

Korrekte Software: Grundlagen und Methoden
Vorlesung 5 vom 19.05.20
Die Floyd-Hoare-Logik

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2020

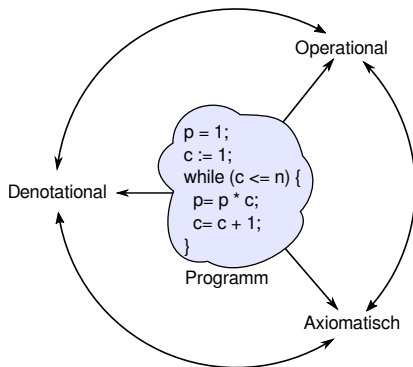


Fahrplan

- ▶ 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
- ▶ Ausblick und Rückblick



Drei Semantiken — Eine Sicht



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.
- ▶ Grundidee: **Zusicherungen** über den Zustand an bestimmten Punkten im Programmablauf.

```
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 ist 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)
```



Arbeitsblatt 5.1: Was berechnet dieses Programm?

```
// (A)
x = 1;
c = 1;
// (B)
while (c <= y) {
  x = 2 * x;
  // (C)
  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
- ▶ Wir benötigen auch **zustandsfreie** Aussagen, um Zustände **vergleichen** zu können.
- ▶ Die Logik **abstrahiert** den Effekt von Programmen durch **Vor- und Nachbedingung**.



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 von Programmen zu logischen Formeln.



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 $b_1 \longrightarrow b_2, \forall v. \dots b, \exists v. \dots b$
- ▶ Formal:
 - Aexpv** $a ::= \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2$
 $\mid f(e_1, \dots, e_n)$
 - Assn** $b ::=$
 $\mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1! = a_2 \mid a_1 <= a_2$
 $\mid ! b \mid b_1 \&\& b_2 \mid b_1 \parallel b_2$
 $\mid b_1 \longrightarrow b_2 \mid p(e_1, \dots, e_n) \mid \backslash \text{forall } v. b \mid \backslash \text{exists } v. b$
 - Assn** $b ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 \leq a_2$
 $\mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2$
 $\mid b_1 \longrightarrow b_2 \mid p(e_1, \dots, e_n) \mid \forall v. b \mid \exists v. b$



Denotationale Semantik von Zusicherungen

- ▶ Erste Näherung: Funktion
 - $\llbracket a \rrbracket_A : \mathbf{Aexpv} \rightarrow (\Sigma \rightarrow \mathbb{Z})$
 - $\llbracket b \rrbracket_B : \mathbf{Assn} \rightarrow (\Sigma \rightarrow \mathcal{B})$
- ▶ **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 $l : \mathbf{Var} \rightarrow \mathbb{Z}$

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

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



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

$$\llbracket b \rrbracket_{B,l}(\sigma) = \text{true}$$



Arbeitsblatt 5.2: Zusicherungen

Betrachte folgende Zusicherung:

$$a \equiv x = 2 \cdot X \longrightarrow x > X$$

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
s_1			
s_2			
s_3			

Fügen Sie eine zusätzliche Bedingung hinzu, so 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 σ , die P erfüllen, gilt: **wenn** die Ausführung von c mit σ in τ terminiert, **dann** erfüllt τ Q .

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

- ▶ 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 σ , die P erfüllen, die Ausführung von c mit σ in τ terminiert, und τ erfüllt Q .

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



Beispiele

- ▶ Folgendes **gilt**: $\models \{\text{true}\} \text{while}(1) \{\text{true}\}$

- ▶ Folgendes gilt **nicht**: $\not\models [\text{true}] \text{while}(1) [\text{true}]$

- ▶ Folgende **gelten**:
 $\models \{\text{false}\} \text{while}(1) \{\text{true}\}$
 $\models [\text{false}] \text{while}(1) [\text{true}]$

Wegen *ex falso quodlibet*: $\text{false} \implies \phi$



Gültigkeit und Herleitbarkeit

- ▶ **Semantische Gültigkeit**: $\models \{P\} c \{Q\}$
 - ▶ Definiert durch denotationale Semantik:
 $\models \{P\} c \{Q\} \iff \forall l. \forall \sigma. \sigma \models^l P \wedge \exists \tau. (\sigma, \tau) \in \llbracket c \rrbracket \implies \tau \models^l Q$
 - ▶ 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

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

- 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:

$$\begin{array}{ll} \frac{}{\vdash \{(x < 10)[5/x]\} x = 5 \{x < 10\}} & \frac{}{\vdash \{x + 1 < 10\} x = x + 1 \{x < 10\}} \end{array}$$



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.

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

- 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:

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



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



Arbeitsblatt 5.3: Ein erster Beweis

Betrachte den Rumpf des Fakultätsprogramms:

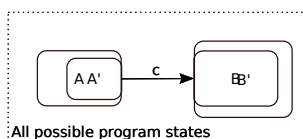
```
// (B)
p = p * c;
// (A)
c = c + 1;
// {p = (c - 1)!}
```

- Welche Zusicherungen gelten
- i** an der Stelle (A)?
- ii** an der Stelle (B)?



Regeln des Floyd-Hoare-Kalküls: Weakening

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



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



Regeln des Floyd-Hoare-Kalküls: Fallunterscheidung

$$\frac{\vdash \{A \wedge b\} c_0 \{B\} \quad \vdash \{A \wedge \neg b\} c_1 \{B\}}{\vdash \{A\} \text{if } (b) \text{ c}_0 \text{ else } \text{c}_1 \{B\}}$$

- 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 5.4: Ein zweiter Beweis

Betrachte folgendes Programm:

```
// (F)
if (x < y) {
  // (E)
  // ...
  z = x;
  // (C)
} else {
  // (D)
  // ...
  z = y;
  // (B)
}
// (A)
```

- 1 Was berechnet dieses Programm?
 - 2 Wie spezifizieren wir das?
 - 3 Wie beweisen wir die Gültigkeit?
- ▶ Die Spezifikation wird zur Nachbedingung (A)
 - ▶ Wir notieren Weakening durch aufeinanderfolgende Bedingungen:


```
// {x < 9}
// {x + 1 < 10}
```

- ▶ Welche Zusicherungen müssen an den Stellen (A) – (F) gelten?
- ▶ Wo müssen wir 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 (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}
// {P1}
x = e;
// {P2}
// {P3}
while (x < n) {
  // {P3 \wedge x < n}
  // {P4}
  z = a;
  // {P3}
}
// {P3 \wedge \neg(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: $P \Rightarrow P_1$,
 $P_2 \Rightarrow P_3$, $P_3 \wedge x < n \Rightarrow P_4$,
 $P_3 \wedge \neg(x < n) \Rightarrow Q$.



Das Fakultätsbeispiel (I)

```
// {1 = 0!}
// {1 = (1-1)!}
p = 1;
// {p = (1-1)!}
c = 1;
// {p = (c-1)!}
while (c <= n) {
  // {p = (c-1)! \wedge c <= n}
  // {p * c = (c-1)! * c}
  // {p * c = c!}
  // {p * c = ((c+1)-1)!}
  p = p * c;
  // {p = ((c+1)-1)!}
  c = c + 1;
  // {p = (c-1)!}
}
// {p = (c-1)! \wedge \neg(c <= n)}
// {p = (c-1)! \wedge c-1 > n}
// ??
// {p = n!}
```



Das Fakultätsbeispiel (II)

```
// {1 = 0! \wedge 0 <= n}
// {1 = (1-1)! \wedge 1-1 <= n}
p = 1;
// {p = (1-1)! \wedge 1-1 <= n}
c = 1;
// {p = (c-1)! \wedge c-1 <= n}
while (c <= n) {
  // {p = (c-1)! \wedge c-1 <= n \wedge c <= n}
  // {p * c = (c-1)! * c \wedge c <= n} !!!
  // {p * c = c! \wedge c <= n}
  // {p * c = ((c+1)-1)! \wedge (c+1)-1 <= n}
  p = p * c;
  // {p = ((c+1)-1)! \wedge (c+1)-1 <= n}
  c = c + 1;
  // {p = (c-1)! \wedge c-1 <= n}
}
// {p = (c-1)! \wedge c-1 <= n \wedge \neg(c <= n)}
// {p = (c-1)! \wedge c-1 <= n \wedge c > n}
// {p = (c-1)! \wedge c-1 <= n \wedge c-1 > n}
// {p = n!}
```



Das Fakultätsbeispiel (komplett)

```
// {1 = 0! \wedge 0 <= n}
// {1 = (1-1)! \wedge 1 <= 1 \wedge 1-1 <= n}
p = 1;
// {p = (1-1)! \wedge 1 <= 1 \wedge 1-1 <= n}
c = 1;
// {p = (c-1)! \wedge 1 <= c \wedge c-1 <= n}
while (c <= n) {
  // {p = (c-1)! \wedge 1 <= c \wedge c-1 <= n \wedge c <= n}
  // {p * c = (c-1)! * c \wedge 1 <= c \wedge c <= n}
  // {p * c = c! \wedge 1 <= c \wedge c <= n}
  // {p * c = ((c+1)-1)! \wedge 1 <= c+1 \wedge (c+1)-1 <= n}
  p = p * c;
  // {p = ((c+1)-1)! \wedge 1 <= c+1 \wedge (c+1)-1 <= n}
  c = c + 1;
  // {p = (c-1)! \wedge 1 <= c \wedge c-1 <= n}
}
// {p = (c-1)! \wedge 1 <= c \wedge c-1 <= n \wedge \neg(c <= n)}
// {p = (c-1)! \wedge c-1 <= n \wedge c > n}
// {p = (c-1)! \wedge c-1 <= n \wedge c-1 > n}
// {p = n!}
```



Arbeitsblatt 5.5: Exponents Revisited

Wir können jetzt das Programm vom Anfang korrekt beweisen:

```
/** ... */
x = 1;
c = 1;
/** x = 2^(c-1) &&& ... */
while (c <= y) {
  /** x = 2^(c-1) &&& ... &&& c <= y */
  /** ... */
  x = 2 * x;
  /** ... */
  c = c + 1;
  /** x = 2^(c-1) &&& ... */
}
/** { x = 2^y &&& ... &&& ! (c <= y) */
/** ... */
/** { x = 2^y } */
```

- ▶ Findet den Rest der Invariante, und
- ▶ Füllt den restlichen Teil aus.



Überblick: die Regeln des Floyd-Hoare-Kalküls

$$\frac{\vdash \{P[e/x]\} x = e \{P\}}{\vdash \{A \wedge b\} c_0 \{B\} \quad \vdash \{A \wedge \neg b\} c_1 \{B\}}{\vdash \{A\} \text{if}(b) c_0 \text{ else } c_1 \{B\}}$$

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

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

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



Zusammenfassung Floyd-Hoare-Logik

- ▶ Die Logik abstrahiert über konkrete Systemzustände durch **Zusicherungen** (Hoare-Tripel $\{P\} c \{Q\}$).
- ▶ Zusicherungen sind boolesche 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).