Grundlagen der Informatik

1. Semester, 1998

Kapitel 2: Algorithmen

2.1. Eigenschaften von Algorithmen
- 2.1.1. Forderungen an Algorithmen
- 2.1.2. Spezifikation von Algorithmen
- 2.1.3. Zusammensetzung von Algorithmen
- 2.1.4. Analyse von Algorithmen

2.2. Korrektheit von Programmen
- 2.2.1. Verifikationsregeln
- 2.2.2. Verzweigungen
- 2.2.3. Iterationen
- 2.2.4. Zuweisungen

Verifikation von Programmen

- Programm Verifikation dient dazu, die Fehlerfreiheit von Programmen zu beweisen, d.h. zu zeigen, daß ein Programm seine Spezifikation erfüllt.

{ ... **Vorbedingung ...** }

Programm

{ ... **Nachbedingung ...** }

- Vorbedingung und Nachbedingung sind logische Formeln, die den Zustand einer Programmausführung beschreiben.

- Verifikation versucht aus der Richtigkeit der Vorbedingung die Richtigkeit der Nachbedingung herzuleiten.

Verifikation von Programmen

- Jeder Programmrumpf hat die Form

  begin

  Anweisung1;

  Anweisung2;

  ....

  AnweisungN

  end

- Die Verifikation von { Vorbedingung } *begin .... end* { Nachbedingung }
  wird in Einzelschritte zerlegt:

1. { Vorbedingung } Anweisung1 { Bedingung1 }
2. { Bedingung1 } Anweisung2 { Bedingung2 }

....

Semantik von Anweisungen

- Ein Korrektheitsbeweis für { Vorbedingung } *Anweisung* { Nachbedingung }
ist nur möglich, falls die "Wirkung" (Semantik) der Anweisung formal erfaßt wird.

- Die Semantik einer Anweisung kann durch Beweisregeln der Form

$$\frac{A1, A2, \ldots, An}{B}$$

erfaßt werden,

wobei A1, A2, ... , An die Vorbedingungen für die Anwendbarkeit der Beweis-regel sind

B die Folgerung ist, welche nach Anwendung der Beweis-regel zutrifft.

Typischerweise haben Voraussetzungen und Folgerungen die Form von Spezifikationen

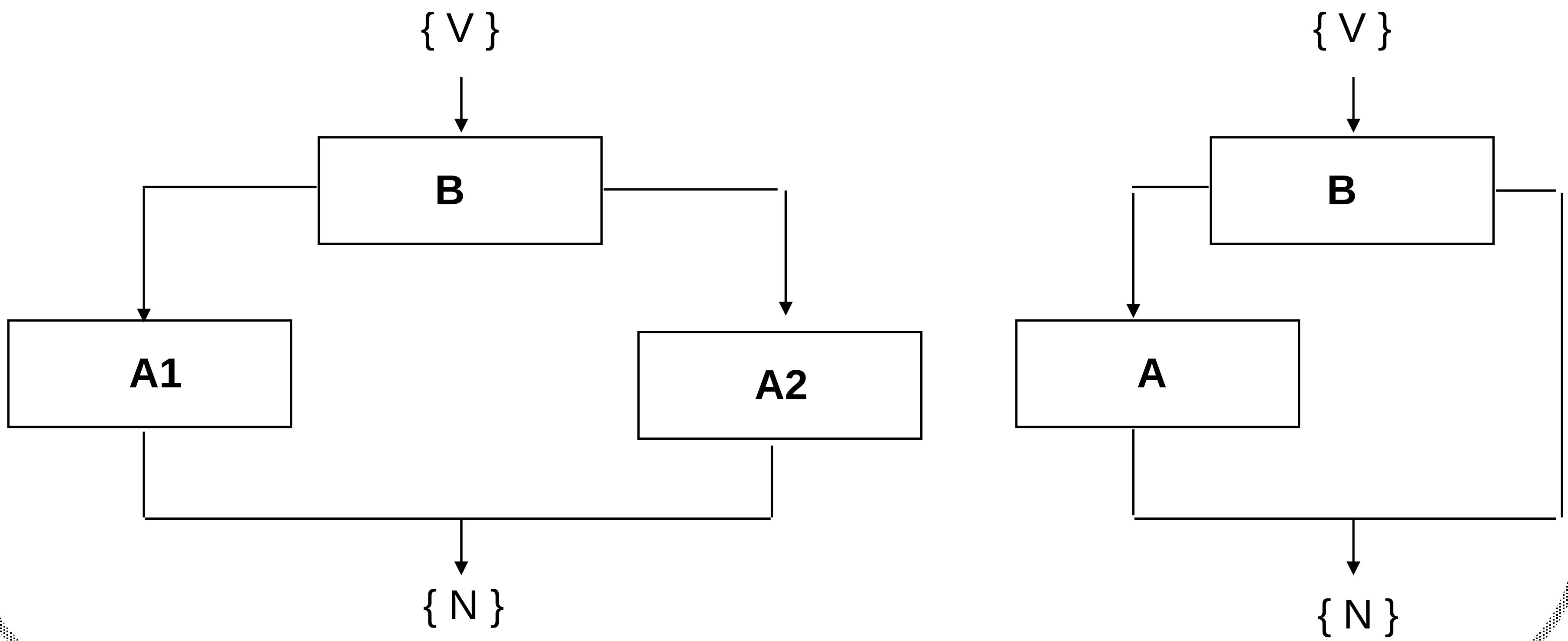
2.2.2 Beweisregeln für Verzweigungen

- Der Kontrollfluß für Verzweigungen

  if B *then* A1 *else* A2

  if B *then* A

  läßt sich beschreiben durch:



Beweisregeln für Verzweigungen

- Für Verzweigungen ergeben sich daraus die Beweisregeln

$$\frac{\{\ V \text{ and } B\ \}\ \ A1\ \{\ N\ \},\ \ \{\ V \text{ and not } B\ \}\ \ A2\ \{\ N\ \}}{\{\ V\ \}\ \ \text{if } B \text{ then } A1 \text{ else } A2\ \{\ N\ \}}$$

und

$$\frac{\{\ V \text{ and } B\ \}\ \ A\ \{\ N\ \},\ \ \{\ V \text{ and not } B\ \}\ \ \Rightarrow\ \{\ N\ \}}{\{\ V\ \}\ \ \text{if } B \text{ then } A\ \{\ N\ \}}$$

- Für case-Anweisungen läßt sich eine entsprechende Beweisregel formulieren

- Der Beweis einer Spezifikation der Form { V } if B then A1 else A2 { N }

  erfolgt in 2 Schritten:

  1. Zeige die Korrektheit von { V and B } A1 { N }

  2. Zeige die Korrektheit von { V and not B } A2 { N }

Beweisregeln für Verzweigungen

Beispiel: Die Spezifikation { Z1, Z2 sind Zahlen vom Typ integer }

if Z1<Z2 then MIN:=Z1 else MIN:=Z2

{ MIN = min(Z1,Z2) }

ist korrekt, denn

1. { Z1<Z2 }

   MIN:=Z1

   { MIN = min(Z1,Z2) }

und

2. { Z1>=Z2 }

   MIN:=Z2

   { MIN = min(Z1,Z2) }

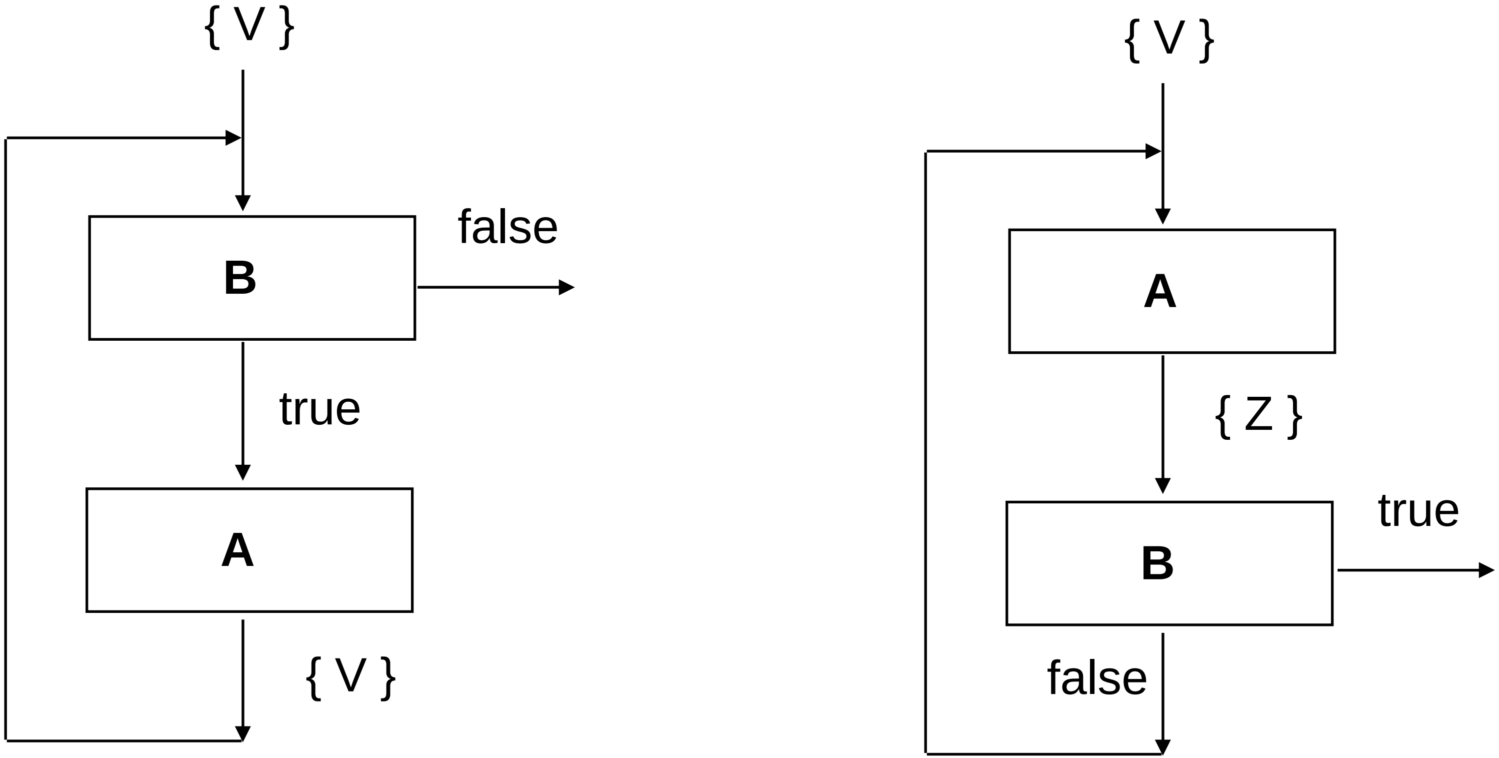
sind korrekt.

2.2.3 Beweisregeln für Iterationen

Der Kontrollfluß für Schleifen

while B *do* A and *repeat* A *until* B

läßt sich beschreiben durch



Beweisregeln für Iterationen

- Aus den Kontrollflußgraphen für Iterationen ergeben sich folgende Beweisregeln:

$$\frac{\{\ V \text{ and } B\ \}\ \ A\ \{\ V\ \}}{\{\ V\ \}\ \ \text{while } B \text{ do } A\ \{\ V \text{ and not } B\ \}}$$

und

$$\frac{\{\ V\ \}\ \ A\ \{\ Z\ \}\ ,\ \{\ Z \text{ and not } B\ \} \Rightarrow \{\ V\ \}}{\{\ V\ \}\ \ \text{repeat } A \text{ until } B\ \{\ Z \text{ and } B\ \}}$$

Beweise für Iterationen

Der Beweis einer Spezifikation der Form { V } while B do A { N } geschieht in 3 Schritten:

1. Schritt: Zeige die Korrektheit von { V and B } A { V }
   Da die Richtigkeit von V bei der Ausführung des Schleifenrumpfes erhalten bleibt, wird V auch Schleifeninvariante genannt.

2. Schritt: Zeige, daß aus V and not B die Richtigkeit von N folgt.

3. Schritt: Zeige, daß die Schleife nach endlich vielen Wiederholungen des Schleifenrumpfes A terminiert.

- Häufig verwendet man auch eine Schleifeninvariante I, die nicht mit V idedentisch ist, aber aus V logisch folgt
  d.h. { I and B } A { I } , { V } => { I }
  { I and not B } => { N }

Beweise für Iterationen

Beispiel: Es soll bewiesen werden, daß

```
SUM := 0;
I := 0;
while I <> 100 do begin I := I + 1;
                        SUM := SUM + I
                  end
```

die Summe 1 + 2 + ... + 100 berechnet.

1. Schritt: Wähle SUM = 1 + 2 + ... + I als Schleifeninvariante INV.

   Dann ist { INV } I := I + 1; SUM := SUM + I { INV } korrekt.

2. Schritt: Aus INV and not (I <> 100) folgt INV and (I = 100)

   d.h. SUM = 1 + 2 + ... + 100

3. Schritt: Die Schleife terminiert, da I die Werte 1, 2, ..., 99, 100 annimmt.

Nach Beendigung der Schleife gilt SUM = 1 + 2 + ... + I and not (I <> 100)

d.h. SUM = 1 + 2 + ... + 100

Beweise für Iterationen

Der Beweis einer Spezifikation der Form { V } repeat A until B { N } erfolgt in 3 Schritten:

1. Schritt: Finde eine Zwischenbedingung Z und beweise die Korrektheit von
   { V } A { Z }

2. Schritt: Zeige, daß aus Z and not B die Richtigkeit von V folgt.

3. Schritt: Zeige, daß die Schleife nach endlich vielen Wiederholungen des Rumpfes terminiert.

- In vielen Fällen sind die Vorbedingung V und die Zwischenbedingung Z identisch, d.h. man verwendet auch hier Schleifeninvarianten.

Beweise für Schleifen

Beispiel: Es soll bewiesen werden, daß

```
SUM := 0;
I := 1;
repeat  SUM := SUM + I;
        I := I + 1
until I = 101
```

die Summe $1 + 2 + \dots\dots + 100$ berechnet

1. Schritt: Wähle $SUM = 1 + 2 + \dots\dots + (I - 1)$ als Schleifeninvariante INV.
   Dann ist { INV } SUM := SUM + I; I := I + 1 { INV } korrekt.
2. Schritt: Aus INV and not (I = 101) folgt natürlich unmittelbar INV
3. Schritt: Die Schleife terminiert, da I der Reihe nach die Werte 1, 2, ..., 100, 101 annimmt.

Nach Beendigung der Schleife gilt $SUM = 1 + 2 + \dots\dots + (I - 1)$ and $(I = 101)$

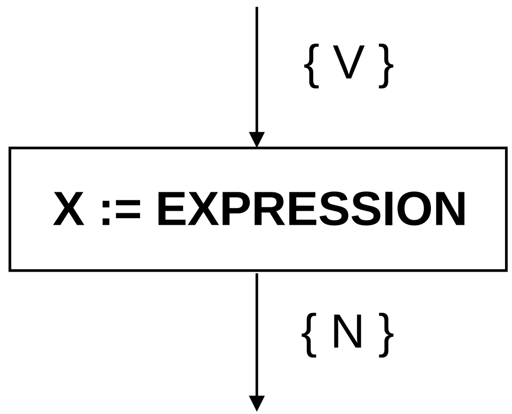
d.h. $SUM = 1 + 2 + \dots\dots + 100$

2.2.4 Beweisregel für Zuweisungen

- Der Kontrollfluß einer Wertzuweisung

X := EXPRESSION

ist wie folgt



Die Vorbedingung V unterscheidet sich von der Nachbedingung N dadurch, daß in N anstelle der Variablen X der Ausdruck EXPRESSION einzusetzen ist.

i.e. V = N [X <- EXPRESSION]

wobei N[X <- EXPRESSION] die Bedingung darstellt, die man erhält, wenn man in N die Variable X durch EXPRESSION ersetzt.

- Beweisregel:

$$\frac{V = N[X \leftarrow \text{EXPRESSION}]}{\{\, V \,\}\ X := \text{EXPRESSION}\ \{\, N \,\}}$$

Beweise für Zuweisungen

Beispiele: Die Spezifikation

$\{ X = a, Y = b \}$
X := X + Y;
Y := X - Y;
X := X - Y
$\{ X = b, Y = a \}$

ist korrekt, was die Anwendung der Beweisregel für Zuweisungen in ruckwärtiger Reihenfolge zeigt:

$\{ X = b, Y = a \}$
X := X - Y
$\{ X - Y = b, Y = a \} \Rightarrow \{ X = a + b, Y = a \}$
Y := X - Y
$\{ X = a + b, X - Y = a \} \Rightarrow \{ X = a + b, Y = b \}$
X := X + Y
$\{ X + Y = a + b, Y = b \} \Rightarrow \{ X = a, Y = b \}$

Sonstige Beweisregeln

- Für alle anderen PASCAL-Anweisungen, wie
  - for .... to ...... do .....
  - case ...... of .... end
  - READ, READLN
  - WRITE, WRITELN
  - GET, PUT
  - with ...... do ......
  - Prozeduraufrufe
  - ....

  lassen sich entsprechende Beweisregeln aufstellen.

- Weitere Details darüber sind enthalten in:

  S. Alagic, M. A. Arbib; The Design of Well-Structured and Correct Programs; Springer Verlag; 1978

Schwächste Vorbedingung

1st Semester, 1999

Beispiele:

Was ist jeweils die schwächste Vorbedingung V der folgenden Programme

(1) *var x, y: integer;*

```
    x := 3 * x + 1;
    if x >= 12 then y:= x else  y := -x;
    y := x * y;
```

Nachbedingung N: y > 0

(2) *var s, t: integer;*

```
    s := 5 * t + 3;
    if s > t then t:= s - 8;
```

Nachbedingung N: -20 <= t <= 20