Übung zu
Algorithmen und Datenstrukturen (für ET/IT)

Wintersemester 2012/13

Jakob Vogel

Computer-Aided Medical Procedures
Technische Universität München

Administratives

Sprechpunkt wird ab sofort zum *Bedarfs-Sprechpunkt*

- Termin weiterhin montags, 13:30 Uhr
- Durchführung nur noch nach Voranmeldung
- Anmeldung per e-Mail oder in der Zentralübung

- Bisher praktisch keine Nachfrage!

Verifikation

- „Verfolgung“ logischer Bedingungen durch Code
  - Vorbedingung $\{VOR\}$
  - Nachbedingung $\{NACH\}$
- Kette von Anweisungen $\{A\}\ \alpha;\beta\ \{B\}$
  - $\{A\}\ \alpha\ \{C\}\ \beta\ \{B\}$
- Bedingte Verarbeitung $\{A\}$ **if** (C) $\{\alpha\}$ **else** $\{\beta\}$ $\{B\}$
  - $\{A \wedge C\}\ \alpha\ \{B\}$
  - $\{A \wedge \neg C\}\ \beta\ \{B\}$
- Schleifen $\{A\}$ **while** (C) $\{\beta\}$ $\{B\}$ mit Invariante P
  - P gilt am Anfang: $A \Rightarrow P$
  - P bleibt gültig: $\{P \wedge C\}\ \beta\ \{P\}$
  - $P \wedge \neg C \Rightarrow B$
- Weiteres Beispiel heute...

Prüfung mittels `assert`-Statement

- C/C++ stellt Statement zur Verfügung:

  `assert(`⟨Bedingung⟩`);`

- Bedingung geprüft *zur Laufzeit* des Programms
  - Keine Aktion falls die Bedingung *wahr* ist.
  - Programmende mit Fehler falls die Bedingung *falsch* ist.
- Etwa für Invarianten oder andere Zwischenbedingungen...
- Import via `#include <assert.h>`
- Sinnvoll eigentlich nur im *Debug*-Modus


Code

Validierung

- Verifikation oft nicht praktikabel oder fehlerhaft
- Bau eines *separaten* Testprogramms mit systematischen Tests
  - Bekannte Eingabewerte, erwartete Ausgabe?
  - Zufällige Eingabewerte, erfüllt Ausgabe Nachbedingung?
- Design der Testfälle ist schwierig, keine (Sonder-)Fälle vergessen!

- Wenn die Vorbedingungen nicht erfüllt sind, ist das Verhalten eigentlich undefiniert. Trotzdem die Eingabewerte prüfen!
  - Auch zur Laufzeit!
  - Sicherheit!
  - Aussagekräftige Fehlermeldungen!

Validierung von Selection Sort

- Sortierung eines Arrays mit bekannter Belegung
  - Erwartetes Ergebnis?
- Sortierung eines Arrays mit zufälliger Belegung
  - Gilt Nachbedingung $A[0] \leq A[1] \leq ...$?
  - Permutation? Aufwändig!
- Sortierung des leeren Arrays?

- Test, ob wirklich mit Selection Sort gearbeitet wird, ist **nicht** möglich! (Auch nicht White-Box-Test!)

Validierung der Multiplikation durch Addition

Multiply(a, b)**:**

$c = 0;$
for $i = 1$ **to** a $\{$
    $c = c + b;$
$\}$
return $c;$

- White-Box-Test zeigt, dass nur Test von a notwendig ist.
- Sinnvolle Testfälle: $a > 0$, $a < 0$ und $a = 0$
- Vergleich von `multiply(a, b)` mit `a * b`...
- Negative Zahlen werden falsch behandelt!


Code

Verifikation der korrigierten Version

Multiply(a, b)**:**

```
if (a = 0 ∨ b = 0) {
        c = 0;
}
else if (a < 0) {
        c = −Multiply(−a, b);
}
else {
        c = 0;
        i = 1;
        while (i ≤ a) {
                c = c + b;
                i = i + 1;
        }
}
return c;
```

Vor- und Nachbedingung

- Die Parameter sind zwei beliebige ganzzahlige Werte

$$VOR \quad = \quad a \in \mathbb{Z} \quad \wedge \quad b \in \mathbb{Z}$$

- Das Ergebnis c ist das Produkt dieser Werte:

$$NACH \quad = \quad (c = a \cdot b)$$

Abkürzungen

Für mehr Übersichtlichkeit ersetzen wir die Anweisungsblöcke zur Berechnung von c durch Symbole α, β und γ

Multiply(a, b)**:**

if $(a = 0 \ \vee \ b = 0)$ {
    $c = 0;$
}
else if $(a < 0)$ {
    $c = -\text{Multiply}(-a, b);$
}
else {
    $c = 0;$
    $i = 1;$
    while $(i \leq a)$ {
        $c = c + b;$
        $i = i + 1;$
    }
}
return $c;$

Abkürzungen

Für mehr Übersichtlichkeit ersetzen wir die Anweisungsblöcke zur Berechnung von c durch Symbole α, β und γ

Multiply(a, b)**:**

if $(a = 0 \ \vee \ b = 0)$ {

α

}

else if $(a < 0)$ {

β

}

else {

γ

}

Korrektheit der **Multiply**: Fallunterscheidung

Zunächst verifizieren wir die Fallunterscheidung:

$\{VOR\}$
if $(a = 0 \ \vee \ b = 0)$ {
α
}
else if $(a < 0)$ {
β
}
else {
γ
}
$\{NACH\}$

Korrektheit der **Multiply**: Fallunterscheidung

Dazu betrachten wir die drei Fälle separat:

- $\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$
- $\{VOR \ \wedge \ \neg(a = 0 \ \vee \ b = 0) \ \wedge \ a < 0\} \ \beta \ \{NACH\}$
- $\{VOR \ \wedge \ \neg(a = 0 \ \vee \ b = 0) \ \wedge \ \neg(a < 0)\} \ \gamma \ \{NACH\}$

Korrektheit der **Multiply**: Fallunterscheidung

Dazu betrachten wir die drei Fälle separat:

- $\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$
- $\{VOR \ \wedge \ a < 0 \ \wedge \ b \neq 0\} \ \beta \ \{NACH\}$
- $\{VOR \ \wedge \ \neg(a = 0 \ \vee \ b = 0) \ \wedge \ \neg(a < 0)\} \ \gamma \ \{NACH\}$

Korrektheit der **Multiply**: Fallunterscheidung

Dazu betrachten wir die drei Fälle separat:

- $\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$
- $\{VOR \ \wedge \ a < 0 \ \wedge \ b \neq 0\} \ \beta \ \{NACH\}$
- $\{VOR \ \wedge \ a > 0 \ \wedge \ b \neq 0\} \ \gamma \ \{NACH\}$

Korrektheit der **Multiply**: Fall 1

$$\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$$

Es gilt ja:

$$a = 0 \ \vee \ b = 0 \quad \Rightarrow \quad a \cdot b = 0$$

α setzt in diesem Fall $c = 0$, also gilt $c = a \cdot b$, und die Nachbedingung ist erfüllt.

- Dieser Fall ist **korrekt**!

Korrektheit der **Multiply**: Fall 2

$$\{VOR \ \wedge \ a < 0 \ \wedge \ b \neq 0\} \ \beta \ \{NACH\}$$

Es gilt ja:

$$a < 0 \quad \Rightarrow \quad a = -|a| = -1 \cdot (-a)$$

Bei $a < 0$ und beliebigem b gilt weiter wegen Assoziativgesetzes:

$$a \cdot b = (-1 \cdot (-a)) \cdot b = -1 \cdot ((-a) \cdot b)$$

β setzt in diesem Fall $c = -\text{Multiply}(-a, b)$, also gilt $c = a \cdot b$, und die Nachbedingung ist erfüllt, falls die Methode im dritten Fall korrekt ist.

- Dieser Fall ist (wahrscheinlich) **korrekt**!

Korrektheit der **Multiply**: Fall 3

$$\{VOR \ \wedge \ a > 0 \ \wedge \ b \neq 0\} \ \gamma \ \{NACH\}$$

Für die Korrektheit der Schleife in γ benötigt man die *Invariante*:

$c = 0;$
$i = 1;$
while $(i \leq a)$ {
    $c = c + b;$
    $i = i + 1;$
}

Der bereits in c aufaddierte Teil und der noch aufzuaddierende Teil müssen zusammen das korrekte Ergebnis $a \cdot b$ ergeben:

$$P \quad = \quad (a \cdot b = c + (a - i + 1) \cdot b)$$

Bei Zählung ab 0 wäre das schöner!

Abkürzungen

Wieder ersetzen wir Anweisungsblöcke:

$\{C_0\}$

$c = 0;$
$i = 1;$

$\{C_1\}$

while $(i \leq a)$ {

$\{C_2\}$

$c = c + b;$
$i = i + 1;$

$\{C_3\}$

}

$\{NACH\}$

Abkürzungen

Wieder ersetzen wir Anweisungsblöcke:

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ $\{$

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

$\}$

$\{NACH\}$

Korrektheit der **Multiply**: Schleife

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ {

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

}

$\{NACH\}$

Die Korrektheit der Schleife wird in drei Schritten gezeigt:

- Schleifeneintritt: $C_1 \Rightarrow P$
- Schleifenkörper: $\{P \ \wedge \ (i \leq a)\} \ \gamma_{\text{körper}} \ \{P\}$
- Schleifenende: $P \ \wedge \ \neg(i \leq a) \Rightarrow NACH$

Korrektheit der **Multiply**: Schleifeneintritt

$$C_1 \Rightarrow P$$

- C_0 ergibt sich aus der Vorbedingung des dritten Falles

$$C_0 \quad := \quad (VOR \ \wedge \ a > 0 \ \wedge \ b \neq 0)$$

- Prüfe $\{C_0\}\ \gamma_{\text{init}}\ \{C_1\}$

$$C_1 \quad := \quad (C_0 \ \wedge \ c = 0 \ \wedge \ i = 1)$$

- Einfügung in die Formel aus P ergibt *wahre* Aussage

$$a \cdot b = c + (a - i + 1) \cdot b = 0 + (a - 1 + 1) \cdot b = a \cdot b$$

- Schleifeneintritt ist **korrekt**!

Korrektheit der **Multiply**: Schleife

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ {

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

}

$\{NACH\}$

Die Korrektheit der Schleife wird in drei Schritten gezeigt:

- Schleifeneintritt: $C_1 \Rightarrow P$
- Schleifenkörper: $\{P \ \wedge \ (i \leq a)\} \ \gamma_{\text{körper}} \ \{P\}$
- Schleifenende: $P \ \wedge \ \neg(i \leq a) \Rightarrow NACH$

Korrektheit der **Multiply**: Schleifenkörper

$$\{C_2 := (P \ \wedge \ (i \leq a))\} \ \gamma_{\text{körper}} \ \{C_3 := P\}$$

- In $\gamma_{\text{körper}}$ werden *neue* Werte gesetzt:

$$c' = c + b \qquad i' = i + 1$$

- Invariante galt *vorher*:

$$a \cdot b = c + (a - i + 1) \cdot b$$

- Prüfung der Invariante *nachher*:

$$\begin{aligned} c' + (a - i' + 1) \cdot b &= (c + b) + (a - (i + 1) + 1) \cdot b \\ &= c + b + (a - i) \cdot b \\ &= c + (a - i + 1) \cdot b \\ &= a \cdot b \end{aligned}$$

- Schleifenkörper ist **korrekt**!

Korrektheit der **Multiply**: Schleife

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ {

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

}

$\{NACH\}$

Die Korrektheit der Schleife wird in drei Schritten gezeigt:

- Schleifeneintritt: $C_1 \Rightarrow P$
- Schleifenkörper: $\{P \ \wedge \ (i \leq a)\} \ \gamma_{\text{körper}} \ \{P\}$
- Schleifenende: $P \ \wedge \ \neg(i \leq a) \Rightarrow NACH$

Korrektheit der **Multiply**: Schleifenende

$$P \ \wedge \ \neg(i \leq a) \Rightarrow \mathit{NACH}$$

- Wegen schrittweiser Erhöhung gilt sogar

$$i = a + 1$$

- Einfügen in Invariante ergibt *NACH*:

$$\begin{aligned} a \cdot b &= c + (a - i + 1) \cdot b \\ &= c + (a - (a + 1) + 1) \cdot b \\ &= c + (a - a - 1 + 1) \cdot b \\ &= c + 0 \cdot b \\ &= c \end{aligned}$$

- Schleifenende (und damit der ganze dritte Fall) **korrekt**!

Korrektheit der **Multiply**: Zusammenfassung

- Untersuchung der drei Fälle der Fallunterscheidung
- Letzter Fall erfordert Untersuchung der Schleife
  - Schleifeneintritt
  - Schleifenkörper
  - Schleifenende

Landau-Symbole

Abschätzung des Wachstumsverhaltens einer (komplexeren) Funktion durch eine andere (intuitivere) Funktion.

$$\begin{aligned}\Theta(g) \quad := \quad \{f : \mathbb{R} \to \mathbb{R} \ : \ &\text{es existieren } c_1, c_2 > 0 \text{ und } n_0 \in \mathbb{N}, \\ &\text{so dass für alle } n \geq n_0 \text{ gilt:} \\ &0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}\end{aligned}$$

$$\begin{aligned}O(g) \quad := \quad \{f : \mathbb{R} \to \mathbb{R} \ : \ &\text{es existieren } c > 0 \text{ und } n_0 \in \mathbb{N}, \\ &\text{so dass für alle } n \geq n_0 \text{ gilt:} \\ &0 \leq f(n) \leq c \cdot g(n)\}\end{aligned}$$

$\Theta(g)$ ist die *asymptotisch scharfe* und $O(g)$ die *asymptotisch obere Schranke*.

$7n^4 = O(n^5)$

Betrachte Ungleichung aus der Definition von $O(n)$:

$$0 \leq 7n^4 \leq c \cdot n^5$$

Dividiere für $n \geq 1$ durch n^4 ohne Umkehr der Ungleichung:

$$0 \leq 7 \leq c \cdot n$$

Für $c = 7$ und $n_0 = 1$, so gilt die Ungleichung für alle $n \geq n_0$. □

$n^2/2 - 2n = \Theta(n^2)$

Betrachte beide Schranken unabhängig voneinander:

$$c_1 \cdot n^2 \leq \frac{n^2}{2} - 2n \qquad \frac{n^2}{2} - 2n \leq c_2 \cdot n^2$$

Dividiere für $n \geq 1$ durch n^2 ohne Umkehr der Ungleichung:

$$c_1 \leq \frac{1}{2} - \frac{2}{n} \qquad \frac{1}{2} - \frac{2}{n} \leq c_2$$

Für $c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$ und $n_0 = 8$ gilt die komplette Ungleichung für alle $n \geq n_0$. □

$2^{n+1} = O(2^n)$

Betrachte wieder die Ungleichung aus der Definition von $O(2^n)$:

$$0 \leq 2^{n+1} \leq c \cdot 2^n$$

Zerlegung ergibt:

$$2^{n+1} = 2 \cdot 2^n$$

Für $c = 2$ und $n_0 = 1$ gilt also die Ungleichung für alle $n \geq n_0$. $\square$

$2^{2n} \neq O(2^n)$

Annahme, dass die Behauptung **nicht** zutrifft. Dann existieren Konstanten c und $n_0 > 0$, so dass für alle $n \geq n_0$ gilt:

$$0 \leq 2^{2n} \leq c \cdot 2^n$$

Zerlegung ergibt:

$$2^{2n} = 2^n \cdot 2^n \leq c \cdot 2^n \Leftrightarrow 2^n \leq c \ .$$

Die Folge 2^n ist aber **unbeschränkt**, und es existiert keine Konstante c, die für beliebige n die Forderung $c \geq 2^n$ erfüllt.

Widerspruch zur Annahme! □