3 Endliche Muster und Konfigurationen

Wir gehen von nun an immer davon aus, dass $0 \in N$ ist.

3.1 Definition Eine Teilmenge $P \subseteq Q$ heißt genau dann *Ruhemenge* oder *passiv*, wenn für alle $l : N \to Q$ mit $\mathrm{ran}(l) \subseteq P$ gilt: $\delta(l) = l(0)$.

Ein Zustand $q \in Q$ heißt genau dann *Ruhezustand*, wenn $\{q\}$ Ruhemenge ist, wenn also mit anderen Worten $\delta(q, q, \ldots, q) = q$ ist.

Ein Zustand $t \in Q$ heißt genau dann *tot*, wenn für alle $l : N \to Q$ mit $l(0) = t$ gilt: $\delta(l) = t$. ◇

Wir gehen von nun an immer davon aus, dass ein ZA C einen ausgezeichneten Ruhezustand q besitzt.

3.2 Beispiele. In Beispiel 1.5 ist $\{\boxed{\blacksquare}, \boxed{}\}$ eine Ruhemenge und $\boxed{\blacksquare}$ ein toter Zustand. In Beweis 2.8 ist $\{\_\} \times B_T$ eine Ruhemenge.

Jeder Zustand einer Ruhemenge ist Ruhezustand. Aber ein ZA kann zwei Ruhezustände q_1 und q_2 besitzen, die zusammen *keine* Ruhemenge bilden.

Im folgenden bezeichne A immer ein *Eingabealphabet*.

3.3 Erinnerung. Bei Turingmaschinen (mit einem eindimensionalen Band) nimmt man an: $A \subseteq B$, die Festlegung eines Anfangszustandes und die eines *Blanksymboles* $\square$. Die *Anfangskonfiguration* zu einem Eingabewort $w = w_1 \cdots w_n \in A^n$ ist dann $c_w = (s_0, b_w, 1)$ mit

$$b_w(i) = \begin{cases} w_i & \text{falls } 1 \leqslant i \leqslant n \\ \square & \text{sonst} \end{cases} .$$

Analoges wollen wir nun für Zellularautomaten definieren. Da aber zumindest auch der zweidimensionale Fall interessiert, wird das ganze etwas komplizierter.

3.4 Für alle $n \in \mathbb{N}_+$ sei $\mathbb{N}_n = \{1, 2, \ldots, n\} \subset \mathbb{N}$.

3.5 Definition Es sei C ein d-dimensionaler ZA mit Raum R und einem ausgezeichneten Ruhezustand q. Ein (d-dimensionales endliches) *Muster* für C ist eine Abbildung $m : T \to Q \setminus \{q\}$, wobei T eine endliche Teilmenge von R ist, der sogenannte *Träger* von m.

Die zu einem Muster m gehörende *Musterkonfiguration* $c_m : R \to Q$ ist durch

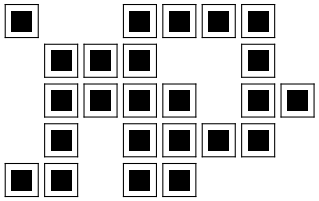
$$c_m(i) = \begin{cases} m(i) & \text{falls } i \in T \\ q & \text{sonst} \end{cases}$$

festgelegt.

Ein *Muster* $m : T \to Q \setminus \{q\}$ *kommt in einer Konfiguration* c *an der Stelle* j *vor*, wenn für alle $i \in T$ gilt: $c(j + i) = m(i)$. ◇

3.6 Beispiele. Typische eindimensionale Muster sind die *Wörter*. Ein $w : \mathbb{N}_n \to Q$ kann offensichtlich als $w \in Q^+$ aufgefasst werden und umgekehrt. Es gibt aber natürlich auch „unzusammenhängende“ eindimensionale Muster wie $m : \{1, 4, 42\} \to Q$.

Einfache Fälle zweidimensionaler Muster sind *Rechtecke*: $m : \mathbb{N}_n \times \mathbb{N}_m \to Q$. Wir werden aber gelegentlich auch mit „unförmigen“ zweidimensionalen Mustern zu tun haben:



Höherdimensionale Muster sind natürlich auch möglich. Nur ist ihre Darstellung so unschön, und in mindestens einem Fall wird auch das zu lösende Problem beim Übergang vom Zweizum Dreidimensionalen viel schwieriger.

3.7 Definition Die *Eingabe* eines Musters m in einen (passenden) ZA geschieht durch die Wahl der zu m gehörenden Musterkonfiguration als *Anfangskonfiguration*.

Eine Konfiguration c ist genau dann *Endkonfiguration*, wenn $\Delta(c) = c$ ist.

Was die *Ausgabe* (das Ergebnis) einer Berechnung sein soll, die in einer Konfiguration c endet, wollen wir nicht im Detail spezifizieren, da das von Fall zu Fall variieren aber stets naheliegend sein wird. ◇

3.8 Definition Für eine Konfiguration $c : R \to Q$ heißt die Menge $T(c) = \{i \mid c_i \neq q\}$ der *Träger* von c. Eine Konfiguration heißt genau dann *endlich*, wenn ihr Träger endlich ist. ◇

Aus endlichen Konfigurationen entstehen im Laufe einer Berechnung stets wieder endliche Konfigurationen. (Warum?)

Wir sind nun in der Lage, sowohl Satz 2.7 genauer formulieren als auch eine Aussage über die umgekehrte Simulation machen:

3.9 Satz.

1. *Zu jeder TM T gibt es einen eindimensionalen, deterministischen, synchron arbeitenden ZA C, der T in dem folgenden Sinne simuliert: Zu jeder Konfiguration c von T gibt es eine Konfiguration $\mathrm{cod}(c)$ von C derart, dass für alle Konfigurationen c von T der ZA angesetzt auf $\mathrm{cod}(c)$ in einem Schritt in die Konfiguration $\mathrm{cod}(\Delta_T(c))$ übergeht.*

2. *Zu jedem eindimensionalen, deterministischen, synchron arbeitenden ZA C gibt es eine TM T, die C in dem folgenden Sinne simuliert: Zu jeder Konfiguration c von C gibt es eine Konfiguration $\mathrm{cod}(c)$ von T derart, dass für alle endlichen (!) Konfigurationen c von C die TM angesetzt auf $\mathrm{cod}(c)$ in endlich vielen Schritten in die Konfiguration $\mathrm{cod}(\Delta_C(c))$ übergeht.*

3. *In beiden Fällen ist* cod *injektiv und berechenbar und das gleiche gilt für* $\mathrm{cod}^{-1} : \mathrm{ran}(\mathrm{cod}) \to \mathrm{dom}(\mathrm{cod})$. cod *ist sogar in intuitivem wie auch formalem Sinne „sehr einfach“.*

Man beweise als Übung den zweiten Teil dieses Satzes.

Wir wollen uns nun mit der Erkennung spezieller endlicher Muster, nämlich Wörtern, beschäftigen. Dazu wird ein Zellularautomat, der eine formale Sprache erkennt, durch ein Siebentupel $C = (R, N, Q, \delta, A, F_+, q)$ festgelegt.

3.10 Definition Es sei das *Eingabealphabet* $A \subset Q \setminus \{q\}$ und $F_+ \subset Q \setminus A$ die Menge *akzeptierender Endzustände*.

Die zu einem Wort $w = w_1 \cdots w_n \in A^+$ gehörende *Anfangskonfiguration* c_w ist die Musterkonfiguration:

$$c_w(i) = \begin{cases} w_i & \text{falls } 1 \leqslant i \leqslant |w| \text{ und } i = (i, 0, \ldots, 0) \\ q & \text{sonst} \end{cases}$$

Eine Endkonfiguration c ist *akzeptierend*, falls $c((1,0,\ldots,0)) \in F_+$ ist und andernfalls *ablehnend*.

Eine endliche Folge $c^0, c^1, \ldots, c^k$ von Konfigurationen wird genau dann eine (vollständige) *Berechnung* genannt, wenn c^k die einzige vorkommende Endkonfiguration ist und für alle $t \in \mathbb{N}$ mit $0 \leqslant t < k$ gilt: $c^{t+1} = \Delta(c^t)$.

Die *von einem Zellularautomaten erkannte Sprache* $L \subseteq A^+$ ist

$$L = \left\{ w \;\middle|\; \text{es gibt eine akzeptierende Endkonfiguration } c_f \text{ und } k \in \mathbb{N}_+ \text{ mit } \Delta^k(c_w) = c_f \right\} \qquad \diamond$$

Man überlege sich, wo die Grenzen obiger Definition sind.

3.11 Beispiel. Als erstes wollen wir eine ganz einfache formale Sprache L_a mit einem eindimensionalen Zellularautomaten mit Nachbarschaft $N = \{0,1\}$ erkennen. Das Eingabealphabet sei $A = \{a,b\}$ und $L_a = \{a\}^+$. Der Zellularautomat muss also nur überprüfen, ob in der Eingabe ein b vorkommt (und sie dann ablehnen) oder nicht. Diese Aufgabe könnte man naürlich auch mit einem endlichen Automaten erledigen, der – sei es nun von vorne oder von hinten – einmal über das Wort hinweggeht. Letzteres „simulieren“ wir mit dem Zellularautomaten. Als Zustandsmenge wählen wir $Q = \{$ a , b , ␣ , <+ , <- $\}$. Neben den Eingabesymbolen und dem Ruhezustand benutzen wir zwei weitere Zustände <+ und <- , wobei ersterer auch akzeptierender Endzustand ist: $F_+ = \{$ <+ $\}$.

Die folgende Tabelle gibt den „wesentlichen Teil“ der lokalen Überführungsfunktion wieder:

$l(0)$	$l(1)$	$\delta(l)$
a	␣	<+
b	␣	<-
a	<+	<+
b	<+	<-
a	<-	<-
b	<-	<-

In allen nicht aufgeführten Fällen sei $\delta(l) = l(0)$, d. h. eine Zelle ändere dann nicht ihren Zustand.

Es ist üblich, Beispielberechnungen eindimensionaler Zellularautomaten in der Form sogenannter *Raum-Zeit-Diagramme* anzugeben. Dabei steht in jeder Zeile der gleiche endliche Ausschnitt einer Konfiguration des Zellularautomaten zu einem Zeitpunkt und in aufeinanderfolgenden Zeilen die Ausschnitte zu aufeinanderfolgenden Zeitpunkten. Ein Raum-Zeit-Diagramm ist also etwas anderes als eine Konfiguration eines zweidimensionalen Zellularautomaten.

Für die Eingabe $w =$ aaaaa ergibt sich zum Beispiel die folgende Berechnung:

$\cdots$	␣	a	a	a	a	a	␣	$\cdots$
$\cdots$	␣	a	a	a	a	<+	␣	$\cdots$
$\cdots$	␣	a	a	a	<+	<+	␣	$\cdots$
$\cdots$	␣	a	a	<+	<+	<+	␣	$\cdots$
$\cdots$	␣	a	<+	<+	<+	<+	␣	$\cdots$
$\cdots$	␣	<+	<+	<+	<+	<+	␣	$\cdots$

Nach $|w \models 5$ Konfigurationsübergängen ist eine akzeptierende Endkonfiguration erreicht. Wie man sieht, identifiziert sich die am weitesten rechts gelegene Zelle mit einem Eingabesymbol als solche (anhand des Zustandes ␣ im rechten Nachbarn) und startet ein *Signal*, das mit einer Geschwindigkeit von einer Zelle pro Schritt nach links läuft.

Wie man an der Berechnung für eine „falsche“ Eingabe noch besser sieht, kann man ein Signal auch dazu benutzen, um Information zu transportieren, hier z. B. + bzw. −:

···	␣	a	b	a	a	a	␣	···
···	␣	a	b	a	a	<+	␣	···
···	␣	a	b	a	<+	<+	␣	···
···	␣	a	b	<+	<+	<+	␣	···
···	␣	a	<-	<+	<+	<+	␣	···
···	␣	<-	<-	<+	<+	<+	␣	···

Nun führen in einem Zellularautomaten bei einem Schritt zwar alle Zellen einen Zustandsübergang durch, in einem intuitiven Sinne sind bei obigem Beispiel aber doch nur sehr wenige Aktivitäten zu beobachten.

3.12 Beispiel. Betrachten wir daher als nächstes die formale Sprache $L_{pal} = \{vxv^R \mid v \in A^* \wedge x \in A\}$ aller Palindrome ungerader Länge. Es sei wieder $A = \{a, b\}$. Als Nachbarschaft erlauben wir $N = \{-1, 0, 1\}$.

Für ein Eingabewort muss überprüft werden, ob das erste und das letzte Symbol übereinstimmen, das zweite und das vorletzte, und so weiter. Das soll bewerkstelligt werden, indem die entsprechenden Symbole mit Hilfe von Signalen aufeinander zu geschoben und in der Mitte verglichen werden.

Es stellt sich nun aber das Problem, dass für die Zellen nur ein rein lokales Verhalten spezifiziert werden kann. Anhand ihrer Umgebung kann eine Zelle aber gar nicht feststellen, ob sie sich in der vorderen oder in der hinteren Hälfte des Eingabewortes befindet. Wohin soll ihr Symbol also geschoben werden? Und wie soll eine Zelle feststellen, ob sie genau in der Mitte des Eingabewortes sitzt und daher die Vergleiche durchführen muss?

Da das alles von Anfang an gar nicht klar ist, wir aber einen *schnellen* Algorithmus wollen, lassen wir den Zellularautomaten ein bisschen „redundant“ arbeiten. Die Eingabesymbole werden prinzipiell in beide Richtungen verschoben und alle Zellen führen Vergleiche der von links und rechts ankommenden Symbole durch.

Wie auch beim „normalen Programmieren“ üblich, strukturieren wir die Daten und verwenden als Zustandsmenge $Q = A \cup \{ \text{␣} \} \cup Q_l \times Q_r \times Q_v \times Q_{lr}$. Dabei werden die Komponenten (oder wie man auch sagt: *Register*) $Q_l = \{$ <a , <b , ␣ $\}$ und $Q_r = \{$ a> , b> , ␣ $\}$ benutzt, um Eingabesymbole nach links bzw. rechts zu schieben. Mit $Q_v = \{$ + , - $\}$ werden Vergleichsergebnisse gespeichert, und $Q_{lr} = \{$ > , < , <+ , <- , ␣ $\}$ wird benutzt, um die Mitte der Eingabe zu finden und das Gesamtvergleichsergebnis an das linke Ende zu transportieren.

Man überlege sich, welche Gestalt F_+ haben sollte.

Insgesamt haben wir $3 + 3 \cdot 3 \cdot 2 \cdot 5 = 93$ Zustände und müssten daher für $93^3 = 804357$ Zustandstripel δ angeben. Das werden wir nicht in aller Ausführlichkeit tun. Zunächst einmal

legen wir für alle $x, y, z \in A$ fest:

$l(-1)$	$l(0)$	$l(1)$	$\delta(l)$
$\sqcup$	y	z	(<y , y> , + , >)
x	y	z	(<y , y> , + , $\sqcup$)
x	y	$\sqcup$	(<y , y> , + , <)
$\sqcup$	y	$\sqcup$	(<y , y> , + , <+)

Dadurch werden im ersten Schritt die Register geeignet initialisiert. Wir stellen nun die Register einer Zelle in übereinander stehenden Kästchen dar. Ist

$$l(-1) = \begin{array}{|c|}\hline <x_l \\ \hline x_r> \\ \hline v_l \\ \hline d_l \\ \hline \end{array} \quad l(0) = \begin{array}{|c|}\hline <y_l \\ \hline y_r> \\ \hline v_m \\ \hline d_m \\ \hline \end{array} \quad l(1) = \begin{array}{|c|}\hline <z_l \\ \hline z_r> \\ \hline v_r \\ \hline d_r \\ \hline \end{array}$$

so sei

$$\delta(l) = \begin{array}{|c|}\hline <z_l \\ \hline x_r> \\ \hline v'_m \\ \hline d'_m \\ \hline \end{array}$$

wobei die dritte und vierte Komponente wie folgt festgelegt sind:

$$v'_m = \begin{cases} + & \text{falls } v_m = \ + \ \wedge z_l = x_r \\ - & \text{sonst} \end{cases}$$

und

$$d'_m = \begin{cases} d_l & \text{falls } d_m = d_r = \ \sqcup \\ d_r & \text{falls } d_m = d_l = \ \sqcup \\ <v'_m & \text{falls } d_l = \ > \ \wedge d_r = \ < \\ <- & \text{falls } d_m = \ > \ \wedge d_r = \ < \\ d_m & \text{falls } d_m \in \{ <+ , <- \} \\ \quad \vdots & \text{usw.} \end{cases}$$

Abbildung 3.1 zeigt die Berechnung für die Eingabe `bababaa`, die kein Palindrom ist. Dabei unterscheide man zwischen den einfachen Querlinien, die Register einer Zelle trennen, und doppelten Querlinien, die Konfigurationen zu aufeinanderfolgenden Zeitpunkten trennen. Zustandskomponenten $\sqcup$ sind der Einfachheit halber durch leere Kästchen dargestellt. Nicht alle Details lassen sich aus den obigen, unvollständigen Angaben für δ ableiten. Man überlege sich, was dort noch zu ergänzen ist.

Im folgenden benutzen wir die Schreibweise Δ^t für die t-fache Komposition von Δ mit sich selbst.

Was dann auch noch fehlt, ist ein Beweis, dass der Zellularautomat tatsächlich das Gewünschte leistet. Dazu mögen einige Hinweise genügen. Es seien im folgenden $w \in A^+$ ein beliebiges Eingabewort der Länge n, $i \in \mathbb{N}_n$ und für alle $t \in \mathbb{N}_0$ mit $0 \leqslant t \leqslant n$ sei $c^t := \Delta^t(c_w)$. Dann kann man per Induktion zeigen, dass unter anderem gilt:

- für die „ersten Register“ $c_i^t[1]$:

$$i \leqslant n + 1 - t \implies c_i^t[1] = w_{i+(t-1)}$$

···		b	a	b	a	b	a	a		···
···		<b	<a	<b	<a	<b	<a	<a		···
···		b>	a>	b>	a>	b>	a>	a>		···
···		+	+	+	+	+	+	+		···
···		>						<		···
···		<a	<b	<a	<b	<a	<a			···
···			b>	a>	b>	a>	b>	a>		···
···		-	+	+	+	+	-	-		···
···			>				<			···
···		<b	<a	<b	<a	<a				···
···				b>	a>	b>	a>	b>		···
···		-	-	+	+	-	-	-		···
···				>		<				···
···		<a	<b	<a	<a					···
···					b>	a>	b>	a>		···
···		-	-	-	-	-	-	-		···
···					<-					···
···		<b	<a	<a						···
···						b>	a>	b>		···
···		-	-	-	-	-	-	-		···
···				<-	<-	<-				···
···		<a	<a							···
···							b>	a>		···
···		-	-	-	-	-	-	-		···
···			<-	<-	<-	<-	<-			···
···		<a								···
···								b>		···
···		-	-	-	-	-	-	-		···
···		<-	<-	<-	<-	<-	<-	<-		···

Abbildung 3.1: Ablehnung eines Nicht-Palindromes.

- für die „zweiten Register“ $c_i^t[2]$:

$$i \geqslant t \implies c_i^t[2] = w_{i-(t-1)}$$

- für die „dritten Register“ $c_i^t[3]$:

$$t \leqslant i \leqslant n+1-t \implies \left(c_i^t[3] = \texttt{+} \iff w_{i-(t-1)} \cdots w_i \cdots w_{i+(t-1)} \in L_{pal}\right)$$

- für die „vierten Register“ $c_i^t[4]$, falls n ungerade ist:

$$t < \frac{n+1}{2} \implies c_i^t[4] = \begin{cases} > & \text{falls } i = t \\ < & \text{falls } i = n+1-t \\ \text{␣} & \text{sonst} \end{cases}$$

$$t \geqslant \frac{n+1}{2} \implies c_i^t[4] = \begin{cases} <+ & \text{falls } w \in L_{pal} \text{ und } i = n+1-t \\ <- & \text{falls } w \notin L_{pal} \text{ und } i = n+1-t \\ \text{␣} & \text{sonst} \end{cases}$$

  Der Fall eines geraden n kann analog beschrieben werden.

Hieraus folgt dann insbesondere

$$c_1^n[4] = \begin{cases} <+ & \text{falls } w \in L_{pal} \\ <- & \text{falls } w \notin L_{pal} \end{cases}.$$

Zusammenfassung

Endliche Konfigurationen sind Konfigurationen, in denen bis auf *endlich* viele Ausnahmen alle Zellen in einem ausgezeichneten Ruhezustand sind.

„Ausschnitte“ endlich vieler Zellen aus R mit der Festlegung ihrer Zustände heißen Muster. Durch Auffüllen mit Ruhezuständen erhält man die zugehörigen Musterkonfigurationen.

Bei der Erkennung formaler Sprachen erfolgt die Eingabe eines Wortes w durch Wahl der zu $(i, 0, \ldots, 0) \mapsto w_i$ gehörenden Musterkonfiguration als Anfangskonfiguration. Die Entscheidung über Annahme oder Ablehnung von w wird anhand des Zustandes von Zelle $(1, 0, \ldots, 0)$ gefällt.

Bei Beschränkung auf endliche Konfigurationen (wie z. B. bei der Erkennung formaler Sprachen) sind Turingmaschinen und Zellularautomaten äquivalent.

