

Grundlagen der Programmierung (Vorlesung 12)

Ralf Möller, FH-Wedel

■ Vorige Vorlesung

- Korrektheit von Anweisungen
- Vorbedingungen und Nachbedingungen
- Einfach- und Mehrfachzuweisung

■ Inhalt dieser Vorlesung

- Wiederholung und Vertiefung des Stoffes aus VL 11
- Fallunterscheidungen

■ Lernziele

- Grundlagen der systematischen Programmentwicklung

4.2.2 Anweisungsfolgen

zusammengesetzte

Bausteine der Programmiersprache

Anweisungen können aufgebaut werden aus

↪ Zuweisungen

↪ Anweisungsfolgen

↪ bedingte Anweisungen

↪ Schleifen-Anweisungen

Anweisungsfolge Zwei (oder mehrere) Anweisungen können zu eine Anweisungsfolge zusammengesetzt werden

Syntax $Anweisung_1 ; Anweisung_2$

$x := 3; y := 5$

Semantik anschaulich:
Nacheinanderausführen der einzelnen Anweisungen in der angegebenen Reihenfolge

Beweisregel

für Anweisungsfolgen

falls

$$(1) \quad \{V\} \ S_1 \ \{P_1\}$$

$$(2) \quad \{P_1\} \ S_2 \ \{P\}$$

dann gilt

$$\{V\} \ S_1; S_2 \ \{P\}$$

Sequentielle Anweisung, Vorbedingung finden

$$\blacksquare \{ V \} x := x + 1; y := y - 1 \{ x + y = s \}$$

$$\blacksquare \{ V \} x := x + 1 \{ P_1 \}; \{ P_1 \} y := y - 1 \{ x + y = s \}$$

$$\blacksquare P_1 : (x + y = s)[y \leftarrow y - 1]$$

$$\equiv x + y - 1 = s$$

$$\blacksquare V : P_1[x \leftarrow x + 1]$$

$$\equiv x + 1 + y - 1 = s$$

$$\equiv x + y = s$$

Unterschied: Parallele und. seq. Zuweisung

- Werte vertauschen
 - $\{ y=w1 \wedge x=w2 \} \ x, y := y, x \ \{ x=w1 \wedge y=w2 \}$
- Geht das sequentiell? Wenn ja, unter welchen Bed.?
 - $\{ V \} \ x := y; y := x \ \{ x=w1 \wedge y=w2 \}$
 - $((x=w1 \wedge y=w2)[y \leftarrow x])[x \leftarrow y]$
 - $(x=w1 \wedge x=w2)[x \leftarrow y]$
 - $y=w1 \wedge y=w2$
- Also nur, wenn $w1 = w2$
(dann natürlich nicht interessant)

Sequentialisierung

- $i, S := i+1, S+i$
- $S := S + i ; i := i + 1$
- Dependenzanalyse notwendig
- Erst die Variablen setzen, die von anderen abhängen (geht nicht immer)
- Ggf. Zwischenvariablen einführen

4.2.3 bedingte Anweisungen

if –Anweisung eine Bedingung (Prädikat, Ausdruck, der zu true oder false ausgewertet wird) und zwei Anweisungen können zu einer bedingten Anweisung zusammengesetzt werden

Syntax if *Bedingung*
 then *Anweisung*₁
 else *Anweisung*₂
 end if

Semantik informell

(1) die *Bedingung* wird ausgewertet

(2a) ist das Resultat true,
wird die *Anweisung*₁ ausgeführt

(2b) ist das Resultat false,
wird die *Anweisung*₂ ausgeführt anschließend
wird

Variante eine *Bedingung* und eine *Anweisung* können zu einer bedingten Anweisung zusammengesetzt werden

Syntax if *Bedingung*
 then *Anweisung*
 end if

Beweisregel

für bedingte Anweisungen

falls

$$(1) \quad \{V_1\} \ S_1 \ \{P\}$$

$$(2) \quad \{V_2\} \ S_2 \ \{P\}$$

dann gilt

$$\{(V_1 \wedge B) \vee (V_2 \wedge \neg B)\}$$

if B then S_1 else S_2 end if $\{P\}$

Beweisregel

für bedingte Anweisungen nur mit then Zweig

falls

$$(1) \quad \{V\} \ S \ \{P\}$$

dann gilt

$$\{(V \wedge B) \vee (P \wedge \neg B)\}$$

if B then S end if $\{P\}$

Fallunterscheidung: Vorbedingung finden

- $\{ V \}$ if $x \geq y$
 then $\{ V_1 \} r := x$
 else $\{ V_2 \} r := y$
 endif $\{ r \geq x \wedge r \geq y \wedge (r = x \vee r = y) \}$
- $V_1: (r \geq x \wedge r \geq y \wedge (r = x \vee r = y))[r \leftarrow x]$
 $\equiv x \geq x \wedge x \geq y \wedge (x = x \vee x = y)$
 $\equiv x \geq y$
- $V_2: y \geq x$ (ähnl. Beweis)
- $V: (B \wedge V_1) \vee (\neg B \wedge V_2) \equiv \text{true}$

Satz

- $(p \wedge q) \vee (\neg p \wedge r) \equiv (p \rightarrow q) \wedge (\neg p \rightarrow r)$
- Erste Anwendung des Satzes:
- Mit diesem Satz kann die schwächste Vorbedingung für die Fallunterscheidung formuliert werden als:
- $\{ (B \rightarrow V_1) \wedge (\neg B \rightarrow V_2) \}$

Zweite Anwendung: Programmkonstruktion

■ Aufgabe:

- Für eine Nachbedingung $P = (B \rightarrow V_1) \wedge (\neg B \rightarrow V_2)$ soll ein Programm gefunden werden, das die Nachbedingung herstellt
- In dieser Form liegen Spezifikationen häufig vor

■ Behauptung: Die Lösung sieht so aus:

- $\{ \text{true} \} \text{ if } B \text{ then } S_1 \text{ else } S_2 \text{ endif } \{ P \}$
- mit $\{ B \} S_1 \{ V_1 \wedge B \}$
- und $\{ \neg B \} S_2 \{ V_2 \wedge \neg B \}$

Begründung

- Wenn $\{ B \} S_1 \{ V_1 \wedge B \}$ korrekt, dann auch
 $\{ B \} S_1 \{ (V_1 \wedge B) \vee (V_2 \wedge \neg B) \}$ (Abschwächung)
- Wenn $\{ \neg B \} S_2 \{ V_2 \wedge \neg B \}$ korrekt, dann auch
 $\{ \neg B \} S_2 \{ (V_1 \wedge B) \vee (V_2 \wedge \neg B) \}$ (Abschwächung)
- Also ist auch die folgende Fallunterscheidung korrekt
 $\{ (B \wedge B) \vee (\neg B \wedge \neg B) \}$ if B then S_1 else S_2
 $\{ (V_1 \wedge B) \vee (V_2 \wedge \neg B) \}$
- Die Vorbedingung ist äquivalent zu true:
 $\{ \text{true} \}$ if B then S_1 else S_2 $\{ (V_1 \wedge B) \vee (V_2 \wedge \neg B) \}$
- Laut obigem Satz ergibt sich die Behauptung durch Transformation der Nachbedingung

Beispiel: Spezifikation für Max

- $\{ (x \geq y \rightarrow r = x) \wedge (x \leq y \rightarrow r = y) \}$
- Idee: In Verzweigung umsetzen!
- Aber B und $\neg B$ notwendig
- Idee: Nachbedingung verstärken!
- $\{ (x \geq y \rightarrow r = x) \wedge (x < y \rightarrow r = y) \}$

Mehrfach –Auswahl

Mehrfach–Verzweigung

↪

in Programmiersprachen gibt es häufig noch eine Mehrwegverzweigung (case -Anweisung).

Syntax

```
case Ausdruck of  
    Wert1 : Anweisung1  
    ⋮  
    Wertn : Anweisungn  
    else : Anweisung0  
end case
```

Bedeutung

erklärt mit bedingter Anweisung

```
Wert := Ausdruck  
if Wert = Wert1  
    then Anweisung1  
else  
    ⋮  
if Wert = Wertn  
    then Anweisungn  
else  
    Anweisung0  
end if ... end if
```

Beispiel: Multiplikation durch Addition

- Ziel: Programmstück konstruieren
- $\text{var } x, y : \mathbb{N}_0, i, j, p : \mathbb{N}_0$
- $\{ \text{true} \}$ "multiplizieren" $\{ x * y = p \}$
- Idee: i, j mit x und y initialisieren (und p mit 0) und dann schrittweise j verkleinern
- $i, j, p := x, y, 0 \{ x * y = p + i * j \}$
- $\{ x * y = p + i * j \} j, p := j-1, E \{ x * y = p + i * j \}$
- Idee: Die Anweisung j mal wiederholen bis $j = 0$
 - $x * y = p + (i * 0) \quad x * y = p$

Zusammenfassung, Kernpunkte



- Logik und die systematische Entwicklung von Programmen
- Programmsequenz
- Fallunterscheidung

Was kommt beim nächsten Mal?



- Korrektheit von Schleifen