

---

# 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 = SimpleIO.getInt("Gib Zahl ein"), i, res;  
    <true>  
    <n=n>  
    i = n;  
    <i=n>  
    <i=n ∧ 1=1>  
    res = 1;  
    <i=n ∧ res=1>  
    <i! · res = n!>  
    while (i > 1) {  
    <i! · res = n! ∧ i>1>  
    <(i-1)! · res · i = n!>  
        res = res * i;  
    <(i-1)! · res = n!>  
        i = i - 1;  
    <i! · res = n!>  
    }  
    <i! · res = n! ∧ i>1> ← Fakultät v. negativen Zahlen ist 1.  
    <res = n!>  
    SimpleIO.output("Fakultaet ist " + res, "Erg");  
}
```

# 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

# Zuweisungsregel

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

$x$  ist Variable,  $t$  ist Ausdruck (ohne Seiteneffekte),  
 $\varphi [x/t]$  ist  $\varphi$  mit allen  $x$  ersetzt durch  $t$

Bsp:  $\langle 5 = 5 \rangle \quad x = 5; \quad \langle x = 5 \rangle$

$$\langle 5 = 5 \rangle$$
$$x = 5;$$
$$\langle x = 5 \rangle$$

# Konsequenzregel 1 (Stärkere Vorbedingung)

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

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

$$\frac{\langle 5 = 5 \rangle \quad \mathbf{x = 5}; \quad \langle \mathbf{x = 5} \rangle \qquad \text{true} \Rightarrow 5 = 5}{\langle \text{true} \rangle \quad \mathbf{x = 5}; \quad \langle \mathbf{x = 5} \rangle}$$

$\langle \text{true} \rangle$   
 $\langle 5 = 5 \rangle$   
 $\mathbf{x = 5};$   
 $\langle \mathbf{x = 5} \rangle$

# Konsequenzregel 2 (Schwächere Nachbedg.)

$$\frac{\langle \varphi \rangle \text{ P } \langle \psi \rangle \quad \psi \Rightarrow \beta}{\langle \varphi \rangle \text{ P } \langle \beta \rangle}$$

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

$$\frac{\langle \text{true} \rangle \quad \mathbf{x} = 5; \quad \langle \mathbf{x} = 5 \rangle \quad \mathbf{x} = 5 \Rightarrow \mathbf{x} \geq 5}{\langle \text{true} \rangle \quad \mathbf{x} = 5; \quad \langle \mathbf{x} \geq 5 \rangle}$$

$\langle \text{true} \rangle$   
 $\langle 5 = 5 \rangle$   
 $\mathbf{x} = 5;$   
 $\langle \mathbf{x} = 5 \rangle$   
 $\langle \mathbf{x} \geq 5 \rangle$

# Sequenzregel

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

Bsp:  $\langle \text{true} \rangle$

$x = 5;$

$\text{res} = x * x + 6;$

$\langle \text{res} = 31 \rangle$

$\langle \text{true} \rangle$

$\langle 5 = 5 \rangle$

$x = 5;$

$\langle x = 5 \rangle$

$\langle x * x + 6 = 31 \rangle$

$\text{res} = x * x + 6;$

$\langle \text{res} = 31 \rangle$



# Bedingungsregel 1

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

Bsp:  $\langle \text{true} \rangle$

```
res = y;  
if (x > y) res = x;  
 $\langle \text{res} = \max(x, y) \rangle$ 
```

denn:  $\langle \text{res} = y \wedge x > y \rangle$   
 $\langle x = \max(x, y) \rangle$   
res = x;  
 $\langle \text{res} = \max(x, y) \rangle$

und  $\langle \text{res} = y \wedge \neg x > y \rangle$   
 $\Rightarrow \langle \text{res} = \max(x, y) \rangle$

```
 $\langle \text{true} \rangle$   
 $\langle y = y \rangle$   
res = y;  
 $\langle \text{res} = y \rangle$   
  
if (x > y) {  
     $\langle \text{res} = y \wedge x > y \rangle$   
     $\langle x = \max(x, y) \rangle$   
    res = x;  
     $\langle \text{res} = \max(x, y) \rangle$  }  
  
 $\langle \text{res} = \max(x, y) \rangle$ 
```

# Bedingungsregel 2

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

Bsp:  $\langle \text{true} \rangle$

```
if (x < 0)
    res = -x;
else
    res = x;
 $\langle \text{res} = |x| \rangle$ 
```

denn:  $\langle \text{true} \rangle$

```
if (x < 0) {
     $\langle \text{true} \wedge x < 0 \rangle$ 
     $\langle -x = |x| \rangle$ 
    res = -x;
     $\langle \text{res} = |x| \rangle$ 
}
else {
     $\langle \text{true} \wedge \neg x < 0 \rangle$ 
     $\langle x = |x| \rangle$ 
    res = x;
     $\langle \text{res} = |x| \rangle$ 
}
 $\langle \text{res} = |x| \rangle$ 
```

# Schleifenregel

$$\frac{\langle \varphi \wedge B \rangle \quad P \quad \langle \varphi \rangle}{\langle \varphi \rangle \text{ while } (B) \{ P \} \langle \varphi \wedge \neg B \rangle}$$

$\langle \text{true} \rangle$

`i = n; res = 1;`

$\langle i = n \wedge res = 1 \rangle$

$\langle \varphi \rangle$

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

$\langle \varphi \wedge \neg i > 1 \rangle$

$\langle res = n! \rangle$

# Schleifenregel

$$\frac{\langle \varphi \wedge B \rangle \quad P \quad \langle \varphi \rangle}{\langle \varphi \rangle \text{ while } (B) \{ P \} \langle \varphi \wedge \neg B \rangle}$$

```
<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! >
```

$\varphi$  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

$$\frac{}{\langle \varphi [x/t] \rangle \quad \mathbf{x} = \mathbf{t}; \quad \langle \varphi \rangle}$$

## ■ Konsequenzregeln

$$\frac{\langle \varphi \rangle \quad \mathbf{P} \quad \langle \psi \rangle \quad \alpha \Rightarrow \varphi}{\langle \alpha \rangle \quad \mathbf{P} \quad \langle \psi \rangle}$$

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

## ■ Sequenzregel

$$\frac{\langle \varphi \rangle \quad \mathbf{P} \quad \langle \psi \rangle \quad \langle \psi \rangle \quad \mathbf{Q} \quad \langle \beta \rangle}{\langle \varphi \rangle \quad \mathbf{P} \quad \mathbf{Q} \quad \langle \beta \rangle}$$

## ■ Bedingungsregeln

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

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

## ■ Schleifenregel

$$\frac{\langle \varphi \wedge \mathbf{B} \rangle \quad \mathbf{P} \quad \langle \varphi \rangle}{\langle \varphi \rangle \quad \mathbf{while} \quad (\mathbf{B}) \quad \{\mathbf{P}\} \quad \langle \varphi \wedge \neg \mathbf{B} \rangle}$$

# Fakultät mit Assertions

---

```
public static void main (String [] arguments) {  
  
    int n = SimpleIO.getInt("Gib Zahl ein"), 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);  
  
    SimpleIO.output("Fakultaet ist " + res, "Ergebnis");  
}
```

# 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 static void main (String [] args) {  
  
    int a = SimpleIO.getInt("Gib erste Zahl ein"),  
        b = SimpleIO.getInt("Gib zweite Zahl ein"), x, res;
```

$\langle a \geq 0 \rangle$

$\langle a = a \wedge b = b \wedge a \geq 0 \rangle$

```
    x = a;
```

$\langle x = a \wedge b = b \wedge a \geq 0 \rangle$

```
    res = b;
```

$\langle x = a \wedge res = b \wedge a \geq 0 \rangle$

$\langle res = a + b - x \wedge x \geq 0 \rangle$

```
    while (x > 0) {
```

$\langle res = a + b - x \wedge x \geq 0 \wedge x > 0 \rangle$

$\langle res + 1 = a + b - (x - 1) \wedge x - 1 \geq 0 \rangle$

```
        x = x - 1;
```

$\langle res + 1 = a + b - x \wedge x \geq 0 \rangle$

```
        res = res + 1;
```

$\langle res = a + b - x \wedge x \geq 0 \rangle$

```
    }
```

$\langle res = a + b - x \wedge x \geq 0 \wedge \neg x > 0 \rangle$

$\langle res = a + b \rangle$

```
    SimpleIO.output(a + " + " + b + " = " + res, "Ergebnis");
```

```
}
```

Vorbedingung:  $a \geq 0$

a	b	x	res
3	4	3	4
3	4	2	5
3	4	1	6
3	4	0	7

$res = a + b - x$   
 $\wedge x \geq 0$

Nachbedingung:  $res = a + b$



# Verifikation der Addition

```
public static void main (String [] args) {  
  
    int a = SimpleIO.getInt("Gib erste Zahl ein"),  
        b = SimpleIO.getInt("Gib zweite Zahl ein"), x, res;  
  
    x = a;  
  
    res = b;  
    //Invariante:  $x \geq 0 \wedge x + res = a + b$   
    //Variante:  $x$   
    while (x > 0) {  
  
        x = x - 1;  
  
        res = res + 1;  
  
    }  
  
    SimpleIO.output(a + " + " + b + " = " + res, "Ergebnis");  
}
```

Vorbedingung:  $a \geq 0$

Terminierung:

(1)  $x > 0 \Rightarrow x \geq 0$

(2)  $\langle x = m \wedge x > 0 \rangle$

$\langle x - 1 < m \rangle$

$x = x - 1;$

$\langle x < m \rangle$

$res = res + 1;$

$\langle x < m \rangle$

Nachbedingung:  $res = a + b$

# Verifikation der Subtraktion

```
public static void main (String [] args) {

    int  x = SimpleIO.getInt("Gib erste Zahl ein"),
        y = SimpleIO.getInt("Gib zweite Zahl ein"), z, res;
```

$\langle x \geq y \rangle$

$\langle y = y \wedge 0 = 0 \wedge x \geq y \rangle$

```
z = y;
```

$\langle z = y \wedge 0 = 0 \wedge x \geq y \rangle$

```
res = 0;
```

$\langle z = y \wedge res = 0 \wedge x \geq y \rangle$

$\langle res = z - y \wedge x \geq z \rangle$

```
while (x > z) {
```

$\langle res = z - y \wedge x \geq z \wedge x > z \rangle$

$\langle res + 1 = z + 1 - y \wedge x \geq z + 1 \rangle$

```
z = z + 1;
```

$\langle res + 1 = z - y \wedge x \geq z \rangle$

```
res = res + 1;
```

$\langle res = z - y \wedge x \geq z \rangle$

```
}
```

$\langle res = z - y \wedge x \geq z \wedge \neg x > z \rangle$

$\langle res = x - y \rangle$

```
SimpleIO.output(x + " - " + y + " = " + res , "Ergebnis");
```

```
}
```

Vorbedingung:  $x \geq y$

Terminierung:

Variante  $x - z$

(1)  $x > z \Rightarrow$

$x - z \geq 0$

(2)  $\langle x - z = m \wedge x > z \rangle$   
 $\langle x - (z + 1) < m \rangle$

$z = z + 1;$

$\langle x - z < m \rangle$

$res = res + 1;$

$\langle x - z < m \rangle$

x	y	z	res
5	2	2	0
5	2	3	1
5	2	4	2
5	2	5	3

$res = z - y \wedge x \geq z$

Nachbedingung:  $res = x - y$