Grundlagen der Programmierung

Vorlesung 7 vom 25.11.2004

Sebastian Iwanowski
FH Wedel

Offene Fragen aus Vorlesung 6

1. Warum werden bei Negationen von Quantoren die Definitionsbereiche nicht negiert ?

$\forall\ y{\in}D\ (P(y))$ ist Kurzschreibweise von: $\forall\ y\ (y{\in}D \rightarrow P(y))$

$\Leftrightarrow \forall\ y\ (y{\notin}D \vee P(y))$

$\exists\ y{\in}D\ (P(y))$ ist Kurzschreibweise von: $\exists\ y\ ((y{\in}D \wedge (P(y))$

Daher gilt für die Negation:

$\neg(\forall\ y{\in}D\ (P(y))) \Leftrightarrow \neg(\forall\ y\ (y{\notin}D \vee P(y))) \Leftrightarrow \exists\, y\, (y{\in}D \wedge \neg P(y))$

$\Leftrightarrow \exists\ y{\in}D\ (\neg P(y))$

$\neg(\exists\ y{\in}D\ (P(y))) \Leftrightarrow \neg(\exists\ y\ (y{\in}D \wedge P(y))) \Leftrightarrow \forall\ y\ (y{\notin}D \vee \neg P(y))$

$\Leftrightarrow \forall\ y{\in}D\ (\neg P(y))$

Offene Fragen aus Vorlesung 6

2. Anwendung auf folgende Aufgabe:

Bilde das Gegenteil von:

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

Lösung:

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

ist Kurzschreibweise von: $\forall y\ (\neg(y<0) \vee ((x>0) \vee ((y+x)\leq 0)))$

„Gegenteil“ soll heißen: Negation der oben angegebenen Formel

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

$\Leftrightarrow \exists y\ ((y<0) \wedge (x\leq 0) \wedge ((y+x)>0)))$

$\Leftrightarrow \exists y<0\ ((x\leq 0) \wedge ((y+x)>0)))$

$\Leftrightarrow \bot$ **für alle x**

Also ist die Negation ein Widerspruch

Offene Fragen aus Vorlesung 6

3. Unterscheide, wo die Negation steht:

Aussagen für negierte Formeln: $(F(x) \leftrightarrow \top) \Leftrightarrow (\neg F(x) \leftrightarrow \bot)$

(gilt für *beliebige* x)

Damit gilt:

$F(x)$ ist Tautologie $\Leftrightarrow$ $\neg F(x)$ ist Widerspruch

$F(x)$ ist erfüllbar $\Leftrightarrow$ $\neg F(x)$ ist widerlegbar

Also ist die Formel $F(x) \Leftrightarrow \forall y{<}0\ ((x{>}0) \vee ((y{+}x){\leq}0))$ eine Tautologie,
da $\neg F(x) \Leftrightarrow \forall y{<}0\ ((x{>}0) \vee ((y{+}x){\leq}0))$ ein Widerspruch ist.

Negierte Aussagen für dieselbe Formel $F(x)$:

$F(x)$ ist Tautologie	$\Leftrightarrow$	$F(x)$ ist **nicht** widerlegbar
$F(x)$ ist widerlegbar	$\Leftrightarrow$	$F(x)$ ist **nicht** Tautologie
$F(x)$ ist erfüllbar	$\Leftrightarrow$	$F(x)$ ist **nicht** widersprüchlich
$F(x)$ ist widersprüchlich	$\Leftrightarrow$	$F(x)$ ist **nicht** erfüllbar

Unterscheide: $F(x)$ ist **nicht** widerlegbar $\neq$ $\neg F(x)$ ist widerlegbar

Grundlagen der Programmierung

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

Verifikation von Zuweisungen

Zuweisung ohne Verwendung der Zuweisungsvariable im Ausdruck:

$\{V(x_1,...,x_k)\}$ φ

$z := A(x_1,...,x_k);$ S

$\{N(x_1,...,x_k,z)\}\}$ ψ

Finde die stärkste Nachbedingung ψ zu gegebenem φ:

$$N(x_1,...,x_k,z) \Leftrightarrow V(x_1,...,x_k) \wedge (z = A(x_1,...,x_k))$$

Beispiel:

$\{ (x > 0) \wedge (y > 0)\}$ φ
$z := x - sqrt\ (y)\ ;$ S
$\{ N \}$ ψ

Stärkste Nachbedingung:

$$N(x,y,z) \Leftrightarrow (x > 0) \wedge (y > 0) \wedge (z = x - sqrt\ (y))$$

Verifikation von Zuweisungen

Zuweisung mit Verwendung der Zuweisungsvariable im Ausdruck:

$\{V_1(x_1,...,x_k) \wedge V_2(x)\}$ φ

$x := A(x_1,...,x_k,x);$ S

$\{N(x_1,...,x_k,x)\}\}$ ψ

Finde die stärkste Nachbedingung ψ zu gegebenem φ:

$N(x_1,...,x_k,x) \Leftrightarrow$
$V_1(x_1,...,x_k) \wedge$
$(x \in$ Wertebereich von $A(x_1,...,x_k,z)$
für $(V_1(x_1,...,x_k) \wedge V_2(z))$

Beispiel:

```
{ (x > 0) }                φ
x := x - sqrt (x) ;        S
{ N }                      ψ
```

Stärkste Nachbedingung:

$N(x) \Leftrightarrow x \in$ Wertebereich von $f(z)= z - sqrt\ (z)$ für $(z > 0)$

Verifikation von Zuweisungen

$\{V(x_1,...,x_k,z)\}$ φ

$z := A(x_1,...,x_k,z);$ S

$\{N_1(x_1,...,x_k) \wedge N_2(z)\}$ ψ

Finde die schwächste Vorbedingung φ zu gegebenem ψ:

$$V(x_1,...,x_k,z) \Leftrightarrow N_1(x_1,...,x_k) \wedge N_2(A(x_1,...,x_k,z)))$$

Beispiel:

```
{V}                          φ
x := x - sqrt (x) ;          S
{ x > 0 }                    ψ
```

Schwächste Vorbedingung:

$V(x) \Leftrightarrow (x - sqrt\ (x) > 0)$

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

Verifikation von Verzweigungen

Verifikationstechnik:

{Vorbedingung}	φ
if Ausdruck	β
then	
{then-Vorbedingung}	φ_1
then-Anweisung	
{then-Nachbedingung}	ψ_1
else	
{else-Vorbedingung}	φ_2
else-Anweisung	
{else-Nachbedingung}	ψ_2
{Nachbedingung}	ψ

Berechnung der schwächsten Vorbedingung: Gegeben ψ, berechne φ

1) Setze $\psi_1 = \psi$ und berechne das schwächste φ_1

2) Setze $\psi_2 = \psi$ und berechne das schwächste φ_2

3) Lösung: $\varphi \Leftrightarrow (\varphi_1 \wedge \beta) \vee (\varphi_2 \wedge \neg\beta)$

Verifikation von Verzweigungen

Verifikationstechnik:

{Vorbedingung}	φ
if Ausdruck	β
then	
{then-Vorbedingung}	φ_1
then-Anweisung	
{then-Nachbedingung}	ψ_1
else	
{else-Vorbedingung}	φ_2
else-Anweisung	
{else-Nachbedingung}	ψ_2
{Nachbedingung}	ψ

Berechnung der stärksten Nachbedingung: Gegeben φ, berechne ψ

1) Setze $\varphi_1 = \varphi \wedge \beta$ und berechne das stärkste ψ_1

1) Setze $\varphi_2 = \varphi \wedge \neg\beta$ und berechne das stärkste ψ_2

3) Lösung: $\psi \Leftrightarrow \psi_1 \vee \psi_2$

Verifikation von Verzweigungen

Manche Programmiersprachen kennen Mehrfachverzweigungen:

```
case Ausdruck of
     Wert_1: Anweisung_1
     Wert_2: Anweisung_2
     ...
     Wert_n: Anweisung_n
     else else-Anweisung
end
```

Ausdruck muss eine Funktion mit diskretem Wertebereich sein, die nur von Variablen abhängen darf, die mit Werten belegt sind.
$\mathbf{Wert_1, ..., Wert_n}$ müssen zulässige Wertebereiche von **Ausdruck** sein.

Funktionsweise:

Zunächst wird **Ausdruck** ausgewertet. Das Ergebnis sei **w**.
Wenn **w** in $\mathbf{Wert_1}$ liegt, wird nur $\mathbf{Anweisung_1}$ ausgeführt.
Wenn **w** in $\mathbf{Wert_2 \setminus Wert_1}$ liegt, wird nur $\mathbf{Anweisung_2}$ ausgeführt.
...
Wenn **w** in $\mathbf{Wert_n \setminus \{Wert_1 \wedge ... \wedge Wert_{n-1}\}}$ liegt, wird nur $\mathbf{Anweisung_n}$ ausgeführt.
Wenn **w** nicht in $\mathbf{\{Wert_1 \wedge ... \wedge Wert_n\}}$ liegt, wird nur die **else-Anweisung** ausgeführt.

Beim nächsten Mal:

Programmentwicklung und –verifikation:
Schleifen