# Vorlesung Informatik 1 (Wintersemester 2020/2021)

## Kapitel 12: Korrektheit von Algorithmen

Martin Frieb
Johannes Metzger

Universität Augsburg
Fakultät für Angewandte Informatik

25. Januar 2021

# Grundlegende Forderung an Algorithmen - Wiederholung

Eine grundlegende Forderung an einen Algorithmus ist, dass er die vorgegebene Problemstellung löst:

- Oft ist die Problemstellung aber zu vage, um das zu überprüfen
- Deshalb präzisiert und formalisiert man die Problemstellung vor deren Bearbeitung. Eine solche Präzisierung nennt man **Problemspezifikation** (diese hilft dann oft auch bei der Lösung des Problems)

Löst ein Algorithmus die durch eine Problemspezifikation gegebene Problemstellung, so nennt man ihn **korrekt**

**Definition 12.1 (Verifikationsverfahren (siehe auch Def. 10.37))**

Verifikationsverfahren dienen dem Nachweis der Korrektheit eines Algorithmus bzgl. einer Problemspezifikation

**Definition 12.2 (Validierungsverfahren (siehe auch Def. 10.38))**

Validierungsverfahren dienen für den Nachweis der Korrektheit einer Problemspezifikation bzgl. eines realen Problems

# Was bedeutet Korrektheit formal?

## Definition 12.3 (Partielle und totale Korrektheit)

Ein Algorithmus `A` heißt

- **partiell korrekt bzgl. einer Problemspezifikation `S`**, falls gilt: Ist `e` eine gültige Eingabe (d.h ein Element der Menge der Eingabedaten aus `S`) und liefert `A` für `e` eine Ausgabe `a`, so ist `a` eine von `S` erlaubte Ausgabe für `e`
- **total korrekt bzgl. einer Problemspezifikation `S`**, falls gilt: `A` ist partiell korrekt und liefert für jede gültige Eingabe `e` eine Ausgabe `a`

Die partielle Korrektheit fordert nicht, dass für alle gültige Eingaben eine Ausgabe geliefert wird, d.h. sie fordert nicht, dass der Algorithmus immer terminiert.

# Wie kann man partielle Korrektheit beweisen?

## Generelle Idee

Gehe wie folgt vor für jede Anweisung im Algorithmus, beginnend mit der ersten Anweisung:

- **Vorbedingung formulieren**: Formuliere eine Zusicherung, die vor der Anweisung erfüllt ist
- **Nachbedingung formulieren**: Folgere aus der Art der Anweisung und der Vorbedingung eine Zusicherung, die nach der Anweisung erfüllt ist
- Die Nachbedingung einer Anweisung dient dann als Vorbedingung für die nachfolgende Anweisung
- **Ziel**: Die Nachbedingung der letzten Anweisung entspricht dem funktionalen Zusammenhang der Ausgabe von der Eingabe gemäß der vorgegebenen Problemspezifikation

## Zusicherung

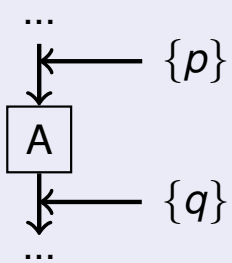
Eine **Zusicherung** ist eine Aussage (formuliert als prädikatenlogische Formel) über die Wertebelegungen der Programmvariablen.

# Zusicherungen in Algorithmusdarstellungen

## Pseudocode

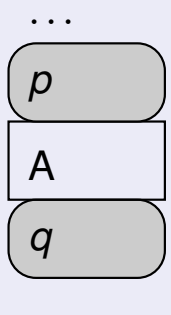
```
1 ...
2 {p}
3 A;
4 {q}
5 ...
```

## Programmablaufplan




- $A$: Anweisung
- $p$: Vorbedingung von $A$
- $q$: Nachbedingung von $A$

## Struktogramm



# Exkurs: Freie und gebundene Variablen in prädikatenlogischen Formeln

Sei $V(A)$ die Menge der in einem Term $A$ auftretenden Variablen. Wir definieren induktiv folgende Mengen für Formeln $p$:

- $FV(p)$: Menge der **freien Variablen** von $p$
- $BV(p)$: Menge der **gebundenen Variablen** von $p$

**Definition 12.4 (Freie und gebundene Variablen)**

- Für $p := (A_1, \ldots, A_n) \in R$ ist $FV(p) := V(A_1) \cup \ldots \cup V(A_n)$ und $BV(p) := \emptyset$
- Für $p := q \wedge r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := q \vee r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := \neg q$ ist $FV(p) := FV(q)$ und $BV(p) := BV(q)$
- Für $p := q \Rightarrow r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := q \Leftrightarrow r$ ist $FV(p) := FV(q) \cup FV(r)$ und $BV(p) := BV(q) \cup BV(r)$
- Für $p := \forall x(q)$ ist $FV(p) := FV(q) \setminus \{x\}$ und $BV(p) := BV(q) \cup \{x\}$
- Für $p := \exists x(q)$ ist $FV(p) := FV(q) \setminus \{x\}$ und $BV(p) := BV(q) \cup \{x\}$

# Exkurs: Substitution in prädikatenlogischen Formeln

Wir definieren induktiv eine Formel $p\{x \to A\}$, die durch **Substitution** einer Variable $x$ in einer Formel $p$ durch einen Term $A$ entsteht.

## Definition 12.5 (Substitution - Terme)

- Es gilt: $x\{x \to A\} := A$
- Für eine Variablen $y$ mit $y \neq x$ gilt: $y\{x \to A\} := y$
- Für Konstante $a$ gilt: $a\{x \to A\} := a$
- Für eine $n$-stellige Operation $op$ und Terme $A_1, \ldots, A_n$ gilt: $op(A_1, \ldots, A_n)\{x \to A\} := op(A_1\{x \to A\}, \ldots, A_n\{x \to A\})$
- Für eine Funktion $f$ und einen Term $B$ gilt: $f(B)\{x \to A\} := f(B\{x \to A\})$

# Exkurs: Substitution in prädikatenlogischen Formeln

Wir definieren induktiv eine Formel $p\{x \to A\}$, die durch **Substitution** einer Variable $x$ in einer Formel $p$ durch einen Term $A$ entsteht.

**Definition 12.6 (Substitution - Formeln)**

- Für $p := (A_1, \ldots, A_n) \in R$ gilt: $p\{x \to A\} := (A_1\{x \to A\}, \ldots, A_n\{x \to A\}) \in R$
- Für $p := q \wedge r$ gilt: $p\{x \to A\} := q\{x \to A\} \wedge r\{x \to A\}$
- Für $p := q \vee r$ gilt: $p\{x \to A\} := q\{x \to A\} \vee r\{x \to A\}$
- Für $p := \neg q$ ist $p\{x \to A\} := \neg q\{x \to A\}$
- Für $p := q \Rightarrow r$ gilt: $p\{x \to A\} := q\{x \to A\} \Rightarrow r\{x \to A\}$
- Für $p := q \Leftrightarrow r$ gilt: $p\{x \to A\} := q\{x \to A\} \Leftrightarrow r\{x \to A\}$
- Für $p := \forall x(q)$ und $r := \exists x(s)$ gilt: $p\{x \to A\} := \forall x(q)$ und $r\{x \to A\} := \exists x(s)$
- Für $p := \forall y(q)$ gilt:
  $p\{x \to A\} := \forall y(q)$, falls $x \notin FV(p)$
  $p\{x \to A\} := \forall y(q\{x \to A\})$, falls $x \in FV(p)$ und $y \notin V(A)$
  $p\{x \to A\} := \forall z(q\{y \to z\}\{x \to A\})$, falls $x \in FV(p)$ und $z \notin V(A) \cup FV(q)$ ($z$ neue Variable)
- Für $p := \exists y(q)$ gelten analoge Definitionen wie für $p := \forall y(q)$.

# Hoare-Kalkül zur Korrektheit von Algorithmen

## Definition 12.7 (Hoare-Tripel)

Ein **Hoare-Tripel** ist eine Aussage der Form

$$\{p\}S\{q\},$$

wobei gilt:

- $S$ ist ein Ausschnitt eines Algorithmus.
- $p$ ist eine prädikatenlogische Formel über die Variablen in $S$, genannt **Vorbedingung von** $S$.
- $q$ ist eine prädikatenlogische Formel über die Variablen in $S$, genannt **Nachbedingung von** $S$.

$\{p\}S\{q\}$ ist **erfüllt**, wenn gilt: Gilt $p$ vor Ausführung von $S$, so gilt $q$ danach, falls $S$ ohne Fehlerabbruch terminiert.

# Hoare-Kalkül zur Korrektheit von Algorithmen

## Definition 12.8 (Zuweisungsregel)

Für einen Term $A$ und eine Formel $p$ ist das Hoare-Tripel

$$\{p\{x \to A\}\}x \leftarrow A;\{p\}$$

erfüllt.

## Beispiel 12.9

- $\{x = a \cdot (i+1)\}i \leftarrow i+1;\{x = a \cdot i\}$

# Hoare-Kalkül zur Korrektheit von Algorithmen

## Definition 12.10 (Konsequenzregel)

Für Formeln $p, q, r, s$ mit $p \Rightarrow q$ und $r \Rightarrow s$ gilt:

$$\{q\}S\{r\} \Rightarrow \{p\}S\{s\}.$$

## Beispiel 12.11

- $\{x \geq 0\}x \leftarrow -x;\{x \leq 0\}$, denn $\{-x \leq 0\}x \leftarrow -x;\{x \leq 0\}$ (Zuweisungsregel) und $x \geq 0 \Rightarrow -x \leq 0$.

# Hoare-Kalkül zur Korrektheit von Algorithmen

## Definition 12.12 (Regel für sequentielle Komposition)

Für Formeln $p, q, r$ gilt:

$$\{p\}S\{q\} \wedge \{q\}T\{r\} \Rightarrow \{p\}ST\{r\}.$$

## Definition 12.13 (Regel für bedingte Anweisungen (Fallunterscheidung))

Für Formeln $p, q, r$ gilt:

$$\{p \wedge q\}S\{r\} \wedge \{p \wedge \neg q\}T\{r\} \Rightarrow \{p\} \text{ Wenn } q \text{ dann } S \text{ sonst } T \{r\}.$$

$q$ ist die Bedingung der Fallunterscheidung

## Definition 12.14 (Regel für wiederholte Anweisungen)

Für Formeln $p, q$ gilt:

$$\{p \wedge q\}S\{p\} \Rightarrow \{p\} \text{ Solange } q \text{ tue } S \{p \wedge \neg q\}.$$

$p$ heißt **Schleifeninvariante**
$q$ ist die Schleifenbedingung

# Hoare-Kalkül zur Korrektheit von Algorithmen

## Definition 12.15 (Regel für wiederholte Anweisungen)

Für Formeln $p, q$ gilt:

$$\{p \wedge q\} S \{p\} \Rightarrow \{p\} \text{ Solange } q \text{ tue } S \{p \wedge \neg q\}.$$

## Korrektheit der Regel für wiederholte Anweisungen

Beweis mit vollständiger Induktion nach der Anzahl $n$ der Schleifendurchläufe.

1. **Induktionsanfang ($n = 0$)**: Wird $S$ keinmal durchlaufen, so ist $\neg q$ erfüllt. Da $p$ per Voraussetzung vor Schleifenbeginn erfüllt ist, gilt insgesamt $p \wedge \neg q$
2. **Induktionsschritt ($n > 0$)**:
   *Induktionsvoraussetzung*: Die Aussage gilt, falls die Schleife nach $n$ Durchläufen terminiert.
   *Zu zeigen*: Die Aussage gilt, falls die Schleife nach $n+1$ Durchläufen terminiert.

   Die Schleife wird mindestens 1-mal durchlaufen. Deshalb ist vor dem ersten Schleifendurchlauf $q$ erfüllt. Außerdem ist per Voraussetzung $p$ vor dem ersten Schleifendurchlauf erfüllt, insgesamt gilt also $p \wedge q$ vor dem ersten Schleifendurchlauf. Per Voraussetzung gilt deshalb $p$ nach dem ersten Schleifendurchlauf. Wegen der Induktionsvoraussetzung gilt dann $p \wedge \neg q$ nach den nächsten $n$ Durchläufen.

# Beispiel für den Nachweis der partiellen Korrektheit

Multiplikation durch Addition

- **Eingabe**: $a, n \in \mathbb{N}_0$
- **Ausgabe**: $x \in \mathbb{N}_0$
- **Funktionaler Zusammenhang**: $x = a \cdot n$

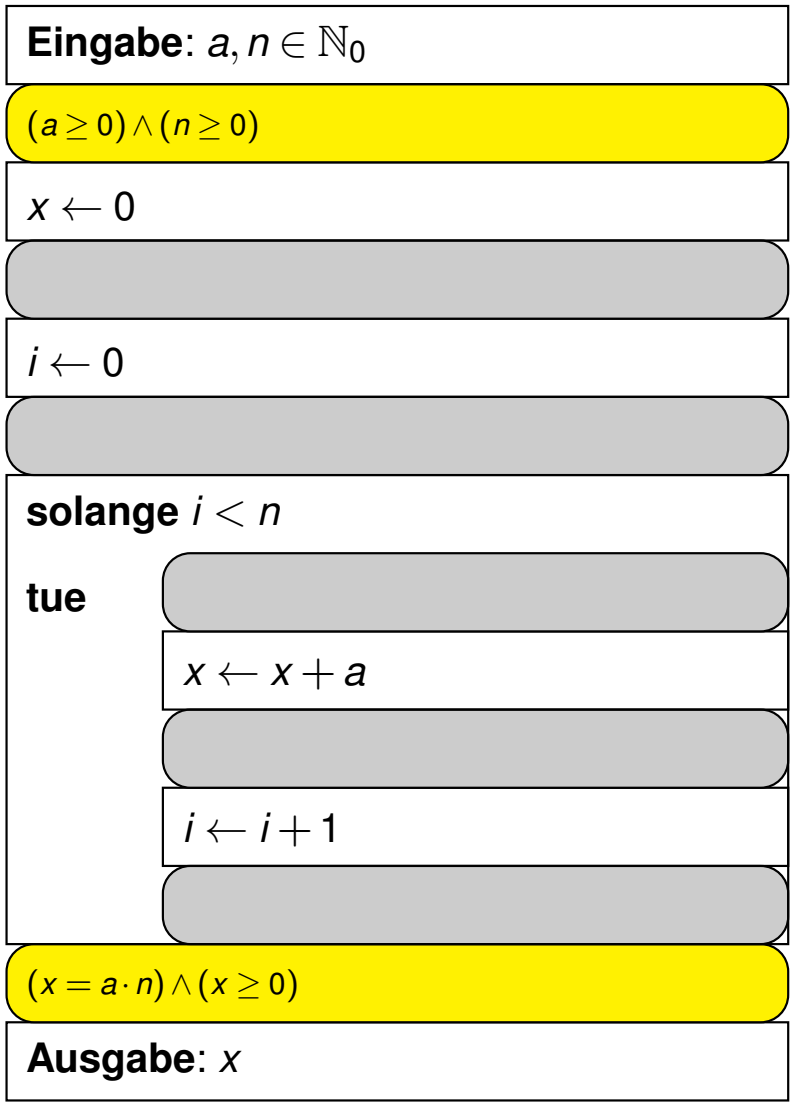
| **Eingabe**: $a, n \in \mathbb{N}_0$ | |
|---|---|
| $x \leftarrow 0$ | |
| $i \leftarrow 0$ | |
| **solange** $i < n$ | |
| **tue** | $x \leftarrow x + a$ |
| | $i \leftarrow i + 1$ |
| **Ausgabe**: $x$ | |

# Beispiel für den Nachweis der partiellen Korrektheit

1. Übernahme aus Problemspezifikation:
   - Eingabe
   - Ausgabe
   - Funktionaler Zusammenhang

Problemspezifikation:

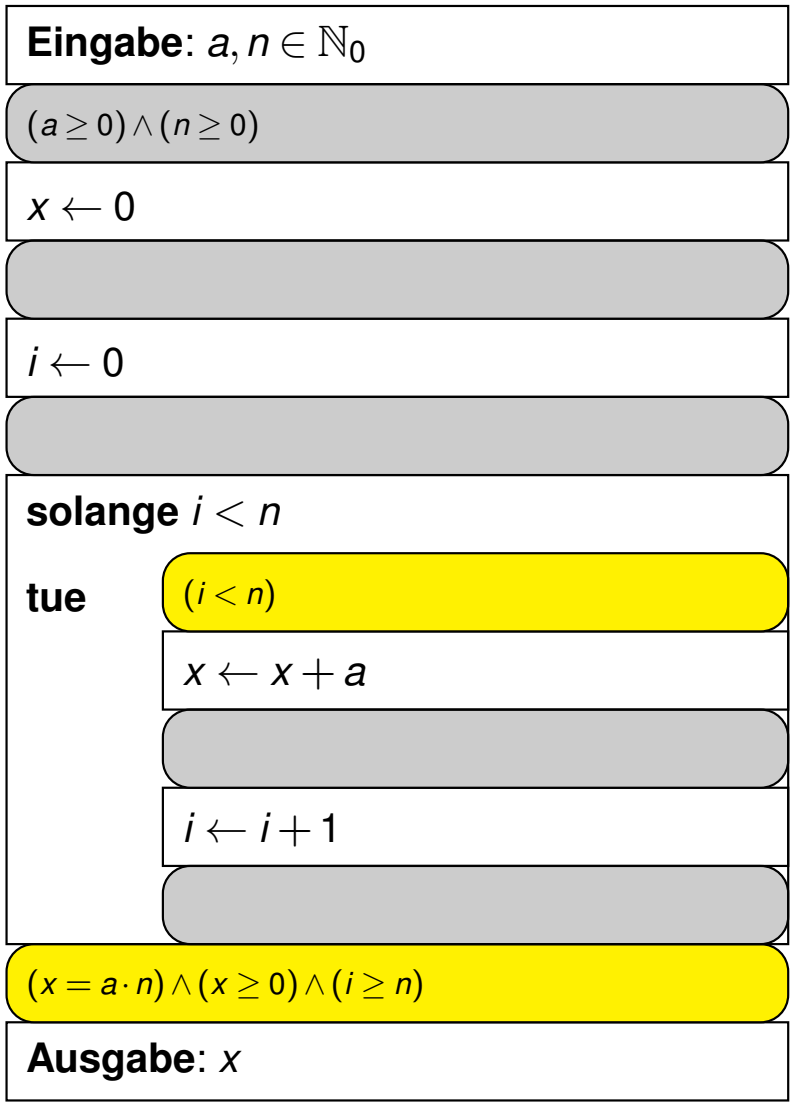
- **Eingabe**: $a, n \in \mathbb{N}_0$
- **Ausgabe**: $x \in \mathbb{N}_0$
- **Funktionaler Zusammenhang**: $x = a \cdot n$



# Beispiel für den Nachweis der partiellen Korrektheit

2. Schleifenbedingung

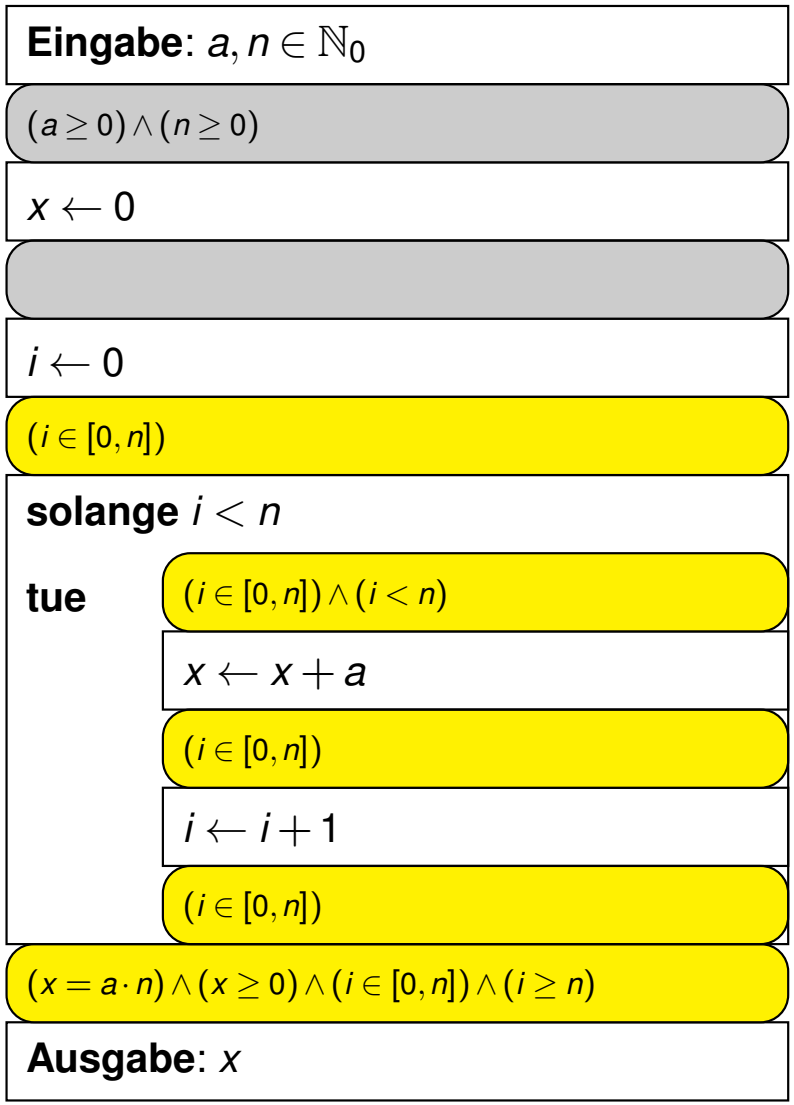
- Schleifenbedingung nach Schleifeneintritt
- negierte Schleifenbedingung nach Verlassen der Schleife



# Beispiel für den Nachweis der partiellen Korrektheit

3. Schleifenzählvariable
   - Vor Schleifeneintritt: $i = 0$
   - Beginn der Schleife: $0 \leq i < n$
   - Ende der Schleife: $0 < i \leq n$
   - Nach der Schleife: $i = n$

Zusammenfassung:
$0 \leq i \leq n \Leftrightarrow i \in [0, n]$



# Beispiel für den Nachweis der partiellen Korrektheit

4. Zuweisungsregel
   - Zuweisungsregel rückwärts: $0 \in [0, n]$
   - Konsequenzregel: $i < n \Rightarrow i + 1 \leq n$
   - Zuweisungsregel: $i \leq n$

**Eingabe**: $a, n \in \mathbb{N}_0$

$(a \geq 0) \wedge (n \geq 0)$

$x \leftarrow 0$

$(0 \in [0, n])$

$i \leftarrow 0$

$(i \in [0, n])$

**solange** $i < n$

**tue**

$(i \in [0, n]) \wedge (i < n)$

$x \leftarrow x + a$

$(i \in [0, n]) \wedge (i + 1 \leq n)$

$i \leftarrow i + 1$

$(i \in [0, n]) \wedge (i \leq n)$

$(x = a \cdot n) \wedge (x \geq 0) \wedge (i \in [0, n]) \wedge (i \geq n)$

**Ausgabe**: $x$

# Beispiel für den Nachweis der partiellen Korrektheit

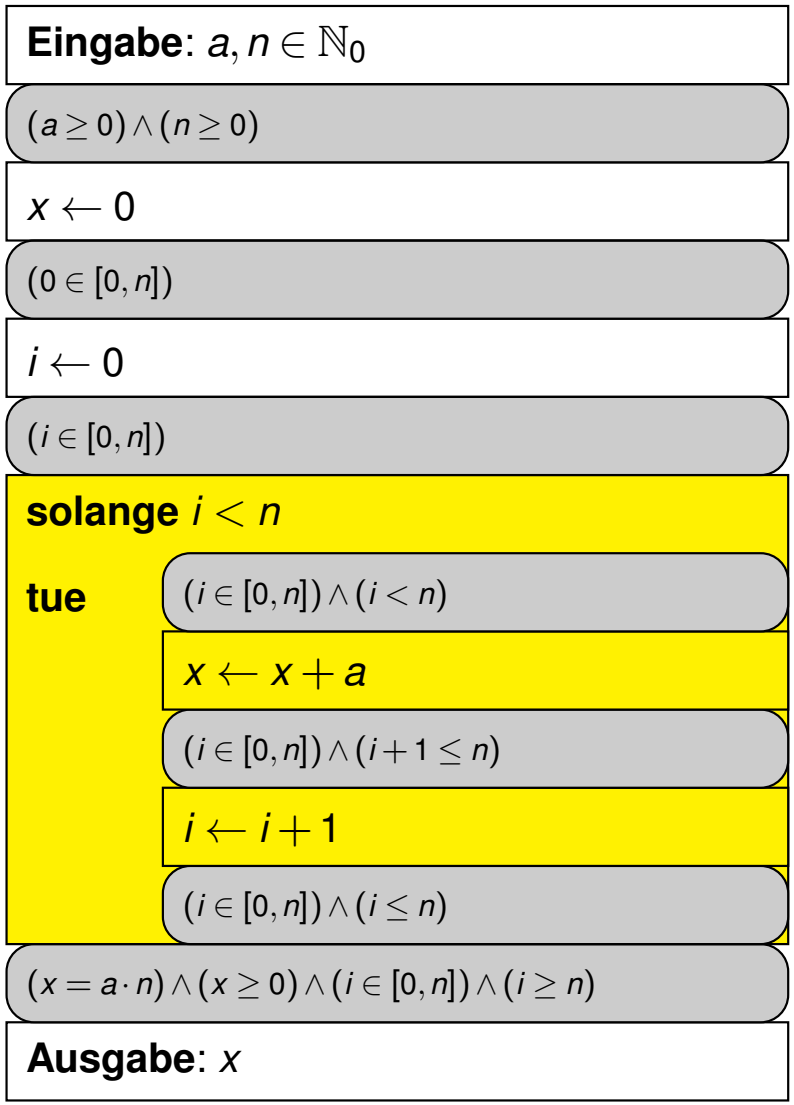
5. Schleifeninvariante

Eingabe: $a = 5$, $n = 2$

| 1. Iteration | 2. Iteration | 3. Iteration |
|---|---|---|
| $0 < 2$ | $1 < 2$ | $2 < 2$ |
| $x \leftarrow 0 + 5$ | $x \leftarrow 5 + 5$ | |
| $i \leftarrow 0 + 1$ | $i \leftarrow 1 + 1$ | |
| | | Ausgabe: 10 |

Ergebnis: $x = a + a + ... + a$

$\Rightarrow x = a \cdot i$



# Beispiel für den Nachweis der partiellen Korrektheit

- Funktionaler Zusammenhang soll sich aus Schleifeninvariante ergeben
- Wieso führen Zwischenergebnisse am Ende zum gewünschten Ergebnis?
- Am Ende gilt: $(i \in [0,n]) \wedge (i \geq n)$
- $\Rightarrow n = i$ (am Ende!)
- Formuliere funktionalen Zusammenhang als $x = a \cdot i$

**Eingabe**: $a, n \in \mathbb{N}_0$

$(a \geq 0) \wedge (n \geq 0)$

$x \leftarrow 0$

$(0 \in [0,n])$

$i \leftarrow 0$

$(i \in [0,n])$

**solange** $i < n$

**tue**

- $(i \in [0,n]) \wedge (i < n)$
- $x \leftarrow x + a$
- $(i \in [0,n]) \wedge (i+1 \leq n)$
- $i \leftarrow i + 1$
- $(i \in [0,n]) \wedge (i \leq n)$

$(x = a \cdot i) \wedge (x \geq 0) \wedge (i \in [0,n]) \wedge (i \geq n)$

**Ausgabe**: $x$

# Beispiel für den Nachweis der partiellen Korrektheit

Schließe rückwärts:
Es gilt auch am Ende der Schleife $x = a \cdot i$

**Eingabe**: $a, n \in \mathbb{N}_0$

$(a \geq 0) \wedge (n \geq 0)$

$x \leftarrow 0$

$(0 \in [0, n])$

$i \leftarrow 0$

$(i \in [0, n])$

**solange** $i < n$

**tue**

$(i \in [0, n]) \wedge (i < n)$

$x \leftarrow x + a$

$(i \in [0, n]) \wedge (i + 1 \leq n)$

$i \leftarrow i + 1$

$(x = a \cdot i) \wedge (i \in [0, n]) \wedge (i \leq n)$

$(x = a \cdot i) \wedge (x \geq 0) \wedge (i \in [0, n]) \wedge (i \geq n)$

**Ausgabe**: $x$

# Beispiel für den Nachweis der partiellen Korrektheit

6. Ausbau der Schleifeninvariante:
   - Zuweisungsregel rückwärts anwenden
   - $x = a \cdot i$ wird rückwärts durch $i \leftarrow i + 1$ zu $x = a \cdot (i + 1)$
   - Fortsetzung mit $x \leftarrow x + a$

**Eingabe**: $a, n \in \mathbb{N}_0$

$(a \geq 0) \wedge (n \geq 0)$

$x \leftarrow 0$

$(0 \in [0, n])$

$i \leftarrow 0$

$(i \in [0, n])$

**solange** $i < n$

**tue**

$x + a = a \cdot (i + 1)) \wedge (i \in [0, n]) \wedge (i < n)$

$x \leftarrow x + a$

$(x = a \cdot (i + 1)) \wedge (i \in [0, n]) \wedge (i + 1 \leq n)$

$i \leftarrow i + 1$

$(x = a \cdot i) \wedge (i \in [0, n]) \wedge (i \leq n)$

$(x = a \cdot i) \wedge (x \geq 0) \wedge (i \in [0, n]) \wedge (i \geq n)$

**Ausgabe**: $x$

# Beispiel für den Nachweis der partiellen Korrektheit

Fortsetzung vor Schleife:

- Zuweisungs-/ Konsequenzregel: $x = a \cdot i$
- Zuweisungsregel rückwärts: $x = a \cdot 0$
- Fortsetzung: $0 = a \cdot 0$
- Schleifeninvariante fertig

**Eingabe**: $a, n \in \mathbb{N}_0$

$(0 = a \cdot 0) \wedge (a \geq 0) \wedge (n \geq 0)$

$x \leftarrow 0$

$(x = a \cdot 0) \wedge (0 \in [0, n])$

$i \leftarrow 0$

$(x = a \cdot i) \wedge (i \in [0, n])$

**solange** $i < n$

**tue**

$(x + a = a \cdot (i + 1)) \wedge (i \in [0, n]) \wedge (i < n)$

$x \leftarrow x + a$

$(x = a \cdot (i + 1)) \wedge (i \in [0, n]) \wedge (i + 1 \leq n)$

$i \leftarrow i + 1$

$(x = a \cdot i) \wedge (i \in [0, n]) \wedge (i \leq n)$

$(x = a \cdot i) \wedge (x \geq 0) \wedge (i \in [0, n]) \wedge (i \geq n)$

**Ausgabe**: $x$

# Beispiel für den Nachweis der partiellen Korrektheit

Aufräumen:

- Von oben nach unten durchgehen
- Nicht benötigte Zusicherungen entfernen
- Schritt ist optional

**Eingabe**: $a, n \in \mathbb{N}_0$

$(0 = a \cdot 0) \wedge (n \geq 0)$

$x \leftarrow 0$

$(x = a \cdot 0) \wedge (0 \in [0, n])$

$i \leftarrow 0$

$(x = a \cdot i) \wedge (i \in [0, n])$

**solange** $i < n$

**tue**

$(x + a = a \cdot (i + 1)) \wedge (i \in [0, n]) \wedge (i < n)$

$x \leftarrow x + a$

$(x = a \cdot (i + 1)) \wedge (i \in [0, n]) \wedge (i + 1 \leq n)$

$i \leftarrow i + 1$

$(x = a \cdot i) \wedge (i \in [0, n])$

$(x = a \cdot i) \wedge (i \in [0, n]) \wedge (i \geq n)$

**Ausgabe**: $x$

# Vorausetzungen für den Nachweis der partiellen Korrektheit

Um formale Zusicherungen formulieren zu können, muss der Algorithmus in einer **formalen programmiersprachenunabhängigen Darstellung** vorliegen, d.h. als Pseudocode, Struktogramm oder Programmablaufplan **ohne natürlichsprachliche Formulierungen**.

## Formale Darstellung

Erlaubt sind mathemetische Operationen und Funktionen:

- Wertzuweisung ($\leftarrow$)
- arithmetische Operationen ($+, -, /, *, \div$, mod )
- Mengenoperationen ($\cap, \cup, \setminus$)
- Vergleiche von mathematischen Objekten ($\leq, \geq, <, >, =, \neq$)
- Logische Operationen ($\neg, \wedge, \vee$)
- Mathematische Funktionen ($x^n, \log_b(x), b^x, \sin(x), \cos(x), \sqrt{x}, |\cdot|, \ldots$)
- ...

# Allgemeines Vorgehen zum Nachweis der partiellen Korrektheit

1. Formuliere Vorbedingung des Algorithmus
   (= Spezifikation der Eingabedaten)
2. Formuliere Nachbedingung des Algorithmus
   (= Spezifikation des funktionalen Zusammenhangs)
3. Finde ausgehend von der Vorbedingung Anweisung für Anweisung passende Zwischenzusicherungen
4. Verliere dabei das Ziel (Nachbedingung des Algorithmus) nicht aus den Augen;
   schließe u.U auch rückwärts
5. Bei der Formulierung von Schleifeninvarianten stelle man sich die Frage: Wieso ergeben die Zwischenergebnisse nach jedem Schleifendurchlauf am Ende das gewünschte Endergebnis?

# Beispiel für den Nachweis der partiellen Korrektheit

- **Eingabe**:
  $a_1 \dots a_n \in A^+$
- **Ausgabe**: $k \in \{0,1\}$
- **Funktionaler Zusammenhang**:
  $((k = 1)$
  $\wedge(\forall i \in [1, n \div 2](a_i = a_{n+1-i})))$
  $\vee((k = 0)$
  $\wedge(\exists i \in [1, n \div 2](a_i \neq a_{n+1-i})))$

(Das Intervall $[1,0]$ entspricht dem **leeren Intervall**)

```
1  Algorithmus : Palindromtest
   Eingabe : a_1 ... a_n ∈ A^+
2  m ← 1;
3  solange (m ≤ n ÷ 2) ∧ (a_m = a_{n+1-m})
     tue
4    m ← m + 1;
5  wenn m > n ÷ 2 dann
6    k ← 1;
7  sonst
8    k ← 0;
   Ausgabe : k
```

# Beispiel für den Nachweis der partiellen Korrektheit

- **Eingabe**:
  $a_1 \dots a_n \in A^+$
- **Ausgabe**: $k \in \{0,1\}$
- **Funktionaler Zusammenhang**:
  $((k=1) \wedge (\forall i \in [1, n \div 2](a_i = a_{n+1-i}))) \vee ((k=0) \wedge (\exists i \in [1, n \div 2](a_i \neq a_{n+1-i})))$

(Das Intervall $[1,0]$ entspricht dem **leeren Intervall**)

**Eingabe :** $a_1 \dots a_n \in A^+$

1 $\{\forall i \in \emptyset(a_i = a_{n+1-i})\}$
2 $m \leftarrow 1;$
3 $\{\forall i \in [1, m-1](a_i = a_{n+1-i})\}$
4 **solange** $(m \leq n \div 2) \wedge (a_m = a_{n+1-m})$ **tue**
5 $\quad \{\forall i \in [1, (m+1)-1](a_i = a_{n+1-i})$
6 $\quad m \leftarrow m+1;$
7 $\quad \{\forall i \in [1, m-1](a_i = a_{n+1-i})\}$
8 $\{\forall i \in [1, m-1](a_i = a_{n+1-i}) \wedge ((m > n \div 2) \vee (a_m \neq a_{n+1-m}))\}$
9 **wenn** $m > n \div 2$ **dann**
10 $\quad \{\forall i \in [1, m-1](a_i = a_{n+1-i}) \wedge (m > n \div 2)\}$
11 $\quad k \leftarrow 1;$
12 $\quad \{\forall i \in [1, n \div 2](a_i = a_{n+1-i}) \wedge (k = 1)\}$
13 **sonst**
14 $\quad \{(m \leq n \div 2) \wedge (a_m \neq a_{n+1-m})\}$
15 $\quad k \leftarrow 0;$
16 $\quad \{(m \leq n \div 2) \wedge (a_m \neq a_{n+1-m}) \wedge (k = 0)\}$
17 $\{(\forall i \in [1, n \div 2](a_i = a_{n+1-i}) \wedge (k = 1))$
18 $\vee ((a_m \neq a_{n+1-m}) \wedge (m \leq n \div 2) \wedge (k = 0))\}$

**Ausgabe :** k

# 12. Korrektheit von Algorithmen

12.1 Partielle Korrektheit

12.2 Totale Korrektheit

# Wie kann man die totale Korrektheit nachweisen?

Für den Nachweis der totalen Korrektheit muss man zusätzlich zur partiellen Korrektheit zeigen:
**Der Algorithmus terminiert für alle (gültigen) Eingaben**

## Generelle Idee

Zeige für jede Schleife mit Hilfe einer **Terminierungsfunktion**, dass diese (für jede Eingabe) nach endlich vielen Durchläufen abbricht.

# Was ist eine Terminierungsfunktion?

**Definition 12.16 (Terminierungsfunktion)**

Sei *Alg* ein Algorithmus und $B(x_1, \ldots, x_n)$ eine Schleifenbedingung von *Alg*, die von den Variablen $x_1, \ldots, x_n$ abhängt.

Für eine Belegung der Variablen $x_1, \ldots, x_n$ mit Werten $a_1, \ldots, a_n$ **vor** einem Schleifendurchlauf sei $a'_1, \ldots, a'_n$ deren Belegung **nach** dem Schleifendurchlauf.

Eine **Terminierungsfunktion** (zu $B$) ist eine Funktion

$$T(x_1, \ldots, x_n) =: y \in \mathbb{Z}$$

mit folgender Eigenschaft: Für jede Belegung $a_1, \ldots, a_n$ von $x_1, \ldots, x_n$, die $B(a_1, \ldots, a_n)$ erfüllt, gilt

- $T(a_1, \ldots, a_n) > 0$
- $T(a_1, \ldots, a_n) > T(a'_1, \ldots, a'_n)$

# Was bedeutet eine Terminierungsfunktion?

Existiert zu einer Schleife eine Terminierungsfunktion $T$, so terminiert die Schleife nach endlich vielen Durchläufen, denn:

- Der Wert von $T$ ist positiv, solange die Schleifenbedingung erfüllt ist.
- Da der Wert von $T$ ganzzahlig ist und mit jedem Schleifendurchlauf abnimmt, muss er nach endlich vielen Schleifendurchläufen 0 oder negativ werden
- Dann ist die Schleifenbedingung nicht mehr erfüllt, d.h. die Schleife terminiert.

# Beispiel für eine Terminierungsfunktion

**Schleife**:

```
1  i ← 1;
2  solange i ≤ 10 tue
3  |  i ← i + 1;
```

**Entwurf Terminierungsfunktion**:

Formuliere Schleifenbedingung so um, dass sie $> 0$ vergleicht

$$i \leq 10 \Leftrightarrow 10 - i \geq 0 \Leftrightarrow 11 - i > 0$$

$\Rightarrow$ Terminierungsfunktion $T(i) := 11 - i$

**Nachweis:**

Für $(i \leq 10)$ gilt:

- $T(i) = 11 - i \geq 11 - 10 = 1 > 0$.
- $T(i) = 11 - i > 11 - (i+1) = 11 - i' = T(i')$.