 1)! ∧ 1 ≤ 1 ∧ 1 - 1 ≤ n}
p = 1;
// {p = (1 - 1)! ∧ 1 ≤ 1 ∧ 1 - 1 ≤ n}
c = 1;
// {p = (c - 1)! ∧ 1 ≤ c ∧ c - 1 ≤ n}
while (c ≤ n) {
    // {p = (c - 1)! ∧ 1 ≤ c ∧ c - 1 ≤ n ∧ c ≤ n}
    // {p * c = (c - 1)! * c ∧ 1 ≤ c ∧ c ≤ n}
    // {p * c = c! ∧ 1 ≤ c ∧ c ≤ n}
    // {p * c = ((c + 1) - 1)! ∧ 1 ≤ c + 1 ∧ (c + 1) - 1 ≤ n}
    p = p * c;
    // {p = ((c + 1) - 1)! ∧ 1 ≤ c + 1 ∧ (c + 1) - 1 ≤ n}
    c = c + 1;
    // {p = (c - 1)! ∧ 1 ≤ c ∧ c - 1 ≤ n}
}
// {p = (c - 1)! ∧ 1 ≤ c ∧ c - 1 ≤ n ∧ ¬(c ≤ n)}
// {p = (c - 1)! ∧ 1 ≤ c ∧ c - 1 ≤ n ∧ c > n}
// {p = (c - 1)! ∧ 1 ≤ c ∧ c - 1 = n}
// {p = n!}
```

Zusammenfassung Floyd-Hoare-Logik

- ▶ Die Logik abstrahiert über konkrete Systemzustände durch **Zusicherungen** (Hoare-Tripel $\{P\} c \{Q\}$).
- ▶ Zusicherungen sind boolsche Ausdrücke, angereichert durch logische Variablen.
- ▶ 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).