

Christian Maurer

Die **RM**aschine
Register

v. 17. September 2011

Dr. Christian Maurer
Keithstr. 16
10787 Berlin

<http://murus.org/>

Die Quelltexte des Modula-2 UML Sums sind mit größter Sorgfalt entwickelt und werden laufend gepflegt. Kein Programmsystem dürfte jedoch jemals frei von Fehlern sein; deshalb ist auch nicht damit zu rechnen, daß *dieses* System fehlerfrei ist: Es darf nur benutzt werden „wie es ist“.

Die Modula-2- und die Java-Quelltexte von Murus sind freie Software. Sie können sie unter den Bedingungen der GNU General Public License, wie von der Free Software Foundation veröffentlicht, weitergeben und/oder modifizieren, entweder gemäß Version 3 der Lizenz oder (nach Ihrer Wahl) jeder späteren Version. Ihre Veröffentlichung erfolgt in der Hoffnung, daß sie Ihnen von Nutzen sein könnten – aber *ohne irgendeine Garantie*, auch ohne die implizite Garantie der *Marktreife* oder der *Verwendbarkeit für einen bestimmten Zweck*.

Der Originaltext der GPL ist im weltweiten Netz unter der Adresse www.gnu.org/licenses/gpl.html zu finden; eine deutsche Übersetzung unter www.gnu.de/documents/gpl-3.0-de.html.

Die Quelltexte von Murus sind nur zu Lehrzwecken konstruiert und haben rein akademischen Wert. Ihre Verwendung in Programmen könnte zu *Schäden* führen, z. B. zur Inbrandsetzung von Rechnern, zur Entgleisung von Eisenbahnzügen, zum GAU in Atomkraftwerken oder zum Absturz des Mondes ...

Satz: Autor mit $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{T}\mathcal{E}\mathcal{X}$

Mach die Dinge so einfach wie möglich,
aber nicht einfacher.

Albert Einstein

Vorwort

Vor vielen Jahren hatte ich am Fachbereich Mathematik und Informatik der Freien Universität Berlin eine Lehrerfortbildungstagung zur Informatik organisiert, auf der u. a. Frau Prof. Dr. Koppelberg einen grundlegenden Vortrag über Registermaschinen gehalten hat.

Begleitend dazu war von mir ein Programm zur Simulation einer Registermaschine zu dem Zweck entwickelt und vorgestellt worden, das Thema im Informatikunterricht nicht nur theoretisch behandeln, sondern die Lösungen von Aufgaben auf dem Rechner auch praktisch ausprobieren zu können.

Dieses Programm wird hier dokumentiert, um es dem Informatikunterricht zugänglich zu machen.

Christian Maurer

Inhaltsverzeichnis

ALLGEMEINES

Arbeitsverzeichnis	1
--------------------------	---

EINFÜHRUNG

Das Konzept	2
Bestandteile einer Registermaschine	3
Register	3

REGISTERMASCHINENPROGRAMME

Programmzeilen	4
Ausführung eines RM-Programms	4
Elementare Anweisungen	5
Makros	6
Komplexere Beispiele	10

DAS PROGRAMM RM

Das Simulationsprogramm $R_{\text{Register}}^{\text{Maschine}}$	12
Allgemeines zur Programmbedienung	13
Hinweise zur Arbeit mit der $R_{\text{Register}}^{\text{Maschine}}$	14

ANHANG

Aufgaben	15
Literatur	16

ALLGEMEINES

$RM_{\text{Maschine Register}}$ ist als Bestandteil des $Modul_{\text{en}}^{Ula} \cup_{\text{iver}}^{Sums}$ nach dessen Installation oder Aktualisierung ebenfalls installiert.

Voraussetzung für die Installation des $Modul_{\text{en}}^{Ula} \cup_{\text{iver}}^{Sums}$ ist die Installation von **Mocka**, dem Modula-2-Compiler der GMD (Hinweise dazu finden sich unter lwb.mi.fu-berlin.de/inf/mocka).

Arbeitsverzeichnis

Das Arbeitsverzeichnis, aus dem heraus RM aufgerufen wird und in dem ggf. Daten abgelegt werden, ist $\$HOME/.RM$ (es sei denn, daß durch die Umgebungsvariable RM ein anderes Arbeitsverzeichnis definiert ist). Damit ist sichergestellt, daß Benutzer/innen mit dem Aufruf von RM ihre eigenen Daten verwalten.

Wenn dieses Verzeichnis noch nicht existiert, wird es beim ersten Aufruf von RM selbsttätig angelegt und es werden Beispieldaten aus dem $Modul_{\text{en}}^{Ula} \cup_{\text{iver}}^{Sum}$ dorthinein kopiert, falls es welche gibt.

EINFÜHRUNG

Das Konzept

Bei der Registermaschine handelt es sich um ein sehr einfaches Maschinenmodell, das geeignet ist, den Gedanken der Berechenbarkeit propädeutisch zu behandeln.

Die Bedeutung dieses Konzepts liegt

- *theoretisch* in seiner Äquivalenz zum Begriff der Berechenbarkeit (durch rekursive Funktionen bzw. Turingmaschinen):
 - *Alles, was sich überhaupt programmieren läßt, läßt sich im Prinzip schon mit Registermaschinen erledigen* (s. Literatur: Artikel von Frau Prof. Dr. S. Koppelberg).
- *unterrichtspraktisch* in seiner Klarheit und Verständlichkeit:
- Registermaschinen-Programme sind wegen ihrer etwas höheren Abstraktionsstufe weniger kompliziert und leichter handhabbar als Turing-Maschinen-Programme, die das gleiche leisten; trotzdem unglaublich ausdrucksstark, weil mit geeigneten Makros quasi beliebig erweiterbar;
- *methodisch* in seinem einführenden Charakter in grundlegende Konzepte der *maschinennahen* sowie allgemein der *imperativen* Programmierung: Der Zustandsbegriff (repräsentiert durch *Wertzuweisungen an Variable*) und die algorithmischen Grundstrukturen (*Sequenz*, *Fallunterscheidung* und *Schleife*) lassen sich auf natürliche Weise mit RM-Programmen entwickeln, die mit minimalem syntaktischen Aufwand erstellt werden.

Es gibt auch konzeptionelle Berührungspunkte mit der funktionalen Programmierung: RM-Programme sind inhärent *rekursiv* und lassen sich als Algorithmen zur *Berechnung von Funktionswerten* auffassen.

Registermaschinen stellen also eine didaktisch-methodisch leicht zugängliche und in vieler Hinsicht durchaus vorteilhafte Variante zur Einführung von Turing-Maschinen dar.

Das Thema ist auch gut für eine Unterrichtsreihe in frühen Phasen des Informatikunterrichts geeignet, weil es keine Vorkenntnisse voraussetzt.

Bestandteile einer Registermaschine

Eine Registermaschine verfügt über

- einen *Datenspeicher* in Form einer Menge von *Registern* und
- einen *Programmspeicher* zur Aufnahme eines Programms in Form einzelner *Programmschritte*.

Die Größe von Daten- und Programmspeicher soll nicht festgelegt sein; es wird aber vorausgesetzt, daß sie groß genug sind.

Darüberhinaus benötigt eine Registermaschine eine Möglichkeit

- zur Belegung der Register mit Daten und zur Ausgabe dieser Daten sowie
- zur Eingabe eines Programms, d. h. zur Belegung des Programmspeichers mit Programmschritten.

Register

Register sind Speicherplätze, die jeweils eine natürliche Zahl aufnehmen können. Diese Inhalte der Register werden auch als ihr *Wert* bezeichnet.

Da die Inhalte der Register im Laufe der Ausführung eines Programms verändert werden (was ja gerade der Zweck von Programmen ist), können die Register auch als *Variable* bezeichnet werden.

Register müssen irgendwie identifiziert werden können, um an ihre Werte heranzukommen. Man könnte sie einfach numerieren und mit ihrer Nummer ansprechen; etwas eleganter ist es natürlich, sie nach bestimmten Konventionen mit einem Namen zu versehen, über den sie angesprochen werden.

Wir bezeichnen sie im folgenden einfach mit kleinen Buchstaben, die auch als Namen von Variablen aufgefaßt werden können.

Zu Beginn der Ausführung eines Registermaschinenprogramms haben alle Register den Wert 0.

REGISTERMASCHINENPROGRAMME

Programmzeilen

Die einzelnen Schritte eines Registermaschinenprogramms (im folgenden kurz RM-Programm genannt) stehen „Zeile für Zeile“ hintereinander im Programmspeicher und werden deshalb als *Programmzeilen* bezeichnet. Sie werden – beginnend bei 0 – fortlaufend durchnummeriert; in der Reihenfolge, wie sie im Programmspeicher stehen.

Eine Programmzeile ist entweder

- eine *elementare Anweisung* oder
- der Aufruf eines *Makros*.

Am Anfang einer Programmzeile, d. h. vor einer elementaren Anweisung oder einem Makroaufruf, darf zusätzlich eine *Marke* stehen. Es darf auch leere Programmzeilen geben, die *nur* aus einer Marke bestehen.

Ausführung eines RM-Programms

Ein RM-Programm wird von einer Registermaschine in der Weise ausgeführt, daß seine Programmzeilen *sequentiell* (d. h. Zeile für Zeile) abgearbeitet werden, wobei bei der ersten Zeile begonnen wird und danach entweder zur nächsten Programmzeile gegangen oder zu einer anderen als der nächsten Zeile gesprungen wird.

Etwas genauer betrachtet, wird das so realisiert:

Als jeweils nächste Programmzeile wird immer *diejenige* Zeile ausgeführt, deren Nummer in einem speziellen Register, dem sogenannten *Programmzähler*, steht.

Anfangs enthält der Programmzähler eine 0; es wird also bei der ersten Programmzeile begonnen. Welche Zahl *nach* der Ausführung einer Programmzeile im Programmzähler steht, d. h. welche Zeile als nächste ausgeführt wird, hängt vom Inhalt dieser Programmzeile ab.

Leere Programmzeilen haben dabei lediglich den Effekt, daß der Programmzähler um 1 erhöht wird.

Wenn der Wert des Programmzählers größer oder gleich der Anzahl der Programmschritte ist, ist die Ausführung des Programms beendet (da die Numerierung mit 0 beginnt, z. B. nach der letzten Programmzeile, falls das keine Sprunganweisung ist).

Elementare Anweisungen

Registermaschinen verfügen nur über eine äußerst beschränkte „Programmiersprache“ mit – der geradezu lächerlichen Anzahl von – lediglich drei *elementaren Anweisungen*: zwei zum Verändern eines Registerinhalts um 1 und eine zum „Springen“ im Programm in Abhängigkeit davon, ob ein Register den Inhalt 0 hat oder nicht.

Im Einzelnen sind das:

$$x = x + 1$$

Effekt:

Der Wert des Registers x ist um 1 erhöht; danach steht die Nummer der nächsten Programmzeile im Programmzähler, was zur Folge hat, daß danach die nächste Programmzeile ausgeführt wird.

$$x = x - 1$$

Effekt:

Wenn der Wert des Registers x größer als 0 war, ist er um 1 erniedrigt, andernfalls unverändert; danach steht – wie oben – die Nummer der nächsten Programmzeile im Programmzähler.

if $x \neq 0$ goto M

Effekt:

Wenn der Wert des Registers x größer als 0 ist, steht die Nummer der ersten Programmzeile mit der Marke M im Programmzähler, was zur Folge hat, daß im Programm zu dieser Zeile „gesprungen“ wird; andernfalls die Nummer der nächsten Zeile, was dazu führt, daß das Programm bei der nächsten Zeile fortgesetzt wird, sofern es noch eine gibt, ansonsten beendet ist. Wenn es keine Programmzeile mit dieser Marke gibt, wird das Programm abgebrochen.

Für den Namen des verwendeten Registers – hier x – kann in den Anweisungen der Name eines beliebigen Registers eingesetzt werden; in diesem Sinne sind diese Zeilen als *Schablonen* für Anweisungen zu verstehen.

Makros

Makros sind RM-(„Unter-“)Programme, die eine sehr weitgehende Möglichkeit eröffnen, den Umfang der RM-(„Programmier“-)Sprache quasi unbegrenzt zu erweitern.

Ein Makro besteht aus

- einer *Definitionszeile*, die durch eine spezielle Marke eingeleitet ist (wir vereinbaren dafür das Ausrufungszeichen !), sowie
- ihrer *Implementierung* in Form einer Folge von Programmzeilen, d. h. von elementaren Anweisungen oder wiederum Makroaufrufen, die den Effekt des Makros darstellt.

Mit der Vereinbarung, daß alle von einem RM-Programm aufgerufenen Makros am Ende des Programms stehen müssen, ist das Ende eines Makros dadurch definiert, daß entweder keine weiteren Programmzeilen folgen oder es sich bei der nächsten Programmzeile um die Definitionszeile eines weiteren Makros handelt.

Wir zeigen zwei ganz einfache Beispiele für Makros.

Erst einmal ein Programm zur Erhöhung eines Registers um 3:

```
! x = x + 3
  x = x + 1
  x = x + 1
  x = x + 1
```

und eins zur Rücksetzung eines Register auf den Wert 0:

```
! z = 0
N z = z - 1
  if z # 0 goto N
```

Die Definitionszeilen dieser Makros sind zu Schablonen für neue (in gewissem Sinne „elementare“) Anweisungen geworden, die in einem RM-Programm verwendet werden können; die *Zahl 3* (im Grunde auch das *Symbol +* in seiner neuen Bedeutung) in diesen Beispielen zu neuen Bestandteilen der RM-Sprache.

Daß in den Programmzeilen eines Makros wiederum Makroaufrufe vorkommen dürfen, heißt, daß Makros (auch mehrfach) ineinander geschachtelt werden können.

Einfache Beispiele dafür sind

```
! d = 3
  d = 0
  d = d + 3
```

und

```
! n = 9
  n = 3
  n = n + 3
  n = n + 3
```

sowie das folgende mit dem Effekt, daß im Programmzähler die Nummer der ersten Programmzeile mit der Marke X steht, falls das Register v den Wert 0 hatte, ansonsten der Programmzähler um 1 erhöht ist:

```
! if v = 0 goto X
  if v # 0 goto V
  v1 = v1 + 1
  if v1 # 0 goto X
V
```

In die RM-Sprache lassen sich neben Zahlen und Symbolen auch *Wörter* einführen; etwa wie im Beispiel

```
! verdreifachen y
  if y = 0 goto E
  y1 = 0
M y1 = y1 + 3
  y = y - 1
  if y # 0 goto M
P y = y + 1
  y1 = y1 - 1
  if y1 # 0 goto P
E
```

womit sich das dritte Beispiel etwas vereinfachen ließe:

```
! n = 9
  s = 3
  verdreifachen s
```

Ein etwas anspruchsvolleres Beispiel mit ineinandergeschachtelten Schleifen ist die Berechnung der Summe der ersten n natürlichen Zahlen:

```

! s = gauss n
% Eff.: s = 0, falls n = 0
% s = 1 + 2 + ... + n sonst
  s = 0
  if n = 0 goto D
  i = 0
  k = 0
A n = n - 1
  i = i + 1
B s = s + 1
  k = k + 1
  i = i - 1
  if i # 0 goto B
C k = k - 1
  i = i + 1
  if k # 0 goto C
  if n # 0 goto A
D

```

Die durch die Marke % eingeleiteten Zeilen sind *Kommentarzeilen*, die bei der Ausführung von RM-Programmen lediglich die Erhöhung des Programmzählers bewirken. Für komplexere Programme sind Kommentare mindestens guter Stil, wenn nicht sogar zur Verständlichkeit zwingend erforderlich (wer versteht denn noch die eigenen komplizierten Gedankengänge, wenn die eine Weile zurückliegen ...).

An diesen Beispielen wird deutlich:

Mit Makros kann die RM-Sprache beliebig erweitert werden.

Bei der Verwendung von Makros in RM-Programmen *müssen* die Namen aller in den Makros vorkommenden Register von den der im Programm verwendeten Register verschieden sein; genauso müssen die Register der verschiedenen Makros sorgfältig auseinandergehalten werden; andernfalls würden kaum übersehbare Konflikte zwischen den Effekten der einzelnen Makros drohen, schlimmstensfalls eine gegenseitige Beeinflussung der Registerwerte.

Entsprechendes gilt natürlich für alle beteiligten Marken, damit der Programmzähler bei Sprunganweisungen auf den korrekten Wert gesetzt, d. h. die jeweils richtige Stelle gefunden wird.

Beim Aufruf eines Makros muß außerdem beachtet werden, daß der syntaktische Aufbau des Aufrufs genau dem der Definitionszeile des Makros entspricht; z. B. wird das Makro mit der Definitionszeile

$$! x = x + y$$

mit $a = a + b$ oder $a = a + a$ aufgerufen, nicht jedoch z. B. mit $a = b + a$ oder $a = b + c$.

Komplexere Beispiele

Das folgende Makro erhöht den Wert des Registers **a** um den Wert des Registers **b**, wenn **a** und **b** verschiedene Register sind:

```

! a = a + b
  if b = 0 goto C
  z0 = 0
A a = a + 1
  z0 = z0 + 1
  b = b - 1
  if b # 0 goto A
B b = b + 1
  z0 = z0 - 1
  if z0 # 0 goto B
C

```

Falls **b** den Wert 0 hat, ist das Makro effektfrei. Ansonsten wird in Schleife A der Wert von **a** um den Wert von **b** erhöht. Weil dabei der Wert von **b** verloren geht, wird er gleichzeitig in das Hilfsregister **z0** kopiert, um dann in Schleife B wieder restauriert zu werden.

Wenn beim Aufruf dieses Programms als Makros allerdings für **a** und **b** die gleichen Register eingesetzt werden (z. B. $x = x + x$), ist dieser Algorithmus nicht korrekt, weil die Schleife A nicht terminiert. Um diesen Fall abzufangen, wird ein weiteres Hilfsregister gebraucht:

```

! a = a + b
  if b = 0 goto D
  z0 = 0
  z1 = 0
A z0 = z0 + 1
  z1 = z1 + 1
  b = b - 1
  if b # 0 goto A
B b = b + 1
  z0 = z0 - 1
  if z0 # 0 goto B
C a = a + 1
  z1 = z1 - 1

```

```
        if z1 # 0 goto C
D
```

Die allgemeine Berechnung einer Summe ist noch aufwendiger. Das folgende Makro ist auch korrekt, falls bei seinem Aufruf irgendwelche der beteiligten Register gleich sind:

```
! a = b + c
% Eff.: a enthält die Summe von b und c
  z0 = 0
  z1 = 0
  if b = 0 goto C
% z0 = z1 = b, b = 0:
A b = b - 1
  z0 = z0 + 1
  z1 = z1 + 1
  if b # 0 goto A
% restauriert:
B b = b + 1
  z1 = z1 - 1
  if z1 # 0 goto B
C if c = 0 goto F
% z0 = b + c, z1 = c, c = 0:
D z0 = z0 + 1
  z1 = z1 + 1
  c = c - 1
  if c # 0 goto D
% c restauriert:
E c = c + 1
  z1 = z1 - 1
  if z1 # 0 goto E
% a = b + c:
F a = 0
  if z0 = 0 goto H
G a = a + 1
  z0 = z0 - 1
  if z0 # 0 goto G
H
```

DAS SIMULATIONSPROGRAMM RM

Das Programm RM

Die $R_{\text{register}}^{\text{Maschine}}$ ist ein Simulationsprogramm zur Ausführung von RM-Programmen. Für die Bezeichner in RM-Programmen, deren Ausführung von der $R_{\text{register}}^{\text{Maschine}}$ simuliert werden soll, müssen folgende Konventionen eingehalten werden:

- *Variable*, d. h. Namen von Registern, müssen mit einem Kleinbuchstaben (von a bis z) anfangen und dürfen danach bis zu zwei Ziffern enthalten,
- *Marken* müssen mit einem Großbuchstaben (von A bis Z) anfangen und dürfen danach bis zu zwei Groß- oder Kleinbuchstaben oder Ziffern enthalten, wobei
- als Marke zur Kennzeichnung der Definitionszeile eines Makros das Ausrufungszeichen „!“ und
- als Marke zur Kennzeichnung einer Kommentarzeile das Prozentzeichen „%“ verwendet wird.

In den Definitionszeilen von Makros – folglich auch in den Zeilen, in denen sie aufgerufen werden – dürfen auch vorkommen:

- *natürliche Zahlen* (sie bestehen nur aus Ziffern),
- *Symbole*, und zwar – abgesehen von „=“ und „+“ – die Zeichen „-“, „*“, „/“, „^“, „<“ und „>“ sowie Klammern „(“ und „)“ und
- selbstdefinierte *Wörter*, die (zur Unterscheidung von Variablen) aus zwei oder mehr *Kleinbuchstaben* bestehen.

Die von einem RM-Programm benutzten Makros müssen textuell *hinter* dem RM-Programm stehen (ihre Reihenfolge untereinander spielt keine Rolle).

Allgemeines zur Programmbedienung

Die Bedienung des Programms ist denkbar einfach.

Neben den Buchstaben-, Ziffern- und Zeichentasten zum Eingeben von Text werden einige Sondertasten zur Korrektur von Eingaben und zur Steuerung des Programmablaufs gebraucht.

Der Eingabekorrektur dienen die folgenden Tasten:

- die Rückschritt- \leftarrow und die Entfernungstaste **Entf** zum Löschen einzelner Zeichen, in Kombination mit der Umschalttaste \uparrow zum Löschen des Eingabefeldes,
- die Pfeiltasten \leftarrow und \rightarrow nach links und rechts sowie
- die Anfangstaste **Pos1** und die Endetaste **Ende** zum Bewegen im Text.
- Mit der Einfügetaste **Einf** wird zwischen dem Einfüge- und dem Überschreibemodus umgeschaltet, wobei der aktuelle Modus an der unterschiedlichen Cursorform erkennbar ist: ein Unterstrich im Einfüge- und ein rechteckiger Block im Überschreibemodus.

Der Programmablauf wird mit

- der Eingabetaste \leftarrow , der Schlußtaste **Esc**, der Rückschritt-Taste \leftarrow ,
 - den Pfeil- \uparrow und \downarrow und den Bildtasten **Bild \uparrow** und **Bild \downarrow** nach oben und unten sowie
 - der Tabulatortaste \leftrightarrow gesteuert;
- gelegentlich in Verbindung mit der Umschalttaste \uparrow .

Fehlermeldungen (in gelber Schrift auf rotem Grund) werden mit der Rücktaste \leftarrow quittiert.

Hinweise zur Arbeit mit der $R_{\text{Register}}^{\text{Maschine}}$

Die Dateinamen von RM-Programmen *müssen* immer die Endung `.rm` haben.

- Mit einem beliebigen Editor ein RM-Programm editieren, z. B. `test.rm`
- Die $R_{\text{Register}}^{\text{Maschine}}$ durch Eingabe von `RM` aufrufen, wobei der Name des auszuführenden RM-Programms (ohne die Endung `.rm`) als Parameter mitgegeben werden kann, z. B. `RM test`,
- den Namen des RM-Programms eingeben und mit der Eingabetaste `↵` bestätigen
- ggf. das RM-Programm editieren (der eingebaute Editor ist aber noch nicht besonders leistungsfähig, deshalb kann zum Editieren längerer RM-Programme auch ein Editor eigener Wahl benutzt werden)
- den Editiermodus mit der Schlußtaste `Esc` verlassen
- die Startwerte der verwendeten Register eingeben
- mit der Eingabetaste `↵` schrittweise durch das RM-Programm laufen (die Registerinhalte werden dabei laufend angezeigt),
- falls gewünscht, mit `Esc` den Schrittmodus verlassen und das RM-Programm bis zum Ende durchlaufen lassen,
- falls gewünscht, die Programmausführung der $R_{\text{Register}}^{\text{Maschine}}$ mit der Kombination `Strg + C` abbrechen und
- nach Ausführung des Programms (was mit der Meldung `Programm ausgeführt` quittiert wird) die $R_{\text{Register}}^{\text{Maschine}}$ mit `Esc` beenden.

ANHANG

Aufgaben

Implementieren Sie die folgenden Makros und testen Sie sie mit der $R_{\text{Register}}^{\text{Maschine}}$:

```
! a = a - b
! a = b - c
! a = 2 * a
! a = a * b
! a = b * c
! a = b ^ 2
! a = a ^ b
! a = b ^ c
! goto M
! if a = b goto M
! if a < b goto M
! a = a div 2
! a = b div 2
! a = b div c
! a = b mod 2
! a = b mod c
! a = ggt b c
! a = kgv b c
! a = fak b
! a = maa b c
! a = min b c
! a = log2 b
! a = logb c
! a = binom b c
! a = fibonacci b
```

Literatur

E. Cohors-Fresenborg: Registermaschinen und Funktionen

In: Osnabrücker Schriften zur Mathematik, Heft 22 (1979), S. 23-54

Ein Schulbuch zur Einführung des Funktionsbegriffs auf der Grundlage von Algorithmen

S. Koppelberg: Was können Algorithmen?

In: Mathematische Aspekte der Angewandten Informatik,

R.-H. Schulz (Hrg.), BI-Wissenschaftsverlag (1994), S. 23–54

Im weltweiten Netz zu finden unter der Adresse

<http://www.emis.de/monographs/schulz/algo.pdf>