* Bsp.: Iterative Berechnung der Summe einer endl. Folge reeller Zahlen

1) $Q \equiv \text{Res} = \sum_{j=1}^{n} a_j$

2) $INV \equiv \text{Res} = \sum_{j=1}^{i} a_j \ \wedge\ 0 \le i \le n$

Implementieren des Körpers S:

$i := i + 1;\ \text{Res} := \text{Res} + a_i;$

$wp(S, Q) \equiv wp(i := i + 1 , wp(\text{Res} := \text{Res} + a_i , Q(i, \text{Res})))$

$\equiv wp(i := i + 1 , Q(i, \text{Res} + a_i))$

$\equiv Q(i + 1, \text{Res} + a_{i+1})$

$\Rightarrow h(i, \text{Res}) = [i + 1, \text{Res} + a_{i+1}]$ als Transformation;
garantiert mit der noch zu def. Wiederholbed. B die Invariante;

Bem.: Die umgek. Reihenfolge in S ist falsch, da dort h(V) die Invariante verletzt.

3) $P \equiv \text{Res} = 0 \ \wedge\ i = 0 \ \wedge\ n \ge 0$

$P \Rightarrow INV$?

Antwort: $\text{Res} = 0 = \sum_{j=1}^{0} a_j \ \wedge\ 0 = i \le n \quad \Rightarrow$ Ja!

4) $B \equiv i < n$

5) $INV(V) \wedge B(V) \Rightarrow INV(V')$?

Antwort: $INV(V') \equiv INV(h(i, \text{Res})) \equiv INV(i + 1, \text{Res} + a_{i+1})$

$\equiv \text{Res} + a_{i+1} = \sum_{j=1}^{i} a_j + a_{i+1} = \sum_{j=1}^{i+1} a_j \ \wedge\ 0 \le i + 1 \le n$

$INV(V) \wedge B(V) \equiv \text{Res} = \sum_{j=1}^{i} a_i \ \wedge\ 0 \le i \le n \ \wedge\ i < n$

$\Rightarrow INV(V')$

6) $INV(V) \wedge \neg B(V) \Rightarrow Q(V)$?

Antwort: $\text{Res} = \sum_{j=1}^{i} a_j \ \wedge\ 0 \le i \le n \ \wedge\ i \ge n \equiv \text{Res} = \sum_{j=1}^{i} a_j \ \wedge\ i = n$

$\equiv \text{Res} = \sum_{j=1}^{n} a_j \equiv Q$ ■

→ Schleife ist erfolgreich konstruiert.

$\Rightarrow$ folg. Implementierung

S
```
Res := 0;
   i := 1;
WHILE (i < n) DO BEGIN
                    i := i + 1;
                Res := Res + a_i
              END;
```

Jedoch: In P ist $n \geq 0$ enthalten $\Rightarrow$

```
IF n >= 0
   THEN S
   ELSE   » Fehlermeldung «
```

* Def.: Korrektheit von Prog. unter versch. Aspekten;

- zunächst – immer wenn A. endet, so erwart. Erg.

  Partielle Korrektheit := Ein A. mit Vorbed. P und Nachbed. Q heißt partiell korrekt g. d., wenn jede Eingabe, welche die Vorbed. P erfüllt und für welche die Ausführung des A. terminiert, die Nachbed. Q impliziert.

  z. B.

  $P \equiv k \in \mathbb{N} \wedge a \in \mathbb{N}$
  S : WHILE $(k \text{ MOD } a) > 0$ DO $k := (k+2) \text{ MOD } a$;
  $Q \equiv (k \text{ MOD } a) = 0$

  $\Rightarrow$ Abbruch nur, wenn Q erfüllt!
  Q nicht erfüllbar, sobald k gerade $\wedge$ a ungerade bzw. k ungerade $\wedge$ a gerade für $k \neq a$.

  HA: Beweis!
  Hinweis: gerade Zahl $\rightarrow \exists\, n \geq 0 : k = 2 \cdot n + 1$
  3 Fälle $k = a$; $k < a$; $k > a$;

$\Rightarrow$ einzige Garantie hier: wenn ein Erg. erzeugt wird, so ist es korrekt;

- zweiter wichtiger Aspekt – Terminiertheit

  Totale Korrektheit := Ein A. mit Vorbed. P und Nachbed. Q heißt (total) korrekt g. d., wenn für jede Eingabe, welche die Vorbed. P erfüllt, die Ausführung des A. anhält und die Ausgabe erfüllt die Nachbed. Q.

* Terminiertheit von Schleifen:

grundlegende Technik, die Endlichkeit von Schleifen zu untersuchen

→ beob. Werte, die sich in best. Richtung mit der Anzahl der Schleifendurchläufe ändern;

Bsp.: A: $\mathbb{N}^2 \to \mathbb{N}$ gemäß

$$A(x, y) = \begin{cases} y + 1, & x = 0; \\ A(x - 1, 1), & y = 0; \\ A(x - 1, A(x, y - 1)), & \text{sonst}; \end{cases}$$

modifizierte (große) Ackermann–Funktion;

```
FUNCTION A (x, y : nichtneg.ganzeZahl) : Integer;
    BEGIN
        IF x = 0
          THEN    A := y +1
          ELSE    IF y = 0
                    THEN    A := A(x – 1, 1)
                    ELSE    A := A(x – 1, A(x, y – 1))
    END;
```

Terminiert? Ja, da Bewegung auf Terminierbed. $x = 0$ zu.

HA: Bestimmen Sie $A(2,2)$ und $A(3,2)$!

Endlichkeitsbeweise oft trivial, sofern A. richtig verstanden.
Nun anderes Bsp. – einf. Aussehen, aber nicht zu beweisen (z.Z.) →

Bsp.: F: $\mathbb{N}_+ \to \mathbb{N}_+$ gemäß

$$F(x) = \begin{cases} 1, & x = 1; \\ F(x / 2), & x > 1 \text{ und gerade}; \\ F(3 \cdot x + 1), & x > 1 \text{ und ungerade}; \end{cases}$$

```
FUNCTION F (x : pos.ganzeZahl) : Integer;
    BEGIN
        IF x = 1
          THEN    F := 1
          ELSE    IF (x gerade)
                    THEN    F := F(x DIV 2)
                    ELSE    F := F(3*x + 1)
    END;
```

besser $\Rightarrow$

```
PROGRAM   Trivial;
      VAR   x : Integer;
      BEGIN
            WRITE('Eingabe x: '); READLN(x);
            WHILE x > 1 DO
                  IF NOT ODD(x)
                     THEN     x := x DIV 2
                     ELSE     x := 3*x + 1;
            WRITE('f(x) = ', x); READLN;
      END.
```

Bem.: Graf. Darstellung hilfreich!

<u>* induktives Beweisen:</u>

z.B. McCarthy's sogen. 91–Algorithmus $\rightarrow$

```
FUNCTION MC (x : nichtneg.ganzeZahl) : Integer;
      BEGIN
            IF x > 100
               THEN     MC := x – 10
               ELSE     MC := MC(MC(x + 11))
      END;
```

<u>Induktionsbehauptung:</u>

$$MC(x) = \begin{cases} x - 10 & \text{bei } x > 100; \\ 91 & \text{sonst;} \end{cases}$$

<u>Induktionsanfang:</u>

Für $x > 100 \rightarrow$ trivial;
Also $x = 100 \Rightarrow MC(100) = MC(MC(111)) = MC(101) = 91.$

<u>Induktionsübergang:</u>

Sei für ein $x < 100$ gezeigt, daß $MC(y) = 91$ für $x \le y \le 100$.
Zeigen für $x - 100$.
a) Falls $\overbrace{x + 10}^{(x-1)+11} > 100 \Rightarrow MC(x - 1) = MC(MC(x + 10)) = MC(x)$
$= 91$ lt. Ind.-vor;

b) Falls $x + 10 \leq 100 \Rightarrow MC(x - 1) = MC(MC(x + 10)) = MC(91)$,
da laut Ind.-vor. $MC(y) = 91$ für $y = x + 10 \leq 100$.

Mit $x + 10 \leq 100$ gilt $x \leq 90 \leq 91$ und somit laut Ind.-vor. wieder $MC(91) = 91$. ■

6.1.3. Test

* formale Beweise → nur für kleinere Programme;
meistens Tests; {weil weniger gedankl. Aufwand}

Testziel: Fehler zu entdecken!

Ist Test erfolgreich → Symptom (Spur) eines Fehlers entdeckt;
danach Spurenverfolgung: Fehlersuche;
schließlich: Fehlerbehandlung;

{Fehler – bug; ~suche – debugging}

Programm(-teil) enthält Fehler, wenn es
(i) nicht tut, was es soll;
(ii) tut, was es nicht soll.

* Test := gezieltes und systematisches Erproben eines A. oder A.-teils mit Hilfe ausgewählter Testdaten (Testfälle).

⇒ repräsentat. Eingaben durch A. bearbeitet und die Ausgaben mit den laut Spezifikationen erwarteten Ausgaben verglichen

in Appelrath/Ludewig 4 Korrektheitsebenen:

Ebene 1: Pr. „läuft“ → syntakt. Korr.;
Ebene 2: Pr. liefert für einige ausgew. Eingabewerte korr. Ergebnisse;
Ebene 3: Pr. liefert für irgendwelche Eingabewerte korr. Ergebnisse;
Ebene 4: Pr. liefert für alle zulässigen Eingabewerte korr. Ergebnisse;

z.B.: a) Algor. MULT1 aus § 2.2.1 ⇒
1. $m = 0 \wedge n > 0$ 2. $m = 0 \wedge n = 0$
3. m – groß $\wedge$ n – klein …;
b) Suche ⇒
1. $a \notin F$ 2. $F = \emptyset$
3. $a = a_1 \vee a_n$ 4. $a = a_i$ für $1 < i < n$
⇒ i.w. zu Ebene 2;

zu Ebene 4:

Für völlige Sicherheit ist A. auf ges. Eingabemenge *E* anzuwenden.
→ prakt. unmöglich (vgl. PROGRAM Trivial)
{Austesten unrealisierbar als Suche der Stecknadel im Heuhaufen!}

! Testen kann nur Anwesenheit von Fehlern nachweisen,
nicht aber deren Abwesenheit.

Wichtig dabei:

a) P und Q
b) Einblick in Anwend.-domäne eines A.

„Auswege“:

(i) konstruktive Lösungen wo möglich
(ii) Prinzip der Lokalität (Prozed., Module, Kapsel)
(iii) Parallelität von A.-Entwurf und Korr.-Beweis
(iv) Test von „Klassen“
(v) graf. Darstellung des/der A.

* Aufg.-teilung und Zielsetzung

Programmierer ⟷ Tester
will keine Fehler — sucht Fehler

⇒ 2 versch. Personen; Gegenstand der SWE (SWT)

* Konstruktion von Testdaten (⇒ Testplan)

Spurensuche → unsystematisch oder gezielt
besser
Auswahl der Testfälle unter folg. Gesichtspunkten:

a) Anford. der Spezifikation werden leicht mißverstanden;

b) Grenzfälle sind bei Implementation oft nicht korr. beh.;

c) Fehlersituation i.d.R. weniger sorgfältig bearbeitet als der Normalfall;

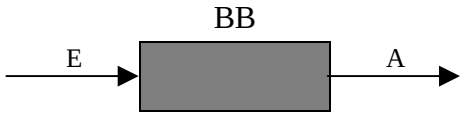
d) Fehler in best. Anw./Verzweig. nur entdeckt, wenn diese durchlaufen:

e) Jede Änderung des Pfades, auf dem ein Pr. durchlaufen wird, kann Fehler offenbaren.

* Auswahlverf. für Testfälle $\Rightarrow$

- Black–Box–Test (Schnittstellentest):

  wenn kein Quellcode verfügbar, so nur an Schnittstellen Test möglich;
  (bzgl. der Spezifikation);



  Spezifikation $\Rightarrow$ $f : \mathrm{E} \rightarrow \mathrm{A}$

  $\{ \mathrm{A}_n = f(\mathrm{E}_n),\ n = 1\ (1)\ \mathrm{N} \} \rightarrow f$ implem. oder nicht?

- White–Box–Test (programmabh. Test):

  Testfälle auch auf Basis des Quelltextes ausgewählt;
  nur hier Aspekte (d) und (e) zu beachten;