II.1. Grundelemente der Programmierung

- 1. Erste Schritte
- 2. Einfache Datentypen
- 3. Anweisungen und Kontrollstrukturen
- 4. Verifikation
- 5. Reihungen (Arrays)

4. Verifikation

- **Spezifikation: Angabe, *was* ein Programm tun soll**
  - natürliche Sprache
  - grafische Sprachen (UML, ...)
  - logische Sprachen (Z, VDM, ...)

- **Testen: Überprüfung für endlich viele Eingaben**

  ⇒ keine 100% Sicherheit

- **Verifikation: Mathematischer Beweis der Korrektheit**
  - Terminierung: Hält Programm immer an?
  - Partielle Korrektheit: Falls Programm anhält, erfüllt es Spezifikation?
  - Totale Korrektheit: Terminierung & Partielle Korrektheit

⇒ Semantik der Programmiersprache

Fakultät

```
public static void main (String [] arguments) {
     int n = IO.eingabe(), i, res;

     i = n;

     res = 1;

     while (i > 1) {

      res = res * i;

      i = i - 1;

     }

     System.out.println("Die Fakultät ist " + res);
 }
```

Verifikation

Programm P

```
i = n;
res = 1;
while (i > 1) {
   res = res * i;
   i = i - 1;
}
```

- **Spezifikation:**
  Programm berechnet (in **`res`**) Fakultät von **n**
- **Terminierung:**
  Programm hält an, weil **`i`** in jedem Schleifendurchlauf kleiner wird
- **Partielle Korrektheit:**
  Nach Ausführung ist **`res`** = **n**!
- **Totale Korrektheit**

Wie beweist man so etwas ?

➡ Verifikation nötig bei sicherheitskritischen Anwendungen

➡ hilft für Programmentwurf und Programmierstil

Partielle Korrektheit: Hoare-Kalkül

- **Spezifikation (zur partiellen Korrektheit)**

$$<\varphi> \ \mathbf{P} \ <\psi>$$

  Wenn vor Ausführung von P Vorbedingung φ gilt

  und Ausführung von P terminiert,

  dann gilt hinterher Nachbedingung ψ.

- **Bsp:** < **`true`** > P < **`res = n!`** >

- Partielle Korrektheit ist *semantische* Aussage

  Hoare-Kalkül: 7 *syntaktische* Regeln zur Herleitung von Korrektheitsaussagen

Zuweisungsregel

$$\frac{\qquad}{\texttt{<}\varphi\texttt{[x/t]> x = t; <}\varphi\texttt{>}}$$

`x` ist Variable, **`t`** ist Ausdruck (ohne Seiteneffekte),
φ **`[x/t]`** ist φ mit allen **`x`** ersetzt durch **`t`**

Bsp: `<5 = 5>   x = 5;   <x = 5>`

```
<5 = 5>

x = 5;

<x = 5>
```

Konsequenzregel 1 (Stärkere Vorbedingung)

$$\frac{<\varphi>\ \mathbf{P}\ <\psi> \qquad\qquad \alpha \Rightarrow \varphi}{<\alpha>\ \mathbf{P}\ <\psi>}$$

Bsp: `<true>  x = 5;  <x = 5>`, denn:

$$\frac{\texttt{<5 = 5> x = 5; <x = 5>} \qquad\qquad \texttt{true} \Rightarrow \texttt{5 = 5}}{\texttt{<true> x = 5; <x = 5>}}$$

```
<true>
<5 = 5>
x = 5;
<x = 5>
```

Konsequenzregel 2 (Schwächere Nachbedg.)

$$\frac{\langle \varphi \rangle \ \mathbf{P} \ \langle \psi \rangle \qquad\qquad \psi \Rightarrow \beta}{\langle \varphi \rangle \ \mathbf{P} \ \langle \beta \rangle}$$

Bsp: $\langle \mathtt{true} \rangle \quad \mathtt{x = 5;} \quad \langle \mathtt{x} \geq \mathtt{5} \rangle$, denn:

$$\frac{\langle \mathtt{true} \rangle \ \mathtt{x = 5;} \ \langle \mathtt{x = 5} \rangle \qquad\qquad \mathtt{x = 5} \Rightarrow \mathtt{x} \geq \mathtt{5}}{\langle \mathtt{true} \rangle \quad \mathtt{x = 5;} \quad \langle \mathtt{x} \geq \mathtt{5} \rangle}$$

$$\begin{array}{c} \langle \mathtt{true} \rangle \\ \langle \mathtt{5 = 5} \rangle \\ \mathtt{x = 5;} \\ \langle \mathtt{x = 5} \rangle \\ \langle \mathtt{x} \geq \mathtt{5} \rangle \end{array}$$

Sequenzregel

$$\frac{<\varphi>\ \mathbf{P}\ <\psi> \qquad\qquad <\psi>\ \mathbf{Q}\ <\beta>}{<\varphi>\ \mathbf{P}\ \mathbf{Q}\ <\beta>}$$

Bsp:

```
<true>
      x = 5;
      res = x * x + 6;
  <res = 31>
```

```
<true>
<5 = 5>

x = 5;

<x = 5>
<x * x + 6 = 31>

res = x * x + 6;

<res = 31>
```

Bedingungsregel 1

$$\frac{\langle\varphi \wedge B\rangle\ P\ \langle\psi\rangle \qquad\qquad \varphi \wedge \neg B \Rightarrow \psi}{\langle\varphi\rangle\ \texttt{if (B) \{P\}}\ \langle\psi\rangle}$$

Bsp:

```
<true>
    res = y;
    if (x > y) res = x;
<res = max(x,y)>
```

denn:

```
<res = y ∧ x > y >
<x = max(x,y)>
    res = x;
<res = max(x,y)>
```

und

```
<res = y ∧ ¬ x > y >
⇒ <res = max(x,y)>
```

```
<true>
<y = y>

res = y;

<res = y>

if (x > y) res = x;

<res = max(x,y)>
```

Bedingungsregel 2

$$\frac{\langle \varphi \wedge B\rangle\ \mathbf{P}\ \langle\psi\rangle \qquad\qquad \langle \varphi \wedge \neg B\rangle\ \mathbf{Q}\ \langle\psi\rangle}{\langle\varphi\rangle\ \texttt{if (B) \{P\} else \{Q\}}\ \langle\psi\rangle}$$

Bsp:

```
<true>
        if (x < 0)
            res = -x;
        else
            res = x;
<res = |x|>
```

denn:

```
<x < 0>
<-x = |x|>

    res = -x;

<res = |x|>
```

und

```
< ¬ x < 0>
<x = |x|>

    res = x;

<res = |x|>
```

Schleifenregel

$$\frac{\langle \varphi \wedge B \rangle \; \mathtt{P} \; \langle \varphi \rangle}{\langle \varphi \rangle \; \mathtt{while\ (B)\ \{P\}} \; \langle \varphi \wedge \neg B \rangle}$$

```
<true>
     i = n; res = 1;
<i = n ∧ res = 1>
<i! * res = n!>
     while (i > 1) {res = res * i; i = i - 1; }
<i! * res = n! ∧ ¬ i > 1>
<res = n! >
```

φ ist Schleifen-invariante

denn:
```
<i! * res = n! ∧ i > 1>
<(i-1)! * (res * i) = n!>
     res = res * i;
     i = i - 1;
<i! * res = n!>
```

Hoare-Kalkül

- Zuweisungsregel

$$\frac{}{\langle \varphi\,[x/t] \rangle \; \texttt{x = t;} \; \langle \varphi \rangle}$$

- Konsequenzregeln

$$\frac{\langle \varphi \rangle \; P \; \langle \psi \rangle \qquad \alpha \Rightarrow \varphi}{\langle \alpha \rangle \; P \; \langle \psi \rangle}$$

$$\frac{\langle \varphi \rangle \; P \; \langle \psi \rangle \qquad \psi \Rightarrow \beta}{\langle \varphi \rangle \; P \; \langle \beta \rangle}$$

- Sequenzregel

$$\frac{\langle \varphi \rangle \; P \; \langle \psi \rangle \qquad \langle \psi \rangle \; Q \; \langle \beta \rangle}{\langle \varphi \rangle \; P \; Q \; \langle \beta \rangle}$$

- Bedingungsregeln

$$\frac{\langle \varphi \wedge B \rangle \; P \; \langle \psi \rangle \qquad \varphi \wedge \neg B \Rightarrow \psi}{\langle \varphi \rangle \; \texttt{if (B) \{P\}} \; \langle \psi \rangle}$$

$$\frac{\langle \varphi \wedge B \rangle \; P \; \langle \psi \rangle \qquad \langle \varphi \wedge \neg B \rangle \; Q \; \langle \psi \rangle}{\langle \varphi \rangle \; \texttt{if (B) \{P\} else \{Q\}} \; \langle \psi \rangle}$$

- Schleifenregel

$$\frac{\langle \varphi \wedge B \rangle \; P \; \langle \varphi \rangle}{\langle \varphi \rangle \; \texttt{while (B) \{P\}} \; \langle \varphi \wedge \neg B \rangle}$$

Fakultät mit Assertions

```
public static void main (String [] arguments) {
     int n = IO.eingabe(), i, res;

     assert n == n;

     i = n;

     assert i == n;
     assert i == n && 1 == 1;

     res = 1;

     assert fac(i) * res == fac(n);

     while (i > 1) {
          assert fac(i) * res == fac(n) && i > 1;
          assert fac(i-1) * (res * i) == fac(n);

          res = res * i;

          assert fac(i-1) * res == fac(n);

          i = i - 1;

          assert fac(i) * res == fac(n);
     }
     assert fac(i) * res == fac(n) && !(i > 1);
     assert res == fac(n);

     System.out.println("Die Fakultät ist " + res);
 }
```

Terminierung

Für jede Schleife **`while (B) {P}`** finde einen **`int`**-Ausdruck V (*Variante* der Schleife), so dass:

$B \Rightarrow V \geq 0$ und $\langle V = m \wedge B \rangle$ **P** $\langle V < m \rangle$

```
while (i > 1) {res = res * i; i = i - 1; }
```

Variante ist **`i`**, denn: $i > 1 \Rightarrow i \geq 0$

$\langle i = m \wedge i > 1 \rangle$

$\langle i-1 < m \rangle$

```
res = res * i; i = i - 1;
```

$\langle i < m \rangle$

Verifikation der Addition

```
public class Plus {

 public static void main (String [] args) {
  System.out.print ("Gib 2 Zahlen ein: ");
  int a = IO.eingabe (), b = IO.eingabe (), x, res;
```

Vorbedingung: **a ≥ 0**

```
  x = a;
  res = b;

 //Invariante: x ≥ 0 ∧ x + res = a + b
 //Variante: x

  while (x > 0) {
     x = x - 1;
     res = res + 1;
  }
```

Nachbedingung: **res = a + b**

```
  System.out.println (a + " + " + b + " = " + res);
}}
```

Verifikation der Subtraktion

```
public class Subtract {

 public static void main (String [] args) {
  System.out.print ("Gib 2 Zahlen ein: ");
  int x = IO.eingabe (), y = IO.eingabe (), z, res;

  z = y;
  res = 0;

 //Invariante: x ≥ z ∧ res = z - y
 //Variante: x - z

  while (x > z) {
     z = z + 1;
     res = res + 1;
  }

  System.out.println (x + " - " + y + " = " + res);
}}
```

Vorbedingung: $x \geq y$

Nachbedingung: $res = x - y$

Verifikation eines Primzahl-Programms

```
public class Prim {
 public static void main (String [] args) {
  System.out.print ("Gib Zahl ein: ");
  int n = IO.eingabe ();
  int wurzel = (int) Math.sqrt (n),
      teiler = 2;
  boolean istPrim = true;

 //Invariante: n ≥ 2 ∧ wurzel = ⌊sqrt(n)⌋ ∧
 //       istPrim = keine Zahl i mit 2 ≤ i < teiler teilt n
 //Variante: wurzel - teiler

  while (teiler <= wurzel) {
        if (n % teiler == 0)  istPrim = false;
        teiler = teiler + 1; }

  System.out.println (n + " prim: " + istPrim);
}}
```

Vorbedingung: $n \geq 2$

Nachbedingung: **istPrim = true** gdw. **n** ist Primzahl