Software-Qualitätsmanagement

Kernfach Angewandte Informatik

Sommersemester 2007

Prof. Dr. Hans-Gert Gräbe

http://bis.informatik.uni-leipzig.de/HansGertGraebe

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zusicherungen

- Konditionierung bedeutet, das Programm in Einheiten von Wenn-Dann-Aussagen zu zerlegen.
- **Zusicherungen** beschreiben dazu bestimmte **Eigenschaften** der Datenlandschaft an vorgegebenen Kontrollpunkten im Programmfluss.

  Logische Aussagen über Werte von Variablen im Programm

  Formulierung auf unterschiedliche Art und Weise:

  - umgangssprachlich, z. B. x ist nicht negativ
  - formal, z. B. $x \geq 0$

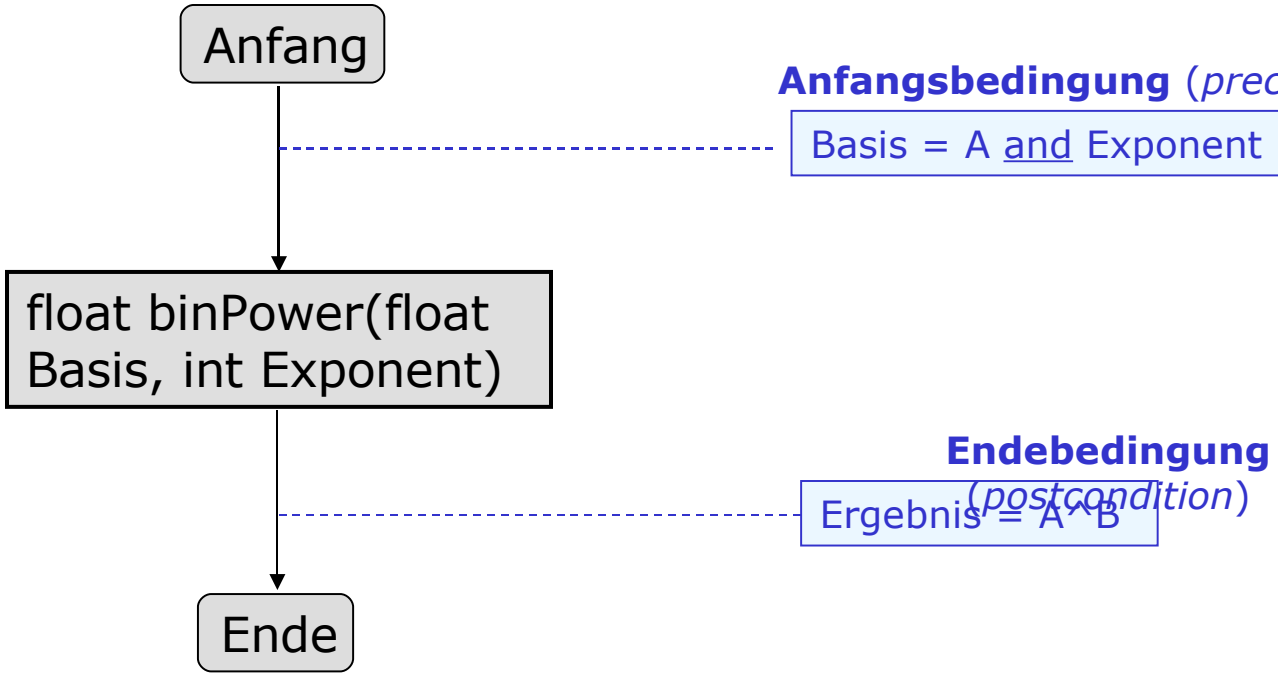
  gebräuchliche Notationen :

  Annotation durch gestrichelte Linien am Programmablaufplan

  Kommentare oder Makros in Programmiersprachen, z. B.
  `assert (x >= 0); //Zusicherung ist ungültig, wenn x negativ ist.`

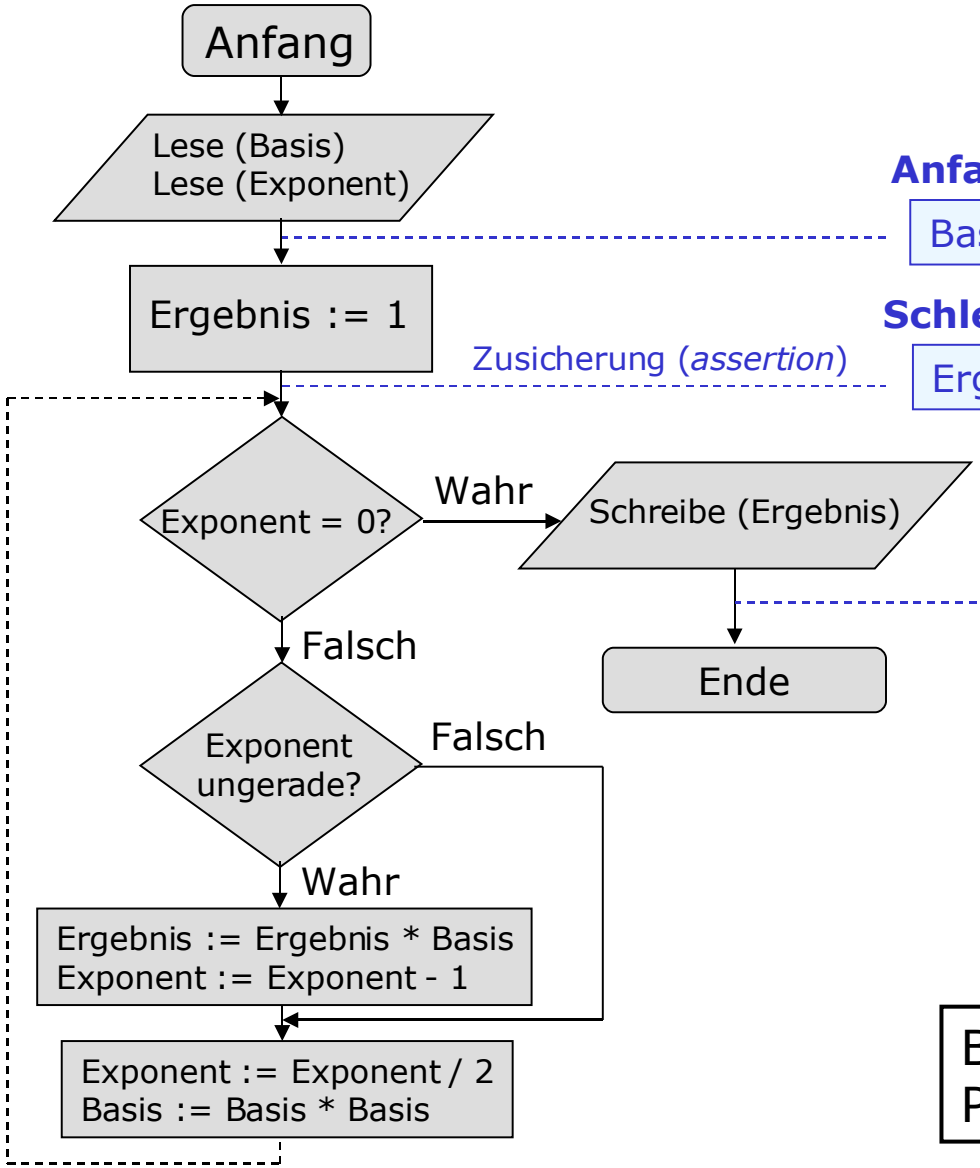
  Ergänzung von Struktogrammen durch abgerundete Rechtecke

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



Beispiel: Binäres Potenzieren, Spezifikation

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



Anfangsbedingung (*precondition*)

Basis = A and Exponent = B and B ≥ 0

Schleifen-Invariante

Zusicherung (*assertion*)

Ergebnis*Basis^Exponent = A^B and B ≥ 0

Endebedingung (*postcondition*)

Ergebnis = A^B

Beispiel: Binäres Potenzieren, Programmablaufplan

Beispiel binäres Potenzieren in Pseudocode-Notation

```
float binPower(float Basis, int Exponent) {
    /* Ass: Basis = A and Exponent = B and B ≥ 0 */ // Anfangsbedingung
    float Ergebnis:=1.0;
    /* Ass: Ergebnis*Basis^Exponent = A^B and Exponent ≥ 0 */
    while (Exponent > 0) {
    /* Schleifeninvariante:
        Ass: Ergebnis*Basis^Exponent = A^B and Exponent > 0 */
        if (isOdd(Exponent)) {
            Ergebnis := Ergebnis * Basis;
            Exponent:= Exponent-1;
        }
        Exponent := Exponent/2;
        Basis := Basis * Basis;
    }
        return Ergebnis;
} /* Ass: Return-Wert = A^B */ // Endbedingung
```

Beispiel gcd-Berechnung mit Euklidschem Algorithmus

```
int gcd(int a, int b) {
    /* Ass: a = A and b = B */ // Anfangsbedingung
    while (b != 0) {
        /* Ass: gcd(a,b) = gcd(A,B) and b ≠ 0 */
        int r = a mod b; a :=b, b:=r;
    }
    /* Ass: b = 0 and a [=gcd(a,b)] = gcd(A,B) */
        return a;
} /* Ass: Return-Wert = gcd(A,B) */ // Endebedingung
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Beispiel Erweiterter Euklidscher Algorithmus

```
(int g, int u, int v) ExtendedEuklid(int a, int b) {
    /* Ass: a = A and b = B */
    int ua:=1; int va:=0; int ub:=0; int vb:=1;
    /* Ass: gcd(a,b) = gcd(A,B) and a = ua*A+va*B and b = ub*A+vb*B */
    while (b != 0) {
        /* Ass: gcd(a,b) = gcd(A,B) and b ≠ 0 and
                    a = ua*A+va*B and b = ub*A+vb*B */
        int q = a div b;
        int r=a-q*b; int uc:=ua-q*ub; int vc:=va-q*vb;
        a :=b, b:=r; ua:=ub; ub:=uc; va:=vb; vb:=vc;
    }
    /* Ass: b = 0 and a [=gcd(a,b)] =gcd(A,B) and a = ua*A+va*B */
        return (a,ua,va);
} /* Ass: g = gcd(A,B) and g = u*A+v*B */
```

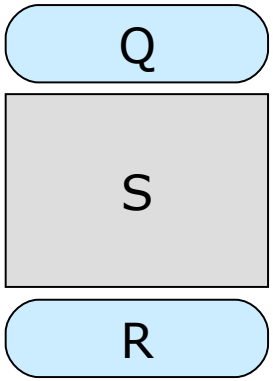
Konditionierung = Programm(teil) in einen Wenn-Dann-Zusammenhang einspannen

Anfangsbedingung (Vorbedingung, *precondition*),

- legt zulässige Werte der Variablen vor dem Ablauf des Programms fest

Endebedingung (Nachbedingung, *postcondition*),

- legt die gewünschten Werte der Variablen, sowie Beziehungen zwischen den Variablen nach dem Programmlauf fest.



Anfangsbedingung

spezifiziertes Programm

Endebedingung

Notation:

in linearem Programmtext: **{Q} S {R}**

bei Spezifikation ohne konkretes Programm: **{Q} . {R}**

Unterscheide zwischen Zuweisung und Zusicherung

Exponent := A (Zuweisung im Programmtext)

Exponent = A (Zusicherung im Kontext)

sowie zwischen Wert vor und nach der Zuweisung.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Verifikation des Programms { **Q** } **S** { **R** }
= Mathematisch exakter Beweis der Aussage

„Wenn vorher **Q** erfüllt ist und **S** ausgeführt wird, dann ist danach **R** erfüllt."

Beispiele

- Verifikation des Beispiels **binPower**
- Verifikation des Beispiels **gcd**
- Verifikation des Beispiels **ExtendedEuklid**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Verifikationsregeln

- **Voraussetzung**: Programm ist aus konditionierten Bausteinen modular zusammengesetzt

  Korrektheit des gesamten Programms ergibt sich aus der korrekten Zusammensetzung korrekter Teilstrukturen
- **Vorgehen**: Komplexes Programm wird verifiziert durch schrittweises Zusammensetzen aus **verifizierten einfacheren Strukturen** nach wenigen einfachen **Verifikationsregeln**.
- Folgende Verifikationsregeln existieren:

  Konsequenz-Regel,

  Zuweisungs-Regel,

  Sequenz-Regel,

  if-Regel und

  while-Regel

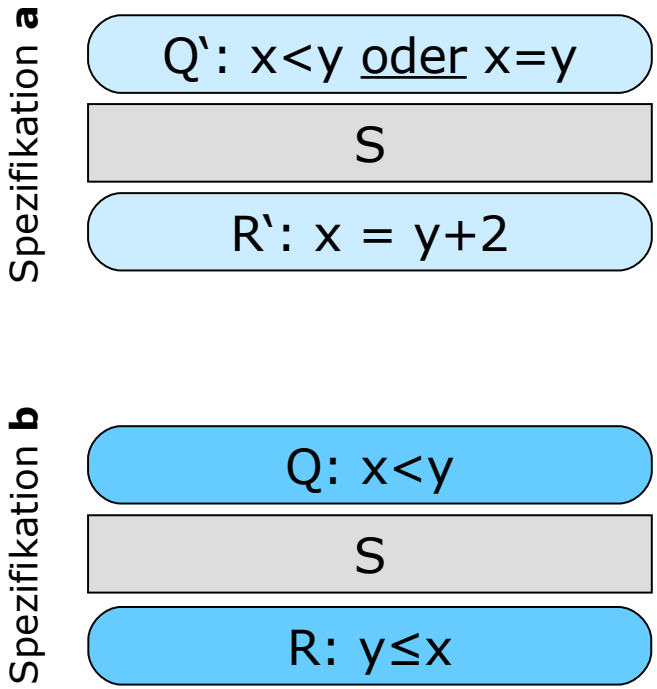
Konsequenz-Regel

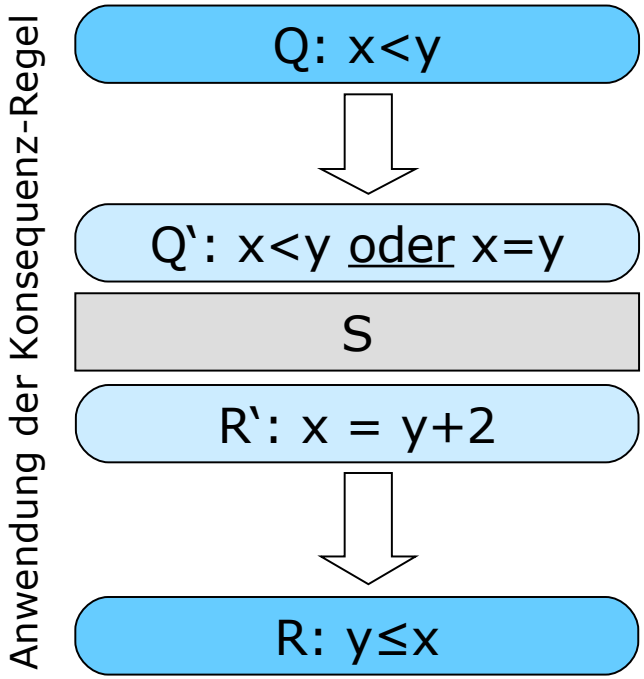
Gilt **{Q'} S {R'}** und

Q' wird durch Q ersetzt, wobei Q schärfer ist als Q'.

R' wird durch R ersetzt, wobei R schwächer ist als R'.

so gilt auch **{Q} S {R}.**





- Geht man vorwärts durch ein Programm, so kann man Bedingungen abschwächen:

  Hinzufügen eines Terms mit **oder**-Verknüpfung

  Weglassen eines **und**-verknüpften Terms

  Schwächere Bedingung

- Bei rückwärtiger Abarbeitung eines Programms, dürfen Bedingungen verschärft werden:

  Hinzufügen eines Terms mit **und**-Verknüpfung

  Weglassen eines **oder**-verknüpften Terms

  Schärfere Bedingung

- Notation von Verifikationsregeln als Schlussregel:

$$\frac{\text{Voraussetzungen}}{\text{Schlussfolgerung}} \qquad \frac{Q \Rightarrow Q', \{Q'\}\, S\, \{R'\}, R' \Rightarrow R}{\{Q\}\, S\, \{R\}}$$

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zuweisungs-Regel

- Die Zuweisung x:=A verändert den Wert von x

  Beispiel: { y+z = 25 } x:=y+z { x = 25 }

- Allgemeine Struktur: **{ R(A) } x:=A { R(x) }**

  so zu verstehen: Hat man einen logischen Ausdruck R=R(x) mit der freien Variablen x und bildet Q::=R(A) durch Ersetzen dieser Variablen mit dem Ausdruck A, so ist die Aussage

  { Q } x:=A { R }

  wahr.

- Regel wird eingesetzt beim Rückwärtsarbeiten, um aus einer Nach- eine Vorbedingung abzuleiten:

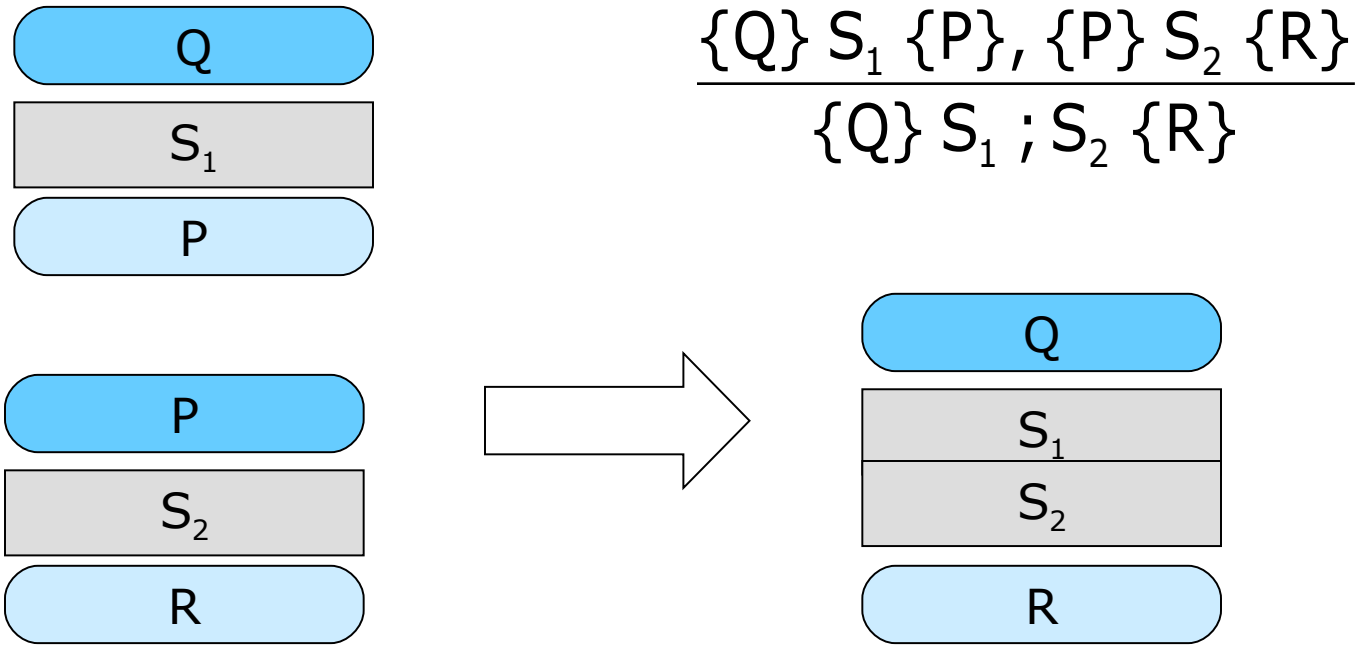
  Bsp.: { Q? } x := x+25 { x = 2y }

  { R(A) } x‘ = x+25 { R(x‘) ::= x‘ = 2y}

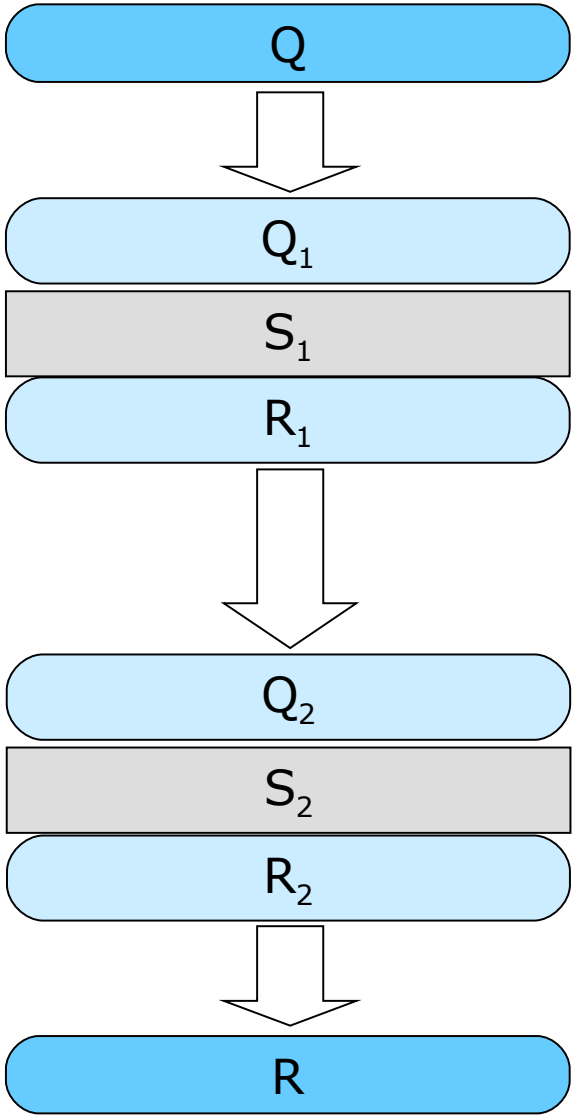
  Vorbedingung { Q ::= 2y = x + 25 }

Sequenz-Regel

- Zwei Programmteile S_1 und S_2 können zusammengesetzt werden, wenn die Nachbedingung von S_1 gleich der Vorbedingung von S_2 ist.

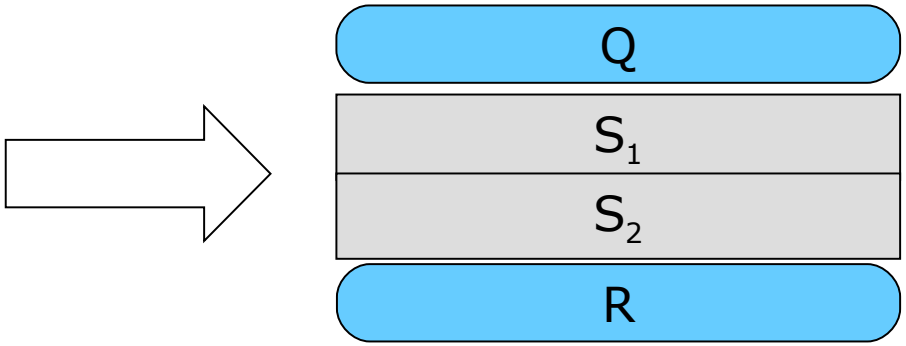


$$\frac{\{Q\}\ S_1\ \{P\},\ \{P\}\ S_2\ \{R\}}{\{Q\}\ S_1\ ;S_2\ \{R\}}$$



Die Sequenz-Regel kann mit Hilfe der Konsequenz-Regel noch verallgemeinert werden:
Es genügt, wenn die Nachbedingung von S_1 „schärfer" ist als die Vorbedingung von S_2, um S_1 und S_2 zu einem Programmstück zusammenzusetzen.

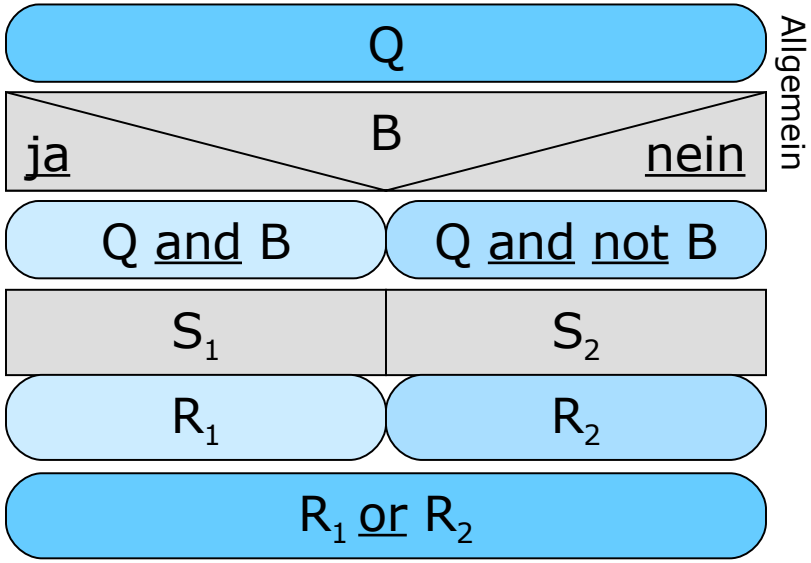
$$\frac{Q \Rightarrow Q_1, \{Q_1\}\, S_1\, \{R_1\}, R_1 \Rightarrow Q_2, \{Q_2\}\, S_2\, \{R_2\}, R_2 \Rightarrow R}{\{Q\}\, S_1\, ; S_2\, \{R\}}$$

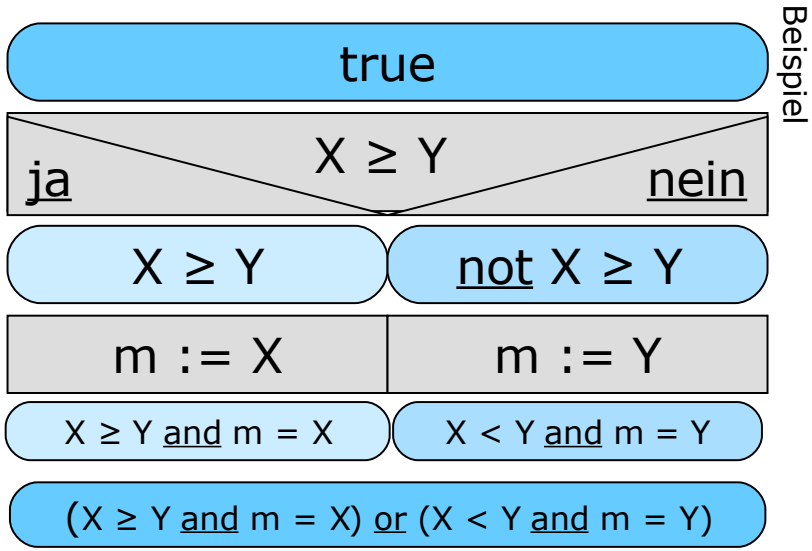


if-Regel

- Gibt an, unter welchen Voraussetzungen zwei Programmstücke S_1 und S_2 und eine Bedingung B zu einer zweiseitigen Auswahl mit der Vorbedingung Q und der Nachbedingung R zusammengesetzt werden können.

$$\frac{\{Q \text{ and } B\}\ S_1\ \{R\},\ \{Q \text{ and not } B\}\ S_2\ \{R\}}{\{Q\}\ \text{if } B \text{ then } S_1 \text{ else } S_2\ \{R\}}$$

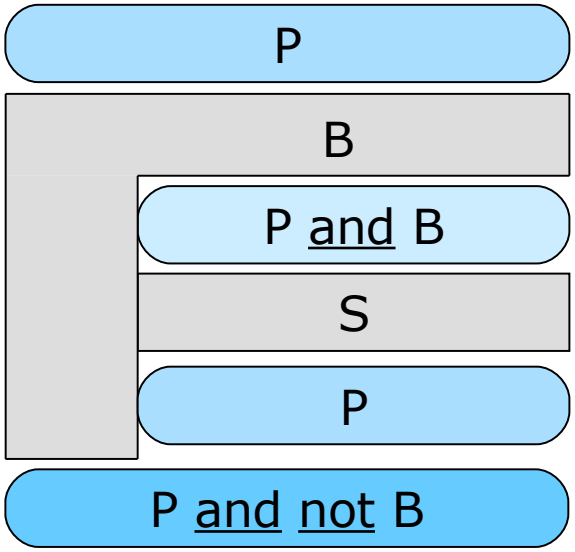




UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

while-Regel

- Bei der Verifikation von Schleifen spielt eine invariante Zusicherung **P**, die **Schleifeninvariante** eine entscheidende Rolle.
- Die Invariante gilt vor der Schleife und nach dem Schleifenrumpf.



$$\frac{\{P \text{ and } B\}\ S\ \{P\}}{\{P\}\ \text{while } B \text{ do } S\ \{P \text{ and not } B\}}$$

Diese Regel beweist nur **partiell** die Korrektheit der Schleife, denn die **Termination** wird durch P nicht garantiert.

Zum Beweis der Termination einer Schleife

- Wiederholungsbedingung B muss irgendwann falsch sein.
- Prüfung der Termination mit Hilfe einer **Terminationsfunktion t**.

  Idee: Die Terminationsfunktion

  t : Programmzustände -> **Z**

  ist nach unten beschränkt **und** wird in jedem Schleifendurchlauf kleiner.

Formale Formulierung der Bedingungen für t:

- { P and B and t = T } S { P and t<T } (T ist freie Variable)
- P and B t ≥ 0

- **Variation**: Kettenbedingung auf Halbordnungen
  - Beispiel: Termordnungen auf dem Term-Monoid $T=T(x_1,...x_n)$
  - es reicht die Kettenbedingung statt Beschränktheit

Konditionierungsregel für Schleifen

Bei gegebener Invariante **P** und Terminationsfunktion **t** muss eine **while**-Schleife folgende Punkte erfüllen:

5. Die Invariante P muss während der Initialisierung der Schleife gesichert werden:

$$\{Q\}\ \text{init}\ \{P\}$$

- P bleibt im Schleifenrumpf S invariant, t wird bei jedem Ausführen des Schleifenrumpfes verringert.

$$\{P \text{ and } B \text{ and } t = T\}\ S\ \{P \text{ and } t < T\}$$

- t ist vor jedem Ausführen des Schleifenrumpfes nicht negativ.

$$P \text{ and } B \quad t \geq 0$$

- Die Nachbedingung R ist eine Folge der Schleifeninvariante.

$$P \text{ and not } B \quad R$$

Entwickeln einer Schleife durch Weglassen einer Bedingung

Gegeben sei eine Spezifikation {Q} . {R: U and V}

- R wird aufgeteilt in {R = P and not B}: P = U, B = not V
  - Die Invariante P ergibt sich durch Weglassen einer Bedingung.
  - Die weggelassene Bedingung {not V} wird zur Abbruchbedingung.

7. Initialisierung der Invarianten P durch ein Programmstück

   {Q} init {P}

8. Entwicklung eines Schleifenrumpfes mit der Spezifikation

   {P and B} S {P}

- Hinzufügen der Terminationsbedingung **t**
  - **t** ergibt sich häufig aus dem Vergleich der Initialisierung mit der Abbruchbedingung {not B}.

Beispiel: *int y = isqrt(int x) mit y= ⌊sqrt(x)⌋*

$\{Q: x \geq 0\}$ und $\{R: y \geq 0 \text{ and } y^2 \leq x \text{ and } x < (y+1)^2\}$

Aufteilen von R: $\{P: y \geq 0 \text{ and } x \geq y^2\}$ und $\{B: x \geq (y+1)^2\}$

Initialisierung: $\{Q: x \geq 0\}\ y:=0\ \{P\}$

Schleife: $\{P \text{ and } B\}\ y:=y+1\ \{P\}$

$\{y \geq 0 \text{ and } x \geq (y+1)^2\}\ y`=y+1\ \{y` \geq 0 \text{ and } x \geq y`^2\}$

Terminationsfunktion: **t** := x − y

$\{P \text{ and } B \text{ and } t=T\}\ y:=y+1\ \{P \text{ and } t<T\}$

Java-Implementierung:

```
int isqrt(int x) {
    int y=0;
    while ((y+1)*(y+1) <= x)
        y=y+1;
    return y;
}
```

oder nach leichter Optimierung

```
int isqrt(int x) {
    int y=1;
    while (y*y <= x)
        y=y+1;
    return (y-1);
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Symbolisches Testen: Überblick

- **Idee**: Die Eingabeparameter des Programms werden mit symbolischen Variablen belegt und längs aller möglicher Kontrollflüsse alle möglichen **Zwischenergebnisse** und **Konditionen** in symbolischer Form bestimmt.

  Methode ist besonders gut geeignet, wenn sich die an Verzweigungspunkten gültigen Kombinationen boolescher Bedingungen vereinfachen lassen und Zwischenergebnisse arithmetischer Natur sind.

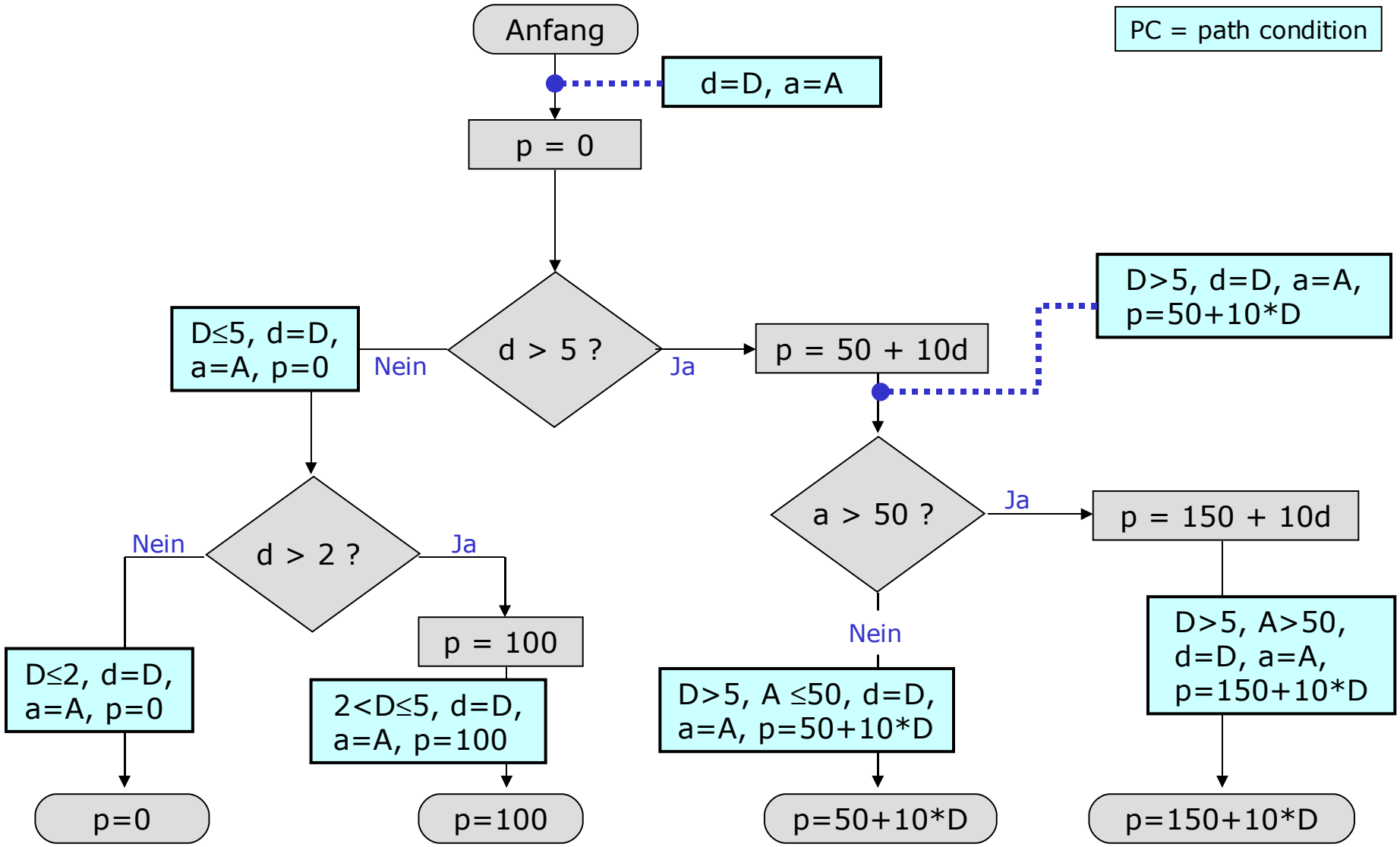
- **Methode hat Beweiskraft** im mathematischen Sinn, wenn die symbolischen Parameter durch alle denkbaren konkreten Parameterwerte ersetzt werden können.

  etwa darf das Zwischenergebnis y/x nicht ohne die Kondition x!=0 auftreten.

- **Schleifen** lassen sich in diesem Ansatz nur bedingt abbilden.

Beispiel

```
int berechnePraemie(int Dienstjahre, int Alter) {
    Praemie = 0;
    if (Dienstjahre > 5) {
        Praemie = 50 + 10 * Dienstjahre;
        if (Alter > 50) Praemie = Praemie + 100;
    }
    else if (Dienstjahre > 2) Praemie = 100;
    return Praemie;
}
```


Anfang
d=D, a=A
p = 0
d > 5 ?
Nein
Ja
D≤5, d=D, a=A, p=0
p = 50 + 10d
D>5, d=D, a=A, p=50+10*D
PC = path condition
d > 2 ?
Nein
Ja
a > 50 ?
Ja
p = 150 + 10d
D≤2, d=D, a=A, p=0
p = 100
2<D≤5, d=D, a=A, p=100
Nein
D>5, A ≤50, d=D, a=A, p=50+10*D
D>5, A>50, d=D, a=A, p=150+10*D
p=0
p=100
p=50+10*D
p=150+10*D

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Beispiel

```
List sort(List(a,b,c)) {
    if (a>b) then (b,a):=(a,b);
    if (b>c) then (c,b):=(b,c);
    if (a>b) then (b,a):=(a,b);
    return List(a,b,c);
}
```