Softwaretechnik und Programmierparadigmen

VL06: Hoare Kalkül

Prof. Dr. Sabine Glesner
FG Programmierung eingebetteter Systeme
Technische Universität Berlin

Motivation

- Contracts zur Spezifikation von Methoden
- Können als Ausgangsbasis für die Implementation dienen
- Frage: Wie kann man nachweisen, dass die Implementierung den Contract erfüllt?
- Antwort 1: Testen der Implementierung gegen den Contract
- Antwort 2: Beweisen, dass das Programm den Contract erfüllt

- Voraussetzungen für Beweise
  - Formale Semantik: hier operationale Semantik der „WHILE-Sprache“
  - Beweiskalkül: hier Hoare Kalkül

Übersicht

- Syntax der WHILE-Sprache
- Semantik der WHILE-Sprache
- Hoare-Kalkül
- Beweise im Hoare-Kalkül

Syntaktische und semantische Sorten

- Zahlen (Num)
- Variable (Var)
- Arithmetische Ausdrücke (Aexp) vom Typ INT (REAL, …)
  - Variablen und
  - zusammengesetzte Ausdrücke
- Boolesche Ausdrücke (Bexp)
- Anweisungen

- Ganze Zahlen $\mathbb{Z}$ (Gleitkommazahlen, …)
- Wahrheitswerte
- Zustände (State)
- Funktionen auf den Zuständen
  - Lookup-Funktionen
  - Zustand modifizieren

Beispiel While-Sprache: Syntax (I)

- n: steht für Zahlen
- x: steht für Variablen
- a: steht für arithmetische Ausdrücke
- b: steht für Boolesche Ausdrücke
- S: steht für Anweisungen (statements)

Beispiel While-Sprache: Syntax (II)

Arithmetische Ausdrücke

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

Boolesche Ausdrücke

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

Anweisungen/Programme

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

Andere übliche Ausdrücke lassen sich
mit den vorhandenen ausdrücken

Beispielprogramme

- Vertauschen zweier Variablen
  z:=x; x:=y; y:=z

- Bedingtes Vertauschen zweier Variablen
  if x=y then skip else (z:=x; x:=y; y:=z)

- Fakultätsfunktion
  y:=1 ; while (x≠1) do (y:=x*y ; x:=x-1)

- Endlosschleife
  while true do skip

Übersicht

- Syntax der WHILE-Sprache
- Semantik der WHILE-Sprache
- Hoare-Kalkül
- Beweise im Hoare-Kalkül

Semantik: Bedeutung für Syntax

- Ganze Zahlen $\mathbb{Z}$= { ..., -2, -1, 0, 1, 2, ... }
- Wahrheitswerte T= {tt, ff}
- Zustand (s): Abbildung $\mathrm{Var} \to \mathbb{Z}$
- Lookup: Abbildung $\mathrm{State} \times \mathrm{Var} \to \mathbb{Z}$
- Update: Abbildung $\mathrm{State} \times \mathrm{Var} \times \mathbb{Z} \to \mathrm{State}$
  - Lookup(Update(s,v,z), v) = z und
  - Lookup(Update(s,v,z), v‘) = Lookup(s,v‘) sonst

Semantik für arithmetische Ausdrücke

- Funktion N: $\mathrm{Num} \rightarrow \mathbb{Z}$
  - weist Zahlen aus dem Programm Zahlen in $\mathbb{Z}$ zu
- Funktion A: $\mathrm{Aexp} \times \mathrm{State} \rightarrow \mathbb{Z}$
  - weist arithemtischen Ausdrücken Zahlen in $\mathbb{Z}$ zu

$A[\![n]\!]s = N[\![n]\!]$ $\qquad$ $A[\![x]\!]s = \mathrm{Lookup}(s,x)$

syntaktisch

semantisch

$$A[\![a_1 + a_2]\!]s = A[\![a_1]\!]s + A[\![a_2]\!]s$$
$$A[\![a_1 * a_2]\!]s = A[\![a_1]\!]s * A[\![a_2]\!]s$$
$$A[\![a_1 - a_2]\!]s = A[\![a_1]\!]s - A[\![a_2]\!]s$$

Beispielableitung

Semantisches „Minus“

- $A[\![((3+x)*y)-2]\!]s = A[\![((3+x)*y)]\!]s - A[\![2]\!]s$

$$= (A[\![(3+x)]\!]s * A[\![y]\!]s) - N[\![2]\!]$$

$$= ((A[\![3]\!]s + A[\![x]\!]s) * A[\![y]\!]s) - 2$$

$$= ((N[\![3]\!]s + A[\![x]\!]s) * A[\![y]\!]s) - 2$$

$$= ((3+ A[\![x]\!]s) * A[\![y]\!]s) - 2$$

$$= ((3+\mathrm{Lookup}(s,x)) * \mathrm{Lookup}(s,y)) - 2$$

Syntaktisches „Minus“

Semantik für Boolesche Ausdrücke

- Funktion B: $\mathrm{Bexp} \times \mathrm{State} \rightarrow \{\mathrm{tt}, \mathrm{ff}\}$

$B[\![\mathrm{true}]\!]s = tt$ $\qquad$ $B[\![\mathrm{false}]\!]s = ff$

$B[\![a_1{=}a_2]\!]s$ = tt if $A[\![a_1]\!]s{=}A[\![a_2]\!]s$ und $B[\![a_1{=}a_2]\!]$ = ff sonst

$B[\![a_1 \leq a_2]\!]s$ = tt if $A[\![a_1]\!]s \leq A[\![a_2]\!]s$ und $B[\![a_1 \leq a_2]\!]s$= ff sonst

$B[\![\neg b]\!]s$ = Not $B[\![b]\!]s$

$B[\![b_1 \wedge b_2]\!]s = B[\![b_1]\!]s$ And $B[\![b_2]\!]s$

Wohldefiniertheit der Semantik

- Syntax spezifiziert durch abstrakte Syntax
  - gibt eindeutig an, wie ein Programm in seine Bestandteile zerlegt ist
- Semantik kompositional definiert basierend auf abstrakter Syntax
  - für arithmetische und Boolesche Ausdrücke schon gesehen
  - für Anweisungen im folgenden
- Da abstrakte Syntax die Dekomposition eines Programms eindeutig festlegt, ist Semantik wohldefiniert.

Semantik für Anweisungen

- Auch strukturell über Aufbau der abstrakten Syntax
- Zwei prinzipielle operationale Möglichkeiten:
  - small-step semantics, auch strukturell operationale Semantik (SOS) genannt
  - big-step semantics, auch natürliche Semantik genannt
- Beide definiert mit Zustandsübergangssystemen
- Beide definieren operationale Semantiken
- Daneben auch denotationelle Semantik
  - Beschreibung von Programmen durch Funktionen von State nach State

Strukturell operationale Semantik (SOS)

- Definiert, wie einzelne Schritte bei der Ausführung eines Programms ablaufen
- Weist Programm + Anfangszustand ein neues Programm und einen neuen Zustand zu
- Bei Terminierung: weist Programm + Anfangszustand einen Endzustand zu

SOS-Semantik für While-Sprache (I)

- Definiert einzelne Schritte bei der Programmberechnung:
- Konfigurationenmenge Γ (Gamma):
  - $\Gamma = \{(S,s) \mid S \in \text{While-Sprache}, s \in \text{State}\} \cup \text{State}$
  - Konfiguration beschreibt Zustand der semantischen Maschine während der Programmausführung
- Übergangsrelation $\Longrightarrow$:
  - $\Longrightarrow \subseteq \{(S,s) \mid S \in \text{While-Sprache}, s \in \text{State}\} \times \Gamma$

SOS-Semantik While-Sprache (II)

- Zwei mögliche Übergänge:
  terminierend oder nicht-terminierend
- Programm ist terminiert: $(S,s) \Longrightarrow s'$
  - Die Berechnung ist mit diesem Schritt beendet.
- Programm ist (noch) nicht terminiert: $(S,s) \Longrightarrow (S',s')$
  - Die Berechnung ist noch nicht beendet.
  - „Neues Programm“ S‘ muss im Zustand s‘ ausgewertet werden.

Ende 20.11.14

Inferenzregeln und Axiome

Voraussetzung (Assumption):

besteht aus direkten Unterprogrammen des Programms in der Konklusion oder aus einem Programm, das daraus konstruiert wurde

Randbedingung

$$\frac{\mathcal{B}[\![b]\!]s = tt \qquad (S_1, s) \Longrightarrow (S_1', s')}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow (S_1', s')}$$

Konklusion

Axiom:
Inferenzregel ohne Voraussetzungen

SOS-Semantik While-Sprache (III)

$(x{:=}a,s) \Longrightarrow s[x \rightarrow \mathcal{A}[\![a]\!]s]$

$(\text{skip},s) \Longrightarrow s$

$$\frac{(S_1,s) \Longrightarrow (S_1', s')}{(S_1;S_2,s) \Longrightarrow (S_1';S_2,s')} \qquad \frac{(S_1,s) \Longrightarrow s'}{(S_1;S_2,s) \Longrightarrow (S_2,s')}$$

$$\frac{\mathcal{B}[\![b]\!]s = tt}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow (S_1,s)} \qquad \frac{\mathcal{B}[\![b]\!]s = ff}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow (S_2,s)}$$

$(\text{while } b \text{ do } S,\ s) \Longrightarrow (\text{if } b \text{ then } (S;\text{while } b \text{ do } S) \text{ else skip},s)$

Alternativen für bedingte Anweisung

$$\frac{\mathcal{B}[\![b]\!]s = tt}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow (S_1, s)}$$

$$\frac{\mathcal{B}[\![b]\!]s = ff}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow (S_2, s)}$$

Alternative:

$$\frac{\mathcal{B}[\![b]\!]s = tt \quad (S_1, s) \Longrightarrow s'}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow s'}$$

$$\frac{\mathcal{B}[\![b]\!]s = tt \quad (S_1, s) \Longrightarrow (S_1', s')}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow (S_1', s')}$$

$$\frac{\mathcal{B}[\![b]\!]s = ff \quad (S_2, s) \Longrightarrow s'}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow s'}$$

$$\frac{\mathcal{B}[\![b]\!]s = ff \quad (S_2, s) \Longrightarrow (S_2', s')}{(\text{if } b \text{ then } S_1 \text{ else } S_2,\ s) \Longrightarrow (S_2', s')}$$

Beispielableitung - Swap

- (z:=x; x:=y; y:=z, [x=5,y=42,z=2])

$\Longrightarrow$(x:=y; y:=z, [x=5,y=42,z=5])

$\Longrightarrow$(y:=z, [x=42,y=42,z=5])

$\Longrightarrow$[x=42,y=5,z=5]

Beispielableitung - Fakultät

(y:=1 ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=3,y=42])

$\Longrightarrow$(while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=3,y=1])

$\Longrightarrow$(if x $\geq$ 1 then (y:=x*y ; x:=x-1) ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1)
else skip, [x=3,y=1])

$\Longrightarrow$((y:=x*y ; x:=x-1) ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=3,y=1])

$\Longrightarrow$(x:=x-1 ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=3,y=3])

$\Longrightarrow$(while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=2,y=3])

$\Longrightarrow$(if x $\geq$ 1 then (y:=x*y ; x:=x-1) ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1)
else skip, [x=2,y=3])

$\Longrightarrow$((y:=x*y ; x:=x-1) ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=2,y=3])

$\Longrightarrow$(x:=x-1 ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=2,y=6])

$\Longrightarrow$(while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=1,y=6])

Beispielableitung - Fakultät

(while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=1,y=6])

$\Longrightarrow$(if x $\geq$ 1 then (y:=x*y ; x:=x-1) ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1)
else skip, [x=1,y=6])

$\Longrightarrow$((y:=x*y ; x:=x-1) ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=1,y=6])

$\Longrightarrow$(x:=x-1 ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=1,y=6])

$\Longrightarrow$(while (x $\geq$ 1) do (y:=x*y ; x:=x-1) , [x=0,y=6])

$\Longrightarrow$(if x $\geq$ 1 then (y:=x*y ; x:=x-1) ; while (x $\geq$ 1) do (y:=x*y ; x:=x-1)
else skip, [x=0,y=6])

$\Longrightarrow$(skip, [x=0,y=6])

$\Longrightarrow$[x=0,y=6]

Übersicht

- Syntax der WHILE-Sprache
- Semantik der WHILE-Sprache
- Hoare-Kalkül
- Beweise im Hoare-Kalkül

Hoare Kalkül

- Operationale und denotationelle Semantiken beschreiben die vollständige Semantik
- Beweise basierend auf ihnen werden schnell zu detailliert. Nicht alle spezifizierten Eigenschaften sind für einen bestimmten Beweis notwendig.
- Daher: Beschränkung auf einige bestimmte Eigenschaften.

Hoare Kalkül

- Beispielprogramm:
  y:=1; while (x≠1) do (y:=x*y;x:=x-1)

- Wenn x=n vor Ausführung gilt, dann gilt y=n! nach Ausführung (sofern die Ausführung terminiert).
  - $\Longrightarrow$ partielle Korrektheit

- Gilt x=n vor Ausführung, dann terminiert das Programm und y hat den Wert n! (Fakultät).
  - $\Longrightarrow$ totale Korrektheit

Vor- und Nachbedingungen

- Zwei Arten von Variablen:
  - Programmvariablen (z.B. x, y)
  - logische Variablen (z.B. n)

- Beispiel:
  $\{ x = n \wedge n \geq 0\}$
  y:=1; while x≠1 do (y:=y*x; x:=x-1)
  $\{ y = n!\}$

Partielle Korrektheit

- Ein Programm ist partiell korrekt bzgl. einer Vor- und einer Nachbedingung, falls immer dann, wenn der initiale Zustand die Vorbedingung erfüllt und wenn das Programm terminiert, der Endzustand die Nachbedingung erfüllt.
- Für alle Zustände s gilt: wenn P(s) und (S,s) $\Longrightarrow^*$ s‘, dann Q(s‘)

Annahme: Programm terminiert nach endlich vielen Schritten

- Notation: {P} S {Q}
- Hoare Kalkül: logisches System, um partielle Korrektheitseigenschaften nachzuweisen.

Beispielprogramme

- Eine Vor- und eine Nachbedingung:

- $\{ x=n \land y=m \}$ z:=x; x:=y; y:=z $\{ y=n \land x=m \}$
- $\{ x=n \land y=m \}$ if x=y then skip else (z:=x; x:=y; y:=z) $\{ y=n \land x=m \}$
- $\{ x=n \land y=m \}$ while true do skip $\{ y=n \land x=m \}$
- $\{ x=n \land y=m \}$ while true do skip { false}

Beobachtung: Diese Vor- und Nachbedingungen sind korrekt.

Hoare Kalkül – Axiome

$\{ P[x \mapsto E] \}\ x:=E\ \{ P \}$

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

$\{ P \}\ \text{skip}\ \{ P \}$

- skip ändert den Zustand nicht und bewahrt dadurch jede Vorbedingung

Hoare Kalkül – Regeln

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

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

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

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

Hoare Kalkül – Regel für Schleifen

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

- 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

Hoare Kalkül – Rule of Consequence

$$\frac{\{\ P'\ \}\ S\ \{\ Q'\ \}}{\{\ P\ \}\ S\ \{\ Q\ \}} \quad \text{falls } P \longrightarrow P' \text{ und } Q' \longrightarrow Q$$

- Wurde ein Hoare Tripel { P‘ } S { Q‘ } gezeigt, darf:
  - die **Vorbedingung verschärft** werden
    $P \longrightarrow P'$
  - die **Nachbedingung abgeschwächt** werden
    $Q' \longrightarrow Q$

Totale Korrektheit: Definition

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

- Für alle Zustände s gilt:
  - wenn P(s) und (S,s) $\Longrightarrow^*$ s‘, dann Q(s‘)
    UND
  - wenn P(s), dann existiert ein s‘ so dass (S,s) $\Longrightarrow^*$ s‘

Totale Korrektheit: Nachweis

- Kritischer Fall: While-Schleife

- Vorgehen:
  - Definiere parametrisierte Vorbedingungen P(z) für nichtnegative ganze Zahlen z.
  - Wenn P(0) gilt, wird die While-Schleife nicht ausgeführt
  - Wenn bei Betreten der While-Schleife P(z) gilt, dann gilt nach Abarbeiten des Rumpfes P(z‘), wobei z‘< z gilt.

Übersicht

- Syntax der WHILE-Sprache
- Semantik der WHILE-Sprache
- Hoare-Kalkül
- Beweise im Hoare-Kalkül

Sequentielles Beispiel: Swap

z:=x;

x:=y;

y:=z;

Swap-Befehle

Sequentielles Beispiel: Swap

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

z:=x;

x:=y;

y:=z;

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

Sequentielles Beispiel: Swap

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

z:=x;

x:=y;

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

y:=z;

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

Ersetzungs-
Axiom wurde
angewandt

Sequentielles Beispiel: Swap

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

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

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

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

Ersetzungs-Axiom wurde angewandt

z:=x;

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

x:=y;

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

y:=z;

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

Sequentielles Beispiel: Swap

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

$\equiv$

Die Ableitung ist äquivalent zur Vorbedingung

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

z:=x;

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

x:=y;

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

y:=z;

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

Fakultätsfunktion

y := 1;

while x $\geq$ 1 do

y := y * x;

x := x - 1

Fakultätsfunktion

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

Vorbedingung

y := 1;

while x ≥ 1 do

    y := y * x;

    x := x - 1

Nachbedingung

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

Fakultätsfunktion

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

y := 1;

while x ≥ 1 do

    y := y * x;

    x := x - 1

$\{ y = n! \}$

Schleifen-Invariante:

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

Fakultätsfunktion

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

y := 1;

while x ≥ 1 do

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

Schleifen-Invariante mit Schleifenbedingung

y := y * x;

x := x - 1

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

Schleifen-Invariante

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

$\{ y = n!\}$

Schleifen-Invariante mit negierter Schleifenbedingung

Fakultätsfunktion

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

y := 1;

while x $\geq$ 1 do

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

Ersetzungs-Axiom

y := y * x;

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

x := x - 1

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

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

$\{ y = n!\}$

Fakultätsfunktion

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

y := 1;

while x ≥ 1 do

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

Ersetzungs-Axiom

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

y := y * x;

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

x := x - 1

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

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

$\{ y = n!\}$

Fakultätsfunktion

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

y := 1;

while x ≥ 1 do

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

$\equiv$

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

y := y * x;

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

x := x - 1

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

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

$\{ y = n!\}$

Äquivalent

Fakultätsfunktion

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

y := 1;

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

Invariante ohne Schleifenbedingung

while x ≥ 1 do

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

$\equiv$

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

y := y * x;

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

x := x - 1

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

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

$\{ y = n!\}$

Fakultätsfunktion

27.11.14

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

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

Ersetzungs-Axiom wurde angewandt

y := 1;

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

while x ≥ 1 do

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

$\equiv$

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

y := y * x;

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

x := x - 1

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

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

$\{ y = n!\}$

Fakultätsfunktion

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

$\equiv$

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

y := 1;

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

while x ≥ 1 do

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

$\equiv$

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

y := y * x;

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

x := x - 1

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

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

$\{ y = n!\}$

Äquivalent

Totale Korrektheit: Nachweis

- Kritischer Fall: While-Schleife

- Vorgehen:
  - Definiere Terminierungsfunktion t(…), die als Eingabe Daten des Programms nimmt und natürliche Zahlen n (also n>=0) zurück gibt.
  - Wenn t(…) = 0 gilt, wird die While-Schleife nicht ausgeführt
  - Wenn bei Betreten der While-Schleife t(…) = n gilt, dann gilt nach Abarbeiten des Rumpfes t(…) = n‘, wobei n‘< n gilt.

Fakultätsfunktion

Terminierungsfunktion für die While-Schleife:

y := 1;

while x $\geq$ 1 do

y := y * x;

x := x - 1

Fakultätsfunktion

y := 1;

while x $\geq$ 1 do

Terminierungsfunktion
für die While-Schleife:
t(x) = x

y := y * x;

x := x - 1

Zusammenfassung Hoare Kalkül

- Unterscheide 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