Software-Qualitätsmanagement

Kernfach Angewandte Informatik

Sommersemester 2004

Prof. Dr. Hans-Gert Gräbe

Gliederung

1. Testen versus Verifizieren
2. Konditionieren von Programmen
   - Zusicherungen
   - Spezifizieren mit Anfangs- und Endebedingung
3. Programmverifikation
   - Verifikationsregeln
   - Termination von Schleifen
   - Entwickeln von Schleifen
4. Symbolisches Testen

1. Testen versus Verifizieren

Testen = Stichprobenartige Überprüfung des Programms

- Auswahl einer möglichst repräsentativen Menge von Eingabedaten
- Test liefert eine gewisse Plausibilität des korrekten Funktionierens, aber keine Sicherheit
- Mit Tests lassen sich nicht nur die Korrektheit, sondern auch andere Parameter (Profiling) erfassen.

Tests haben Überzeugungskraft, aber keine Beweiskraft im streng deduktiven Verständnis.
Es bleibt immer ein Rest von Unsicherheit.

1. Testen versus Verifizieren

Verifikation = Formal exakte Methode, um durch theoretische Analyse die Konsistenz zwischen Spezifikation und Implementierung für *alle* möglichen Eingabedaten zu **beweisen**.

- Beweis im mathematisch deduktiven Verständnis

Grundstruktur mathematischer Beweise

- modularer Aufbau aus Aussage-Bausteinen mit „Wenn..., dann...“ Struktur
- Verifikation eines neuen Bausteins durch **Beweis** = Zusammensetzen einer Argumentation aus bereits verifizierten Bausteinen nach den Regeln der (math.) Logik

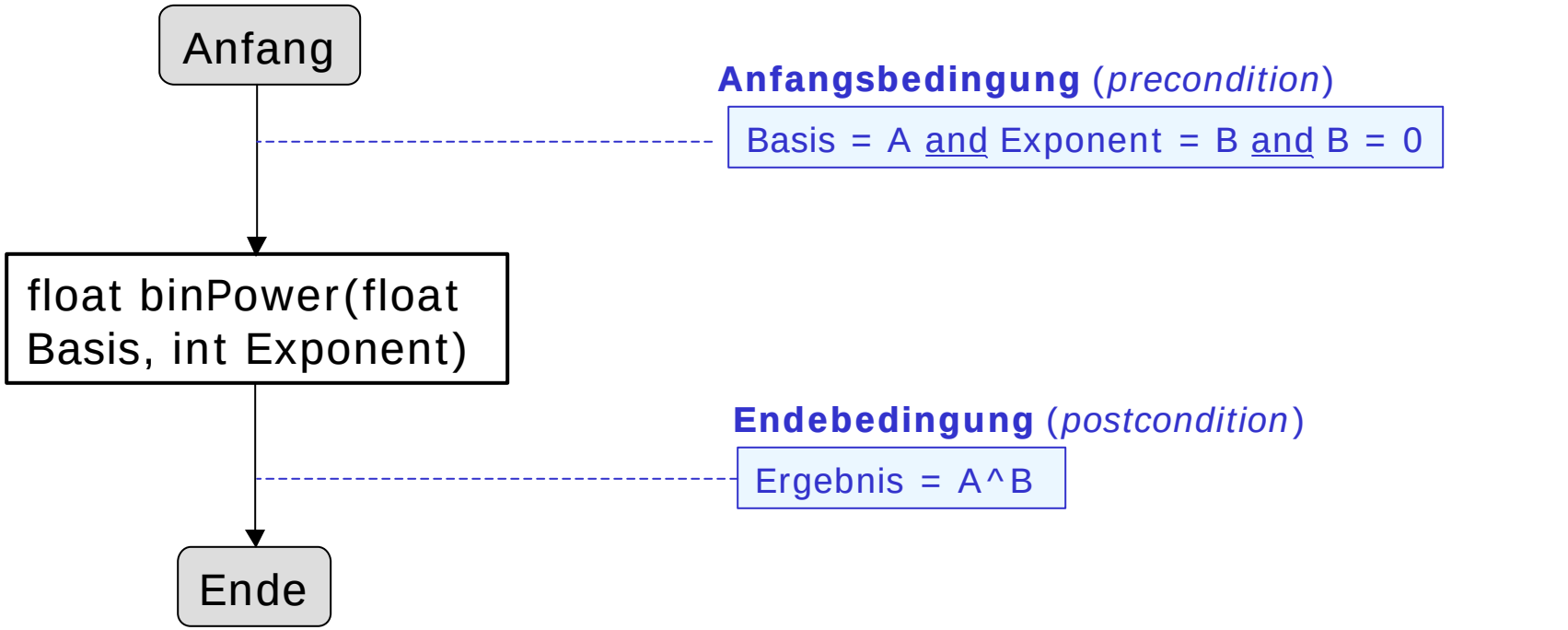
Programm = schrittweise Transformation der Eingabe- in die Ausgabedaten nach einem vorgegebenen Algorithmus

- modularer Aufbau, der für Zwecke der Verifikation entsprechend **konditioniert** werden muss.

Zusicherungen

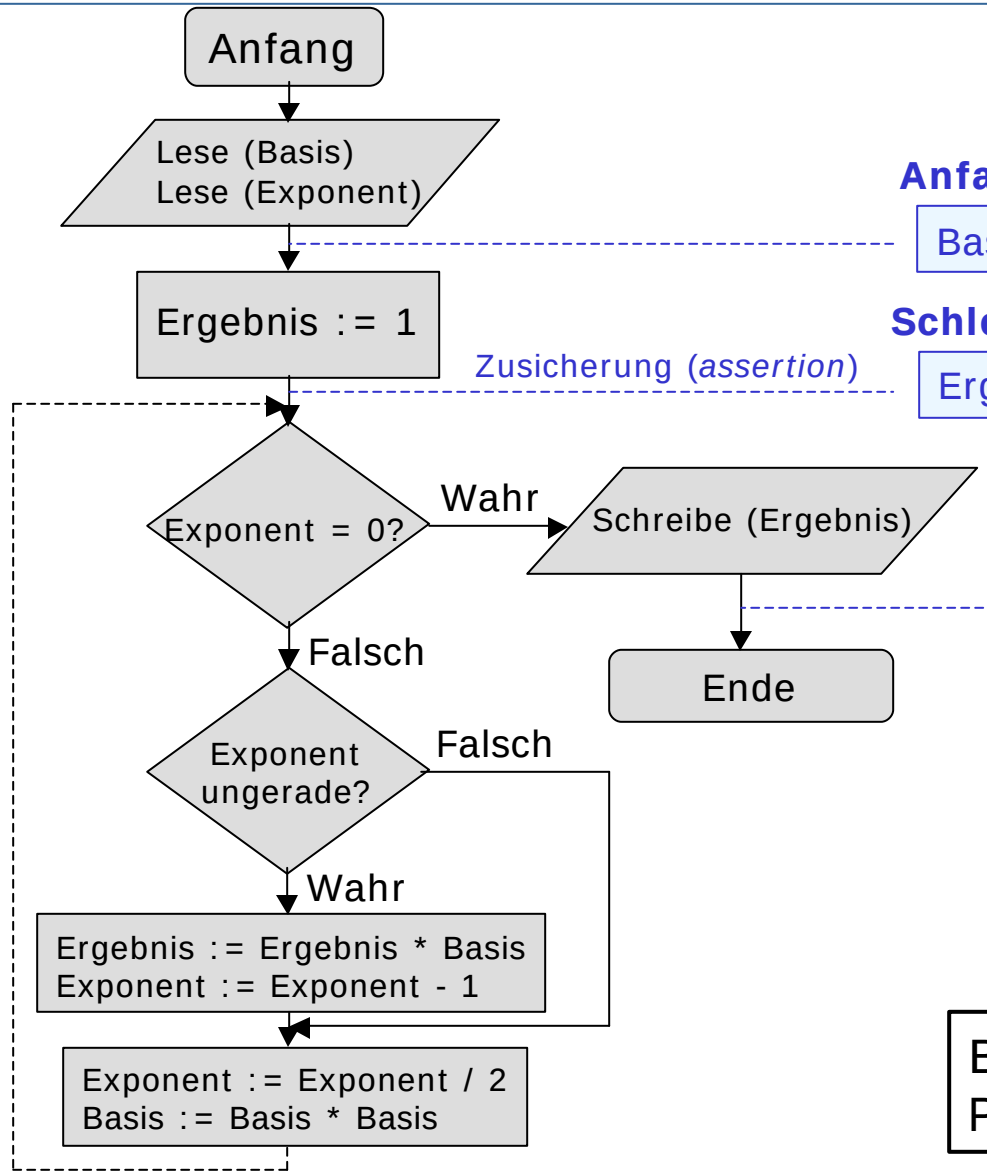
- Konditionierung bedeutet, das Programm in Einheiten von Wenn-Dann-Aussagen zu zerlegen.
- **Zusicherungen** beschreiben dazu bestimmte **Eigenschaften** der Datenlandschaft an vorgegebenen Kontrollpunkten im Programmfluss.
  - Logische Aussagen über Werte von Variablen im Programm
  - Formulierung auf unterschiedliche Art und Weise:
    - umgangssprachlich, z. B. x ist nicht negativ
    - formal, z. B. x = 0
  - gebräuchliche Notationen :
    1. Annotation durch gestrichelte Linien am Programmablaufplan
    2. Kommentare oder Makros in Programmiersprachen, z. B. assert (x >= 0); //Zusicherung ist ungültig, wenn x negativ ist.
    3. Ergänzung von Struktogrammen durch abgerundete Rechtecke

2. Konditionierung von Programmen



Beispiel: Binäres Potenzieren, Spezifikation

2. Konditionierung von Programmen



Anfangsbedingung (*precondition*)

Basis = A and Exponent = B and B = 0

Schleifen-Invariante

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
```

2. Konditionierung von Programmen

Beispiel gcd-Berechnung mit Euklidschem Algorithmus

```
int gcd(int a, int b) {
    /* Ass: a = A and b = B */ // Anfangsbedingung
    /* Ass: gcd(a,b) = gcd(A,B) */
    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
```

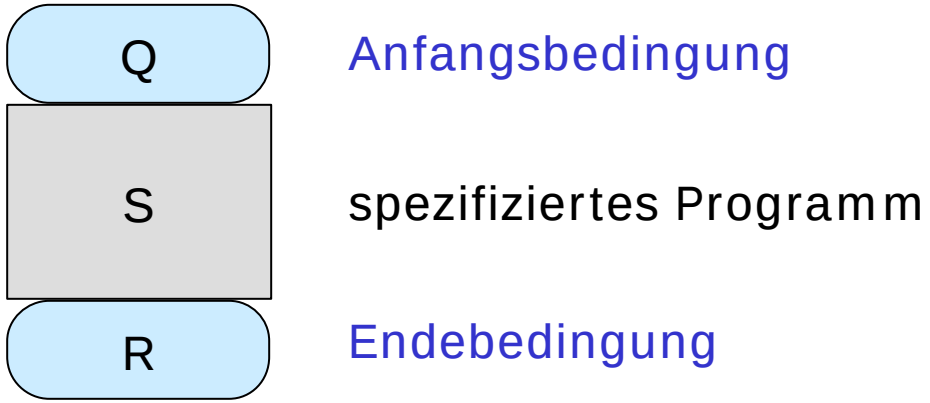
Beispiel Erweiterter Euklidscher Algorithmus

```
(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,ub);
} /* Ass: g=gcd(A,B) and g = u*A+v*B */
```

2. Konditionierung von Programmen

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.



2. Konditionierung von Programmen

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.

3. Programmverifikation

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**

Verifikationsregeln

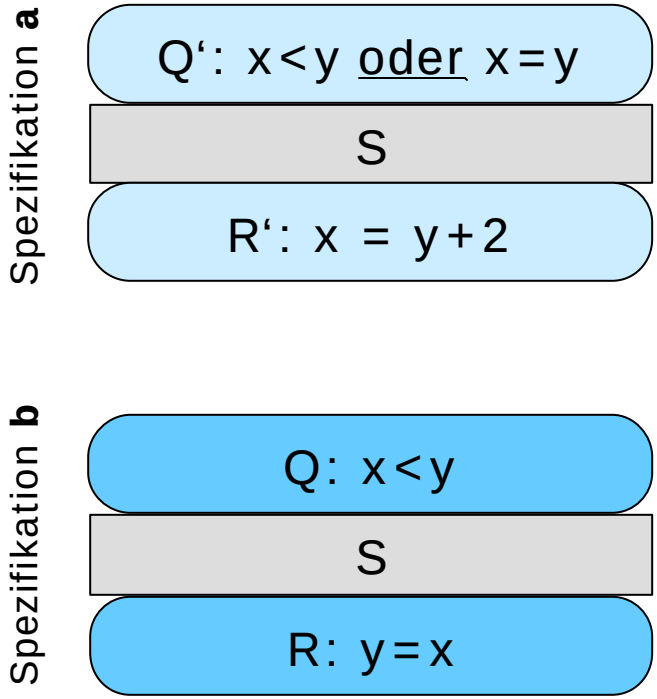
- **Voraussetzung**: Programm ist aus konditionierten Bausteinen modular zusammengesetzt
  - Korrektheit des gesamten Programms ergibt sich aus der korrekten Zusammensetzung korrekter Teilstrukturen
- **Vorgehen**: Komplexes Programm wird verifiziert durch schrittweises Zusammensetzen aus **verifizierten einfacheren Strukturen** nach wenigen einfachen **Verifikationsregeln**.
- Folgende Verifikationsregeln existieren:
  - Konsequenz-Regel,
  - Zuweisungs-Regel,
  - Sequenz-Regel,
  - **if**-Regel und
  - **while**-Regel

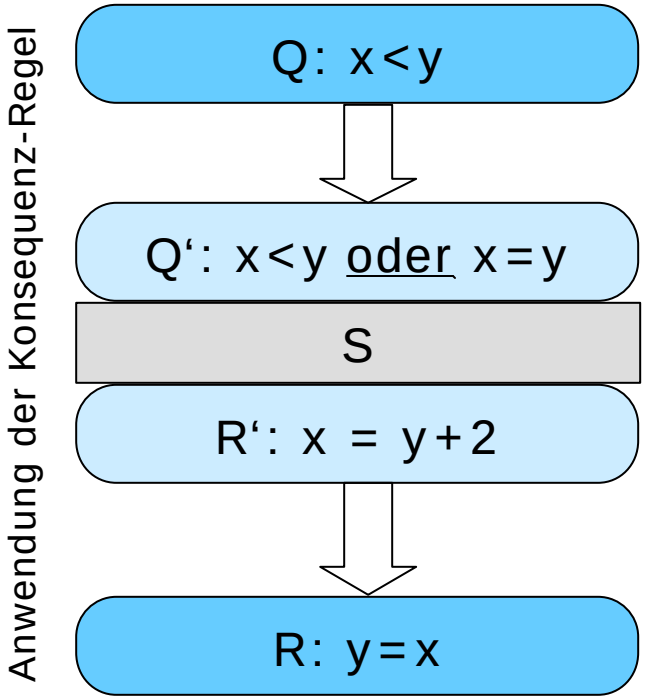
Konsequenz-Regel

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}.**





3. Programmverifikation

- 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:

$$\frac{\text{Voraussetzungen}}{\text{Schlussfolgerung}}$$

$$\frac{Q \Rightarrow Q',\ \{Q'\}\ S\ \{R'\},\ R' \Rightarrow R}{\{Q\}\ S\ \{R\}}$$

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:

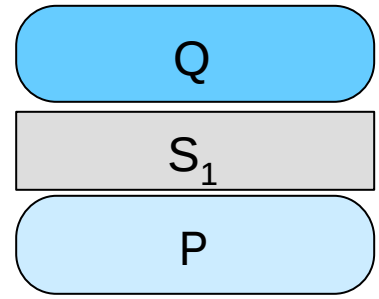
  Bsp.: { Q? } x := x+25 { x = 2y }

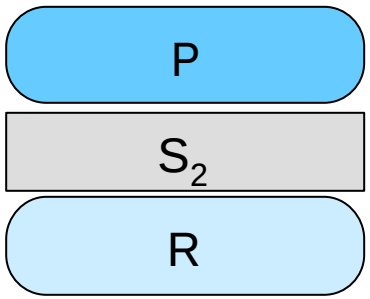
  { R(A) } x‘ = x+25 { R(x‘) ::= x‘ = 2y}

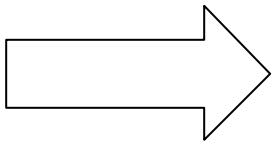
  ➔ Vorbedingung { Q ::= 2y = x + 25 }

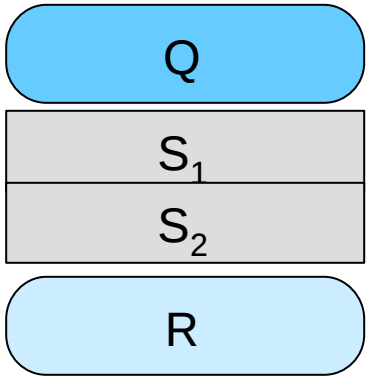
Sequenz-Regel

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



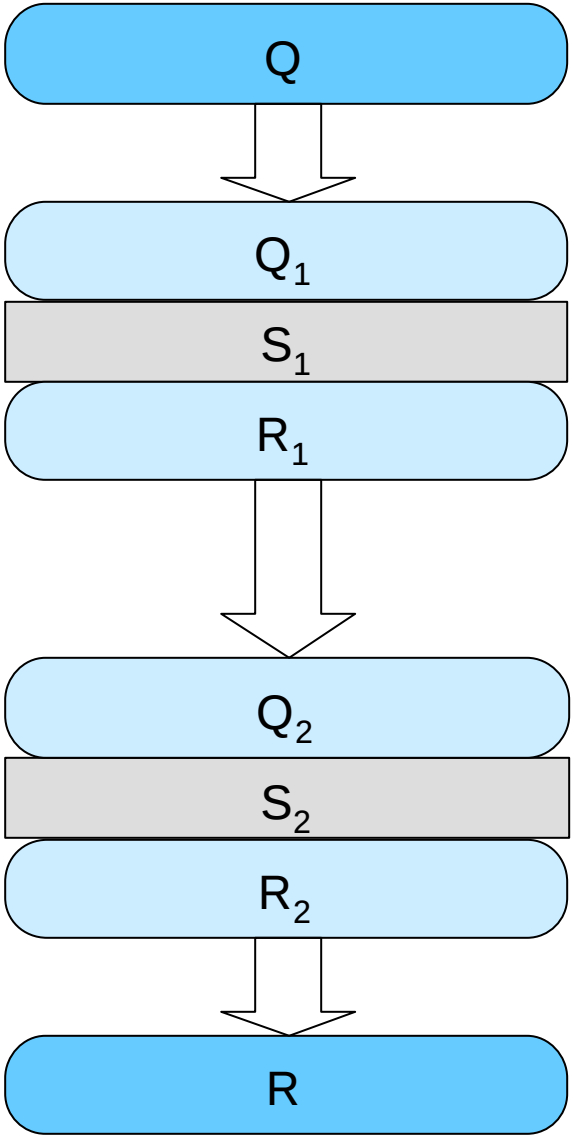






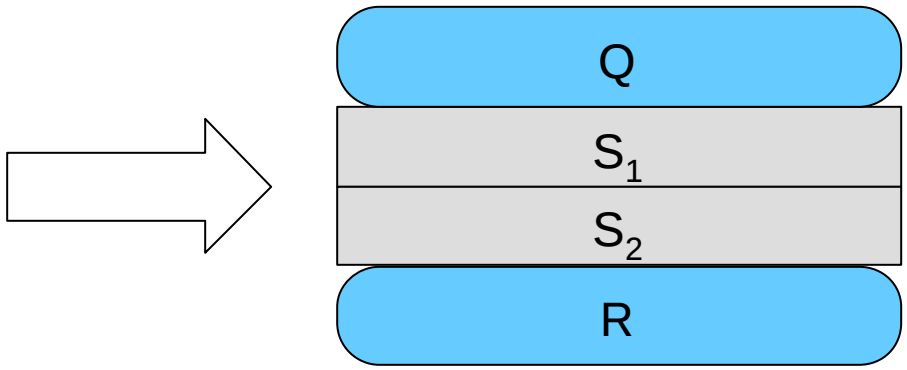
$$\frac{\{Q\}\ S_1\ \{P\},\ \{P\}\ S_2\ \{R\}}{\{Q\}\ S_1\ ;S_2\ \{R\}}$$

3. Programmverifikation



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.

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