

Korrekte Software: Grundlagen und Methoden

Vorlesung 2 vom 10.04.17: Die Floyd-Hoare-Logik

Serge Autexier, Christoph Lüth

Universität Bremen

Sommersemester 2017



Organisatorisches

Die Übung am Donnerstag, 13.04.17, muss leider ausfallen!



Fahrplan

- ▶ Einführung
- ▶ Die Floyd-Hoare-Logik
- ▶ Operationale Semantik
- ▶ Denotationale Semantik
- ▶ Äquivalenz der Operationalen und Denotationalen Semantik
- ▶ Korrektheit des Hoare-Kalküls
- ▶ Vorwärts und Rückwärts mit Floyd und Hoare
- ▶ Funktionen und Prozeduren
- ▶ Referenzen und Speichermodelle
- ▶ Verifikationsbedingungen Revisited
- ▶ Vorwärtsrechnung Revisited
- ▶ Programmsicherheit und Frame Conditions
- ▶ Ausblick und Rückblick



Floyd-Hoare-Logik: Idee

- ▶ Was wird hier berechnet? $p = n!$
- ▶ Wie können wir das beweisen?
- ▶ Wir berechnen symbolisch, welche Werte Variablen über den Programmverlauf annehmen.
- ▶ Um Aussagen über ein Programm zu beweisen, benötigen wir einen Formalismus (eine Logik), die es erlaubt, Zusicherungen über Werte von Variablen zu bestimmten Ausführungszeitpunkten (im Programm) aufzuschreiben und zu beweisen.
- ▶ Dazu müssen wir auch die Bedeutung (Semantik) des Programmes definieren — die Frage "Was tut das Programm" mathematisch exakt beantworten.

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



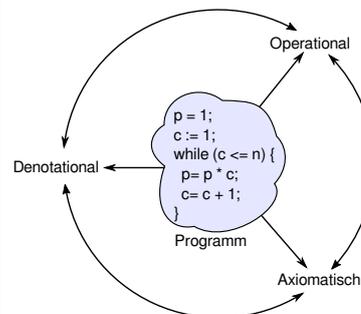
Semantik von Programmiersprachen

Drei wesentliche Möglichkeiten:

- ▶ **Operationale Semantik** beschreibt die Bedeutung eines Programmes, indem die Ausführung auf einer abstrakten Maschine beschrieben wird.
- ▶ **Denotationale Semantik** bildet jedes Programm auf ein mathematisches Objekt (meist ein partielle Funktion zwischen Systemzuständen) ab.
- ▶ **Axiomatische Semantik** beschreibt die Bedeutung eines Programmes durch Beweisregeln, mit welchem sich gültige Eigenschaften herleiten lassen. Das prominenteste Beispiel hierzu ist die Floyd-Hoare-Logik.



Drei Semantiken — Eine Sicht



- ▶ Jede Semantik ist eine Sicht auf das Programm.
- ▶ Diese Semantiken sollten alle äquivalent sein. Wir müssen sie also in Beziehung setzen, und zeigen dass sie die gleiche Sicht ergeben.
- ▶ Für die axiomatische Semantik (die Floyd-Hoare-Logik) ist das die Frage der Korrektheit der Regeln.



Floyd-Hoare-Logik

- ▶ Grundbaustein der Floyd-Hoare-Logik sind **Zusicherungen** der Form $\{P\} c \{Q\}$ (**Floyd-Hoare-Tripel**), wobei P die **Vorbedingung** ist, c das Programm, und Q die **Nachbedingung**.
- ▶ Die Logik hat sowohl **logische Variablen** (zustandsfrei), und **Programmvariablen** (deren Wert sich über die Programmausführung ändert).
- ▶ Die Floyd-Hoare-Logik hat ein wesentliches **Prinzip** und einen **Trick**.
- ▶ Das **Prinzip** ist die Abstraktion vom Programmzustand durch eine logische Sprache; insbesondere wird die **Zuweisung** durch **Substitution** modelliert.
- ▶ Der **Trick** behandelt Schleifen: Iteration im Programm entspricht Rekursion in der Logik. Ein Beweis ist daher induktiv, und benötigt eine Induktionsannahme — eine **Invariante**.



Unsere Programmiersprache

Wir betrachten einen Ausschnitt der Programmiersprache $C(C_0)$.

Ausbaustufe 1 kennt folgende Konstrukte:

- ▶ Typen: **int**;
- ▶ Ausdrücke: Variablen, Literale (für ganze Zahlen), arithmetische Operatoren (für ganze Zahlen), Relationen ($=$, $!$, $<$, $<=$, $>$, $>=$), boolesche Operatoren ($\&\&$, $\|\|$);
- ▶ Anweisungen:
 - ▶ Fallunterscheidung (**if...else...**), Iteration (**while**), Zuweisung, Blöcke;
 - ▶ Sequenzierung und leere Anweisung sind implizit



C0: Ausdrücke und Anweisungen

Aexp $a ::= \mathbf{N} \mid \mathbf{Loc} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid a_1 / a_2$
Bexp $b ::= \mathbf{0} \mid \mathbf{1} \mid a_1 == a_2 \mid a_1 != a_2$
 $\mid a_1 <= a_2 \mid !b \mid b_1 \&\& b_2 \mid b_1 \parallel b_2$
Exp $e ::= \mathbf{Aexp} \mid \mathbf{Bexp}$
Stmt $c ::= \mathbf{Loc} = \mathbf{Exp};$
 $\mid \mathbf{if} (b) c_1 \mathbf{else} c_2$
 $\mid \mathbf{while} (b) c$
 $\mid \{c^*\}$



Floyd-Hoare-Tripel

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

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

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 .

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



Regeln der Floyd-Hoare-Logik

- ▶ Die Floyd-Hoare-Logik 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 der Floyd-Hoare-Logik: 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:

$$\frac{}{\vdash \{5 < 10 \iff (x < 10)[5/x]\} x = 5 \{x < 10\}} \quad \frac{}{\vdash \{x < 9 \iff x + 1 < 10\} x = x + 1 \{x < 10\}}$$



Regeln der Floyd-Hoare-Logik: Fallunterscheidung und Sequenzierung

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

- ▶ 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.

$$\frac{\vdash \{A\} c \{B\} \quad \vdash \{B\} \{c_s\} \{C\}}{\vdash \{A\} \{c c_s\} \{C\}}$$

- ▶ Hier wird ein Zwischenzustand B benötigt.



Regeln der Floyd-Hoare-Logik: Iteration

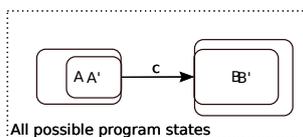
$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \mathbf{while}(b) c \{A \wedge \neg b\}}$$

- ▶ 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 .



Regeln der Floyd-Hoare-Logik: 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

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

$$\frac{}{\vdash \{A\} \{\} \{A\}} \quad \frac{\vdash \{A\} c \{B\} \quad \vdash \{B\} \{c_s\} \{C\}}{\vdash \{A\} \{c c_s\} \{C\}}$$

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

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \mathbf{while}(b) c \{A \wedge \neg b\}} \quad \frac{A' \implies A \quad \vdash \{A\} c \{B\} \quad B \implies B'}{\vdash \{A'\} c \{B'\}}$$



Eigenschaften der Floyd-Hoare-Logik

Korrektheit

Wenn $\vdash \{P\} c \{Q\}$, dann $\models \{P\} c \{Q\}$

- ▶ Wenn wir eine Korrektheitsaussage herleiten können, dann gilt sie auch.
- ▶ Wird gezeigt, indem wir $\models \{P\} c \{Q\}$ durch die anderen Semantiken definieren, und zeigen, dass alle Regeln diese Gültigkeit erhalten.

Relative Vollständigkeit

Wenn $\models \{P\} c \{Q\}$, dann $\vdash \{P\} c \{Q\}$ (bis auf Weakening)

- ▶ Wenn eine Korrektheitsaussage nicht bewiesen werden kann (aber sie stimmt), dann liegt das immer daran, dass eine **logische Aussage** (in einer Anwendung der Weakening-Regelx) nicht bewiesen werden kann.
- ▶ Das ist zu erwarten: alle interessanten Logiken sind unvollständig.



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: $P \implies P_1$,
 $P_2 \implies P_3$, $P_3 \wedge x < n \implies P_4$,
 $P_3 \wedge \neg(x < n) \implies Q$.



Warum Verifikation?

Hier sind Varianten des Fakultätsbeispiels.
Welche sind korrekt?

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



Eine Handvoll Beispiele

```
// {y = Y ∧ y ≥ 0}           // {0 ≤ a}
x = 1;                       t = 1;
while (y != 0) {             s = 1;
  y = y - 1;                 i = 0;
  x = 2 * x;                 while (s <= a) {
}                             t = t + 2;
// {x = 2^Y}                 s = s + t;
// {a ≥ 0 ∧ b ≥ 0}           i = i + 1;
r = a;                       }
q = 0;                       // {i^2 ≤ a ∧ a < (i+1)^2}
while (b <= r) {             }
  r = r - b;
  q = q + 1;
}
// {a = b * q + r ∧ 0 ≤ r ∧ r < b}
```



Zusammenfassung

- ▶ Floyd-Hoare-Logik zusammengefasst:
 - ▶ Die Logik abstrahiert über konkrete Systemzustände durch **Zusicherungen** (Hoare-Tripel $\models \{P\} c \{Q\}$).
 - ▶ Zusicherungen sind boolesche Ausdrücke, angereichert durch logische Variablen und Programmvariablen.
 - ▶ Wir können partielle Korrektheitsaussagen der Form $\models \{P\} c \{Q\}$ herleiten (oder totale, $\models [P] c [Q]$).
 - ▶ 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).
- ▶ Die Korrektheit hängt sehr davon ab, wie **exakt** wir die **Semantik** der Programmiersprache beschreiben können.

