

6 DER BEGRIFF DES ALGORITHMUS

MUHAMMAD IBN MŪSĀ AL-KHWĀRIZMĪ lebte ungefähr von 780 bis 850. Abbildung 6.1 zeigt eine (relativ beliebte weil copyright-freie) Darstellung auf einer russischen Briefmarke anlässlich seines (jedenfalls ungefähr) 1200. Geburtstages. Im Jahr 830 (oder wenig früher) schrieb al-Khwārizī ein wichtiges Buch mit dem



Abbildung 6.1: Muhammad ibn Mūsā al-Khwārizmī; Bildquelle: http://commons.wikimedia.org/wiki/Image:Abu_Abdullah_Muhammad_bin_Musa_al-Khwarizmi.jpg (4.11.2010)

Titel „Al-Kitāb al-mukhtaṣar fī ḥisāb al-ğabr wa'l-muqābala“. (An anderer Stelle findet man als Titel „Al-Kitāb al-mukhtaṣar fī ḥisāb al-jabr wa-l-muqābala“.) Die deutsche Übersetzung lautet in etwa „Das kurzgefasste Buch zum Rechnen durch Ergänzung und Ausgleich“. Aus dem Wort „al-ğabr“ bzw. „al-jabr“ entstand später das Wort *Algebra*. Inhalt des Buches ist unter anderem die Lösung quadratischer Gleichungen mit einer Unbekannten.

Einige Jahre früher (825?) entstand das Buch, das im Original vielleicht den Titel „Kitāb al-Jam' wa-l-tafrīq bi-ḥisāb al-Hind“ trug, auf Deutsch „Über das Rechnen mit indischen Ziffern“. Darin führt al-Khwārizī die aus dem Indischen stammende Zahl Null in das arabische Zahlensystem ein und führt die Arbeit mit Dezimalzahlen vor. Von diesem Buch existieren nur noch Übersetzungen, zum Beispiel auf Lateinisch aus dem vermutlich 12. Jahrhundert. Abbildung 6.2 zeigt einen Ausschnitt aus einer solchen Übersetzung.

Von diesen Fassungen ist kein Titel bekannt, man vermutet aber, dass er „Algoritmi de numero Indorum“ oder „Algorismi de numero Indorum“ gelautet haben könnte, also ein Buch „des Al-gorism über die indischen Zahlen“. Das „i“ am Ende von „Algorismi“ wurde später fälschlicherweise als Pluralendung des Wortes

*Herkunft des Wortes
Algorithmus*

Algorithmus angesehen.

Womit wir etwas zur Ethymologie eines der wichtigsten Begriffe der Informatik gesagt hätten.

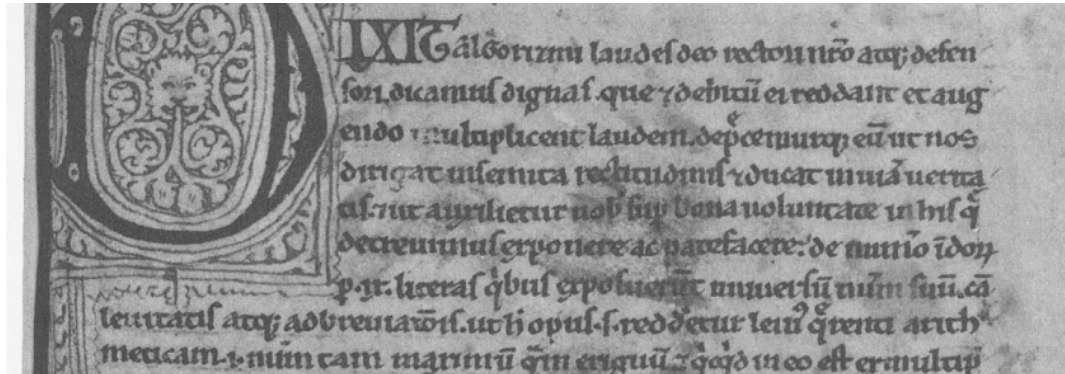


Abbildung 6.2: Ausschnitt einer Seite einer lateinischen Übersetzung der Arbeit von al-Khwārizmī über das „Rechnen mit indischen Ziffern“; Bildquelle: http://en.wikipedia.org/wiki/Image:Dixit_algorizmi.png (4.11.2010)

6.1 LÖSEN EINER SORTE QUADRATISCHER GLEICHUNGEN

Angenommen, man hat eine quadratische Gleichung der Form $x^2 + bx = c$, wobei b und c positive Zahlen sind. Dann, so al-Khwarizmi, kann man die positive Lösung dieser Gleichung bestimmen, indem man nacheinander wie folgt rechnet:

$$h \leftarrow b/2 \quad (6.1)$$

$$q \leftarrow h^2 \quad (6.2)$$

$$s \leftarrow c + q \quad (6.3)$$

$$w \leftarrow \sqrt{s} \quad (6.4)$$

$$x \leftarrow w - h \quad (6.5)$$

(Natürlich ist die Beschreibung der durchzuführenden Rechnungen bei al-Khwarizmi anders.)

Wir haben hier eine Notation gewählt, bei der in jeder Zeile ein Pfeil \leftarrow steht. Links davon steht ein symbolischer Name. Rechts vom Pfeil steht ein arithmetischer Ausdruck, in dem zum einen die beiden Namen b und c für die Eingangs-

größen vorkommen, zum anderen symbolische Namen, die schon einmal in einer *darüberliegenden* Zeile auf der linken Seite vorkamen.

Durch eine solche *Zuweisung* wird die linke Seite zu einem Namen für den Wert, den man aus der rechten Seite ausrechnen kann.

Man kann Schritt für Schritt alle Zuweisungen „ausführen“. Es passieren keine Unglücke (s ist nie negativ.) Man kann sich überlegen, dass am Ende x immer einen Wert bezeichnet, der die quadratische Gleichung $x^2 + bx = c$ erfüllt.

6.2 ZUM INFORMELLEN ALGORITHMUSBEGRIFF

Das gerade besprochene Beispiel besitzt einige Eigenschaften, die man allgemein beim klassischen Algorithmusbegriff fordert:

- Der Algorithmus besitzt eine *endliche Beschreibung* (ist also ein Wort über einem Alphabet).
- Die Beschreibung besteht aus *elementaren Anweisungen*, von denen jede offensichtlich effektiv in einem Schritt ausführbar ist.
- *Determinismus*: Zu jedem Zeitpunkt ist eindeutig festgelegt, welches die nächste elementare Anweisung ist, und diese Festlegung hängt nur ab
 - von schon berechneten Ergebnissen und
 - davon, welches die zuletzt ausgeführte elementare Anweisung war.
- Aus einer *endlichen Eingabe* wird eine *endliche Ausgabe* berechnet.
- Dabei werden *endlich viele Schritte* gemacht, d. h. nur endlich oft eine elementare Anweisung ausgeführt.
- Der Algorithmus funktioniert für *beliebig große Eingaben*.
- Die *Nachvollziehbarkeit/Verständlichkeit* des Algorithmus steht für jeden (mit der Materie vertrauten) außer Frage.

Zwei Bemerkungen sind hier ganz wichtig:

- So plausibel obige Forderungen sind, so informell sind sie aber andererseits: Was soll z. B. „offensichtlich effektiv ausführbar“ heißen? Für harte Beweise benötigt man einen präziseren Algorithmusbegriff.
- Im Laufe der Jahre hat sich herausgestellt, dass es durchaus auch interessant ist, Verallgemeinerungen des oben skizzierten Algorithmusbegriffes zu betrachten. Dazu gehören zum Beispiel
 - randomisierte Algorithmen, bei denen manchmal die Fortsetzung nicht mehr eindeutig ist, sondern abhängig von Zufallsereignissen,
 - Verfahren, bei denen nicht von Anfang an alle Eingaben zur Verfügung stehen, sondern erst nach und nach (z. B. Cacheverwaltung in Prozessoren), und
 - Verfahren, die nicht terminieren (z. B. Ampelsteuerung).

6.3 ZUR KORREKTHEIT DES ALGORITHMUS ZUR LÖSUNG EINER SORTE QUADRATISCHER GLEICHUNGEN

Al-Khwarizmi gibt einen sehr schönen Beweis dafür an, dass die Rechnungen in (6.1)-(6.5) das Ergebnis liefern. Er beruht auf einer geometrischen Überlegung (siehe z. B. <http://www-history.mcs.st-and.ac.uk/~history/Biographies/Al-Khwarizmi.html>, 4.11.2010).

Man kann in diesem konkreten Fall auch einfach den am Ende berechneten Wert z. B. in die linke Seite der Ausgangsgleichung einsetzen und nachrechnen, dass sich als Ergebnis c ergibt. Wir schreiben die fünf Zuweisungen noch einmal auf und fügen nach jeder eine logische Formel ein, die eine gültige Aussage über berechnete Werte macht. Den Formeln ist ein doppelter Aufstrich // vorangestellt, mit dem in Java Kommentare gekennzeichnet werden. Man nennt so etwas auch eine *Zusicherung*:

Zusicherung

```
//  $b > 0 \wedge c > 0$ 
 $h \leftarrow b/2$ 
//  $h = b/2$ 
 $q \leftarrow h^2$ 
//  $q = b^2/4$ 
 $s \leftarrow c + q$ 
//  $s = c + b^2/4$ 
 $w \leftarrow \sqrt{s}$ 
//  $w = \sqrt{c + b^2/4}$ 
 $x \leftarrow w - h$ 
//  $x = \sqrt{c + b^2/4} - b/2$ 
//  $x^2 + bx = (\sqrt{c + b^2/4} - b/2)^2 + b(\sqrt{c + b^2/4} - b/2)$ 
//  $x^2 + bx = c + b^2/4 - b\sqrt{c + b^2/4} + b^2/4 + b\sqrt{c + b^2/4} - b^2/2$ 
//  $x^2 + bx = c$ 
```

Genauer und besser ist es, wenn man zum Verständnis einer Zusicherung nicht alle vorherigen noch einmal studieren muss, sondern nur die unmittelbar vorangehende. Damit dann in unserem Beispiel trotzdem klar ist, was nach der Zuweisung

$x \leftarrow w - h$ der Fall ist, muss man die Information, dass $h = b/2$ ist, Schritt für Schritt mitführen. Das ergibt folgendes Bild:

// $b > 0 \wedge c > 0$

$h \leftarrow b/2$

// $h = b/2$

$q \leftarrow h^2$

// $q = b^2/4 \wedge h = b/2$

$s \leftarrow c + q$

// $s = c + b^2/4 \wedge h = b/2$

$w \leftarrow \sqrt{s}$

// $w = \sqrt{c + b^2/4} \wedge h = b/2$

$x \leftarrow w - h$

// $x = \sqrt{c + b^2/4} - b/2$

// $x^2 + bx = (\sqrt{c + b^2/4} - b/2)^2 + b(\sqrt{c + b^2/4} - b/2)$

// $x^2 + bx = c + b^2/4 - b\sqrt{c + b^2/4} + b^2/4 + b\sqrt{c + b^2/4} - b^2/2$

// $x^2 + bx = c$

6.4 WIE GEHT ES WEITER?

Wir wollen im Laufe der Vorlesung zum Beispiel sehen, wie man zumindest in nicht allzu schwierigen Fällen *allgemein* vorgehen kann, um sich davon zu überzeugen, dass solche Folgen von Rechnungen das richtige Ergebnis liefern. Das gehört dann zum Thema *Verifikation* von Algorithmen.

Dafür benötigt man zumindest die folgenden „Zutaten“:

- einen präzisen Algorithmenbegriff,
- eine präzise Spezifikation des „richtigen Verhaltens“ (bei uns meist der Zusammenhang zwischen gegebenen Eingaben und dazu gehörenden Ausgaben) und
- präzise Methoden, um z. B. zu beweisen, dass das Verhalten eines Algorithmus der Spezifikation genügt.
- Dazu kommt in allen Fällen auch eine präzise Notation, die zumindest bei der Verarbeitung durch Rechner nötig ist.

In einigen der nachfolgenden Einheiten werden wir diese Punkte aufgreifen und in verschiedenen Richtungen weiter vertiefen.

- Präzisierungen des Algorithmusbegriffes gibt es viele, und Sie werden schon in diesem Semester mehrere kennenlernen:
 - Sie kennen inzwischen z. B. Grundzüge einer Programmiersprache. Die ist praktisch, wenn man tatsächlich Algorithmen so aufschreiben will, dass sie ein Rechner ausführen können soll.
 - Die ist aber andererseits unpraktisch, wenn man z. B. beweisen will, dass ein bestimmtes Problem durch keinen Algorithmus gelöst werden kann. Dann sind einfachere Modelle wie Registermaschinen oder Turingmaschinen besser geeignet.
- „Ordentliche“ Notationen dafür, was „das richtige Verhalten“ eines Algorithmus ist, gibt es viele. Wie im vorangegangenen Abschnitt werden wir gleich und in einer späteren Einheit logische Formeln benutzen.
- Dort werden wir auch eine Methode kennenlernen, um zu beweisen, dass ein Algorithmus „das Richtige tut“.
- Präzise Notationen sind nicht nur wichtig, um sich etwas sicherer sein zu können, keine Fehler gemacht zu haben. Sie sind auch unabdingbar, wenn man Aufgaben dem Rechner übertragen will. Und dazu gehören nicht nur „Rechnungen“, sondern z. B. auch
 - die Analyse von Beschreibungen aller Art (z. B. von Programmen oder Ein-/Ausgabe-Spezifikationen) auf syntaktische Korrektheit
 - Beweise (etwa der Korrektheit von Algorithmen bezüglich spezifizierter Anforderungen).

6.5 EIN ALGORITHMUS ZUR MULTIPLIKATION NICHTNEGATIVER GANZER ZAHLEN

Betrachten wir als erstes den in Abbildung 6.3 dargestellten einfachen Algorithmus, der als Eingaben eine Zahl $a \in \mathbb{C}_8$ und eine Zahl $b \in \mathbb{N}_0$ erhält. Variablen X_0 und Y_0 werden mit a und b initialisiert und Variable P_0 mit 0.

Es werden zwei binäre Operationen **div** und **mod** benutzt, die wie folgt definiert sind: Die Operation **mod** liefere für zwei Argumente x und y als Funktionswert $x \bmod y$ den Rest der ganzzahligen Division von x durch y . Und die Operation **div** liefere für zwei Argumente x und y als Funktionswert $x \mathbf{div} y$ den Wert der ganzzahligen Division von x durch y . Mit anderen Worten gilt für nichtnegative ganze Zahlen x und y stets:

$$x = y \cdot (x \mathbf{div} y) + (x \mathbf{mod} y) \quad \text{und} \quad 0 \leq (x \mathbf{mod} y) < y \quad (6.6)$$

```

// Eingaben:  $a \in \mathbb{G}_8, b \in \mathbb{N}_0$ 
 $P_0 \leftarrow 0$ 
 $X_0 \leftarrow a$ 
 $Y_0 \leftarrow b$ 
 $x_0 \leftarrow X_0 \bmod 2$ 
// — Algorithmusstelle —  $i = 0$ 
 $P_1 \leftarrow P_0 + x_0 \cdot Y_0$ 
 $X_1 \leftarrow X_0 \mathbf{div} 2$ 
 $Y_1 \leftarrow 2 \cdot Y_0$ 
 $x_1 \leftarrow X_1 \bmod 2$ 
// — Algorithmusstelle —  $i = 1$ 
 $P_2 \leftarrow P_1 + x_1 \cdot Y_1$ 
 $X_2 \leftarrow X_1 \mathbf{div} 2$ 
 $Y_2 \leftarrow 2 \cdot Y_1$ 
 $x_2 \leftarrow X_2 \bmod 2$ 
// — Algorithmusstelle —  $i = 2$ 
 $P_3 \leftarrow P_2 + x_2 \cdot Y_2$ 
 $X_3 \leftarrow X_2 \mathbf{div} 2$ 
 $Y_3 \leftarrow 2 \cdot Y_2$ 
 $x_3 \leftarrow X_3 \bmod 2$ 
// — Algorithmusstelle —  $i = 3$ 

```

Abbildung 6.3: Ein einfacher Algorithmus für die Multiplikation einer kleinen Zahl a mit einer anderen Zahl b .

Machen wir eine Beispielrechnung für den Fall $a = 6$ und $b = 9$. Für die in Abbildung 6.3 mit dem Wort „Algorithmusstelle“ markierten Zeilen sind in der nachfolgenden Tabelle in jeweils einer Zeile die Werte angegeben, die dann die Variablen P_i , X_i , Y_i und x_i haben. Am Ende erhält man in P_3 den Wert 54. Das ist das Produkt der Eingabewerte 6 und 9.

	P_i	X_i	Y_i	x_i
$i = 0$	0	6	9	0
$i = 1$	0	3	18	1
$i = 2$	18	1	36	1
$i = 3$	54	0	72	0

Im folgenden wollen wir auf einen Beweis hinarbeiten, der zeigt, dass das immer so ist: Unter den Voraussetzungen, die zu Beginn zugesichert werden enthält nach Beendigung des Algorithmus das zuletzt ausgerechnete P_3 den Wert $P_3 = a \cdot b$. Wie geht das?

Da P_3 unter anderem mit Hilfe von P_2 ausgerechnet wird, ist es vermutlich hilfreich auch etwas darüber zu wissen; und über P_1 auch, usw. Analog sollte man am besten etwas über alle x_i wissen sowie über alle X_i und Y_i .

Angenommen, uns gelingt es, „etwas Passendes“ hinzuschreiben, d. h. eine logische Formel, die eine Aussage \mathcal{A}_i über die interessierenden Werte P_i , x_i , X_i und Y_i macht. Was dann? Sie ahnen es vermutlich: vollständige Induktion.

Induktionsbeweise sind am Anfang nicht immer ganz leicht. Aber hier stehen wir vor einem weiteren Problem: Wir müssen erst einmal Aussagen \mathcal{A}_i finden, die wir erstens beweisen können, und die uns zweitens zum gewünschten Ziel führen. Passende Aussagen zu finden ist nicht immer ganz einfach und Übung ist sehr(!) hilfreich. Hinweise kann man aber oft durch Betrachten von Wertetabellen wie weiter vorne angegeben finden. Im vorliegenden Fall liefert ein bisschen Herumspielen irgendwann die Beobachtung, dass jedenfalls im Beispiel für jedes $i \in \mathbb{G}_4$ die folgende Aussage wahr ist:

$$\forall i \in \mathbb{G}_4 : X_i \cdot Y_i + P_i = a \cdot b$$

Das formen wir noch in eine Aussage für alle nichtnegativen ganzen Zahlen um:

$$\forall i \in \mathbb{N}_0 : i < 4 \implies X_i \cdot Y_i + P_i = a \cdot b$$

Wir beweisen nun also durch vollständige Induktion die Formel $\forall i \in \mathbb{N}_0 : \mathcal{A}_i$, wobei \mathcal{A}_i die Aussage ist:

$$i < 4 \implies X_i \cdot Y_i + P_i = a \cdot b .$$

Induktionsanfang $i = 0$: Aufgrund der Initialisierungen der Variablen ist klar:

$$X_0 Y_0 + P_0 = ab + 0 = ab.$$

Induktionsvoraussetzung: für ein beliebiges aber festes i gelte:

$$i < 4 \implies X_i \cdot Y_i + P_i = a \cdot b.$$

Induktionsschluss: zu zeigen ist nun:

$$i + 1 < 4 \implies X_{i+1} \cdot Y_{i+1} + P_{i+1} = a \cdot b.$$

Sei also $i + 1 < 4$ (andernfalls ist die Implikation ohnehin wahr). Dann ist auch $i < 4$ und nach Induktionsvoraussetzung gilt: $X_i \cdot Y_i + P_i = a \cdot b$.

Wir rechnen nun:

$$\begin{aligned} X_{i+1} \cdot Y_{i+1} + P_{i+1} &= (X_i \mathbf{div} 2) \cdot 2Y_i + P_i + x_i Y_i \\ &= (X_i \mathbf{div} 2) \cdot 2Y_i + P_i + (X_i \mathbf{mod} 2) Y_i \\ &= (2(X_i \mathbf{div} 2) + (X_i \mathbf{mod} 2)) Y_i + P_i \\ &= X_i Y_i + P_i \\ &= ab. \end{aligned}$$

Dabei gelten die beiden ersten Gleichheiten wegen der Zuweisungen im Algorithmus, die vierte wegen Gleichung 6.6 und die letzte nach Induktionsvoraussetzung.

Wissen wir nun, dass am Ende des Algorithmus $P = ab$ ist? Offensichtlich noch nicht: Wir haben bisher nur bewiesen, dass nach der letzten Anweisung gilt: $P_3 + X_3 Y_3 = ab$. Der weiter vorne angegebenen Wertetabelle entnimmt man, dass jedenfalls in der Beispielrechnung am Ende $X_3 = 0$ gilt. Wenn es gelingt zu beweisen, dass auch das für *alle* Eingaben $a \in \mathbb{G}_8$ und $b \in \mathbb{N}_0$ gilt, dann sind wir fertig. Wie beweist man das? Wie Sie sich vielleicht schon bewusst gemacht haben, werden die X_i der Reihe nach immer kleiner. Und zwar immer um mindestens die Hälfte, denn $X_i \mathbf{div} 2 \leq X_i/2$. Mit anderen Worten:

$$\begin{aligned} X_0 &\leq a \\ X_1 &\leq X_0/2 \leq a/2 \\ X_2 &\leq X_1/2 \leq a/4 \\ &\vdots \end{aligned}$$

Ein Induktionsbeweis, der so einfach ist, dass wir ihn hier schon nicht mehr im Detail durchführen müssen, zeigt: $\forall i \in \mathbb{N}_0 : i < 4 \implies X_i \leq a/2^i$. Insbesondere ist also $X_2 \leq a/4$. Nun ist aber nach Voraussetzung $a < 8$, und folglich $X_2 < 8/4 = 2$. Da X_2 eine ganze Zahl ist, ist $X_2 \leq 1$. Und daher ist das zuletzt berechnete $X_3 = X_2 \mathbf{div} 2 = 0$.

6.6 DER ALGORITHMUS ZUR MULTIPLIKATION NICHTNEGATIVER GANZER ZAHLEN MIT EINER SCHLEIFE

Nun schreiben wir als den Algorithmus etwas anders auf. Man kann nämlich folgendes beobachten: Wenn man erst einmal der Reihe nach x_{i+1} , P_{i+1} , X_{i+1} und Y_{i+1} berechnet hat, braucht man x_i , P_i , X_i und Y_i nicht mehr. Deswegen kann man die Indizes weglassen, und erhält die vereinfachte Darstellung aus Abbildung 6.4.

Schleife

Nun hat man dreimal genau den gleichen Algorithmustext hintereinander. Dafür führen wir als Abkürzung eine *Schleife* ein, genauer gesagt eine sogenannte **for**-Schleife. Generell schreiben wir das in der folgenden Struktur auf:

```
for  $\langle$ Schleifenvariable $\rangle \leftarrow \langle$ Startwert $\rangle$  to  $\langle$ Endwert $\rangle$  do  
     $\langle$ sogenannter Schleifenrumpf, der  
     $\langle$ aus mehreren Anweisungen bestehen darf $\rangle$   
od
```

Die Bedeutung soll sein, dass der Schleifenrumpf nacheinander für jeden Wert der \langle Schleifenvariable \rangle durchlaufen wird: als erstes für den \langle Startwert \rangle . Bei jedem weiteren Durchlauf werde die \langle Schleifenvariable \rangle um 1 erhöht. Der letzte Durchlauf finde für den \langle Endwert \rangle statt. Außerdem wollen wir vereinbaren, dass der Schleifenrumpf überhaupt nicht durchlaufen werde, falls der angegebene \langle Endwert \rangle echt kleiner ist als der \langle Anfangswert \rangle .

Abbildung 6.5 zeigt, was sich in unserem Beispiel ergibt. Es ist ein besonders einfacher Fall, in dem der Schleifenrumpf gar nicht die Schleifenvariable (hier i) benutzt. Im allgemeinen soll das aber erlaubt sein.

Wir haben am Ende des Algorithmus noch die inzwischen bewiesene Zusage $P = ab$ notiert.

Interessant ist es nun, sich noch einmal klar zu machen, was nach dem Entfernen der Indizes aus den Aussagen $\mathcal{A}_i \equiv P_i + X_i Y_i = ab$ wird: Alle sehen gleich aus: $P + XY = ab$. Inhaltlich war \mathcal{A}_i aber eine Aussage darüber, was „an Algorithmusstelle i “ gilt, also wie wir nun sagen können, nach i Schleifendurchläufen bzw. vor dem $i + 1$ -ten Schleifendurchlauf. Wir hatten bewiesen, dass aus der Gültigkeit von \mathcal{A}_i die von \mathcal{A}_{i+1} folgt. Nach dem Entfernen der Indizes sind aber \mathcal{A}_i und \mathcal{A}_{i+1} identisch. Wir haben also gezeigt:

Wenn die Aussage $P + XY = ab$ vor dem einmaligen Durchlaufen des Schleifenrumpfes gilt, dann gilt sie auch hinterher wieder.

```

// Eingaben:  $a \in \mathbb{G}_8, b \in \mathbb{N}_0$ 
 $X \leftarrow a$ 
 $Y \leftarrow b$ 
 $P \leftarrow 0$ 
 $x \leftarrow X \bmod 2$ 
// — Algorithmusstelle —  $i = 0$ 
 $P \leftarrow P + x \cdot Y$ 
 $X \leftarrow X \operatorname{div} 2$ 
 $Y \leftarrow 2 \cdot Y$ 
 $x \leftarrow X \bmod 2$ 
// — Algorithmusstelle —  $i = 1$ 
 $P \leftarrow P + x \cdot Y$ 
 $X \leftarrow X \operatorname{div} 2$ 
 $Y \leftarrow 2 \cdot Y$ 
 $x \leftarrow X \bmod 2$ 
// — Algorithmusstelle —  $i = 2$ 
 $P \leftarrow P + x \cdot Y$ 
 $X \leftarrow X \operatorname{div} 2$ 
 $Y \leftarrow 2 \cdot Y$ 
 $x \leftarrow X \bmod 2$ 
// — Algorithmusstelle —  $i = 3$ 

```

Abbildung 6.4: Vereinfachter Darstellung des Algorithmus aus Abbildung 6.3.

variante

Man sagt auch, dass diese Aussage eine *Schleifeninvariante* ist.

Der Induktionsanfang war nichts anderes als der Nachweis, dass die Schleifeninvariante vor dem ersten Betreten der Schleife stets gilt. Der Induktionsschritt war der Nachweis, dass die Wahrheit der Schleifeninvariante bei jedem Durchlauf erhalten bleibt.

Wenn also die Schleife jemals zu einem Ende kommt (und etwas anderes ist bei einer **for**-Schleife wie eben beschrieben gar nicht möglich), dann gilt die Schleifeninvariante auch zum Schluss.

```

// Eingaben:  $a \in \mathbb{C}_8, b \in \mathbb{N}_0$ 
 $X \leftarrow a$ 
 $Y \leftarrow b$ 
 $P \leftarrow 0$ 
 $x \leftarrow X \bmod 2$ 
for  $i \leftarrow 0$  to 2 do
    // — Algorithmusstelle —  $i$ 
     $P \leftarrow P + x \cdot Y$ 
     $X \leftarrow X \operatorname{div} 2$ 
     $Y \leftarrow 2 \cdot Y$ 
     $x \leftarrow X \bmod 2$ 
    // — Algorithmusstelle —  $i + 1$ 
od
// Ergebnis:  $P = a \cdot b$ 

```

Abbildung 6.5: Multiplikationsalgorithmus mit einer Schleife für größenbeschränkten Faktor a .

Zum Abschluss wollen wir den Algorithmus nun noch so verallgemeinern, dass er für alle $a \in \mathbb{N}_0$ funktioniert und nicht nur für Zahlen kleiner als 8. Wenn man sich die obigen Beweise alle noch einmal ansieht, stellt man fest, dass die Bedingung $a < 8$ nur an einer Stelle verwendet wurde, nämlich beim Nachweis, dass $X_3 = 0$ ist.

Für einen Anfangswert von z. B. 4711 ist man natürlich nach drei Schleifendurchläufen noch nicht bei 0. Damit man für größere Anfangswerte „irgendwann“ bei 0 angelangt, muss man öfter den Wert X_i halbieren. Es ist also eine größere Anzahl n von Schleifendurchläufen notwendig. Wieviele es sind, sieht man an der schon besprochenen Ungleichung $X_i \leq a/2^i$. Man ist fertig, wenn vor dem letzten Durchlauf gilt: $X_{n-1} \leq 1$. Das ist sicher richtig, wenn $a/2^{n-1} \leq 1$ ist, also $a \leq 2^{n-1}$. Im Fall $a > 0$ gilt das, wenn $n - 1 \geq \log_2 a$. Im Fall $a = 0$ ist $a \leq 2^{n-1}$ auch immer wahr. Um diese Fallunterscheidung nicht immer aufschreiben zu müssen, vereinbaren wir: $\log_2 0 = 0$.

Insgesamt ergibt sich der in Abbildung 6.6 dargestellte Algorithmus.

```

// Eingaben:  $a \in \mathbb{N}_0, b \in \mathbb{N}_0$ 
 $X \leftarrow a$ 
 $Y \leftarrow b$ 
 $P \leftarrow 0$ 
 $x \leftarrow X \bmod 2$ 
 $n \leftarrow 1 + \lceil \log_2 a \rceil$ 
for  $i \leftarrow 0$  to  $n - 1$  do
     $P \leftarrow P + x \cdot Y$ 
     $X \leftarrow X \text{ div } 2$ 
     $Y \leftarrow 2 \cdot Y$ 
     $x \leftarrow X \bmod 2$ 
od
// Ergebnis:  $P = a \cdot b$ 

```

Abbildung 6.6: Algorithmus zu Multiplikation beliebiger nichtnegativer ganzer Zahlen.