Beweisverfahren

Im Folgenden stehen **A** und **B** für Aussagen. Eine Aussage hat den Booleschen Wert **`True`** oder **`False`**. Mit ¬**A** bezeichnen wir die Verneinung der Aussage **A**.

Die Boolesche **`or`**-Verknüpfung wird auch mit dem Symbol $\vee$, die Boolesche **`and`**-Verknüpfung mit dem Symbol $\wedge$ bezeichnet.

Wenn aus „**A** == **`True`**" folgt: „**B** == **`True`**", schreiben wir:
A $\Rightarrow$ **B** (lies: *„aus A folgt B"* oder *„A impliziert B"*)

Die Implikation **A** $\Rightarrow$ **B** ist ebenfalls eine Aussage, die den Wert **`True`** oder **`False`** annehmen kann.

Die Implikation **B** $\Rightarrow$ **A** heißt die Umkehrung der Implikation **A** $\Rightarrow$ **B** .

Falls die Implikationen **A** $\Rightarrow$ **B** und **B** $\Rightarrow$ **A** jeweils den Wert **`True`** haben, heißen die Aussagen **A** und **B** äquivalent: **A** $\Leftrightarrow$ **B** (lies: *„A äquivalent zu B"*)

kurz: **(A** $\Rightarrow$ **B** $\wedge$ **B** $\Rightarrow$ **A)** $\Leftrightarrow$ **(A** $\Leftrightarrow$ **B)**

Merke: **(A** $\Rightarrow$ **B)** $\Leftrightarrow$ **(¬B** $\Rightarrow$ **¬A)**

Beispiel 1

A = „Die Qualifikation im Prüfungsbereich ist erreicht"
B = „Im Prüfungsbereich wurden mindestens 100 Punkte erzielt"

¬**A** = „Die Qualifikation im Prüfungsbereich ist nicht erreicht"
¬**B** = „Im Prüfungsbereich wurden weniger als 100 Punkte erzielt"

Die Implikation **A** $\Rightarrow$ **B** hat den Wahrheitswert **`True`**:

> Wenn die Qualifikation im Prüfungsbereich erreicht ist, wurden im Prüfungsbereich mindestens 100 Punkte erzielt.

Ebenso hat die zu **A** $\Rightarrow$ **B** äquivalente Implikation ¬**B** $\Rightarrow$ ¬**A** den Wahrheitswert **`True`**:

> Wenn im Prüfungsbereich weniger als 100 Punkte erzielt wurden, ist die Qualifikation im Prüfungsbereich nicht erreicht.

Dagegen hat die Umkehrung **B** $\Rightarrow$ **A** den Wahrheitswert **`False`**:
Die Implikation

> Wenn im Prüfungsbereich mindestens 100 Punkte erzielt wurden, ist die Qualifikation im Prüfungsbereich erreicht.

ist falsch!

1. Direkter Beweis

Der direkte Beweis verifiziert unmittelbar die Implikation **A** $\Rightarrow$ **B** .

Beispiel 2 (Satz des Pythagoras):

A = „Das Dreieck ΔABC hat einen rechten Winkel"
B = „Das Quadrat einer der Seiten ist gleich der Summe der Quadrate der beiden anderen Seiten"

oder:

Gegeben: ΔABC
A = „ $\gamma = \angle ACB = 90^\circ$ "
B = „ $a^2 + b^2 = c^2$ "

Dann gilt der Satz des Pythagoras:

Die Implikationen **A** $\Rightarrow$ **B** und **B** $\Rightarrow$ **A** sind jeweils wahr, die Aussagen **A** und **B** sind somit äquivalent: **A** $\Leftrightarrow$ **B**

In Worten:

SATZ: Ein Dreieck ΔABC hat einen rechten Winkel genau dann, wenn das Quadrat einer der Seiten gleich der Summe der Quadrate der beiden anderen Seiten ist. Dabei liegt der rechte Winkel der größten Seite gegenüber.

Beispiel 3

Gegeben sei eine differenzierbare Funktion f.
A = „Die Funktion f nimmt an der Stelle $x=x_E$ ein lokales Extremum an"
B = „$f'(x_E) = 0$"

Dann gilt: Die Implikation **A** $\Rightarrow$ **B** ist wahr.

In Worten:

SATZ: Wenn die Funktion f an der Stelle $x=x_E$ ein lokales Extremum annimmt, folgt: $f'(x_E) = 0$.

Äquivalente Formulierung:
Wegen **(A** $\Rightarrow$ **B)** $\Leftrightarrow$ **(** $\neg$**B** $\Rightarrow$ $\neg$**A)** läßt sich der vorstehende Satz auch wie folgt formulieren:

SATZ: Wenn $f'(x_E) \neq 0$, folgt:
Die Funktion f nimmt an der Stelle $x=x_E$ kein Extremum an.

Die Bedingung $f'(x_E) = 0$ ist für die Existenz eines lokalen Extremums an der Stelle $x=x_E$ zwar notwendig, aber keineswegs hinreichend, wie folgendes Beispiel zeigt:

Die Ableitung der differenzierbaren Funktion $f(x) = x^3$ verschwindet an der Stelle x=0, aber f nimmt an der Stelle x=0 ein Extremum nicht an (vielmehr hat der Graph von f an der Stelle x=0 einen Wendepunkt mit waagerechter Tangente).

Die Umkehrung **B** $\Rightarrow$ **A** ist – wegen vorstehenden Gegenbeispiels – somit falsch!

2. Indirekter Beweis

Anstatt die Implikation **A** $\Rightarrow$ **B** zu verifizieren, verifiziert man die äquivalente Implikation $\neg$**B** $\Rightarrow$ $\neg$**A** .

Beispiel 4

Definition: Eine Zahl x heißt rational, wenn sie sich als Bruch darstellen läßt, andernfalls heißt x irrational.

SATZ: $\sqrt{2}$ ist irrational.

Präzisere Formulierung vorstehender Behauptung:

SATZ: Wenn das Quadrat einer Zahl x den Wert 2 hat, folgt: x ist irrational.

A = „ x ist eine Zahl mit $x^2 = 2$ "
B = „ x ist irrational "
$\neg$**A** = „ x ist eine Zahl mit $x^2 \neq 2$ "
$\neg$**B** = „ x ist rational "

Anstatt $\mathbf{A} \Rightarrow \mathbf{B}$ zu verifizieren, verifizieren wir die äquivalente Implikation $\neg\mathbf{B} \Rightarrow \neg\mathbf{A}$:

SATZ: Wenn x rational ist, folgt: x^2 kann den Wert 2 nicht annehmen.

Beweis:

Annahme: x ist rational, und x^2 hat den Wert 2.

x rational $\Rightarrow$ Es gibt ganze Zahlen p und q, $q \neq 0$, mit $x = p/q$;
oBdA setzen wir voraus, daß p und q teilerfremd sind, daß der Bruch p/q also gekürzt ist.

$\Rightarrow$ $x^2 = p^2/q^2$

$\Rightarrow$ $2 = p^2/q^2$

$\Rightarrow$ $2\,q^2 = p^2$

$\Rightarrow$ p^2 ist gerade

$\Rightarrow$ p ist gerade

$\Rightarrow$ Es gibt eine ganze Zahl k mit $p = 2k$

$\Rightarrow$ $2\,q^2 = (2k)^2$

$\Rightarrow$ $2\,q^2 = 4\,k^2$

$\Rightarrow$ $q^2 = 2\,k^2$

$\Rightarrow$ q^2 ist gerade

$\Rightarrow$ q ist gerade

$\Rightarrow$ Es gibt eine ganze Zahl m mit $q = 2m$

$\Rightarrow$ p und q sind gerade, haben jeweils den Teiler 2; p und q sind also entgegen der Voraussetzung nicht teilerfremd.

Damit haben wir einen Widerspruch zu der Annahme konstruiert, daß x ein gekürzter Bruch ist und daß das Quadrat von x den Wert 2 hat; somit gibt es keine rationale Zahl, deren Quadrat den Wert 2 hat.

Beispiel 5

Die Informatik kennt folgende als **Halteproblem** bezeichnete Problemstellung:

Behauptung: Es gibt kein allgemeines Entscheidungsverfahren, welches als Eingabe den Quelltext eines beliebigen Programms **p** sowie dessen Eingabedaten **x** hat und entscheidet, ob das Programm **p** mit Eingabedaten **x** nach endlich vielen Schritten terminiert oder nicht terminiert.

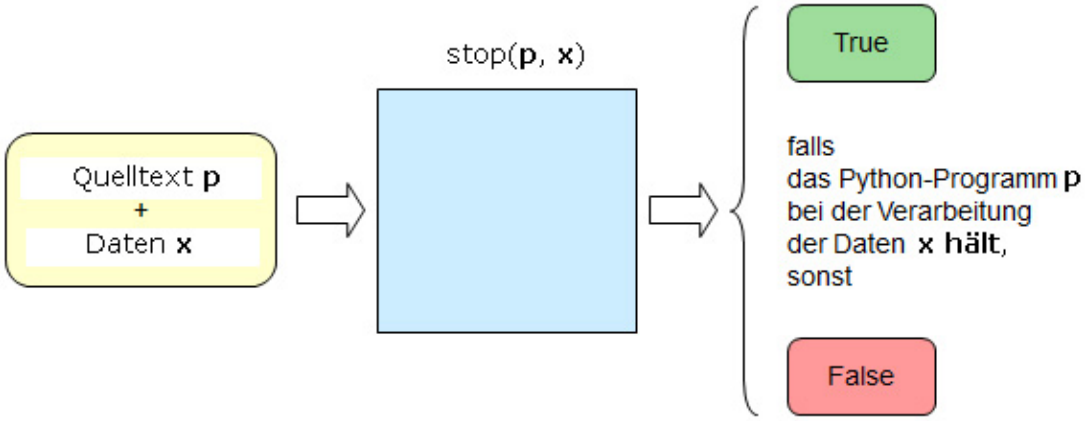
Der Beweis läßt sich indirekt führen, indem man unter der Annahme, daß es ein solches Entscheidungsverfahren gibt, einen Widerspruch zur Annahme herleitet.

Der strenge Beweis zum Halteproblem rekurriert auf das von Alan Turing konzipierte Modell der Turing-Maschine; wir verfolgen hier eine Argumentation, die auf Schulniveau nachvollziehbar ist, und lehnen uns dabei an eine höhere Programmiersprache (hier: Python) an:

Den Algorithmus, der die Entscheidung über die Terminierung eines Programms **p** mit Eingabedaten **x** liefert, formulieren wir als boolesche Funktion **„stop"**, welche als Eingabe das Programm **p** sowie dessen Eingabedaten **x** erhält. **„stop"** liefert den Wert **`True`**, falls **p** angewendet auf **x** terminiert; falls **p** angewendet auf **x** nicht terminiert, liefert **„stop"** den Wert **`False`**.

Die im Quelltext der Funktion **„stop"** benutzte Boolesche Variable **`condition`** nimmt dabei den Wert **`True`** an, falls **p**, angewandt auf **x**, terminiert; andernfalls erhält **`condition`** den Wert **`False`**.

```
def stop(p, x):
      if condition == True: return True
      else:                 return False
```



Das folgende Programm **„strange"** hat als Eingabe ein Programm **p** (formuliert als Quelltext **p**) und benutzt die oben definierte boolesche Funktion **„stop"**.

Insbesondere ist zulässig, daß **„strange"** seinen eigenen Quelltext **p** als Eingabe erhält:

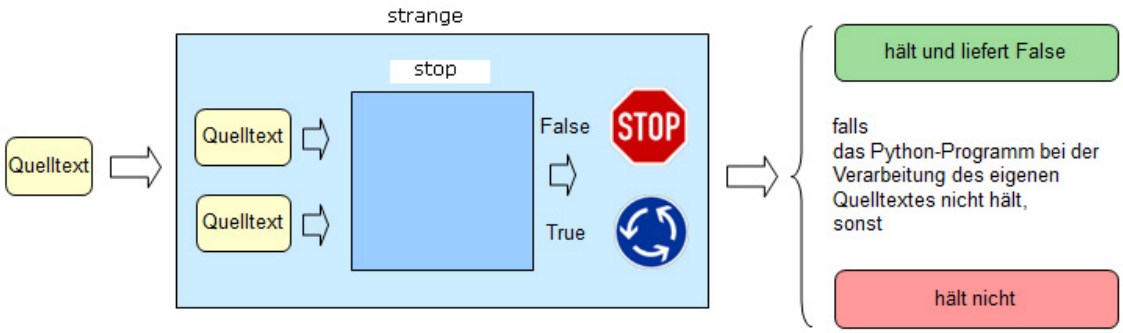
```
# strange

def stop(p, x):
     if condition: return True
     else:         return False

# Eingabe des Quelltextes p
p = input()

if stop(p, p):
    while True: pass

print('strange terminiert')
```



Beachte:
Falls **„stop"** den Wert **`True`** annimmt, gerät **„strange"** in eine Endlosschleife (in Python ist **`pass`** eine leere Anweisung, bei der nichts geschieht), falls **„stop"** den Wert **`False`** erhält, terminiert **„strange"** und gibt den Text „strange terminiert" aus.

Da **„strange"** als allgemeines Verfahren jeden Quelltext **p** akzeptiert, ist es insbesondere zulässig, daß **„strange"** seinen eigenen Quelltext **p** als Eingabedaten erhält, **„strange"** somit auf sich selbst angewendet wird.

Unter der Voraussetzung, daß es eine Boolesche Funktion **`stop(p, x)`** gibt, die entscheidet, ob das Programm **p** mit Eingabedaten **x** terminiert oder nicht terminiert, untersuchen wir folgenden Fälle:

1. Fall:

Annahme:
„strange" terminiert, falls **„strange"** als Eingabe seinen eigenen Quelltext **p** erhält.

⇒ Die Funktion **`stop(p, p)`** nimmt den Wahrheitswert **`True`** an.
⇒ **„strange"** gerät in die Endlosschleife **`while True: pass`**
⇒ **„strange"** terminiert nicht, im Widerspruch zu der Annahme, daß **„strange"** nach Eingabe seines eigenen Quelltexts anhält.

2. Fall:

Annahme:
„strange" terminiert nicht, falls **„strange"** als Eingabe seinen eigenen Quelltext **p** erhält.

⇒ Die Funktion **`stop(p, p)`** nimmt den Wahrheitswert **`False`** an.
⇒ **„strange"** terminiert mit Ausgabe des Text-Strings „strange terminiert", im Widerspruch zu der Annahme, daß **„strange"** nach Eingabe seines eigenen Quelltexts nicht anhält.

Da sich in beiden Fällen ein Widerspruch zur Annahme ergibt, kann es eine Boolesche Funktion **`stop(p, x)`**, die entscheidet, ob **p** mit Eingabedaten **x** terminiert oder nicht terminiert, nicht geben.

3. Das Beweisverfahren „Vollständige Induktion"

Sei **A(n)** eine von der natürlichen Zahl **n** abhängige **Aussage**, $n \in \{0, 1, 2, . \ . \}$.

Um zu beweisen, daß **A(n)** wahr ist für alle $n \in \{0, 1, 2, 3, \ldots . \}$, verifizieren wir:

(1) A(0) ist wahr (Induktionsanfang)

(2) Die Implikation [A(n) $\Rightarrow$ A(n+1)] ist wahr (Induktionsschritt)

Bevor wir dieses Verfahren auf den Korrektheitsbeweis von Algorithmen anwenden, sollten wir es bei einfachen innermathematischen Problemen einüben und verstehen.

Beispiel 6:

Behauptung: $1^2 + 2^2 + 3^2 + \ldots\ldots\ldots + n^2 = n(n+1)(2n+1)/6$

Beweis:

Definiere **A(n) := „$1^2 + \ldots\ldots + n^2 = n(n+1)(2n+1)/6$"**

(Beachte: A(n) ist eine Gleichung, insbesondere also eine Aussage, die genau zwei boolesche Werte annehmen kann: TRUE oder FALSE.)

Induktionsanfang (n=1):

A(1)=TRUE ,
denn
A(1) $\Leftrightarrow$ [$1^2 = 1 \cdot (1+1)(2 \cdot 1+1)/6$] $\Leftrightarrow$ [$1 = 1 \cdot 2 \cdot 3/6$] $\Leftrightarrow$ [1=1]
die letzte Aussage hat trivialerweise den Wert TRUE.

Induktionsschritt:

Unter der Annahme, daß A(n) TRUE ist, verifizieren wir, daß dann auch A(n+1) den Wert TRUE annimmt.

Sei also A(n) TRUE, das heißt

$1^2 + 2^2 + 3^2 + \ldots\ldots\ldots + n^2 = n(n+1)(2n+1)/6$ ist richtig für beliebiges n (diese Annahme heißt auch **Induktionsvoraussetzung**).

Wir betrachten A(n+1), also die Gleichung

$1^2 + 2^2 + \ldots\ldots\ldots + (n+1)^2 = (n+1)[(n+1)+1][2(n+1)+1]/6,$

die wir unter der Annahme, daß A(n) TRUE ist, als TRUE qualifizieren werden.

$$1^2 + 2^2 + \ldots\ldots\ldots + (n+1)^2 = [1^2 + 2^2 + \ldots\ldots + n^2] + (n+1)^2$$

wegen A(n) = TRUE folgt

$$= n(n+1)(2n+1)/6 + (n+1)^2$$
$$= (n+1)[\, n(2n+1)/6 + (n+1)]$$
$$= (n+1)[\, n(2n+1) + 6(n+1)]/6$$

$$= (n + 1)[\, 2n^2+n + 6n + 6)]/6$$
$$= (n + 1)[\, 2n^2+ 7n + 6)]/6$$
$$= (n + 1)[\, (n + 2)(2n + 3)]/6$$
$$= (n + 1)[(n+1) + 1][2(n+1) + 1]/6$$

Somit folgt unter der Annahme „A(n)=TRUE", daß „A(n+1)=TRUE" wahr ist, und in Verbindung mit dem Induktionsanfang „A(1)=TRUE" ergibt sich die Behauptung für alle Werte von n.

Die in den Beispielen 8 und 9 formulierten Problemstellungen bearbeite man als Übungsaufgaben (n sei eine natürliche Zahl):

Beispiel 8:

Behauptung: $1^3 + 2^3 + 3^3 + \ldots\ldots + n^3 = n^2(n+1)^2/4$

Beispiel 9:

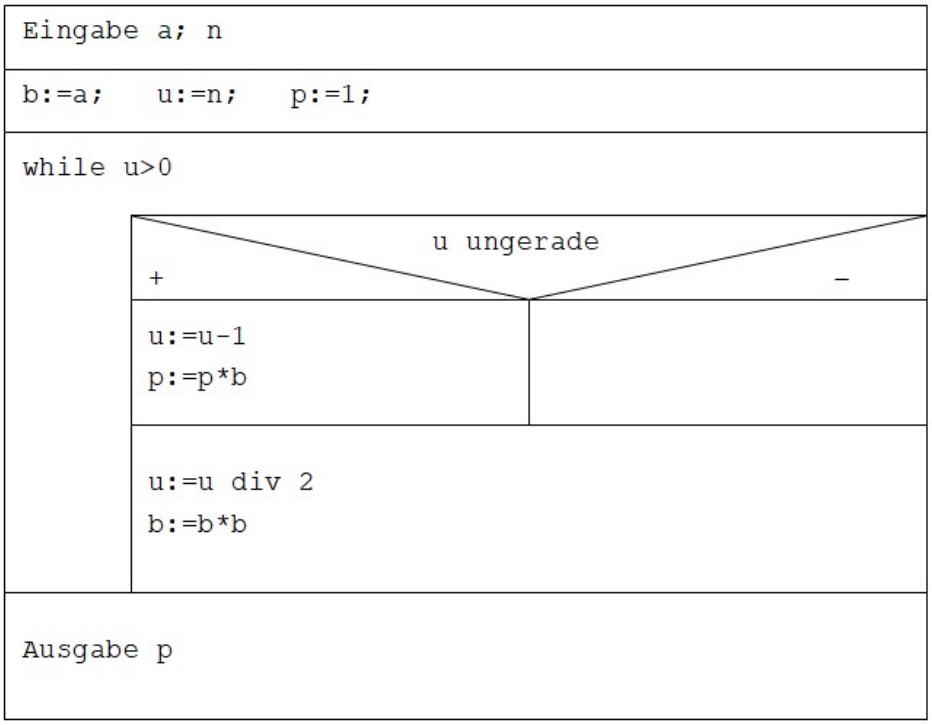
Behauptung: Die Bernoullische Ungleichung $(1 + x)^n > 1 + n \cdot x$

ist wahr für alle natürlichen Zahlen n mit $n \geq 2$ und für reelle Zahlen x mit $x \neq 0$ und $1+x > 0$.

Beispiel 10 (Korrektheitsbeweis für einen Algorithmus)**:**

n sei eine natürliche Zahl (n > 0), und a sei reell mit $a \neq 0$.

Behauptung:
Nach Eingabe von **a** und **n** liefert der folgende als Struktogramm gegebene Algorithmus die Potenz $\mathbf{p = a^n}$.



Hinweis: mit dem Symbol := wird die Wertzuweisung bezeichnet.

Um obenstehende Behauptung zu beweisen, verifizieren wir zunächst:

Die Beziehung

$$\mathbf{p \cdot b^u = a^n}$$

ist vor und nach jedem Schleifendurchlauf erfüllt, also invariant gegenüber Schleifendurchläufen. Eine solche Gleichung heißt auch ***Schleifeninvariante***.

Der Algorithmus bricht ab, sobald u den Wert 0 annimmt; da u bei jedem Schleifendurchlauf um 1 vermindert wird, falls u ungerade ist, in jedem Fall aber durch 2 ganzzahlig dividiert wird, ist die Bedingung u = 0, mit der der Algorithmus abbricht, nach endlich vielen Schleifendurchläufen mit Sicherheit erfüllt.

Für u=0 schreibt sich die Schleifeninvariante:

$$\mathbf{p \cdot b^0 = a^n}$$

$$\Leftrightarrow \qquad \mathbf{p = a^n}$$

Damit ist gezeigt, daß bei Abbruch des Algorithmus die Zahl $\mathbf{a^n}$ ausgegeben wird, falls die Beziehung $\mathbf{p \cdot b^u = a^n}$ sich als Schleifeninvariante erweist.

Wir verifizieren die Behauptung, daß $\mathbf{p \cdot b^u = a^n}$ Schleifeninvariante ist, vermöge **vollständiger Induktion** über den **Index i**, der den i-ten Schleifendurchlauf bezeichnet (i = 1, 2, 3, ...).

Mit $\mathbf{p_i}$, $\mathbf{b_i}$ und $\mathbf{u_i}$ bezeichnen wir die Werte der Variablen **p** , **b** und **u** vor dem i-ten Schleifendurchlauf.

Induktionsanfang (i=1):

Vor dem 1. Schleifendurchlauf gilt wegen $p_1 = 1$, $b_1 = a$ und $u_1 = n$:

$p_1 \cdot b_1^{u_1} = 1 \cdot a^n = a^n$, somit ist die Beziehung $\mathbf{p \cdot b^u = a^n}$ für i=1 erfüllt.

Induktionsschritt:

Wir nehmen an, daß die Beziehung $\mathbf{p \cdot b^u = a^n}$ vor dem i-ten Schleifendurchlauf erfüllt ist, daß also gilt:

$p_i \cdot b_i^{u_i} = a^n$ (*)

Wir verifizieren, daß unter der Induktionsannahme (*) die Beziehung $\mathbf{p \cdot b^u = a^n}$ auch nach dem i-ten, also vor dem (i + 1)-ten Schleifendurchlauf erfüllt ist.

Dazu drücken wir die Werte $\mathbf{p_{i+1}}$, $\mathbf{b_{i+1}}$ und $\mathbf{u_{i+1}}$ der Variablen **p** , **b** und **u** durch die Werte $\mathbf{p_i}$, $\mathbf{b_i}$ und $\mathbf{u_i}$ aus. Da die Eigenschaft von u, gerade oder ungerade zu sein, auf die Berechnung der neuen Werte von **p** , **b** und **u** Einfluß hat, müssen wir eine Fallunterscheidung vornehmen:

α) u sei ungerade vor dem i-ten Schleifendurchlauf, also odd(u_i) = TRUE .

$$p_{i+1} = p_i \cdot b_i \quad \Leftrightarrow \quad p_i = p_{i+1} / b_i$$
$$b_{i+1} = b_i \cdot b_i \quad \Leftrightarrow \quad b_i = \sqrt{b_{i+1}}$$
$$u_{i+1} = (u_i - 1)/2 \quad \Leftrightarrow \quad u_i = 2 \cdot u_{i+1} + 1$$

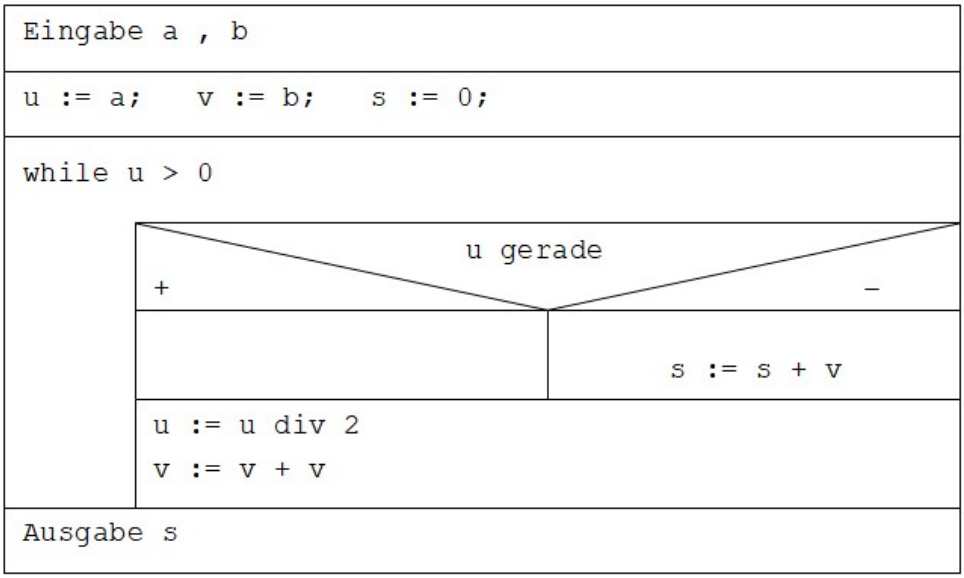
Wenn wir in die Gleichung (*) die für p_i , b_i und u_i erhaltenen Werte einsetzen, folgt (beachte die Schreibweise b^u = b^u):

$$\begin{aligned} a^n &= p_i \cdot b_i^{u_i} \\ &= (p_{i+1} / b_i) \cdot (\sqrt{b_{i+1}})^{(2 \cdot u_{i+1} + 1)} \\ &= (p_{i+1} / \sqrt{b_{i+1}}) \cdot (\sqrt{b_{i+1}})^{(2 \cdot u_{i+1} + 1)} \\ &= p_{i+1} \cdot b_{i+1}^{u_{i+1}} \end{aligned}$$

β) u sei gerade vor dem i-ten Schleifendurchlauf, also odd(u_i) = FALSE .
Übungsaufgabe!

Beispiel 11
(Korrektheitsbeweis für den Algorithmus **„Ägyptische Multiplikation"**)**:**

Der folgende als Struktogramm gegebene Algorithmus verlangt natürliche Zahlen **a** und **b** als Eingabe und liefert das Ergebnis **s**:



a) Begründe: Der Algorithmus terminiert für alle zulässigen Eingabewerte.

b) Verifiziere mit dem Beweisverfahren der Vollständigen Induktion:
Die Beziehung

$$s + u \cdot v = a \cdot b$$

ist Schleifeninvariante.

c) Folgere: Der Algorithmus liefert mit **s** das Produkt der Eingabewerte **a** und **b**.

Bemerkung: *Dieser Algorithmus benutzt die Verdopplung und die ganzzahlige Division durch 2 als wesentliche Rechenoperationen. Diese Rechenoperationen lassen sich im Dualsystem besonders einfach realisieren: Durch einen „leftshift" um eine Stelle wird eine Dualzahl verdoppelt, mit einem „rightshift" um eine Stelle wird eine Dualzahl ganzzahlig halbiert.*
leftshift: 00011101 → 00111010 (dezimal: 29 + 29 = 58)
rightshift: 00100111 → 00010011 (dezimal: 39 div 2 = 19)