Grundlagen der Programmierung

Vorlesung 6
Sebastian Iwanowski
FH Wedel

Verifikation von Zuweisungen

Der allgemeine Fall einer Zuweisung:

`{ V (x) }`	φ
`x := f(x) ;`	**S**
`{ N (x) }`	ψ

Verfahren zur Berechnung der schwächsten Vorbedingung zu gegebener Nachbedingung:

- Ersetze in N (x) jedes x durch f (x) ➔ Das Ergebnis ist die schwächste Vorbedingung.

Verfahren zur Berechnung der stärksten Nachbedingung zu gegebener Vorbedingung:

- Ersetze in V (x) jedes x durch f (x) ➔ Das Ergebnis ist eine Nachbedingung,
  aber nicht unbedingt die stärkste !

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:

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

Verifikation von Zuweisungen

Zuweisung mit Verwendung der Zuweisungsvariable im Ausdruck:

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

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

$$\{N(x_1,...,x_k,x)\}\} \qquad \psi$$

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 $A(x_1,...,x_k,z)$

wenn gilt: $(V_1(x_1,...,x_k) \wedge V_2(z))$

Beispiel: $\{ (x > 0) \}$ φ

x := x - sqrt (x) ; S

$\{ N \}$ ψ

Stärkste Nachbedingung:

$N(x) \Leftrightarrow x \in$ Wertebereich von $f(z) = z - \mathrm{sqrt}\ (z)$ 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:

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

Programmverifikation allgemein

Nachtrag zu Hoare-Tripeln:

$\{ \varphi \}$	S	$\{ \psi \}$	bedeutet:
Vorbedingung	**Anweisung**	**Nachbedingung**	

Bei Vorliegen von φ gilt nach Beendigung der Anweisung S die Bedingung ψ.

<u>Merke:</u> φ_1 ist schwächer als φ_2 bedeutet: $\varphi_2 \rightarrow \varphi_1$ (φ_2 ist stärker als φ_1)

Also gilt: $\top$ (w) ist die schwächste aller Bedingungen und $\perp$ (f) die stärkste.

Folgerung: Die schwächste Vorbedingung für $\top$ ist die Bedingung, die garantiert, dass S zu einem Ende kommt.

Forderung (Axiom): Die schwächste Vorbedingung für $\perp$ ist $\perp$.

Dijkstra: Gesetz des ausgeschlossenen Wunders.

Grundlagen der Programmierung

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

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) then-Vorbedingung ⇔ (Vorbedingung ∧ Ausdruck)
2) else-Vorbedingung ⇔ (Vorbedingung ∧ ¬ Ausdruck)
3) then-Nachbedingung ⇒ Nachbedingung
4) else-Nachbedingung ⇒ 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:

{Vorbedingung} if then ... else ... {Nachbedingung}

$\Leftrightarrow$

1) ({Vorbedingung $\wedge$ Ausdruck} then-Anweisung {Nachbedingung})

2) $\wedge$ ({Vorbedingung $\wedge$ $\neg$ Ausdruck} else-Anweisung {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: $\{\ \varphi_1 \land (y>0)\ \}\ z := x \cdot y\ \{\ z \geq 0\ \}$

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

Lösung: $\varphi \Leftrightarrow (\varphi_1 \land (y>0)) \lor (\varphi_2 \land (y\leq 0))$

Verifikation von Verzweigungen

Verifikationstechnik:

```
{Vorbedingung}                        φ
if Ausdruck                           β
then
      {then-Vorbedingung}             φ1
      then-Anweisung
      {then-Nachbedingung}            ψ1
else
      {else-Vorbedingung}             φ2
      else-Anweisung
      {else-Nachbedingung}            ψ2
{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}           φ1
      then-Anweisung
      {then-Nachbedingung}          ψ1
else
      {else-Vorbedingung}           φ2
      else-Anweisung
      {else-Nachbedingung}          ψ2
{Nachbedingung}                     ψ
```

Berechnung der stärksten Nachbedingung: Gegeben φ, berechne ψ

1) Setze $\varphi_1 = \varphi \wedge \beta$ und berechne das stärkste ψ_1
2) Setze $\varphi_2 = \varphi \wedge \neg\beta$ und berechne das stärkste ψ_2
3) Lösung: $\psi \Leftrightarrow (\psi_1 \wedge \beta) \vee (\psi_2 \wedge \neg\beta)$

Verifikation von Verzweigungen

Verifikationstechnik: Achtung Falle !

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

Zusammenhang der Anweisungen S, T und E:

$$\{\varphi\}\ S\ \{\psi\} \Leftrightarrow (\{\varphi \wedge \beta\}\ T\ \{\psi\}) \wedge (\{\varphi \wedge \neg\beta\}\ E\ \{\psi\})$$

Es wird hier keine Aussage über den Zusammenhang von φ_1, φ_2, φ, ψ_1, ψ_2, ψ gemacht !

Zusammenhang der Bedingungen φ_1, φ_2, φ, ψ_1, ψ_2, ψ:

1) $\varphi \Leftrightarrow (\varphi_1 \wedge \beta) \vee (\varphi_2 \wedge \neg\beta)$

2) $\psi \Leftrightarrow (\psi_1 \wedge \beta) \vee (\psi_2 \wedge \neg\beta)$

Verifikation von Verzweigungen

Mehrfachverzweigungen: *Nur andere Schreibweise!*

```
case Ausdruck of
        Wert₁: Anweisung₁
        Wert₂: Anweisung₂
        ...
        Wertₙ: Anweisungₙ
        else else-Anweisung
end
```

Ausdruck muss eine Funktion mit diskretem Wertebereich sein, die nur von Variablen abhängen darf, die mit Werten belegt sind.
$\mathbf{Wert}_1, \ldots, \mathbf{Wert}_n$ müssen zulässige Wertebereiche von **Ausdruck** sein.

Funktionsweise: *Daher Verifikation analog*

Zunächst wird **Ausdruck** ausgewertet. Das Ergebnis sei **w**.
Wenn **w** in $\mathbf{Wert}_1$ liegt, wird nur $\mathbf{Anweisung}_1$ ausgeführt.
Wenn **w** in $\mathbf{Wert}_2 \setminus \mathbf{Wert}_1$ liegt, wird nur $\mathbf{Anweisung}_2$ ausgeführt.
...
Wenn **w** in $\mathbf{Wert}_n \setminus \{\mathbf{Wert}_1 \wedge \ldots \wedge \mathbf{Wert}_{n-1}\}$ liegt, wird nur $\mathbf{Anweisung}_n$ ausgeführt.
Wenn **w** nicht in $\{\mathbf{Wert}_1 \wedge \ldots \wedge \mathbf{Wert}_n\}$ liegt, wird nur die **else-Anweisung** ausgeführt.

Beim nächsten Mal:

Programmentwicklung und –verifikation: Schleifen