JOHANNES MAGENHEIM
ILLUSTRIERT VON THOMAS A. MÜLLER

INFORMATIK macchiato

CARTOONKURS FÜR SCHÜLER UND STUDENTEN

2. Auflage

ALWAYS LEARNING
PEARSON

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de abrufbar.

10 9 8 7 6 5 4 3 2 1
15 14 13

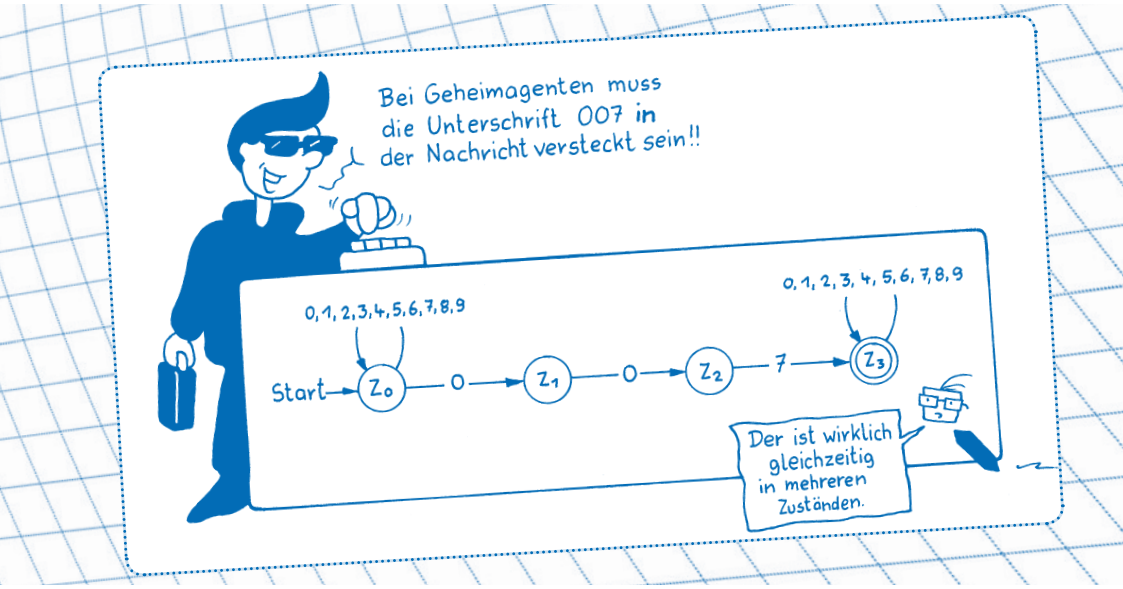
ISBN 978-3-86894-181-4

Martin-Kollar-Str. 10-12, D-81829 München
Alle Rechte vorbehalten
www.pearson.de
A part of Pearson plc worldwide

Programmleitung: Birger Peil, bpeil@pearson.de
Lektorat: Irmgard Wagner, irmwagner@t-online.de
Fachlektorat: Professor Karl C. Posch, Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie, Technische Universität Graz,
2. Fachlektor: StR Dr. Michael Savoric, Hohenstaufen-Gymnasium in Kaiserslautern
Korrektorat: Petra Kienle, Fürstenfeldbruck
Herstellung: Martha Kürzl-Harrison, mkuerzl@pearson.de
Satz: m2 design, Sterzing, www.m2-design.org
Druck und Verarbeitung: GraphyCems, Villatuerta

Printed in Spain

Nichtdeterministische Automaten

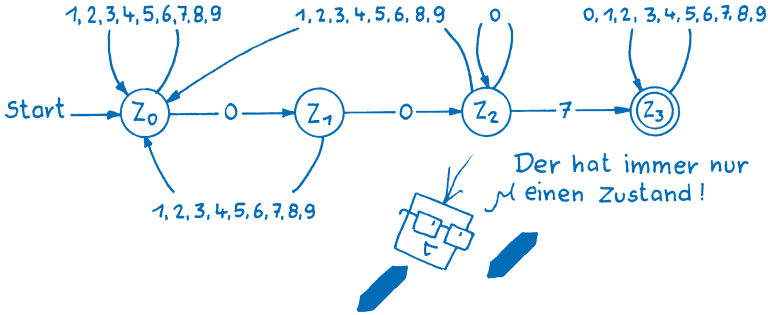


Bei der Eingabe einer **PIN** gibt es nur richtig oder falsch. Wie sieht dann der Automat aus, der die Nachricht des Geheimagenten Tom auf Korrektheit überprüfen kann? Dazu können vorher beliebige Ziffern stehen, dann kommt die Unterschrift 007 und danach können natürlich noch mal beliebige Ziffern folgen.

Oh, was ist denn da passiert? Man kommt aus dem Zustand z_0 mit der Eingabe der Zahl 0 sowohl in den Zustand z_0 als auch in den Zustand z_1. Dadurch ist die Übergangsfunktion nicht eindeutig und wir müssen beide Möglichkeiten testen, um zu entscheiden, ob die Eingabe akzeptiert werden kann. Solche Automaten nennt man **nichtdeterministisch**, sie können gleichzeitig in mehreren Zuständen sein! So wird vom Automaten nach Eingabe von 0 sowohl der Zustand z_0 als auch z_1 angenommen. Bei der Eingabe der nächsten 0 sind sogar drei Zustände möglich: z_0, z_1 und z_2. Wenn bei einer Entscheidung in mindestens einer Alternative ein Endzustand erreicht werden kann, dann wird das Wort akzeptiert. Anschaulich kann man sich das so vorstellen, dass der Automat alle Möglichkeiten durchprobiert und versucht, mit mindestens einer Möglichkeit zum Endzustand zu kommen.

Ein **nichtdeterministischer erkennender Automat** besitzt eine nichteindeutige Zustandsübergangsfunktion, die einem Zeichen aus dem Eingabealphabet E und einem Zustand Z mehrere Nachfolgezustände zuordnen kann. Er akzeptiert eine Eingabe, wenn es für die Eingabe mindestens einen möglichen Weg durch den Graphen zu einem Endzustand gibt.

Da ein nichtdeterministischer Automat gleichzeitig in mehreren Zuständen ist, ist er nicht so einfach zu verstehen. Was für Vorteile hat denn dann der Nichtdeterminismus? Das wird deutlich, wenn man das Problem ohne Nicht-determinismus löst. Der Automat wird dann deutlich komplizierter, da er bei jeder falschen Ziffer, die vor der 007-Folge steht, wieder in den Ausgangs-zustand zurückkehren muss.



Es gibt natürlich noch Automaten, die deutlich mehr können und damit komplizierter sind. Mehr Infos zu diesen **Kellerautomaten** und **Turingmaschinen** auf der Webseite.

Aufgaben zum Ausprobieren finden Sie im Anhang.

Automaten sind ein wichtiges Modell der Informatik, mit dem man auch reale Automaten beschreiben kann.

Automaten

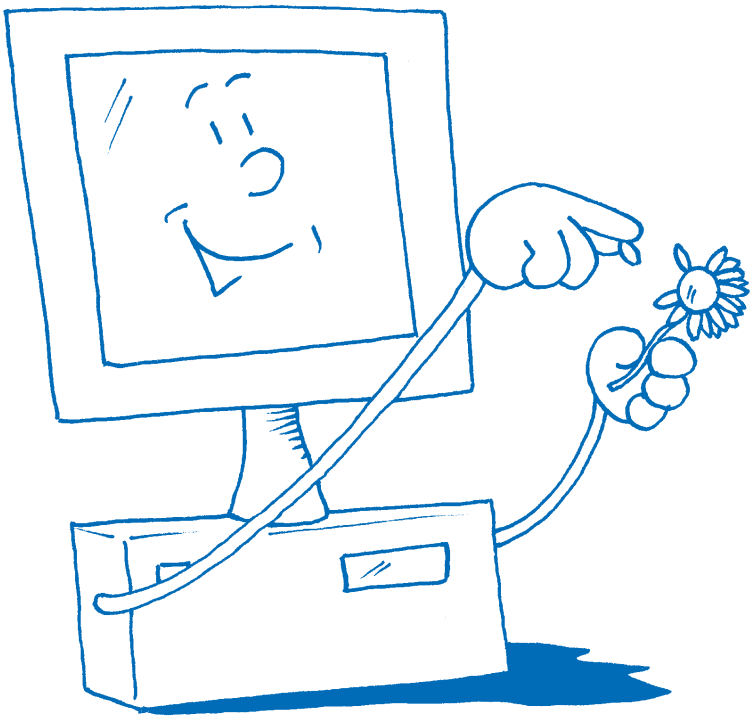
- lesen bestimmte **Eingabe**folgen,
- speichern ihr Wissen mit Hilfe von **Zuständen** und
- erzeugen dabei eine **Ausgabe**. Dieses kann eine Zeichenfolge sein oder auch nur eine Ausgabe wahr/falsch.

Nichtdeterministische Automaten verhalten sich auch nach festen Regeln, können jedoch gleichzeitig in mehreren Zuständen sein.

Je mehr Zustandsmöglichkeiten ein Automat besitzt, desto komplexere Eingabefolgen kann er erkennen. Diese akzeptierten Eingabefolgen oder **Wörter** bilden die **Sprache** des Automaten.

5

Wenn es klappt, dann kann es dauern

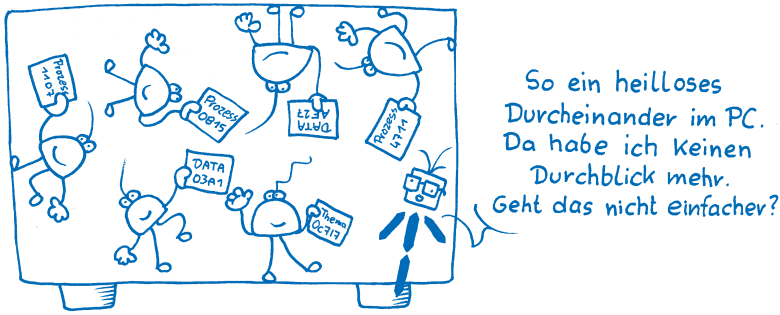


Berechenbarkeit und Komplexität

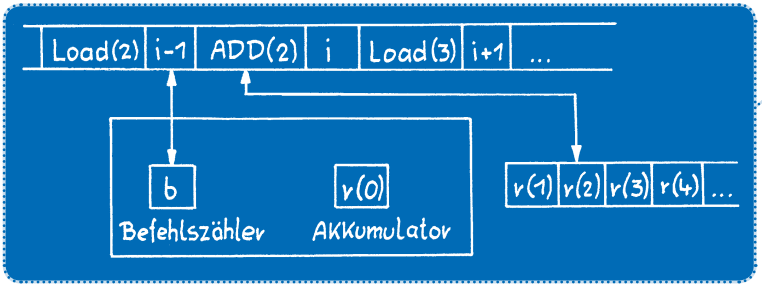
Wenn es klappt, dann kann es dauern

Berechenbarkeit

Lisa sitzt vor einer schwierigen **Mathematikaufgabe**. Die von Hand zu lösen, würde Stunden dauern. Sie fragt Tom, ob er ihr nicht schnell ein Programm schreiben könne, das die Aufgabe löst, denn der Computer kann ja alles schnell berechnen. Tom versucht ihr zu erklären, dass ein Computer auch nicht immer schnell ist. Verschiedene Prozesse und Teile von Programmen müssen bearbeitet werden und dafür benötigt jeder Prozess Rechenzeit. Die Rechenzeiten der Prozesse unterscheiden sich und müssen von einem weiteren Prozess organisiert werden, der wiederum Rechenzeit benötigt. Als er an dieser Stelle ankommt, unterbricht ihn Lisa. Das sei alles interessant, aber lässt sich das nicht auch einfacher erklären?



Tom erzählt ihr von einer sogenannten **Registermaschine**, die alles kann, wozu auch ein moderner PC fähig ist. Der Speicher einer solchen Maschine besteht aus sogenannten Registern, die einfach durchnummeriert sind; daher der Name. Registermaschinen sind im Vergleich zu modernen Computern viel leichter zu verstehen, dafür sind sie aber auch um ein Vielfaches langsamer. Eine Registermaschine wird mit einem Programm gespeist, das aus einer Reihe von einfachen Befehlen besteht, z.B. LOAD, STORE, ADD, GOTO.



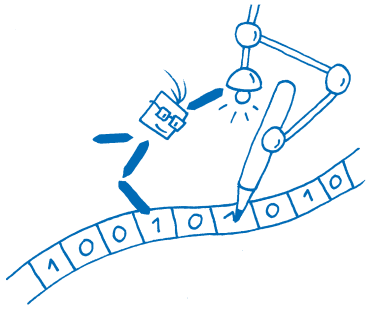
Die einzelnen Zeilen des Programms sind durchnummeriert. Um sich zu merken, welche Befehlszeile als Nächstes bearbeitet werden muss, gibt es einen Befehlszähler. Die Daten können aus den Registern in den Akkumulator geladen und dort durch die Befehle bearbeitet werden.

Lisa unterbricht ihn wieder an dieser Stelle. Da muss man sich ja hier jede Menge Befehle und Begriffe merken. Sie fragt Tom, ob man einen Rechner nicht ohne die vielen verschiedenen Befehle erklären könne.

Man kann einen Rechner noch auf eine andere Art modellieren, mit der sogenannten deterministischen **Turingmaschine**, kurz **DTM**. Diese Maschinen sind ein theoretisches Rechenmodell, für das man Programme erstellen kann, die das Gleiche berechnen wie eine Registermaschine. Dabei haben sie jedoch nur wenige mögliche Operationen.

Eine Turingmaschine besteht aus einem **Schreib-/Lesekopf** und einem unendlich langen Band, das als Speicher dient. Das Band ist in Zellen eingeteilt, in die verschiedene Symbole geschrieben werden können. Ist eine Zelle leer, wird das hier mit „_“ gekennzeichnet.

Eine DTM benötigt noch ein Programm und eine Menge verschiedene Symbole, die vom Band gelesen oder auf das Band geschrieben werden können.



Das hier verwendete **Alphabet** Σ besteht aus den **Symbolen** „0, 1, _". In anderen Darstellungen kann es aber auch aus anderen Symbolen und einer anderen Anzahl von Symbolen bestehen.

Die Anzahl der Symbole von Σ ist aber immer endlich. Zu Beginn eines Programms kann eine Eingabe auf dem Band stehen, wobei sich dann der Schreib-/Lesekopf immer am linken Ende der Eingabe befindet.

Das Programm einer Turingmaschine besteht aus einer endlichen Menge von Zuständen $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ und einer endlichen Menge δ von **Übergangsfunktionen**, die das Verhalten der DTM, wie Lesen oder Schreiben, festlegen. Zudem werden im Allgemeinen zwei weitere Zustände benötigt: ein Zustand, in dem das Programm akzeptierend beendet wird, q_{accept}, und einer, in dem das Programm ablehnend beendet wird, q_{reject}. Die Übergangsfunktionen regeln, was passiert, wenn sich die Turingmaschine in einem bestimmten Zustand befindet und ein Symbol vom Band liest. Dann wird der Zustand gewechselt, ein Symbol auf das Band geschrieben und der Schreib-/Lesekopf nach rechts oder links bewegt. Mathematisch schreibt man:

$$\delta(Q, \Sigma) \rightarrow (Q, \Sigma, \{R, L\})$$

Die Programme können in Tabellenform geschrieben werden. Dabei steht in der ersten Spalte der Zustand, in dem sich die Turingmaschine gerade befindet, und in der ersten Zeile das Symbol, das sich gerade

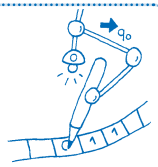
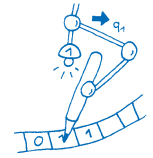
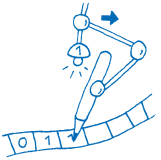
unter dem Schreib-/Lesekopf befindet. In jeder Zelle der Tabelle steht eine Folgekonfiguration; also der Zustand, in den die DTM wechselt, das Symbol, das geschrieben wird, und die Bewegungsrichtung des Schreib-/Lesekopfs. Wenn sich z.B. die DTM im Zustand q_0 befindet und eine 1 liest, wechselt die DTM in den Zustand q_1, schreibt eine 1 und bewegt den Schreib-/Lesekopf nach rechts.

Zu Beginn befindet sich hier ein Programm immer im Zustand q_0.

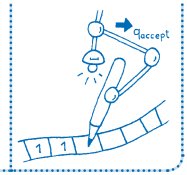
Zustand \ Symbol	0	1	_
q_0	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_{reject}, \_, R)$
q_1	$(q_0, 0, R)$	$(q_1, 1, R)$	$(q_{accept}, \_, R)$

Bei diesem Programm wird der Schreib-/Lesekopf der DTM einfach nur so lange nach rechts bewegt, bis das erste „_"-Symbol gelesen wird. Die Maschine hält nur dann im akzeptierenden Zustand, wenn die Eingabe mit einer 1 endet. Ansonsten ist die Eingabe für diese DTM ungültig.

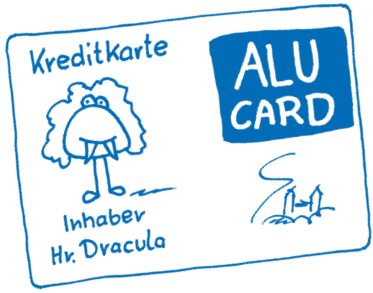
Ist beispielsweise die Eingabe „011", werden die folgenden Schritte durchgeführt:

1	Die DTM ist im Zustand q_0 und liest eine 0. Sie bleibt im Zustand q_0, schreibt eine 0 und bewegt den Kopf nach rechts. Formal schreibt man: $(q_0, 0) \rightarrow (q_0, 0, R)$	
2	Die DTM ist im Zustand q_0 und liest eine 1. Sie wechselt in den Zustand q_1, schreibt eine 1 und bewegt den Schreib-/Lesekopf nach rechts. $(q_0, 1) \rightarrow (q_1, 1, R)$	
3	Die DTM ist im Zustand q_1 und liest eine 1. Sie bleibt im Zustand q_1, schreibt eine 1 und bewegt den Schreib-/Lesekopf nach rechts. $(q_1, 1) \rightarrow (q_1, 1, R)$	

4 Die DTM ist im Zustand q_1 und liest ein _. Sie wechselt in den Zustand q_{accept}, schreibt ein _ und bewegt den Schreib-/Lesekopf nach rechts.
$(q_1, \_) \rightarrow (q_{accept}, \_, R)$



Mit Turingmaschinen lassen sich so verschiedene Berechnungen durchführen. Außerdem kann man untersuchen, ob Eingaben gewisse Eigenschaften haben, z.B. ob die Eingabe, wie hier, mit einer 1 endet. Eingaben werden auch als Worte bezeichnet. Mengen, die aus **Worten** bestehen, die alle mindestens eine gemeinsame Eigenschaft haben, werden als **Sprache** bezeichnet. Ein Beispiel hierfür ist die Sprache **Palindrom**. Ein Palindrom ist ein Wort, das vorwärts und rückwärts gelesen dasselbe ergibt. In der natürlichen Sprache ist z.B. „anna" oder „draculasalucard" ein Palindrom; „dampfschifffahrt" hingegen nicht.



Das Alphabet der hier verwendeten Turingmaschinen besteht nur aus den Symbolen 0 und 1. Die im Folgenden vorgestellte Turingmaschine kann also nur untersuchen, ob eine Folge von Einsen und Nullen ein Palindrom bilden. Diese Maschine guckt sich zuerst das erste Symbol an, merkt sich dieses und löscht es vom Band. Zum Markieren, welche Symbole vom Band schon gelesen worden sind, würde ein größeres Bandalphabet benötigt. Stattdessen kann hier das vorderste Symbol gelöscht werden, weil es nur mit dem letzten Zeichen auf dem Band verglichen werden muss. Der Schreib-/Lesekopf wird nach rechts bewegt, bis das letzte Symbol erreicht ist. Wenn das letzte Symbol mit dem gespeicherten nicht übereinstimmt, stoppt die Maschine und lehnt das Wort mit q_{reject} ab. Stimmen die Symbole überein, löscht sie das letzte Symbol vom Band und der Schreib-/Lesekopf wird so lange nach links bewegt,

bis das erste „_“ gefunden wird. Der Kopf wird dann um eine Position nach rechts bewegt und das Symbol gelesen. Ist dieses Symbol wiederum ein „_“, hält die Turingmaschine akzeptierend; ansonsten wird der Algorithmus von vorne ausgeführt.

δ	0	1	_
q_0	(q_1,_, R)	(q_2,_, R)	(q_{accept},_, L)
q_1	(q_1,0, R)	(q_1, 1, R)	(q_3,_,L)
q_2	(q_2,0, R)	(q_2, 1, R)	(q_4,_, L)
q_3	(q_5,_, L)	(q_{reject}, 1, L)	(q_{accept},_, L)
q_4	(q_{reject},0, L)	(q_5,_, L)	(q_{accept},_, L)
q_5	(q_5,0, L)	(q_5, 1, L)	(q_0,_, R)

Eine äquivalente Darstellung der Tabelle als Graph kann man auf der Website zum Buch im Internet sehen.

Die Turingmaschine kann sich Symbole merken, ohne diese auf das Band zu schreiben. Dies funktioniert, indem die Maschine bei dem einen Symbol in den einen und bei einem anderen Symbol in einen anderen Zustand wechselt und von dort aus auf unterschiedliche Weise weiterarbeitet. Man sagt auch, dass das Symbol im Zustand gespeichert wird.

Bei der DTM, die die Sprache Palindrom entscheidet, kann man das an dem Zustand q_0 sehen: Wird am Anfang eine 0 gelesen, wechselt die Turingmaschine in Zustand q_1 und geht nach rechts; liest sie eine 1, so wechselt sie in q_2 und geht ebenfalls nach rechts.

Mit Turingmaschinen kann man dasselbe machen wie mit einem PC, jedoch unterscheiden sich die bisher betrachteten von normalen Computern in einem wesentlichen Punkt: Sie können nur eine Funktion berechnen, während Computer eine Vielzahl verschiedener Programme durchführen können. Man kann jedoch die Funktionsweise einer Turingmaschine als eine Folge von Einsen und Nullen codieren. Eine solche Codierung nennt man **Gödelnummer**. Wie eine solche Turingmaschine

genau aussieht, die so **codiert** wurde, wird hier nicht beschrieben. In der Informatik schreibt man für eine codierte DTM M kurz $\langle M \rangle$. Wird der Codierung eine zusätzliche Eingabe w, die aus Zeichen aus der Menge Σ besteht, übergeben, schreibt man $\langle M \rangle w$. Diese zusammengesetzte Eingabe kann man einer sogenannten **Universellen Turingmaschine** übergeben, die dann die codierte DTM M mit der Eingabe w simuliert.

Neben der Simulation von Computern kann man mit Turingmaschinen eine weitere wichtige Sache machen: Man kann zeigen, dass es Probleme gibt, die **nicht berechenbar** sind. Ein bekanntes Beispiel hierfür ist das sogenannte **Halteproblem**. Die Frage, die sich stellt, ist, ob eine DTM konstruiert werden kann, die entscheidet, ob eine beliebige, codierte DTM $\langle M \rangle$ mit einer Eingabe w hält oder ob sie in eine Endlosschleife läuft. Die Menge der Worte, die aus einer codierten DTM $\langle M \rangle$ bestehen, die mit einer Eingabe w halten, wird kurz als H bezeichnet; die codierte Turingmaschine zu der Sprache Palindrom mit einer beliebigen endlichen Eingabe ist ein Wort der Sprache H, da sie sicher im Zustand q_{accept} oder q_{reject} hält. Die Menge H wird formal beschrieben durch:

Copyright