Grundlagen der Programmierung

Vorlesung 6 vom 18.11.2004

Sebastian Iwanowski
FH Wedel

Prädikatenlogik

Arithmetische Vergleichsprädikate:

Bilde das Gegenteil von:

1) $(x>0) \vee ((y+x)\leq 0)$

2) $\forall y<0\,((x>0) \vee ((y+x)\leq 0))$

Grundlagen der Programmierung

1. Einführung
    - Grundlegende Eigenschaften von Algorithmen und Programmen
2. Logik
    - Aussagenlogik
    - Prädikatenlogik
3. Programmentwicklung und –verifikation
    - Grundlagen der Programmverifikation
    - Verbundanweisungen
    - Verzweigungen
    - Schleifen
    - Modularisierung
    - Rekursion
4. Entwurf und Analyse von Algorithmen
    - Klassifikation von Algorithmen
    - Programmierung von Algorithmen
    - Bewertung von Algorithmen

Programmentwicklung

Konstruktionsproblem:

Gegeben eine Spezifikation:

Funktion, die einem Argument einen Funktionswert zuordnet

Entwirf ein Programm, das jedes Argument des Definitionsbereichs als Eingabe akzeptiert und den zugehörigen Funktionswert als Ausgabe produziert.

Verifikationsproblem:

Gegeben eine Spezifikation und ein Programm:

Beweise, dass das Programm für jedes Argument der Spezifikation als Eingabe den zugehörigen Funktionswert der Ausgabe berechnet.

Zusatzaufgaben:

1. Was wird bei anderen Eingaben als den zulässigen Argumenten berechnet ?
2. Welche Bedingungen muss die Eingabe erfüllen, um bestimmte Ausgaben auszuschließen ?

Programmverifikation

Etwas allgemeiner:

Gegeben eine Spezifikation:

Funktion, die Startzuständen eindeutige Endzuständen zuordnet.

Gegeben ein Programm:

Beweise, dass das Programm für jeden Startzustand den von der Spezifikation geforderten Endzustand erreicht und danach stoppt.

Zusatzaufgaben:

1. Welche Zustände werden bei anderen als den geforderten Startzuständen erreicht ?
2. Welche Bedingungen müssen die Startzustände erfüllen, um bestimmte Endzustände auszuschließen ?

Anmerkung: Die gleichen Aufgaben können natürlich auch für beliebige Zwischenzustände untersucht werden.

Programmverifikation

Formalismus zum Lösen der Verifikationsaufgaben: Hoare-Tripel

$$\{ \varphi \} \quad P \quad \{ \psi \}$$

Vorbedingung Programm Nachbedingung

Da ein Programm eine Sequenz von Anweisungen ist, kann man dieses Vorgehen auf die einzelnen Anweisungen reduzieren:

$$\{ \varphi \} \quad S \quad \{ \psi \}$$

Vorbedingung Anweisung Nachbedingung

Fragestellungen:

1. Gegeben φ, finde stärkste Nachbedingung ψ
2. Gegeben ψ, finde schwächste Vorbedingung φ

Programmverifikation

Beispiel für das Arbeiten mit Hoare-Tripeln:

```
{ V }          φ
z := x • y ;   S
{ z ≥ 0 }      ψ
w := sqrt (z)
```

$\varphi_1 \Leftrightarrow (x > 0) \wedge (y > 0)$ ist eine Vorbedingung für ψ: $\{\varphi_1\}\ S\ \{\psi\}$

$\varphi_2 \Leftrightarrow (x < 0) \wedge (y < 0)$ ist auch eine Vorbedingung für ψ: $\{\varphi_2\}\ S\ \{\psi\}$

Welche ist die schwächste Vorbedingung V ?

$V \Leftrightarrow ((x>0)\wedge(y>0)) \vee ((x<0)\wedge(y<0)) \vee (x=0) \vee (y=0)$
ist die schwächste Vorbedingung für ψ:

$\{V\}\ S\ \{\psi\}$ Außerdem gilt: $\{\varphi\}\ S\ \{\psi\} \Rightarrow (\varphi \rightarrow V)$

Programmverifikation

Logischer Zusammenhang von Vorbedingungen:

$$(\{\varphi_1\}\ S\ \{\psi\}) \wedge (\varphi_2 \rightarrow \varphi_1) \quad \Rightarrow \quad \{\varphi_2\}\ S\ \{\psi\}$$

Die Vertauschung gilt nicht:

~~$(\{\varphi_1\}\ S\ \{\psi\}) \wedge (\varphi_1 \rightarrow \varphi_2) \quad \Rightarrow \quad \{\varphi_2\}\ S\ \{\psi\}$~~

Logischer Zusammenhang von Nachbedingungen:

$$(\{\varphi\}\ S\ \{\psi_1\}) \wedge (\psi_1 \rightarrow \psi_2) \quad \Rightarrow \quad \{\varphi\}\ S\ \{\psi_2\}$$

Die Vertauschung gilt nicht:

~~$(\{\varphi\}\ S\ \{\psi_1\}) \wedge (\psi_2 \rightarrow \psi_1) \quad \Rightarrow \quad \{\varphi\}\ S\ \{\psi_2\}$~~

Verifikation von Zuweisungen

Definition einer Zuweisung:

`x := Ausdruck`

Hierbei ist **`x`** ein beliebiger Variablenname und **`Ausdruck`** eine beliebige Funktion, die unter Umständen von Variablen abhängt. Die Variablen in **`Ausdruck`** müssen zum Zeitpunkt der Anweisung Werte haben.

Funktionsweise einer Zuweisung:

{ Prädikate für die Werte von x_1, ..., x_k
und eventuell weitere Prädikate }

$x := Ausdruck (x_1, ..., x_k)$

{ Neues Prädikat für den Wert von x und
eventuell weitere Prädikate }

Zunächst wird **`Ausdruck`** $(x_1, ..., x_k)$ ausgewertet.
Der sich ergebende Funktionswert wird danach in die Variable x geschrieben.

Verifikation von Zuweisungen

Beispiel für die Verifikation einer Zuweisung:

```
{ (x > 0) ∧ (y > 0)}      φ

z := x - sqrt (y) ;       S

{ N }                     ψ
```

Welche ist die stärkste Nachbedingung N ?

Welche ist die schwächste Vorbedingung für ψ ⇔ (z > 0) ?

Verifikation von Zuweisungen

Die Zuweisungsvariable darf auch im zu berechnenden Ausdruck vorkommen:

```
{ (x > 0) }                 φ

x := x - sqrt (x) ;         S

{ N }                       ψ
```

Welche ist die stärkste Nachbedingung N ?

Welche ist die schwächste Vorbedingung für ψ ⇔ (x > 0) ?

Verifikation von Verbundanweisungen

Definition einer Verbundanweisung:

```
begin
     Anweisung 1;
     Anweisung 2;
     ...
     Anweisung n
end
```

Hierbei dürfen die Anweisungen 1 bis n beliebige Anweisungen sein: von einfachen Zuweisungen bis hin zu ineinandergeschachtelten Kontrollstrukturen.

Funktionsweise einer Verbundanweisung:

Die Anweisungen werden der Reihe nach **hintereinander** ausgeführt.
Eine **parallele** Ausführung findet **nicht** statt.

Verifikation von Verbundanweisungen

Verifikationstechnik:

```
{Vorbedingung}
begin
     Anweisung 1;
     {Zwischenbedingung 1}
     Anweisung 2;
     {Zwischenbedingung 2}
     ...
     {Zwischenbedingung n-1}
     Anweisung n
end
{Nachbedingung}
```

Achtung:

Geübte Verifizierer schreiben nicht jede Zwischenbedingung auf. Sie müssen aber alle Zwischenbedingungen im Kopf durcharbeiten, da es keine parallele Abarbeitung gibt !

Verifikation von Verbundanweisungen

Beispiel für die Verifikation einer Verbundanweisung:

Spezifikation: 2 mit Werten belegte Variablen sollen ihre Werte tauschen.

Programm:

```
{ (x = Wert1) ∧ (y = Wert2)}  φ
begin
      x := y ;
      { (x = Wert2) ∧ (y = Wert2)}
      y := x ;
      { (x = Wert2) ∧ (y = Wert2)}
end
{ (x = Wert2) ∧ (y = Wert1)}  ψ
```

Unter welchen Bedingungen ist das Programm korrekt ?

Antwort: nur wenn **`Wert1 = Wert2`**

Verifikation von Verbundanweisungen

Wie tauscht man verschiedene Werte ?

Lösung:

```
{ (x = Wert1) ∧ (y = Wert2)}  φ
begin
     z := x ;
     { (x = Wert1) ∧ (y = Wert2) ∧ (z = Wert1)}
     x := y ;
     { (x = Wert2) ∧ (y = Wert2) ∧ (z = Wert1)}
     y := z ;
     { (x = Wert2) ∧ (y = Wert1) ∧ (z = Wert1)}
end
{ (x = Wert2) ∧ (y = Wert1)}  ψ
```

Keine Einschränkung der Vor- oder Nachbedingung !

Damit ist bewiesen, dass das Programm die Spezifikation erfüllt.

Verifikation von Verzweigungen

Definition einer Verzweigung:

```
if Ausdruck
then
     then-Anweisung
else
     else-Anweisung
```

`Ausdruck` muss eine logische Funktion sein, die nur von Variablen abhängen darf, die mit Werten belegt sind.

Funktionsweise:

Zunächst wird **`Ausdruck`** ausgewertet.
Wenn **`Ausdruck`** wahr ist, wird nur die **`then-Anweisung`** ausgeführt.
Wenn **`Ausdruck`** falsch ist, wird nur die **`else-Anweisung`** ausgeführt.

Verifikation von Verzweigungen

Verifikationstechnik:

```
{Vorbedingung}
if Ausdruck
then
        {then-Vorbedingung}
        then-Anweisung
        {then-Nachbedingung}
else
        {else-Vorbedingung}
        else-Anweisung
        {else-Nachbedingung}
{Nachbedingung}
```

Aufgrund der Funktionsweise einer Verzweigung muss gelten:

1) $\text{then-Vorbedingung} \Leftrightarrow (\text{Vorbedingung} \wedge \text{Ausdruck})$
2) $\text{else-Vorbedingung} \Leftrightarrow (\text{Vorbedingung} \wedge \neg\ \text{Ausdruck})$
3) $\text{then-Nachbedingung} \Rightarrow \text{Nachbedingung}$
4) $\text{else-Nachbedingung} \Rightarrow \text{Nachbedingung}$

Verifikation von Verzweigungen

Verifikationstechnik:

```
{Vorbedingung}
if Ausdruck
then
        {then-Vorbedingung}
        then-Anweisung
        {then-Nachbedingung}
else
        {else-Vorbedingung}
        else-Anweisung
        {else-Nachbedingung}
{Nachbedingung}
```

Damit gilt:

$$\{\text{Vorbedingung}\}\ \text{if then ... else ...}\ \{\text{Nachbedingung}\}$$

$$\Leftrightarrow$$

1) $(\{\text{Vorbedingung} \wedge \text{Ausdruck}\}\ \text{then-Anweisung}\ \{\text{Nachbedingung}\})$
2) $\wedge\ (\{\text{Vorbedingung} \wedge \neg\ \text{Ausdruck}\}\ \text{else-Anweisung}\ \{\text{Nachbedingung}\})$

Verifikation von Verzweigungen

Beispiel für die Verifikation einer Verzweigung:

```
{ Vorbedingung }    φ

if (y>0)
  then
    z := x • y
  else
    z := x / y

{ z ≥ 0 }           ψ
```

Welches ist die schwächste Vorbedingung φ für ψ ?

1. Aufgabe: $\{ \varphi_1 \wedge (y>0) \}\ z := x \cdot y\ \{ z \geq 0 \}$

2. Aufgabe: $\{ \varphi_2 \wedge (y \leq 0) \}\ z := x / y\ \{ z \geq 0 \}$

Lösung: $\varphi \Leftrightarrow (\varphi_1 \wedge (y>0)) \vee (\varphi_2 \wedge (y \leq 0))$

Beim nächsten Mal:

Programmentwicklung und –verifikation:
Verzweigungen (Fortsetzung)
Schleifen (Anfang)