

TuringKara – Zweidimensionale Turing-Maschinen

Die theoretische Informatik ist ein wichtiger Teil jeder Informatikausbildung. Universelle Berechnungsmodelle spielen dabei eine zentrale Rolle. Sie entstanden aus der Frage „Was ist algorithmisch berechenbar und was nicht?“. Für Anfänger ist es oftmals überraschend, dass universelle Berechnungsmodelle mit sehr einfachen Grundoperationen auskommen, solange sie zwei Eigenschaften besitzen: unbeschränkte Zeit und unbeschränkten Speicher. Eine tiefer gehende und anschaulich geschriebene Einführung in die Themen der Berechenbarkeit findet sich in David Harels *Das Affenpuzzle und weitere bad news aus der Computerwelt* [Har02].

Turing-Maschinen sind ein universelles Berechnungsmodell. Schülerinnen und Schüler, die bereits mit Kara vertraut sind, können den Übergang zu Turing-Maschinen einfach vollziehen. Die Kontrolllogik ist die gleiche wie bei endlichen Automaten. Der Unterschied liegt im externen Speicher: Turing-Maschinen arbeiten auf einem unbeschränkten Speichermedium. Typischerweise wird ein eindimensionales Band verwendet. Aus der Sicht der Berechenbarkeit argumentierte Turing, dass ein eindimensionales Band für Turing-Maschinen ausreichend sei [Tur37]:

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares.

TuringKara verwendet hingegen bewusst eine zweidimensionale Welt. Aus Sicht der theoretischen Berechenbarkeit bringt ein zweidimensionales Speichermedium zwar keine Vorteile. Es hat jedoch Vorteile in didaktischer Hinsicht: Das Lösen von Aufgaben wird vereinfacht, da sich die zu verarbeitenden Daten viel besser anordnen lassen und so der Zugriff einfacher wird. Dadurch wird die Anzahl der Bewegungen des Lese-/Schreibkopfes stark reduziert.

6.1 Die TuringKara-Umgebung

Abbildungen 6.1 und 6.2 zeigen die TuringKara-Umgebung, die sich an die vertraute Kara-Umgebung anlehnt. Anstelle des Marienkäfers wird ein Lese-/Schreibkopf programmiert, der als Quadrat dargestellt ist. Der Lese-/Schreibkopf hat keinen Richtungssinn und kann in alle vier Himmelsrichtungen bewegt werden.

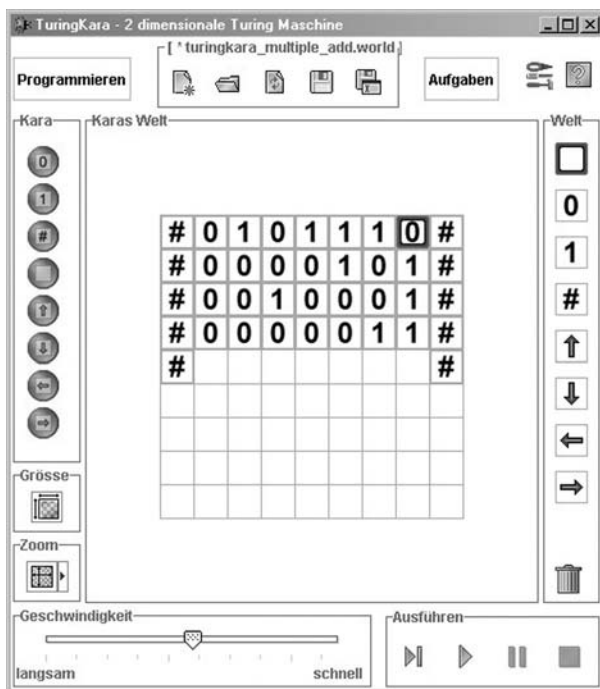


Abb. 6.1. Die TuringKara-Umgebung (Welteditor)

Die Felder der Welt können folgende Symbole aufnehmen: 0, 1, #, □ (leeres Feld) sowie die vier Pfeilsymbole \leftarrow , \rightarrow , \uparrow , \downarrow . Der Lese-/Schreibkopf kann jedes Symbol auf jedes Feld schreiben, unabhängig von dessen gegenwärtigem Inhalt. Die Symbole selbst haben keine semantische Bedeutung und können beliebig verwendet werden. Sie wurden für die TuringKara-Umgebung gewählt, weil sie bei vielen Aufgaben anschaulich sind.

Der Programmeditor ist praktisch identisch zur Kara-Umgebung. Anstelle der Sensoren wird abgefragt, was der Lesekopf auf dem aktuellen Feld sieht. So wird zum Beispiel in der Übergangstabelle von Abbildung 6.2 der erste Übergang gewählt, wenn der Lese-/Schreibkopf auf einer 0 oder einer 1 ist. Steht er auf einem #, so wird der zweite Übergang gewählt, und für den

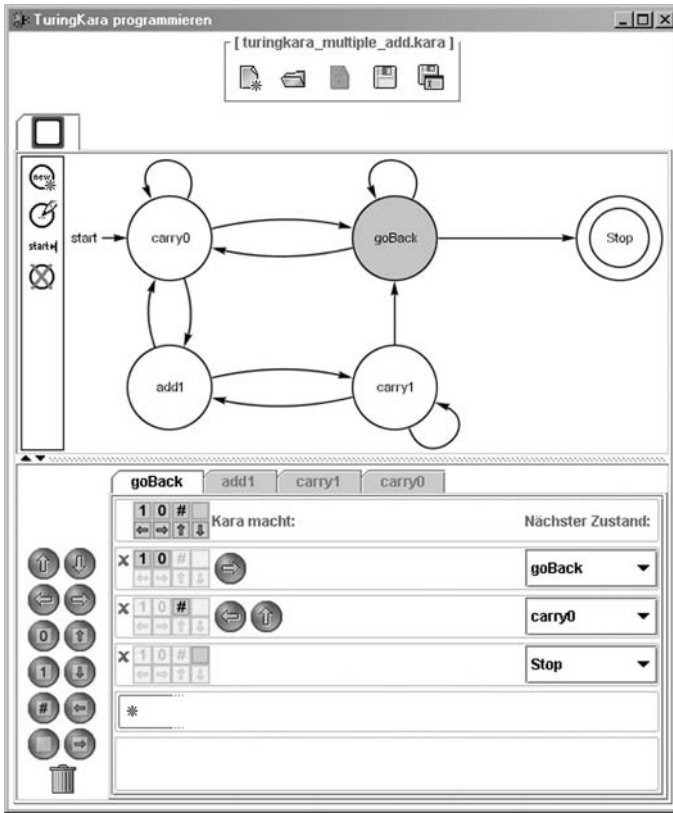


Abb. 6.2. Die TuringKara-Umgebung (Programmeditor)

Fall eines leeren Feldes der dritte Übergang. Steht der Lesekopf auf einem Pfeilsymbol, tritt ein Fehler auf.

Die Definition von TuringKara-Maschinen unterscheidet sich geringfügig von der Definition einer Standard-Turing-Maschine. In einer Standard-Turing-Maschine spezifiziert ein Übergang genau ein Symbol, das geschrieben wird, und einen Bewegungsbefehl für den Lese-/Schreibkopf. Bei TuringKara-Maschinen kann für einen Übergang eine beliebige Anzahl Schreib- und Bewegungsbefehle definiert werden. Standard- und TuringKara-Maschinen sind äquivalent, und die Umwandlung von TuringKara- zu Standard-Maschinen ist einfach. Aus didaktischer Sicht hat das TuringKara-Modell den Vorteil, dass es für viele Turing-Maschinen die Anzahl der Zustände deutlich verringert. Die Maschinen werden dadurch überschaubarer und erleichtern beim Programmieren die Konzentration auf das Wesentliche.

6.2 Beispielaufgaben

Typische Aufgabenstellungen für eindimensionale Turing-Maschinen lassen sich natürlich auch mit TuringKara lösen. Dazu gehören Entscheidungsprobleme wie „Ist die Zeichenkette auf dem Band Element der Sprache $\{0^n 1^n\}$?“. Wir lassen diese Beispiele hier auf der Seite und wenden uns Aufgaben zu, deren Lösungen die zweidimensionale Welt nutzen.

Beispiel 1: Boole'sche Operationen

Mit TuringKara lassen sich Boole'sche Funktionen wie AND, OR und XOR einfach implementieren. So lässt sich zeigen, wie einer Turing-Maschine das Rechnen beigebracht werden kann. Der Vorteil der zweidimensionalen Welt zeigt sich bei diesen Beispielen auf eindrückliche Art und Weise. Abbildung 6.3 zeigt den Ablauf einer XOR-Rechnung zweier Bitstrings in TuringKara.

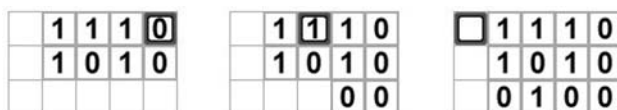


Abb. 6.3. XOR-Berechnung zweidimensional

Abbildung 6.4 zeigt eine TuringKara-Maschine, die zwei untereinander stehende binäre Bitstrings XOR rechnet. Die Maschine hat lediglich die drei Zustände `read`, `00/01` und `10/11`. Die Maschine berechnet das Resultat bitweise von rechts nach links. Der Zustand `read` liest das aktuelle Bit des ersten Bitstrings und bewegt den Lese-/Schreibkopf auf das entsprechende Bit im zweiten String. Die Zustände `00/01` und `10/11` berechnen das Resultat, schreiben es und bewegen den Lese-/Schreibkopf auf das nächste Bit des ersten Strings. Im Gegensatz zu einer eindimensionalen Lösung ist das Programm sehr einfach: Bei einer eindimensionalen Welt braucht es einige Zustände mehr, die lediglich den Lese-/Schreibkopf hin- und herbewegen.

Beispiel 2: Binäre Addition

Eine ganze Reihe von TuringKara-Beispielen veranschaulicht, wie man die Algorithmen der Primarschul-Arithmetik formal exakt darstellen kann: Addition, Subtraktion, Multiplikation und Division. In der Schule haben wir alle gelernt, wie man diese Operationen mit Hilfe von kariertem Papier durchführt. Das karierte Blatt Papier entspricht der zweidimensionalen Welt von TuringKara. Die Grundrechenarten Addition, Subtraktion und Multiplikation lassen sich in TuringKara recht einfach implementieren, einzig die Division ist etwas aufwändiger. Als erstes Beispiel betrachten wir die Addition einer beliebigen

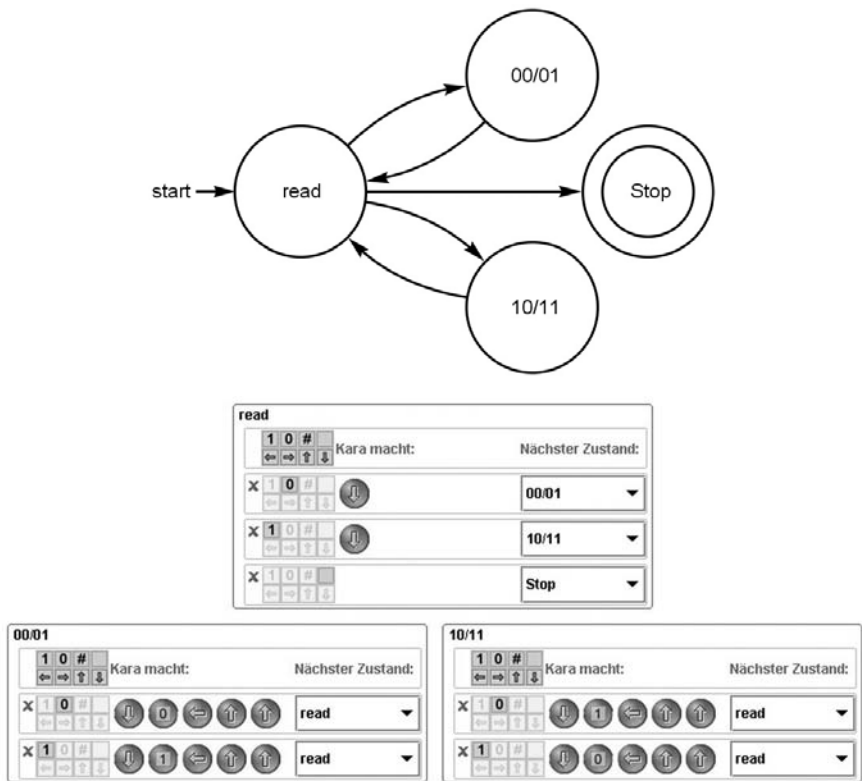


Abb. 6.4. Programm für XOR-Berechnung

Anzahl binärer Zahlen. Auch bei diesem Beispiel zeigt sich deutlich der Vorteil der zweidimensionalen Welt.

Die zu addierenden Zahlen werden untereinander geschrieben und links und rechts durch #-Markierungen begrenzt, die in einem Abstand von $k \geq 1$ Feldern angebracht sind. Die Summe wird modulo 2^k berechnet, also exakt, falls k im Vergleich zu den Summanden genügend gross ist. Die erste Zeile wird zur zweiten addiert, und die partielle Summe überschreibt diese. Danach geht es Zeile um Zeile weiter, bis zuletzt nur noch die Zeile mit der Summe übrig bleibt (Abbildung 6.5).

Das Programm für die Addition ist recht einfach (Abbildung 6.5). Der Automat startet im Zustand `carry0` und durchläuft die aktuelle Zahl von rechts nach links. Nur wenn das aktuelle Bit eine 1 ist, muss es im Zustand `add1` addiert werden. Je nachdem, ob bei dieser Addition ein Übertrag entsteht oder nicht, ist der Folgezustand `carry1` oder `carry0`. Wenn die Endmarkierung # erreicht wird, läuft der Automat im Zustand `goBack` nach rechts zurück und beginnt mit der Addition der nächsten beiden Zahlen.

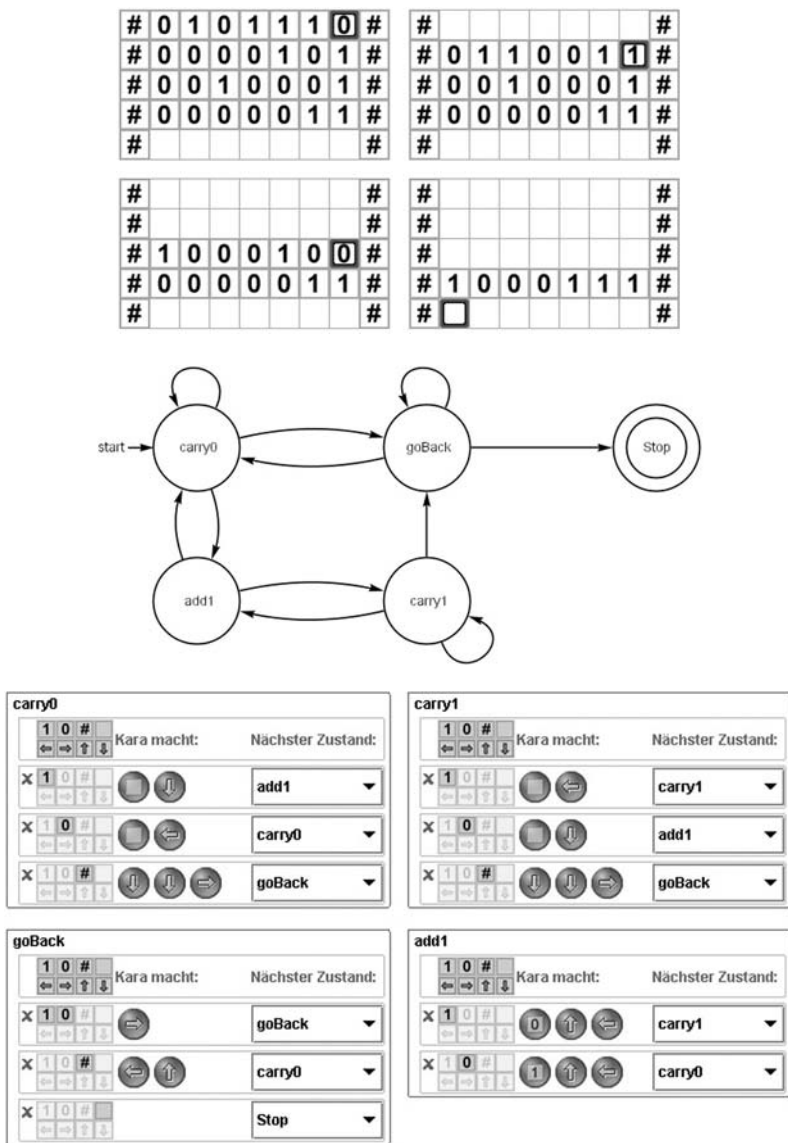


Abb. 6.5. Addition mehrerer Binärzahlen

Beispiel 3: Binäre Multiplikation

Als weiteres Beispiel zeigen wir die Multiplikation zweier Zahlen. Anstelle des aus der Schule bekannten Verfahrens verwenden wir die sogenannte Zigeuner-Multiplikation. Die erste Zahl wird halbiert, während die zweite verdoppelt wird. Ist der erste Multiplikand ungerade und hinterlässt daher die ganzzahlige Halbierung einen Rest, wird dieser Rest, das heisst, der zweite Multiplikand, zum Zwischenresultat addiert. Dieses Verfahren wird wiederholt bis der erste Multiplikand eins ist (Abbildung 6.6). Das Halbieren und das Verdoppeln einer binären Zahl kann einfach mit einem Shift nach rechts und einem Shift nach links erreicht werden.

5 * 4	1	0	1	#	1	0	0
0	0	0	0	0	0	0	
2 * 8	1	0	#	1	0	0	
4	0	0	0	0	1	0	
1 * 16	1	#	1	0	0	0	
4	0	0	0	0	1	0	
20	0	0	1	0	1	0	

Abb. 6.6. Multiplikation von 5 * 4 in TuringKara

Abbildung 6.6 zeigt den Ablauf der Multiplikation in der Welt. Die beiden Multiplikanden stehen auf der ersten Zeile getrennt durch ein #, das aktuelle Zwischenresultat steht auf der zweiten Zeile. Die Division und Multiplikation der Multiplikanden findet ausschliesslich auf der oberen Zeile statt. Dieser Algorithmus ist dank der zweidimensionalen Welt mit TuringKara relativ einfach mit nur neun Zuständen umsetzbar. Vier Zustände übernehmen die Addition und können gleichsam als Makro vom Multiplikations-Programm übernommen werden. Die anderen fünf Zustände halbieren und verdoppeln die Multiplikanden (Abbildung 6.7).

Beispiel 4: Traversieren eines Labyrinthes

Das vollständige Besuchen eines Labyrinths ist eine klassische Problemstellung der Algorithmik. Typischerweise wird zur Lösung ein rekursiver Algorithmus implementiert. Die rekursiven Aufrufe der Suchmethode speichern dabei, in welchen Richtungen das Labyrinth schon besucht wurde. Ein anderer Lösungsansatz benutzt eine Datenstruktur, in der für jedes Feld des Labyrinths festgehalten wird, aus welcher Richtung das Feld betreten wurde. Diese Methode lässt sich einfach mit TuringKara implementieren.

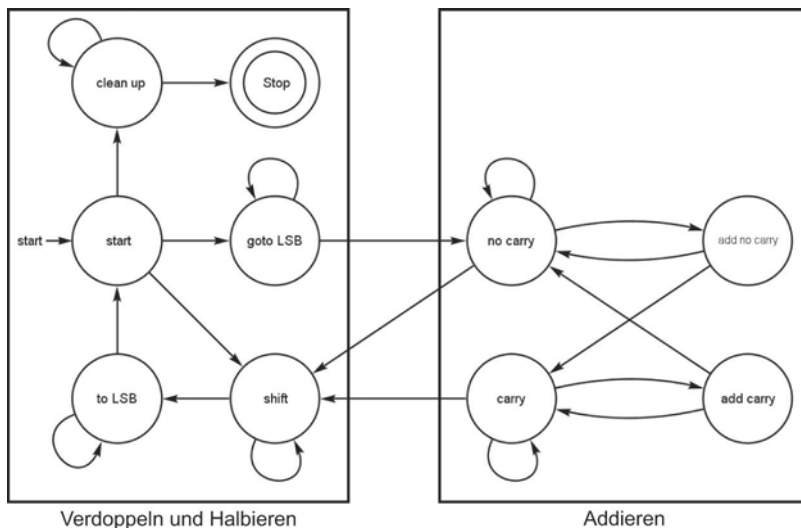


Abb. 6.7. TuringKara-Programm für Multiplikation

Abbildung 6.8 zeigt ein durch # begrenztes Labyrinth. Jedes Feld innerhalb des Labyrinths soll besucht und mit einer 0 markiert werden. Die Pfeile geben an, aus welcher Richtung ein Feld betreten wurde.

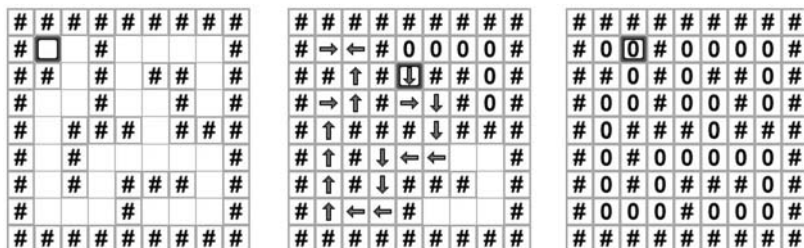
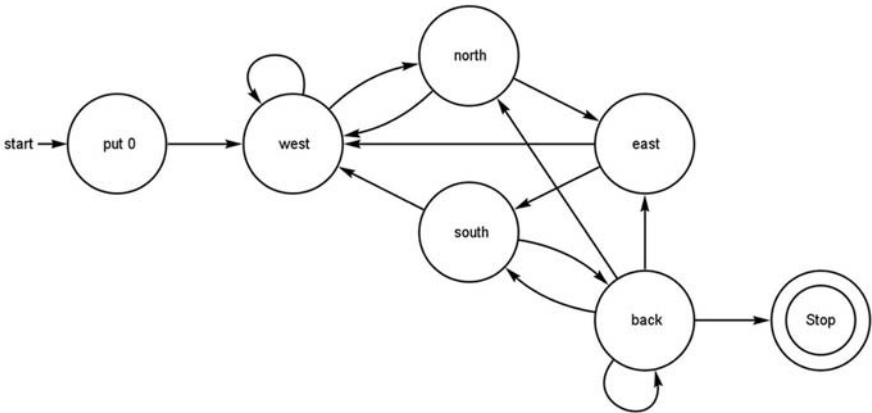


Abb. 6.8. Labyrinth vollständig mit 0 füllen

Das Programm für die Traversierung des Labyrinths ist in Abbildung 6.9 dargestellt. Es implementiert eine Standard-Tiefensuche. Der Startzustand put 0 schreibt als Stopmarkierung eine 0. Die Zustände west, north, east und south funktionieren alle nach dem gleichen Muster und suchen in den Nachbarfeldern des aktuellen Feldes, auf dem der Lese-/Schreibkopf steht, nach einem freien Feld. Im Zustand west bewegt sich der Lese-/Schreibkopf vom aktuellen Feld auf das benachbarte Feld links. Falls dieses Feld leer ist, wird ein Pfeil nach rechts geschrieben, um festzuhalten, von wo aus das Feld be-

treten wurde. Falls das Feld schon besucht wurde, geht der Lese-/Schreibkopf weiter zum Feld oberhalb des Ausgangsfeldes und wechselt in den Zustand *north*. Wird rundherum kein freies Nachbarfeld gefunden, läuft der Zustand *back* den Pfeilen entlang rückwärts, bis bei einem Feld ein freies Nachbarfeld gefunden wird. Dann beginnt die Suche von vorne.



west

10#

Kara macht:

Nächster Zustand:

X10#

0

⇨

⇩

north

X10#

⇨

⇩

west

north

10#

Kara macht:

Nächster Zustand:

X10#

⇩

⇨

east

X10#

⇩

⇨

west

back

10#

Kara macht:

Nächster Zustand:

X10#

0

⇨

⇩

north

X10#

0

⇩

⇨

east

X10#

0

⇩

⇨

south

X10#

0

⇩

⇨

back

X10#

⇨

⇩

Stop

Abb. 6.9. Auszug aus dem Programm für Labyrinth-Traversierung

6.3 Die universelle Turing-Maschine

Das Konzept der universellen Turing-Maschine (UTM) spielt in der theoretischen Informatik eine wichtige Rolle. Es illustriert die Idee von programmierbaren Maschinen und zeigt, dass es Maschinen gibt, die jede beliebige andere Maschine simulieren können.

Die UTM nimmt als Eingabe die Beschreibung einer Turing-Maschine (Programm und Band) und simuliert deren Ausführung. Wir haben in TuringKara eine UTM mit 41 Zuständen implementiert. Sie kann Standard-Turing-Maschinen simulieren, die ein eindimensionales Band mit den Symbolen 0, 1, #, \square benutzen. Obwohl die UTM selbst komplex ist, kann ihr Ablauf dank der zweidimensionalen Welt intuitiv und visuell einfach nachvollzogen werden. Auf diese Art kann mit TuringKara Studierenden ein Gefühl für das Konzept der UTM vermittelt werden.

Damit eine Turing-Maschine durch die TuringKara-UTM simuliert werden kann, muss sie zunächst in der Welt kodiert werden. Die Kodierung eines einzelnen Zustandsübergangs der zu simulierenden Turing-Maschine ist in Abbildung 6.10 dargestellt. Die zu simulierende Maschine muss eine Standard-Turing-Maschine sein. Deshalb ist für jeden Übergang von einem Zustand s_1 nach s_2 folgendes definiert: Das aktuelle Symbol a , das diesen Übergang auswählt, das zu schreibende Symbol b sowie der auszuführende Bewegungsbefehl m . Die Symbole a und b müssen im Alphabet $\{0, 1, \#, \square\}$ enthalten sein, und m muss eines der Symbole $\{\leftarrow, \square, \rightarrow\}$ sein, wobei \square für „keine Bewegung“ steht. Die Zustände werden für die Codierung binär nummeriert. Das Beispiel in Abbildung 6.10 rechts zeigt die Codierung eines Übergangs von Zustand 01 in sich selbst, sofern das aktuelle Symbol a eine 1 ist. Geschrieben wird eine 0, und der Lese-/Schreibkopf wird nach rechts bewegt.

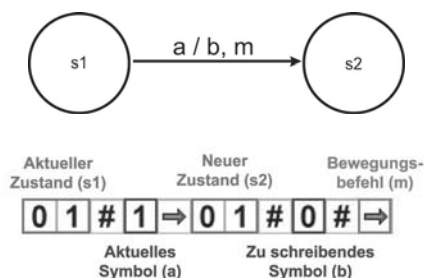


Abb. 6.10. Zustandsübergang und seine Kodierung für die TuringKara-UTM

Als ein einfaches, konkretes Beispiel kodieren wir einen Bitstring-Invertierer für die TuringKara-UTM. Der Invertierer startet am linken Ende eines Bitstrings. Er negiert so lange die einzelnen Bits, nach rechts laufend, bis er das Stoppsymbol # erreicht. Das Programm ist in Abbildung 6.11 als TuringKara-Maschine dargestellt.

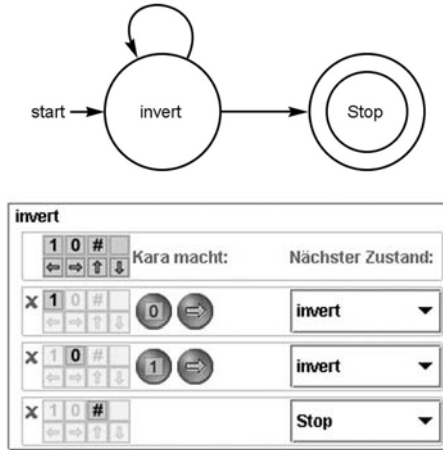


Abb. 6.11. Bitstring-Invertierer in TuringKara

Damit die TuringKara-UTM die Bitstring-Invertierer-Maschine simulieren kann, muss jeder Übergang entsprechend obigem Schema in der Welt kodiert werden. Die Zustände der Maschine müssen eindeutige Nummern haben, damit die Übergänge korrekt angegeben werden können. Zudem müssen der Startzustand und das Band in der Welt kodiert werden. Die vollständige Kodierung der Bitstring-Invertierer-Maschine ist in Abbildung 6.12 (rechts) dargestellt. Zusätzlich zur eigentlichen Kodierung werden noch einige Leerzeilen und Markierungen benötigt, damit die UTM arbeiten kann.



Abb. 6.12. Kodierung des Bitstring-Invertierers für die TuringKara-UTM

Man kann die TuringKara-UTM auf einer Welt laufen lassen, die eine codierte Turing-Maschine enthält, und zuschauen, wie sie Übergang für Übergang diese Maschine ausführt. Von der Idee her ist die Arbeit der UTM einfach und lässt sich in zwei Phasen aufteilen. In der ersten Phase ermittelt die UTM den nächsten Übergang, der durch den aktuellen Zustand und das aktuelle Symbol auf dem Band bestimmt wird. Dazu muss die UTM im der Welt das aktuelle Symbol auf dem Band betrachten und dann zu der Liste mit Übergängen laufen, um den entsprechenden Übergang zu suchen. In der zweiten Phase läuft die UTM in der Welt vom aktuellen Übergang wieder zurück zum Band, schreibt dort das Symbol dieses Übergangs auf das Band und bewegt den Lese-/Schreibkopf entsprechend. Diese beiden Phasen werden wiederholt, bis die Maschine terminiert. Abbildung 6.13 zeigt den Ablauf am Beispiel der Invertierer-Maschine. Die Abbildung macht auch deutlich, wie die UTM Informationen in der Welt speichert und verschiebt, da sie sich diese Informationen nicht im Zustandsraum merken kann. Wirklich anschaulich wird die Simulation durch die UTM allerdings erst dann, wenn man sie in TuringKara laufen lässt.

#	#	#	#	#	#	#	#	#	#
0	1	0	1	0	1	0	1	0	1

Lesen (Phase 1): Die UTM liest das aktuelle Symbol (0) des Bandes.

⇒	1	#	0						
	1	#	1	⇒	1	#	0	#	⇒

Suchen (Phase 1): Die Markierung in der Liste der Übergänge wurde gefunden. Damit beginnt die Suche nach dem nächsten Übergang. Dazu wurde das aktuelle Symbol des Bandes mitgeführt.

⇒	←								
	1	#	0	⇒	1	#	1	#	⇒

Gefunden (Phase 1): Der nächste Übergang wurde ermittelt. Damit geht die UTM über zu Phase 2.

#									
#	#	#	#	#	#	#	1	⇒	#
#									

Suchen (Phase 2): Das zu schreibende Symbol (1) und der auszuführende Bewegungsbefehl (Schritt nach rechts) werden zum Band verschoben.

1	⇒	#	#	#	#	#	#	#	#
#									
0	1	0	1	0	1	0	1	0	1

Gefunden (Phase 2): Die aktuelle Position auf dem Band wurde ermittelt. Somit kann der Übergang ausgeführt werden.

#	1								
#	#	#	#	#	#	#	#	#	#
#									
1	1	0	1	0	1	0	1	0	1

Lesen (Phase 1): Das zu schreibende Symbol wurde geschrieben, und die Markierung für die Position auf dem Band verschoben. Das aktuelle Symbol auf dem Band wird gelesen.

⇒	1	#	1						
	1	#	0	⇒	1	#	1	#	⇒

Suchen (Phase 1): Die Suche nach dem richtigen Übergang für das aktuelle Symbol beginnt erneut.

Abb. 6.13. Ausführung der TuringKara-UTM für den Bitstring-Invertierer

6.4 Nicht-berechenbar: Kara, der fleissige Biber

Universelle Berechnungsmodelle spielen in der Theoretischen Informatik eine wichtige Rolle. Die Church-Turing-These besagt, dass jede Funktion, die intuitiv „berechenbar“ ist, auch mit einer Turing-Maschine berechnet werden kann. Das Standard-Beispiel eines unentscheidbaren Problems ist das Halteproblem. Es gibt keine Turing-Maschine, die entscheiden kann, ob eine beliebige andere Turing-Maschine je anhält oder nicht.

Von Tibor Rado stammt 1962 ein Beispiel einer nicht berechenbaren Funktion [Rad62], deren Output sich gut visualisieren lässt. Man betrachtet alle Turing-Maschinen mit Alphabet $\{1, \square(\text{leeres Feld})\}$ mit n Zuständen, die beginnend auf einem leeren Band irgendwann anhalten. Der Stop-Zustand wird dabei nicht gezählt. Die Funktion $\Sigma(n)$ bezeichnet die maximale Anzahl Markierungen (nicht notwendigerweise zusammenhängend), die eine Turing-Maschine mit n Zuständen auf dem Band hinterlassen kann, bevor sie anhält. Eine Turing-Maschine, welche $\Sigma(n)$ Markierungen (Einsen) liefert, wird ein „fleissiger Biber“ genannt.

Schon Turing-Maschinen mit einer kleinen Anzahl Zustände sind in der Lage, eine sehr grosse Anzahl Markierungen auf dem Band zu hinterlegen. Die fleissigen Biber beginnen klein: $\Sigma(1) = 1$, $\Sigma(2) = 4$, $\Sigma(3) = 6$, $\Sigma(4) = 13$. H. Marxen und J. Buntrock haben gezeigt, dass der fleissige Biber mit 5 Zuständen mindestens $\Sigma(5) \geq 4098$ Markierungen und der fleissige Biber mit 6 Zuständen mindestens $\Sigma(6) \geq 1.29 \cdot 10^{865}$ Markierungen auf dem Band hinterlassen [MB90].

Die Funktion $\Sigma(n)$ ist wohldefiniert, aber nicht berechenbar. In Dewdney's *Turing Omnibus* findet sich ein Beweis der Unberechenbarkeit der Funktion $\Sigma(n)$ [Dew97]. Dewdney zitiert auch ein Resultat von M. W. Green von 1964, das eindrücklich zeigt, was unberechenbar bei diesem Beispiel bedeutet: $\Sigma(n)$ wächst schneller als jede berechenbare Funktion. Das heisst, für jede berechenbare Funktion f gibt es eine Konstante n_0 mit der Eigenschaft:

$$f(n) < \Sigma(n), \forall n > n_0$$

Fleissige Biber mit n Zuständen sind schwierig zu finden. Für jeden Zustand einer Turing-Maschine ausser dem Stop-Zustand gibt es zwei Übergänge, so dass es insgesamt $2n$ Übergänge gibt. Jeder Übergang kann eines von zwei Symbolen (1 oder \square) schreiben und kann entweder nach links oder nach rechts gehen. Jeder Übergang kann zu einem von $n + 1$ Zuständen führen. Für n Zustände gibt es damit $(4(n + 1))^{2n}$ verschiedene Turing-Maschinen. Die Zahl der Turing-Maschinen wächst also exponentiell mit der Anzahl Zustände. Zudem ist es im Allgemeinen nicht möglich zu entscheiden, ob eine gegebene Turing-Maschine anhalten wird oder nicht. Eine „Brute-Force“-Suche hat daher wenig Aussicht auf Erfolg. Dewdney gibt noch einen weiteren Grund an, warum fleissige Biber so schwierig zu finden sind [Dew97]:

One reason for the enormous difficulty of the busy beaver problem lies in the ability of relatively small Turing machines to encode profound unproved mathematical conjectures such as Fermat's last „theorem“ or the Goldbach conjecture (every even number is the sum of two primes). Knowledge of whether such machines halt is tantamount to proving or disproving such a conjecture.

Turing-Maschinen für fleissige Biber lassen sich gut mit TuringKara simulieren, zumindest solche mit nicht allzu grossen Bandlängen. Ein fleissiger Biber mit 6 Zuständen lässt sich kaum mehr simulieren, da eine Welt dieser Grösse den Speicherbedarf sprengen würde. Abbildung 6.14 zeigt das Zustandsdiagramm für einen fleissigen TuringKara mit 2 Zuständen, der 4 Markierungen auf dem Band hinterlässt.

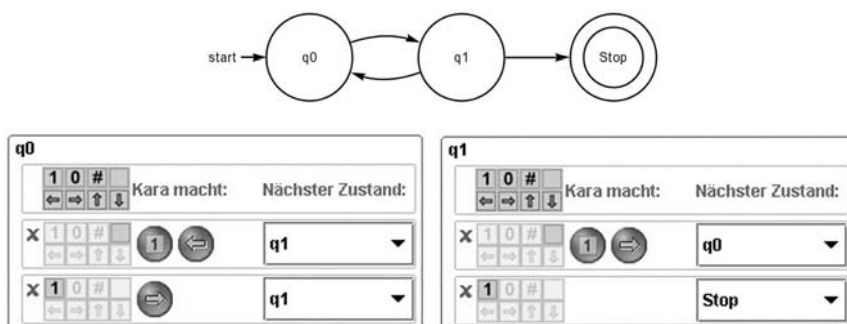


Abb. 6.14. Biber-Automat mit 2 Zuständen

Mit TuringKara lassen sich nicht nur eindimensionale Biber simulieren, sondern auch zweidimensionale. Wie viele Markierungen kann eine TuringKara-Maschine mit drei Zuständen maximal produzieren, bevor sie terminiert? Zwecks Übereinstimmung mit den Busy-Beaver-Spielregeln machen wir die folgenden Einschränkungen: nur die Symbole 1 und □ (leeres Feld) dürfen verwendet werden, und die Übergänge müssen der Definition von Standard-Turing-Maschinen entsprechen. Jeder Übergang besteht also aus genau einem Schreibbefehl gefolgt von einem Bewegungsbefehl. Abbildung 6.15 zeigt eine TuringKara-Maschine, von der wir vermuten, dass es sich um einen fleissigen TuringKara-Biber-Kandidaten mit drei Zuständen handelt. Die Maschine wurde ermittelt durch eine Brute-Force-Suche über alle TuringKara-Maschinen, die obigen Einschränkungen genügen und innerhalb von 100 Übergängen terminieren.

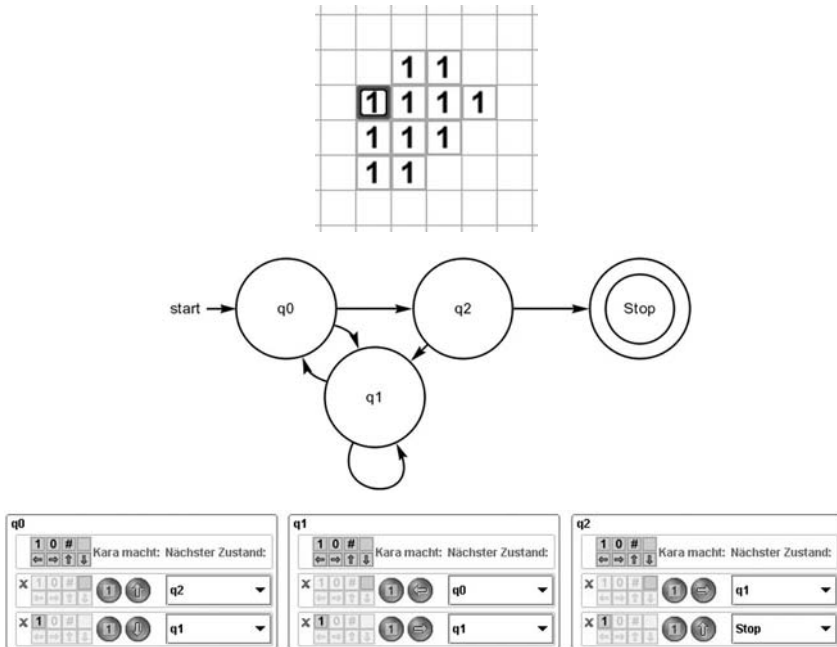


Abb. 6.15. Kandidat für fleissigen TuringKara-Biber mit drei Zuständen

Die aufgeführten Beispiele zeigen, dass sich mit TuringKara grundlegende Konzepte der Theoretischen Informatik auf lehrreiche Art vermitteln lassen. Theorie als solche muss nicht grau sein: Abstraktion ist ein äusserst nützlicher Prozess zur Reduktion auf das Wesentliche. Abstraktion heisst aber nicht, dass die zugrunde liegenden Gedankengänge in aller Allgemeinheit, eben abstrakt, vermittelt werden sollen. Gerade im Unterricht sollten die zentralen Themen an einfachen, überzeugenden Beispielen dargestellt werden.

Programmieren mit Kara

Ein spielerischer Zugang zur Informatik

Reichert, R.; Nievergelt, J.; Hartmann, W.

2005, X, 152 S., Softcover

ISBN: 978-3-540-23819-5