Korrekte Software: Grundlagen und Methoden

Vorlesung 5 vom 19.05.20

Die Floyd-Hoare-Logik

Serge Autexier, Christoph Lüth

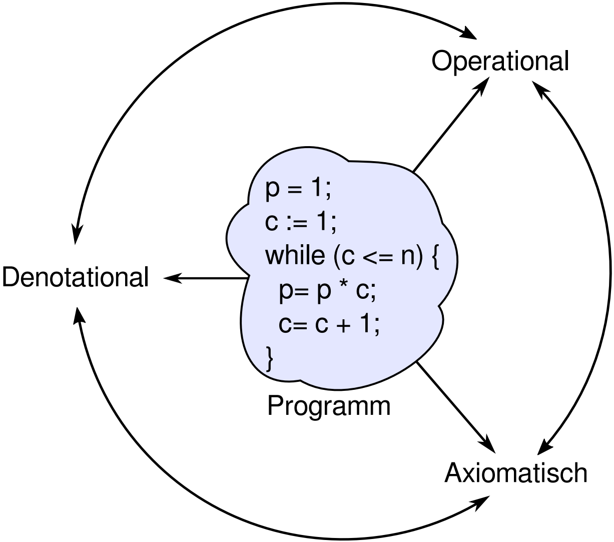
Universität Bremen

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= 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**?

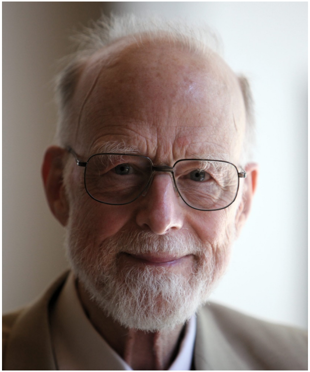
```
p= 1;
c= 1;
while (c <= n) {
  p = p * c;
  c = c + 1;
}
```

- Operationale/denotionale 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 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-Hore-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-Hore-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** and **Bexp** durch
  - **Logische** Variablen **Var** $v := N, M, L, U, V, X, Y, Z$
  - Definierte Funktionen und Prädikate über **Aexp** $n!, x^y, \ldots$
  - Implikation und Quantoren $b_1 \longrightarrow b_2, \forall v.\, b, \exists v.\, b$
- Formal:

$$
\begin{array}{rl}
\textbf{Aexpv} \; a ::= & \mathbf{Z} \mid \mathbf{Idt} \mid \mathbf{Var} \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \\
& \mid f(e_1, \ldots, e_n) \\
\textbf{Assn} \; b ::= & \\
& \mathbf{1} \mid \mathbf{0} \mid a_1 == a_2 \mid a_1 != a_2 \mid a_1 <= a_2 \\
& \mid \,!\, b \mid b_1 \,\&\&\, b2 \mid b_1 \,||\, b_2 \\
& \mid b_1 --> b_2 \mid p(e_1, \ldots, e_n) \mid \backslash\textbf{forall}\; v.\, b \mid \backslash\textbf{exists}\; v.\, b
\end{array}
$$

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** $n!, x^y, \ldots$
  - Implikation und Quantoren $b_1 \longrightarrow b_2, \forall v.\,. b, \exists v.\,. b$
- Formal:

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

$$\textbf{Assn} \quad b ::= \mathit{true} \mid \mathit{false} \mid a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b2 \mid b_1 \vee b_2 \mid b_1 \longrightarrow b_2 \mid p(e_1, \ldots, e_n) \mid \forall v.\, b \mid \exists v.\, b$$

Denotationale Semantik von Zusicherungen

- Erste Näherung: Funktion

$$\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexpv} \to (\Sigma \rightharpoonup \mathbb{Z})$$
$$\llbracket b \rrbracket_{\mathcal{B}} : \textbf{Assn} \to (\Sigma \rightharpoonup \mathcal{B})$$

  - **Konservative** Erweiterung von $\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$

- Aber: was ist mit den logischen Variablen?

Denotationale Semantik von Zusicherungen

- Erste Näherung: Funktion

$$\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexpv} \to (\Sigma \rightharpoonup \mathbb{Z})$$
$$\llbracket b \rrbracket_{\mathcal{B}} : \textbf{Assn} \to (\Sigma \rightharpoonup \mathcal{B})$$

  - **Konservative** Erweiterung von $\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexp} \to (\Sigma \rightharpoonup \mathbb{Z})$

- Aber: was ist mit den logischen Variablen?
- Zusätzlicher Parameter **Belegung** der logischen Variablen $I : \textbf{Var} \to \mathbb{Z}$

$$\llbracket a \rrbracket_{\mathcal{A}} : \textbf{Aexpv} \to (\textbf{Var} \to \mathbb{Z}) \to (\Sigma \rightharpoonup \mathbb{Z})$$
$$\llbracket b \rrbracket_{\mathcal{B}} : \textbf{Assn} \to (\textbf{Var} \to \mathbb{Z}) \to (\Sigma \rightharpoonup \mathcal{B})$$

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

$$[\![b]\!]^I_{\mathcal{B}}(\sigma) = \mathit{true}$$

Arbeitsblatt 5.2: Zusicherungen

Betrachte folgende Zusicherung:

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

Gegeben folgende Belegungen $I_1, \ldots, I_3$ und Zustände $s_1, \ldots, s_3$:

$$s_1 = \langle x \mapsto 0 \rangle, s_2 = \langle x \mapsto 1 \rangle, s_3 = \langle x \mapsto 5 \rangle$$
$$I_1 = \langle X \mapsto 0 \rangle, I_2 = \langle X \mapsto 2 \rangle, I_3 = \langle X \mapsto 10 \rangle$$

Unter welchen Belegungen und Zuständen ist a wahr?

	I_1	I_2	I_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.

$$\models \{P\}\, c\, \{Q\} \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^I P \wedge \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \tau \models^I Q$$

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

$$\models [P]\, c\, [Q] \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^I P \Longrightarrow \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \wedge \tau \models^I Q$$

Beispiele

- Folgendes **gilt**:

$$\models \{\mathit{true}\}\ \textbf{while}(1)\{\ \}\ \{\mathit{true}\}$$

Beispiele

- Folgendes **gilt**:

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

- Folgendes gilt **nicht**:

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

Beispiele

- Folgendes **gilt**:

$$\models \{\mathit{true}\}\ \textbf{while}(1)\{\ \}\ \{\mathit{true}\}$$

- Folgendes gilt **nicht**:

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

- Folgende **gelten**:

$$\models \{\mathit{false}\}\ \textbf{while}\ (\mathbf{1})\ \{\,\}\ \{\mathit{true}\}$$
$$\models [\mathit{false}]\ \textbf{while}\ (\mathbf{1})\ \{\,\}\ [\mathit{true}]$$

Wegen *ex falso quodlibet*: $\mathit{false} \Longrightarrow \phi$

Gültigkeit und Herleitbarkeit

- **Semantische Gültigkeit:** $\models \{P\}\, c\, \{Q\}$
  - Definiert durch denotationale Semantik:

    $$\models \{P\}\, c\, \{Q\} \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^{I} P \wedge \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \tau \models^{I} Q$$

  - Problem: müssten Semantik von c ausrechnen

Gültigkeit und Herleitbarkeit

- **Semantische Gültigkeit:** $\models \{P\}\, c\, \{Q\}$
  - Definiert durch denotationale Semantik:
    $$\models \{P\}\, c\, \{Q\} \Longleftrightarrow \forall I.\, \forall \sigma.\, \sigma \models^I P \wedge \exists \tau.\, (\sigma, \tau) \in [\![c]\!]_{\mathcal{C}} \Longrightarrow \tau \models^I Q$$
  - 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

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

- 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:

```
//
x = 5
//{x < 10}
```

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:

```
//{(x < 10)[5/x]}
x = 5
//{x < 10}
```

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:

```
//{(x < 10)[5/x] ⟺ 5 < 10}
x = 5
//{x < 10}
```

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:

```
//{(x < 10)[5/x] <=> 5 < 10}
x = 5
//{x < 10}
```

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

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:

```
//{(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

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

- Hier wird eine Zwischenzusicherung B benötigt.

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

- Trivial.

Ein allererstes Beispiel

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

- Was berechnet dieses Programm?

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:

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

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:

$$
\dfrac{
\dfrac{}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; \\ \quad \{?\}\end{array}}
\qquad
\dfrac{}{\begin{array}{l}\vdash \{?\} \\ \quad y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
}{
\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}
}
$$

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?
- $\vdash \{x = X \land y = Y\}\, p\, \{y = X \land x = Y\}$

Herleitung:

$$\dfrac{\dfrac{}{\begin{array}{l}\vdash \{x = X \land y = Y\} \\ \quad z = x; x = y; \\ \quad \{z = X \land x = Y\}\end{array}} \qquad \dfrac{}{\begin{array}{l}\vdash \{z = X \land x = Y\} \\ \quad y = z; \\ \quad \{y = X \land x = Y\}\end{array}}}{\begin{array}{l}\vdash \{x = X \land y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \land x = Y\}\end{array}}$$

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:

$$
\dfrac{
\dfrac{}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; \\ \quad \{?\}\end{array}}
\quad
\dfrac{}{\begin{array}{l}\vdash \{?\} \\ \quad x = y; \\ \quad \{z = X \wedge x = Y\}\end{array}}
}{
\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; \\ \quad \{z = X \wedge x = Y\}\end{array}
}
\quad
\dfrac{}{\begin{array}{l}\vdash \{z = X \wedge x = Y\} \\ \quad y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
$$

$$
\dfrac{\begin{array}{c}\vdots\end{array}}{\begin{array}{l}\vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \wedge x = Y\}\end{array}}
$$

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:

$$
\dfrac{
  \dfrac{
    \dfrac{}{\begin{array}{l} \vdash \{x = X \wedge y = Y\} \\ \quad z = x; \\ \quad \{z = X \wedge y = Y\} \end{array}}
    \quad
    \dfrac{}{\begin{array}{l} \vdash \{z = X \wedge y = Y\} \\ \quad x = y; \\ \quad \{z = X \wedge x = Y\} \end{array}}
  }{
    \begin{array}{l} \vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; \\ \quad \{z = X \wedge x = Y\} \end{array}
  }
  \quad
  \dfrac{}{\begin{array}{l} \vdash \{z = X \wedge x = Y\} \\ \quad y = z; \\ \quad \{y = X \wedge x = Y\} \end{array}}
}{
  \begin{array}{l} \vdash \{x = X \wedge y = Y\} \\ \quad z = x; x = y; y = z; \\ \quad \{y = X \wedge x = Y\} \end{array}
}
$$

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)!}
```

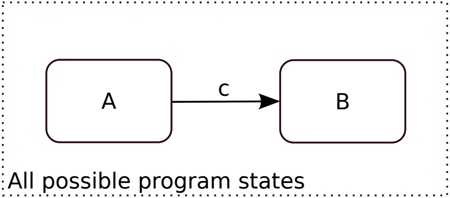
- Welche Zusicherungen gelten

  1. an der Stelle (A)?

  2. an der Stelle (B)?

Regeln des Floyd-Hoare-Kalküls: Weakening

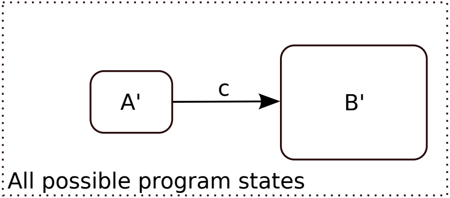
$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$



- $\models \{A\}\, c\, \{B\}$: 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. $P \Longrightarrow Q$.

Regeln des Floyd-Hoare-Kalküls: Weakening

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$



- $\models \{A\}\, c\, \{B\}$: 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. $P \Longrightarrow Q$.
- Wir können A zu A' einschränken ($A' \subseteq A$ oder $A' \Longrightarrow A$), oder B zu B' vergrößern ($B \subseteq B'$ oder $B \Longrightarrow B'$), und erhalten $\models \{A'\}\, c\, \{B'\}$.

Regeln des Floyd-Hoare-Kalküls: Fallunterscheidung

$$\frac{\vdash \{A \land b\}\, c_0\, \{B\} \qquad \vdash \{A \land \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

- In der Vorbedingung des **if**-Zweiges gilt die Bedingung b, und im **else**-Zweig gilt die Negation $\neg b$.
- Beide Zweige müssem 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?

Arbeitsblatt 5.5: 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

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

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

Das Fakultätsbeispiel (I)

```
// {1 = 0!}
// {1 = (1 − 1)!}
p= 1;
// {p = (1 − 1)!}
c= 1;
// {p = (c − 1)!}
while (c<= n) {
  // {p = (c − 1)! ∧ 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)! ∧ ¬(c ≤ n)}
// {p = (c − 1)! ∧ c − 1 ≥ n}
// ??
// {p = n!}
```

Das Fakultätsbeispiel (II)

```
// {1 = 0! ∧ 0 ≤ n}
// {1 = (1 − 1)! ∧ 1 − 1 ≤ n}
p= 1;
// {p = (1 − 1)! ∧ 1 − 1 ≤ n}
c= 1;
// {p = (c − 1)! ∧ c − 1 ≤ n}
while (c<= n) {
  // {p = (c − 1)! ∧ c − 1 ≤ n ∧ c ≤ n}
  // {p ∗ c = (c − 1)! ∗ c ∧ c ≤ n} !!!
  // {p ∗ c = c! ∧ c ≤ n}
  // {p ∗ c = ((c + 1) − 1)! ∧ (c + 1) − 1 ≤ n}
  p= p∗c;
  // {p = ((c + 1) − 1)! ∧ (c + 1) − 1 ≤ n}
  c= c+1;
  // {p = (c − 1)! ∧ c − 1 ≤ n}
  }
// {p = (c − 1)! ∧ c − 1 ≤ n ∧ ¬(c ≤ n)}
// {p = (c − 1)! ∧ c − 1 ≤ n ∧ c > n}
// {p = (c − 1)! ∧ c − 1 ≤ n ∧ c − 1 ≥ n}
// {p = n!}
```

Das Fakultätsbeispiel (komplett)

```
// {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)! ∧ c − 1 ≤ n ∧ c > n}
// {p = (c − 1)! ∧ c − 1 ≤ n ∧ c − 1 ≥ n}
// {p = n!}
```

Arbeitsblatt 5.6: 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

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

$$\frac{\vdash \{A \land b\}\, c_0\, \{B\} \qquad \vdash \{A \land \neg b\}\, c_1\, \{B\}}{\vdash \{A\}\ \textbf{if}\ (b)\ c_0\ \textbf{else}\ c_1\, \{B\}}$$

$$\frac{\vdash \{A \land b\}\, c\, \{A\}}{\vdash \{A\}\ \textbf{while}(b)\ c\, \{A \land \neg b\}}$$

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

$$\frac{A' \Longrightarrow A \qquad \vdash \{A\}\, c\, \{B\} \qquad B \Longrightarrow B'}{\vdash \{A'\}\, c\, \{B'\}}$$

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).