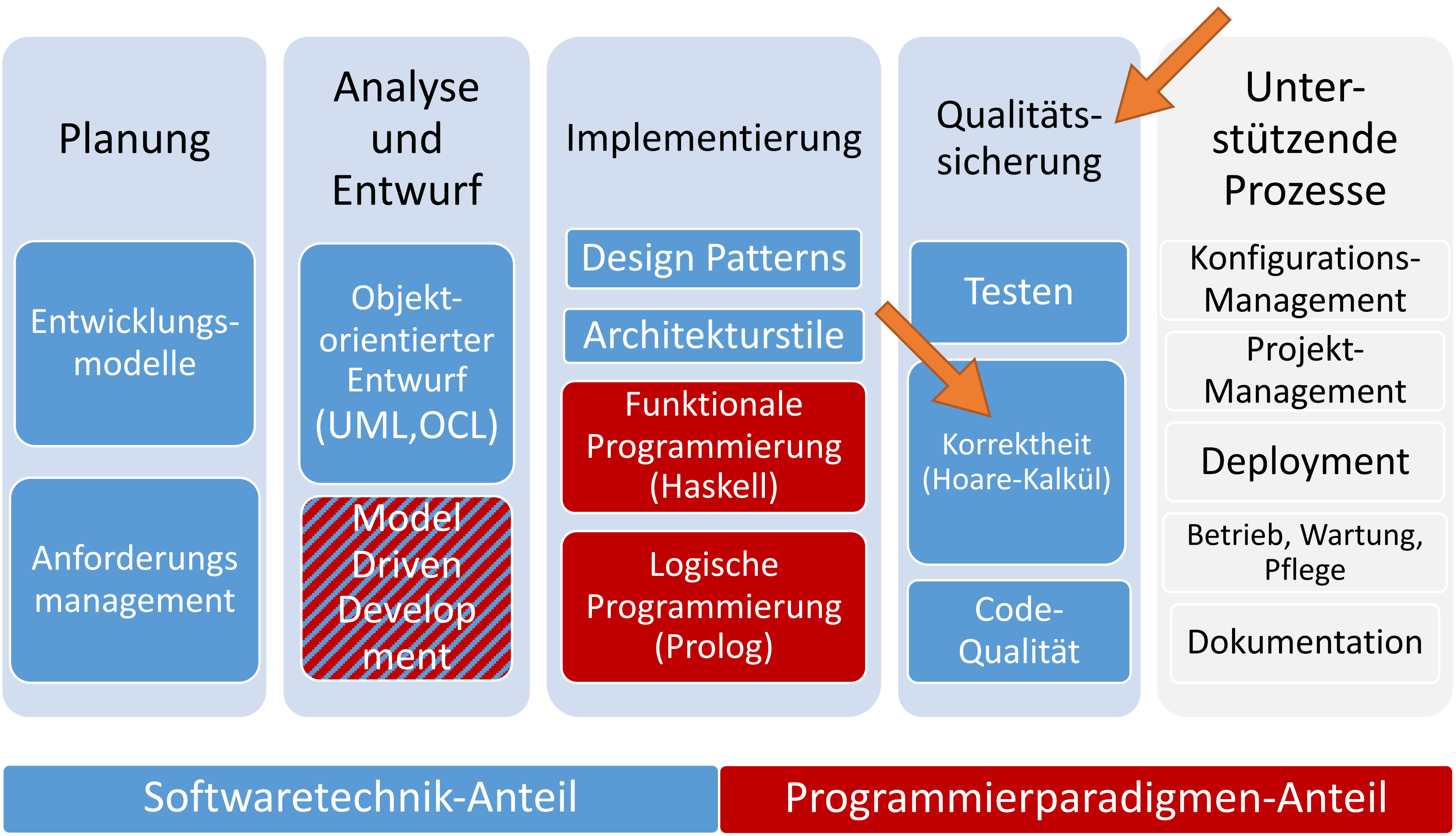
Softwaretechnik und Programmierparadigmen

06 Korrektheit

Prof. Dr. Sabine Glesner
Software and Embedded Systems Engineering
Technische Universität Berlin

Diese VL



Planung
Entwicklungs-modelle
Anforderungs management
Analyse und Entwurf
Objekt-orientierter Entwurf (UML,OCL)
Model Driven Develop ment
Implementierung
Design Patterns
Architekturstile
Funktionale Programmierung (Haskell)
Logische Programmierung (Prolog)
Qualitäts-sicherung
Testen
Korrektheit (Hoare-Kalkül)
Code-Qualität
Unter-stützende Prozesse
Konfigurations-Management
Projekt-Management
Deployment
Betrieb, Wartung, Pflege
Dokumentation
Softwaretechnik-Anteil
Programmierparadigmen-Anteil

Inhalt

Korrektheit

- Einführung Software-Qualität
- WHILE-Sprache
- Hoare-Kalkül
- Nachweis partieller Korrektheit
- Nachweis totaler Korrektheit
- Tool Support

Inhalt

Korrektheit

- Einführung Software-Qualität
- WHILE-Sprache
- Hoare-Kalkül
- Nachweis partieller Korrektheit
- Nachweis totaler Korrektheit
- Tool Support

Was ist Software-Qualität?

„Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen“

DIN-ISO-Norm 9126

Was sagt uns das?

- Sehr allgemeine Definition
- Software-Qualität ist vielfältig
- Messbarkeit der Qualität anhand von definierten Kriterien
- Unterschiedliche Auffassungen des Begriffs möglich

Qualitätsmerkmale

Kundenorientiert

- Funktionalität
- Zuverlässigkeit
- Effizienz
- Benutzbarkeit
- …

Herstellerorientiert

- Übertragbarkeit
- Änderbarkeit
- Testbarkeit
- Transparenz
- …

Qualitätssicherung

Prozessqualität

Befasst sich mit der Verbesserung der Entstehung des Software-Produkts (Prozess).

- Managementprozesse und Entwicklungsmodelle
- Software-Infrastruktur (Build-Automatisierung, Testautomatisierung, …)

Produktqualität

Befasst mit der Verbesserung der genannten Qualitätsmerkmale des Software-Produkts.

- Korrektheit
- Testen
- Konventionen
- Kommentare
- Statische Analyse
- Metriken
- …

Prozessqualität vs. Produktqualität

Der Entwicklungsprozess sollte Methoden zur Verbesserung der Produktqualität an sinnvollen Stellen einsetzen.

Die Verbesserung der Prozessqualität beinhaltet die Optimierung der Maßnahmen zur Verbesserung der Produktqualität

Qualitätssicherung

Prozessqualität

Befasst sich mit der Verbesserung der Entstehung des Software-Produkts (Prozess).

- Managementprozesse und Entwicklungsmodelle
- Software-Infrastruktur (Build-Automatisierung, Testautomatisierung, …)

Produktqualität

Befasst mit der Verbesserung der genannten Qualitätsmerkmale des Software-Produkts.

- Korrektheit ← Heute
- Testen
- Konventionen
- Kommentare
- Statische Analyse
- Metriken
- …

Motivation - Korrektheit

Wer sagt uns eigentlich, dass unsere **Implementierung**…

pre: *self.kunde.id->includes(kundeId) and neueAdr <> “*

```
void emailAendern(Integer kundeId, String neueAdr) {
  Kunde k = kunde.get(kundeId);
  String email = k.getEmail();
  email = neueAdr;
  // TODO:  mrk;31/12/14 Why does this not work?
  // FIXME: joe;03/01/15 Mark, please fix this!
  // FIXME: joe;10/01/15 Mark?
  // XXX:   eve;04/02/16 Mark has left the building.
}
```

post: *self.kunde->any(id = kundeId).email = neueAdr*

…zur **Spezifikation** passt?

Motivation - Korrektheit

Wir kennen Contracts zur **formalen Spezifikation** von Methoden

➢ Können als Ausgangsbasis für die Implementierung dienen

Wie kann man nachweisen, dass die Implementierung den Contract erfüllt?

Antwort 1: Testen der Implementierung gegen den Contract

➢ Viel Testen hilft, ist aber kein Beweis!

Antwort 2: Beweisen, dass das Programm den Contract erfüllt

➢ Voraussetzung für Beweise: Beweiskalkül (hier: Hoare Kalkül)

Inhalt

Korrektheit

- Einführung Software-Qualität
- WHILE-Sprache
- Hoare-Kalkül
- Nachweis partieller Korrektheit
- Nachweis totaler Korrektheit
- Tool Support

Wozu die While-Spache?

Moderne Sprachen haben (viele) komplexe Features

➢ Umfang zu groß bzw. Konstrukte zu schwierig für Demonstration einfacher Ideen

While-Sprache als formales Berechnungsmodell

- Übersichtlicher Sprachumfang
- Einfache Semantik

➢ Wir verwenden die WHILE-Sprache zur Demonstration. Hoare-Regeln lassen sich aber für beliebige Sprach-Konstrukte erstellen

➢ Viele Konstrukte gängiger Sprachen (z.B. Java) können in die While-Sprache umgeformt werden (turingmächtig)

WHILE Syntax

Arithmetische Ausdrücke

$$a ::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2$$

Boolesche Ausdrücke

$$b ::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 \leq a_2 \mid \neg\, b \mid b_1 \wedge b_2$$

Anweisungen/Programme

$$S ::= x := a \mid \mathbf{skip} \mid S_1 ;\ S_2 \mid$$
$$\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi} \mid \mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}$$

n: steht für syntaktische Zahlen

x: steht für syntaktische Variablen

S: steht für Anweisungen (Statements)

Beispielprogramme

Zwei Variablen werden vertauscht wenn sie nicht gleich sind:

```
if x=y then
    skip
else
    z:=x;
    x:=y;
    y:=z
fi
```

If B **then** S_1 **else** S_2 **fi**

Skip-Anweisung: Blöcke dürfen laut Syntax nicht leer sein

Sequenz: Semikolon verbindet Instruktionen zu Blöcken

Am Ende kein Semikolon (keine nachfolgende Instruktion)

Beispielprogramme

Fakultätsfunktion y = fak(x):

$$\begin{array}{l} \texttt{y:=1;} \\ \textbf{while } \overbrace{\texttt{x > 1}}^{B} \textbf{ do} \\ \quad S\left\{\begin{array}{l}\texttt{y:=x*y;}\\ \texttt{x:=x-1}\end{array}\right. \\ \textbf{od}\end{array}$$

while B **do** S **od**

Endlosschleife:

while true **do** skip **od**

Inhalt

Korrektheit

- Einführung Software-Qualität
- WHILE-Sprache
- Hoare-Kalkül
- Nachweis partieller Korrektheit
- Nachweis totaler Korrektheit
- Tool Support

Hoare-Kalkül

Entwickelt von **Sir Charles Antony Richard Hoare**

- Geboren 1934
- Emeritierter Professor der Universität Oxford
- Momentan leitender Forscher bei Microsoft Research in Cambridge
- Einige wichtige Leistungen sind:
  - CSP (Communicating Sequential Processes)
  - Monitor-Konzept
  - Hoare-Kalkül
  - Quicksort
- Turing Award 1980
- John-von-Neumann-Medaille 2011

Hoare Kalkül

Idee des Hoare-Kalküls: Die Verifikation eines Programms besteht aus zwei Schritten.

- Beweis der partiellen Korrektheit
- Beweis der Terminierung

Am Beispiel Fakultäts-Funktion:

```
y:=1;
while  x>1 do
        y:=x*y;
        x:=x-1
od
```

Gilt x=n vor Ausführung, dann gilt y=n! nach Ausführung, **sofern die Ausführung terminiert**

$\Longrightarrow$ partielle Korrektheit

Gilt x=n vor Ausführung, **dann terminiert das Programm** und y=n!

$\Longrightarrow$ totale Korrektheit

Partielle Korrektheit

Ein Programm ist partiell korrekt bzgl. einer Vorbedingung *P* und einer Nachbedingung *Q*, falls immer dann, wenn der initiale Zustand die Vorbedingung erfüllt und, wenn das Programm terminiert, der Endzustand die Nachbedingung erfüllt.

- ➢ Hoare-Tripel: {P} S {Q}

Vorbedingung | Programm | Nachbedingung

Annahme: Programm terminiert nach endlich vielen Schritten

Vorbedingung: Nur wenn diese erfüllt ist kann die Nachbedingung garantiert werden (durch Beweis)

Nachbedingung: Gefordertes Ergebnis (Spezifikation, z.B. Post-Condition eines Contracts)

➢Vorbedingungen ergeben sich oft aus dem Beweis

Bedingungen

Zentrales Element eines Hoare-Beweises sind Bedingungen bzw. Zusicherungen (assertions)

- Formeln der **Prädikatenlogik**
- Werden in geschwungenen Klammern hinzugefügt

Enthalten zwei Arten von Variablen:

- Programmvariablen (z.B. x, y)
- Zusätzlich logische Variablen (z.B. n)

Hier: Logische Variable n für den Anfangszustand von x …

$\{x = n \wedge n \geq 0\}$

```
y:=1;
while x>1 do
        y:=x*y;
        x:=x-1
od
```

$\{y = n!\}$

Damit dieser für das Ergebnis zur Verfügung steht

Beispiele Vor-und Nachbedingungen

Vorbedingung (Anfangswerte in logischen Variablen „merken“) von allen Eingaben erfüllbar

{x=n ∧ y=m}

```
if x=y then
   skip
else
   z:=x;
   x:=y;
   y:=z
fi
```

{ y=n ∧ x=m }

Nachbedingung: Werte wurden vertauscht (oder waren gleich)

Nachbedingung kann nur für x>=0 vom Programm erfüllt werden

{x>=0}

```
c:=0;
sum:=0;
while c < x do
   c := c+1;
   sum := sum+c
od
```

{ sum=$\sum_{i=0}^{x} i$}

Nachbedingung: Variable sum enthält Summe der Zahlen bis x

Inhalt

Korrektheit

- Einführung Software-Qualität
- WHILE-Sprache
- Hoare-Kalkül
- Nachweis partieller Korrektheit
- Nachweis totaler Korrektheit
- Tool Support

Überblick

Hoare Kalkül besteht aus **Axiomen** und **Inferenzregeln** für alle Konstrukte einer Programmiersprache

- ➢ definiert was ein Konstrukt bewirkt (wie wird es ausgeführt)
- ➢ gibt der **Syntax** eine **Semantik**

Axiome sind Grundsätze eines Systems, ohne weitere Begründung

Inferenzregeln ermöglichen das Zerlegen komplexerer Ausdrücke

- Stellen eine logische Schlussfolgerung dar

Formale Beschreibung der Konstrukte ermöglicht es, das Programm für **alle Eingaben** „durchzurechnen“

Hoare Kalkül – Axiome

Skip-Axiom

Skip ändert den Zustand nicht und bewahrt dadurch jede Vorbedingung

{ P } **skip** { P }

$\{x = y + 5\}$

skip

$\{x = y + 5\}$

Ersetzungs-Axiom (Zuweisung)

Ersetze jede Vorkommen von x in der Nachbedingung P durch syntaktischen Ausdruck E

➢ Wird von unten nach oben angewandt

{ P[x ← E] } x:=E { P }

$\{y - 1 < y\}$ P[x ← E]

a := y - 1;

$\{a < y\}$ P

Inferenzregeln

Jedes Kalkül gibt eine Menge von Inferenzregeln vor

- Eine Inferenzregel beschreibt, unter welchen Annahmen eine bestimmte Schlussfolgerung abgeleitet werden kann

$$\frac{\text{Annahmen}}{\text{Schlussfolgerung}} \qquad \frac{\textit{Wenn}}{\textit{Dann}}$$

- Beispiel aus dem Kalkül des natürlichen Schließens (Modus Ponens):

$$\frac{A \qquad A \rightarrow B}{B}$$

„Wenn A und A impliziert B gilt, dann gilt B“

Hoare Kalkül – Regeln

$$\frac{\{ P \}\, S_1\, \{ R \} \qquad \{ R \}\, S_2\, \{ Q \}}{\{ P \}\, S_1 ; S_2\, \{ Q \}}$$

Sequenzregel

Hoare-Tripel können sequentiell komponiert werden, wenn die Nachbedingung der ersten Anweisung mit der Vorbedingung der zweiten Anweisung kompatibel ist

Inline

$$\{P\}$$
$$S_1$$
$$\{R\}$$
$$S_2$$
$$\{Q\}$$

Hoare Kalkül – Regeln

$$\frac{P \longrightarrow P' \qquad \{\,P'\,\}\,S\,\{\,Q'\,\} \qquad Q' \longrightarrow Q}{\{\,P\,\}\,S\,\{\,Q\,\}}$$

Rule of Consequence

- die **Vorbedingung** darf **verschärft** werden
- die **Nachbedingung** darf **abgeschwächt** werden

Inline

$$\begin{gathered}\{P\}\\ \downarrow\\ \{P'\}\\ S\\ \{Q'\}\\ \downarrow\\ \{Q\}\end{gathered}$$

Verschärfung

P P‘

Beispiel: $\{\,x < y\,\} \longrightarrow \{\,x \le y\}$

Q‘ Q

Abschwächung

Sequentielles Beispiel: Swap

$\{x = n \land y = m\}$

Vorbedingung P

z:=x;

x:=y;

y:=z

$\{x = m \land y = n\}$

Nachbedingung Q

Sequentielles Beispiel: Swap

$\{x = n \land \ y = m\}$

$\{\qquad\qquad\qquad\}$

z:=x;

$\{\qquad\qquad\qquad\}$

x:=y;

$\{\qquad\qquad\qquad\}$

y:=z

$\{x = m \land y = n\}$

Sequenzregel

Sequentielles Beispiel: Swap

$\{x = n \wedge \ y = m\}$

$\{ \qquad\qquad\qquad \}$

z:=x;

$\{ \qquad\qquad\qquad \}$

x:=y;

$\{x = m \wedge z = n\}$

Ersetzungs-Axiom wurde angewandt

y:=z

$\{x = m \wedge y = n\}$

Sequentielles Beispiel: Swap

$\{x = n \land \ y = m\}$

$\{ \qquad\qquad\qquad \}$

z:=x;

$\{y = m \land z = n\}$

Ersetzungs-Axiom wurde angewandt

x:=y;

$\{x = m \land z = n\}$

y:=z

$\{x = m \land y = n\}$

Sequentielles Beispiel: Swap

$\{x = n \wedge \ y = m\}$

$\{y = m \wedge x = n\}$

Ersetzungs-Axiom wurde angewandt

z:=x;

$\{y = m \wedge z = n\}$

x:=y;

$\{x = m \wedge z = n\}$

y:=z

$\{x = m \wedge y = n\}$

Sequentielles Beispiel: Swap

$\{x = n \land y = m\}$

$\leftrightarrow$

Die Ableitung ist äquivalent zur Vorbedingung

$\{y = m \land x = n\}$

z:=x;

$\{y = m \land z = n\}$

x:=y;

$\{x = m \land z = n\}$

y:=z

$\{x = m \land y = n\}$

Hoare Kalkül – Regeln

$$\frac{\{\, B \land P \,\}\ S_1\ \{\, Q \,\} \qquad \{\neg B \land P \,\}\ S_2\ \{\, Q \,\}}{\{\, P \,\}\ \textbf{if}\ B\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{\, Q \,\}}$$

If-Regel

- Für die if-Anweisung müssen beide möglichen Fälle entsprechend berücksichtigt werden

```
{P}
if B then
    { B ∧ P }
    S1
    {Q}
else
    {¬B ∧ P }
    S2
    {Q}
fi
{Q}
```

Beispiele rechnen wir in der Übung

Hoare Kalkül – Regeln

While-Regel

- Beweise für Schleifen basieren auf Invarianten I
- Die Invariante muss vom Schleifenrumpf S bewahrt werden
- Wenn vor der Schleife I gilt, gilt I auch nach der Schleife, zusätzlich gilt die Schleifenbedingung nicht

➢ keine Vorgabe zur Ermittlung der Schleifeninvariante: muss manuell ermittelt werden

$$\frac{\{\, B \land I \,\}\; S \;\{\, I \,\}}{\{\, I \,\}\; \textbf{while}\; B\; \textbf{do}\; S\; \textbf{od}\; \{\neg B \land I \,\}}$$

Gefundene Invariante eingesetzt in das Programm

```
{I}
while B do
    { B ∧ I }
    S
    {I}
od
{¬B ∧ I }
```

Fakultätsfunktion

$\{ x = n \wedge n \geq 0 \}$

Vorbedingung

y := 1;

while x ≥ 1 **do**

Wie könnte eine **Schleifeninvariante** zum Nachweis der Fakultätsfunktion aussehen?

    y := y * x;

    x := x - 1

od

$\{ y = n! \}$

Nachbedingung

Fakultätsfunktion

$\{ x = n \land n \geq 0 \}$

y := 1;

while x ≥ 1 **do**

y := y * x;

x := x - 1

od

$\{ y = n!\}$

Beobachtungen

1. x ist immer positiv
2. Werte für n = 4 (Beispiel):

x	n!	x!	y
4	24	24	1
3	24	6	4
2	24	2	12
1	24	1	24
0	24	1	24

Schleifen-Invariante:

$$\{y = \frac{n!}{x!} \land x \geq 0\}$$

Anwendung der Schleifenregel

$\{ x = n \land n \geq 0 \}$

y := 1;

$\{y = \frac{n!}{x!} \land x \geq 0\}$

Schleifen-Invariante

while x $\geq$ 1 **do**

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x \geq 1 \}$

Schleifen-Invariante mit Schleifenbedingung

y := y * x;

x := x - 1

$\{y = \frac{n!}{x!} \land x \geq 0\}$

Schleifen-Invariante

od

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x < 1 \}$

Schleifen-Invariante mit **negierter** Schleifenbedingung

$\{ y = n!\}$

Fakultätsfunktion

$\{ x = n \land n \geq 0 \}$

y := 1;

$\{y = \frac{n!}{x!} \land x \geq 0\}$

while x $\geq$ 1 **do**

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x \geq 1 \}$

y := y * x;

x := x - 1

$\{y = \frac{n!}{x!} \land x \geq 0\}$

od

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Rule of Consequence

$$\left\{ y = \frac{n!}{x!} \land x \geq 0 \land x < 1 \right\}$$
$$\Rightarrow \left\{ y = \frac{n!}{x!} \land x = 0 \right\}$$
$$\Rightarrow \left\{ y = \frac{n!}{0!} \right\}$$
$$\Rightarrow \left\{ y = \frac{n!}{1} \right\}$$
$$\Rightarrow \{ y = n!\}$$

Fakultätsfunktion

$\{ x = n \wedge n \geq 0 \}$

y := 1;

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

while x ≥ 1 **do**

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

Ersetzungsaxiom wurde angewandt

x := x - 1

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

od

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Fakultätsfunktion

$\{ x = n \wedge n \geq 0 \}$

y := 1;

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

while x ≥ 1 **do**

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \}$

$\{ y \cdot x = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

Ersetzungsaxiom wurde angewandt

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

x := x - 1

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

od

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Fakultätsfunktion

$\{ x = n \wedge n \geq 0 \}$

y := 1;

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

while x $\geq$ 1 **do**

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \}$

$\leftrightarrow$ Äquivalent

$\{ y \cdot x = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

x := x - 1

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

od

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Fakultätsfunktion

$\{ x = n \land n \geq 0 \}$

$\{1 = \frac{n!}{x!} \land x \geq 0\}$

Ersetzungsaxiom wurde angewandt

y := 1;

$\{y = \frac{n!}{x!} \land x \geq 0\}$

While-Schleife fertig: Darüber geht es weiter

while x ≥ 1 **do**

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x \geq 1 \}$

$\leftrightarrow$

$\{ y \cdot x = \frac{n!}{(x-1)!} \land x - 1 \geq 0 \}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \land x - 1 \geq 0 \}$

x := x - 1

$\{y = \frac{n!}{x!} \land x \geq 0\}$

od

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Fakultätsfunktion

$\{ x = n \wedge n \geq 0 \}$

$\longrightarrow$

$\{1 = \frac{n!}{x!} \wedge x \geq 0\}$

Rule of Consequence

y := 1;

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

while x ≥ 1 **do**

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \}$

$\leftrightarrow$

$\{ y \cdot x = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

x := x - 1

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

od

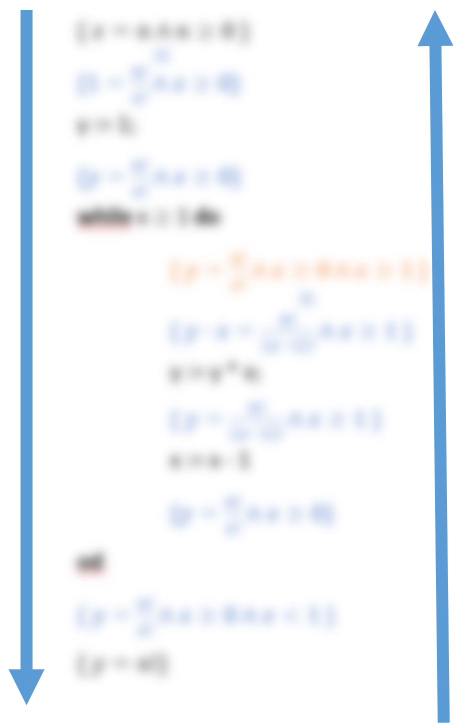
$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Fahrplan zur Anwendung des Hoare-Kalküls

Muss gelten:
Aus Vorbedingung folgt Nachbedingung (RoC / Äquivalenz)



Beweisführung meist rückwärts

- ➢ vom spezifizierten Ergebnis (post) die (möglichst schwächsten) Vorbedingungen ableiten

Vorgehensweise: Vom aktuellen Punkt aus gesehen nach oben arbeiten. Zwei Möglichkeiten:

- Darüber steht ein Programmausdruck
  - ➢ Regel anwenden
- Darüber steht eine Bedingung
  - ➢ Äquivalenz/RoC prüfen/ableiten
  - ➢ Beweis schließen

Beispiele Vor-und Nachbedingungen

$\{x = n \land y = m\}$
```
c:=0;
sum:=0;
while true do

    skip
od
```

$\{y = n \land x = m\}$

$\{true\}$
```
c:=0;
sum:=0;
while true do

    skip
od
```

$\{false\}$

Sind die Vor-und Nachbedingungen für die gegebenen Programme korrekt?

Beispiele Vor-und Nachbedingungen

$\{x = n \land y = m\}$
```
c:=0;
sum:=0;
while true do
```
$\{true\}$
```
    skip
od
```
$\{false\}$

$\downarrow$

$\{y = n \land x = m\}$

$\{true\}$
```
c:=0;
sum:=0;
while true do
```
$\{true\}$
```
    skip
od
```
$\{false\}$

$\downarrow$

$\{false\}$

Partiell sind die Programme korrekt - aber sie terminieren nicht!

Inhalt

Korrektheit

- Einführung Software-Qualität
- WHILE-Sprache
- Hoare-Kalkül
- Nachweis partieller Korrektheit
- Nachweis totaler Korrektheit
- Tool Support

Totale Korrektheit

Ein Programm S ist **total korrekt** bzgl. Vorbedingung P und Nachbedingung Q, wenn es partiell korrekt ist und **immer terminiert**

$$\frac{\{B \land I \land (t = m)\}\; S\; \{I \land (t < m)\}, \quad I \land B \Rightarrow t \geq 0}{\{I\}\; \boldsymbol{while}\; B\; \boldsymbol{do}\; S\; \boldsymbol{od}\; \{\neg B \land I\}}$$

Finde für jede Schleife eine Terminierungsfunktion $t(\dots) \mapsto \mathbb{N}$, so dass

1. $I \land B \Rightarrow t \geq 0$ — Zu Beginn des Schleifenrumpfes ist *t* immer positiv, aber…
2. $\{B \land I \land (t = m)\}\; S\; \{I \land (t < m)\}$ — …nimmt in jedem Durchlauf ab

Wenn eine solche Terminierungsfunktion (auch: **Schleifenvariante**) für jede Schleife gefunden werden kann, **dann** ist die totale Korrektheit bewiesen

Fakultätsfunktion

$\{\, x = n \wedge n \geq 0 \,\}$

$\rightarrow$

$\{1 = \frac{n!}{x!} \wedge x \geq 0\}$

y := 1;

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

while x ≥ 1 **do**

$\{\, y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \,\}$

$\leftrightarrow$

$\{\, y \cdot x = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \,\}$

y := y * x;

$\{\, y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \,\}$

x := x - 1

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

od

$\{\, y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < 1 \,\}$

$\longrightarrow$

$\{\, y = n!\}$

Wie könnte eine Terminierungsfunktion zum Nachweis der totalen Korrektheit aussehen?

Die Schleife zählt den Wert von *x* runter und terminiert bei *x = 0*.

Mögliche Terminierungsfunktion:
t(x) = x

Fakultätsfunktion

$\{ x = n \land n \geq 0 \}$

$\rightarrow$

$\{1 = \frac{n!}{x!} \land x \geq 0\}$

y := 1;

$\{y = \frac{n!}{x!} \land x \geq 0\}$

while x ≥ 1 **do**

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x \geq 1 \}$

$\leftrightarrow$

$\{ y \cdot x = \frac{n!}{(x-1)!} \land x - 1 \geq 0 \}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \land x - 1 \geq 0 \}$

x := x - 1

$\{y = \frac{n!}{x!} \land x \geq 0\}$

od

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Mögliche Terminierungsfunktion:
t(x) = x

Erster Teil des Beweises:

$$I \land B \Rightarrow t \geq 0$$

Fakultätsfunktion

$\{ x = n \wedge n \geq 0 \}$

$\rightarrow$

$\{1 = \frac{n!}{x!} \wedge x \geq 0\}$

y := 1;

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

while x ≥ 1 **do**

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \}$

$\leftrightarrow$

$\{ y \cdot x = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \}$

x := x - 1

$\{y = \frac{n!}{x!} \wedge x \geq 0\}$

od

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < 1 \}$

$\longrightarrow$

$\{ y = n!\}$

Mögliche Terminierungsfunktion:

t(x) = x

Erster Teil des Beweises:

$$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \} \Rightarrow x \geq 0$$

OK

Fakultätsfunktion

while x $\geq$ 1 **do**

$$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \wedge x = m\}$$

$$\{ y \cdot x = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \wedge \qquad \}$$

y := y * x;

$$\{ y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \wedge \qquad \}$$

x := x - 1

$$\{y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < m\}$$

S

od

Mögliche Terminierungsfunktion:
t(x) = x

Erster Teil des Beweises:

$$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \} \Rightarrow x \geq 0$$

Zweiter Teil des Beweises:

$$\{B \wedge I \wedge (t = m)\} \, S \, \{I \wedge (t \ < m)\}$$

Fakultätsfunktion

while x ≥ 1 **do**

S:

$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \wedge x = m\}$

$\{ y \cdot x = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \wedge x - 1 < m\}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \wedge x - 1 \geq 0 \wedge x - 1 < m\}$

x := x - 1

Ersetzungsaxiom

$\{y = \frac{n!}{x!} \wedge x \geq 0 \wedge x < m\}$

od

Mögliche Terminierungsfunktion:
t(x) = x

Erster Teil des Beweises:

$$\{ y = \frac{n!}{x!} \wedge x \geq 0 \wedge x \geq 1 \} \Rightarrow x \geq 0$$

Zweiter Teil des Beweises:

$$\{B \wedge I \wedge (t = m)\} \; S \; \{I \wedge (t < m)\}$$

Fakultätsfunktion

Korrekt, da
$x = m \Rightarrow x - 1 < m$

while x ≥ 1 **do**

S

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x \geq 1 \land x = m \}$

$\rightarrow$

$\{ y \cdot x = \frac{n!}{(x-1)!} \land x - 1 \geq 0 \land x - 1 < m \}$

y := y * x;

$\{ y = \frac{n!}{(x-1)!} \land x - 1 \geq 0 \land x - 1 < m \}$

x := x - 1

$\{ y = \frac{n!}{x!} \land x \geq 0 \land x < m \}$

od

Mögliche Terminierungsfunktion:
t(x) = x

Erster Teil des Beweises:

$$\{ y = \frac{n!}{x!} \land x \geq 0 \land x \geq 1 \} \Rightarrow x \geq 0$$

Zweiter Teil des Beweises:

$$\{ B \land I \land (t = m) \} \; S \; \{ I \land (t < m) \}$$

Inhalt

Korrektheit

- Einführung Software-Qualität
- WHILE-Sprache
- Hoare-Kalkül
- Nachweis partieller Korrektheit
- Nachweis totaler Korrektheit
- Tool Support

Tool Support

- Viele Werkzeuge zur Programmverifikation basieren auf dem Hoare-Kalkül
- Einfache Regeln müssen dann nicht mehr von Hand ausgeführt werden
- Invarianten sind jedoch in der Regel schwer zu finden und können nicht immer automatisch bestimmt werden

„**Verifikation von C-Programmen in der Praxis**“
Inf und TI Master, 3 LP, jedes SoSe:
Infos über *sese.tu-berlin.de* oder direkt *168200*

Beispiel-Werkzeuge

- VCC: Verifying C Compiler von Microsoft Research
- KeY: Integrated Deductive Software Design

key-project.org

KeY

Kooperation vom KIT und der TU Darmstadt

Berechnet Beweisverpflichtungen aus Java-Programmen und JML

- Basiert auf einem erweiterten Hoare-Kalkül mit Zustandsveränderungen
- Ermöglicht symbolische Ausführung von JAVA Code
- Erzeugt automatisch Gegenbeispiele und Testfälle

Wurde unter anderem verwendet um einen Fehler in JAVAs Sortieralgorithmus nachweisen

Als Plug-In für Eclipse erhältlich
http://www.key-project.org/download/

requires = pre
ensures = post

```
/*@
@ public normal_behavior
@ requires true;
@ ensures \result == (unsuccessfulOperations<=3);
@ assignable \nothing;
@*/
public /*@pure@*/ boolean isValid() {
    if (unsuccessfulOperations<=3) {
        return true;
    } else {
        return false;
    }
}
```

Zusammenfassung Hoare Kalkül

- Unterscheidet partielle versus totale Korrektheit
- Definiert durch logische Kalküle
- Beschreibt nicht notwendigerweise die vollständige Semantik, sondern eventuell nur Ausschnitte davon
- Damit können Beweise für relevante Eigenschaften vereinfacht werden
- Tool support z.B. mit KeY oder dem VCC
- Ermöglicht den Nachweis, dass eine Java Methode eine OCL Spezifikation auch tatsächlich umsetzt

Lernziele

- ❑ Wie lässt sich die Korrektheit einer Implementierung nachweisen?
- ❑ Was versteht man unter einem Hoare-Tripel?
- ❑ Welche Axiome gibt es im Hoare-Kalkül für die WHILE-Sprache?
- ❑ Wie lassen sich komplexere Terme zerlegen?
- ❑ Welche Inferenzregeln gibt es für die WHILE-Sprache?
- ❑ Was versteht man unter einer Schleifeninvariante und wofür wird sie benötigt?
- ❑ Was versteht man unter der Rule of Consequence?
- ❑ Was ist der unterschied zwischen partieller und totaler Korrektheit?
- ❑ Wie lässt sich totale Korrektheit nachweisen?
- ❑ Was versteht man unter einer Terminierungsfunktion?
- ❑ Was muss für sie gelten um totale Korrektheit nachzuweisen?