Grundlagen der Theoretischen Informatik

Sebastian Iwanowski
FH Wedel

Kap. 4: Verifikationstechniken
Teil 3: Verifikation von Schleifen

Verifikation und Konstruktion von Schleifen

Definition einer Schleife:

```
while Schleifenbedingung do
Rumpfanweisung
```

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

Funktionsweise:

1) Zunächst wird **`Schleifenbedingung`** ausgewertet.

2) Wenn **`Schleifenbedingung`** falsch ist, wird die Schleife sofort beendet.
   Wenn **`Schleifenbedingung`** wahr ist, wird **`Rumpfanweisung`** ausgeführt.
   Dann wird bei Schritt 1) fortgefahren.

Verifikation von Schleifen

Verifikationstechnik:

W

```
{Vorbedingung}                    φ
while Schleifenbedingung do       β
      {Eintrittsbedingung}        φ_i
      Rumpfanweisung              S
      {Austrittsbedingung}        ψ_i
{Nachbedingung}                   ψ
```

Definition:

Es sei φ_i die Eintrittsbedingung vor der i-ten Ausführung der Rumpfanweisung und ψ_i die Austrittsbedingung nach der i-ten Ausführung der Rumpfanweisung
Die Schleife werde nach k Ausführungen beendet.

Dann gilt:

1) $\varphi_1 \Leftrightarrow \varphi \wedge \beta$ $\quad \{\varphi_1\}$ **Rumpfanweisung** $\{\psi_1\}$

2) $\varphi_2 \Leftrightarrow \psi_1 \wedge \beta$ $\quad \{\varphi_2\}$ **Rumpfanweisung** $\{\psi_2\}$

.

i) $\varphi_i \Leftrightarrow \psi_{i-1} \wedge \beta$ $\quad \{\varphi_i\}$ **Rumpfanweisung** $\{\psi_i\}$

.

k) $\varphi_k \Leftrightarrow \psi_{k-1} \wedge \beta$ $\quad \{\varphi_k\}$ **Rumpfanweisung** $\{\psi_k\}$

k+1) $\psi \Leftrightarrow \psi_k \wedge \neg\beta$ $\quad \{\varphi_k\}$ **Rumpfanweisung** $\{\psi_k\}$

Verifikation von Schleifen

Verifikationstechnik:

W

```
{Vorbedingung}                    φ
while Schleifenbedingung do       β
      {Eintrittsbedingung}        φ_i
      Rumpfanweisung              S
      {Austrittsbedingung}        ψ_i
{Nachbedingung}                   ψ
```

Gegeben ψ, berechne φ: Wie findet man die schwächste Vorbedingung P für φ ?

Beobachtung:

0) Sei P_0 die schwächste Vorbedingung, falls die Schleife gar nicht durchlaufen wird.

1) Sei P_1 die schwächste Vorbedingung, falls die Schleife genau einmal durchlaufen wird.

i) Sei P_i die schwächste Vorbedingung, falls die Schleife genau i-mal durchlaufen wird.

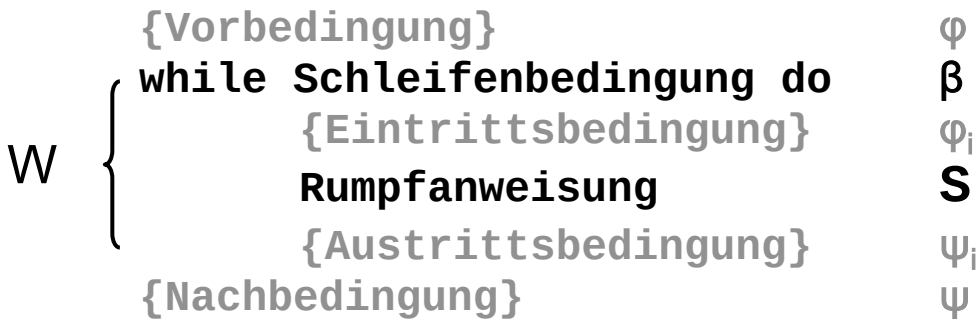
Lösung: Dann gilt: $P = P_0 \vee P_1 \vee \ldots \vee P_i \vee \ldots$

Problem: Im allgemeinen Fall könnten sich die P_i's alle unterscheiden !

Damit kann keine allgemeine Lösungstechnik angegeben werden !

Verifikation von Schleifen

Verifikationstechnik:



Einfachere Aufgabe:

Gegeben φ und ψ:

Beweise, dass gilt: **{φ} W {ψ} !**

Verifikation von Schleifen

Wesentliche Schritte bei der Verifikation von Schleifen:

1) **Beweise, dass die Berechnungen der Schleife kontinuierlich auf dem richtigen Weg sind,**
sodass nach der Schleife das Richtige berechnet ist.

(falls das Programm dort jemals ankommt)

Irgendetwas muss unverändert richtig sein → Invariante (Bedingung)

2) **Beweise, dass die Schleife zum Ende kommt.**

Irgendetwas muss sich im Laufe der Schleife ändern → Variante (Zahl)

Verifikation von Schleifen

Zur Erinnerung:

$\varphi_i \Leftrightarrow \psi_{i-1} \wedge \beta$ **d.h.:** $\{\psi_{i-1} \wedge \beta\}$ **Rumpfanweisung** $\{\psi_i\}$

Verwendung einer Bedingung I (Invariante) zum Beweis der Gültigkeit des Zusammenhangs zwischen Vor- und Nachbedingung:

Die Invariante I soll Folgendes erfüllen:

1) $\varphi \wedge \beta \Rightarrow I$ *Aus Vorbedingung und Schleifenbedingung folgt die Invariante.*

2) $\{I \wedge \beta\}\ S\ \{I\}$ *Innerhalb der Schleife gilt die Invariante **vor und nach** der Rumpfanweisung. Die Schleifenbedingung gilt zumindest **vor** der Rumpfanweisung.*

3) $I \wedge \neg\beta \Rightarrow \psi$ *Aus der Invariante und der Nichterfüllung der Schleifenbedingung muss die geforderte Nachbedingung folgen.*

Problem: *Damit ist **noch nicht** gewährleistet, dass die Schleife terminiert.*

Verifikation von Schleifen

Zur Erinnerung:

$\varphi_i \Leftrightarrow \psi_{i-1} \wedge \beta$ **d.h.:** $\{\psi_{i-1} \wedge \beta\}$ **Rumpfanweisung** $\{\psi_i\}$

Verwendung einer Variante $z \in \mathbb{Z}$ zur Gewährleistung der Terminierung der Schleife:

Die Variante z soll Folgendes erfüllen:

1) $\varphi \wedge \beta \Rightarrow \exists z_0 \in \mathbb{Z} : z = z_0$ *z hat bei Schleifeneintritt einen definierten Wert.*

2) $\{I \wedge \beta \wedge (z = z_0)\}$ S $\{z < z_0\}$ *z hat nach Ausführung der Rumpfanweisung einen kleineren Wert als vorher.*

3) $I \wedge (z \leq 0) \Rightarrow \neg\beta$ *Der Wert von z sorgt dafür, dass irgendwann einmal die Schleifenbedingung verletzt wird.*

Lösung: ***Damit ist gewährleistet, dass die Schleife terminiert.***

Verifikation von Schleifen

Verifikation von Schleifen mit vollständiger Induktion:

Es muss in Induktionsverankerung und im Induktionsschluss von i auf i+1 bewiesen werden, dass die Schleife immer terminiert und das Richtige berechnet.

Die Induktionsbehauptung muss also sowohl eine Invariante als auch eine Variante im Sinne der vorigen Definition enthalten.

Typisches Beispiel für die Wahl der Induktionsvariablen i:

i = Anzahl der bisher durchgeführten Schleifendurchläufe

Typische Beweistechnik im Induktionsschluss:

1) Unterscheide die Belegung der Variablen nach i bzw. i+1 Schleifendurchläufen

2) Setze die verschiedenen Belegungszustände in Beziehung zueinander und verwende die Induktionsannahme für i.

Verifikation von Schleifen

Beispiel 1a:

```
{ Vorbedingung }    φ

f := 1;
k := 0;
while (k < n) do
  begin
     k := k + 1;
     f := k • f
  end

{ Nachbedingung }   ψ
```

Was berechnet dieses Programmstück ?

Gibt es Einschränkungen für die Bedingungen φ oder ψ ?

Verifikation von Schleifen

Beispiel 1b:

```
{ Vorbedingung }    φ

f := 1;
k := n;
while (k > 0) do
  begin
     f := k • f;
     k := k - 1;
  end

{ Nachbedingung }   ψ
```

Was berechnet dieses Programmstück ?

Gibt es Einschränkungen für die Bedingungen φ oder ψ ?

Verifikation von Schleifen

Beispiel 2:

```
Gegeben seien n Zahlen a[1] .. a[n];

{ Vorbedingung }    φ

k := 0;
while (k < n) do
  begin
     k := k + 1;
     if m < a[k]
       then
          m := a[k]
  end

{ Nachbedingung }   ψ
```

Was berechnet dieses Programm ?

Gibt es Einschränkungen für die Bedingungen φ oder ψ ?