# Beschreibungskomplexität von Zellularautomaten

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Biologie und Informatik
der Johann Wolfgang Goethe–Universität
in Frankfurt am Main

von

**Andreas Malcher**

aus Frankfurt am Main

Frankfurt am Main 2004
(D F 1)

vom Fachbereich Biologie und Informatik der
Johann Wolfgang Goethe-Universität als Dissertation angenommen.

| | |
|---|---|
| Dekan: | Prof. Dr. Heinz-Dieter Osiewacz |
| Gutachter: | Prof. Dr. Detlef Wotschke |
| | Prof. Dr. Jürgen Dassow |
| | PD Dr. Martin Kutrib |
| Datum der Disputation: | 19. Juli 2004 |

# Inhaltsverzeichnis

# Kapitel 1

# Einführung

In der Theorie der Automaten und der formalen Sprachen betrachtet man verschiedene Beschreibungsmechanismen wie zum Beispiel unterschiedliche Automaten- und Grammatikmodelle und untersucht diese Mechanismen hinsichtlich ihrer generativen Mächtigkeit. Ein klassisches Ergebnis ist die sogenannte Chomskyhierarchie [HU79], die verschiedenen Grammatiktypen entsprechende Sprachklassen zuordnet, die echt ineinander enthalten sind. So charakterisieren reguläre, kontextfreie und kontextsensitive Grammatiken die regulären, kontextfreien und kontextsensitiven Sprachen. Unbeschränkte Grammatiken charakterisieren die rekursiv aufzählbaren Sprachen. Für alle genannten Sprachklassen sind auch Charakterisierungen durch Automatenmodelle bekannt. So können die regulären, kontextfreien, kontextsensitiven und rekursiv aufzählbaren Sprachen durch endliche Automaten, Kellerautomaten, linear beschränkte Automaten und Turingmaschinen beschrieben werden. Beweise für diese Charakterisierungen finden sich beispielsweise in [HU79]. Alle genannten Automatenklassen haben gemeinsam, daß sie sequentielle Modelle sind, denn alle Modelle besitzen eine endliche Zustandskontrolle, eine Speicherstruktur, zum Beispiel einen Keller oder ein Band, und verarbeiten in jedem Berechnungsschritt höchstens ein Symbol der Eingabe. Ist die gesamte oder ein Teil der Eingabe gelesen und ein akzeptierender Zustand erreicht, dann wird die Eingabe akzeptiert.

Es ist eine naheliegende Verallgemeinerung, anstatt einem sequentiellen Automaten *Systeme* von sequentiellen Automaten zu betrachten. In diesem Zusammenhang tauchen sofort einige Fragen auf:

- Wird die Eingabe von diesem System sequentiell oder parallel verarbeitet?
- Wie wird die Eingabe akzeptiert? Welche Sprachklassen werden akzeptiert?
- Wie ist die Kooperation zwischen den verschiedenen Automaten organisiert? Arbeiten die einzelnen Automaten synchron oder asynchron?
- Wie kommunizieren die verschiedenen Automaten miteinander? Wie sind die Kommunikationsstrukturen konzipiert?
- Gibt es sinnvolle Beschränkungen an die „Menge“ der zu kommunizierenden Information zwischen den Automaten?

- Besteht das System aus homogenen oder heterogenen Komponenten? Besteht es beispielsweise nur aus endlichen Automaten oder aus kooperierenden endlichen Automaten und Kellerautomaten?

Neben diesen Fragen stellen sich auch Fragen bezüglich der Beschreibungskomplexität eines Systems von Automaten im Vergleich mit einem einzelnen Automaten. Natürlich kann eine formale Sprache durch viele verschiedene Mechanismen beschrieben werden. Im Rahmen der Beschreibungskomplexität wird untersucht, wie sich die Größe der Beschreibung einer Sprache verändern kann, wenn die Sprache mit verschiedenen Mechanismen beschrieben wird. Im Zusammenhang mit Systemen von Automaten werden wir untersuchen, wie und in welchem Maße die Größe einer Beschreibung durch den Einsatz von mehreren kooperierenden Automaten reduziert werden kann.

Systeme kooperierender Automaten gibt es in den unterschiedlichsten Ausprägungen und werden in der Literatur entsprechend behandelt. Exemplarisch werden wir an dieser Stelle einige Modelle kurz diskutieren. Mehrkopfautomaten [Ros66, HI68] kann man in gewisser Weise als einfachstes Modell kooperierender Automaten auffassen, denn ein endlicher Automat wird mit einer festen Anzahl von Leseköpfen ausgestattet, die sich in einer oder beiden Richtungen auf der Eingabe bewegen können. In jedem Berechnungsschritt liest ein Kopf ein Zeichen der Eingabe und der Automat nimmt aufgrund dieser Informationen und dem aktuellen Zustand einen Folgezustand an. Die Eingabe ist akzeptiert, sobald der Mehrkopfautomat einen akzeptierenden Zustand annimmt. Wir haben also ein Modell mit einer zentralen Steuereinheit und die Kooperation zwischen der Steuereinheit und den Komponenten besteht im Lesen der Eingabe und der Positionierung der Köpfe. Als ähnliches Modell wurden auch Mehrkopfkellerautomaten [HI68] betrachtet. Ein System verschiedener endlicher Automaten, die durch geeignete Protokolle miteinander kommunizieren, wird in [BZ83] und [Kle96] beschrieben, wobei in der letzten Arbeit neben verschiedenen Beschränkungen an die Kommunikationsstrukturen zwischen den Automaten und den Auswirkungen auf die generative Mächtigkeit, auch Beschreibungskomplexitätsfragen betrachtet werden. Ein parallel arbeitendes Modell kooperierender endlicher Automaten wird in [MVMM02] eingeführt und untersucht. In diesem Modell wird die Eingabe parallel gelesen und von den einzelnen Automaten verarbeitet. Die Kommunikation zwischen den Automaten ist derart, daß ein Automat bei Bedarf den aktuellen Zustand eines anderen Automaten anfordern und mit dieser Information dann weiterarbeiten kann. Unterschiedliche Varianten dieser Systeme und Ergebnisse zur generativen Mächtigkeit finden sich in [MVMM02]. In ähnlicher Weise werden kooperierende Kellerautomaten in [CVMVMV00] untersucht. Neben Systemen kooperierender Automaten werden auch Systeme kooperierender Grammatiken untersucht. Eine Zusammenfassung über Grammatiksysteme findet man in [DPR97] und [CVDKP94]. Man betrachtet dort Systeme kontextfreier Grammatiken, die nacheinander oder gleichzeitig auf einer oder mehreren Satzformen arbeiten. Es gibt Grammatiksysteme in vielen unterschiedlichen Ausprägungen. Es wird unter anderem zwischen sequentieller und paralleler Verarbeitung, zwischen der Anzahl der Komponenten und zwischen den Arbeitsweisen der einzelnen Komponenten unterschieden.

Im allgemeinen sind kooperierende Systeme generativ mächtiger als eine einzelne Komponente. So können Mehrkopfautomaten und das in [MVMM02] eingeführte parallele Automatensystem kontextfreie und kontextsensitive Sprachen akzeptieren. Weiterhin können kooperierende Kellerautomaten und Grammatiksysteme durch geeignete Kooperation kontextsensitive

Sprachen akzeptieren. In diesem Zusammenhang sollte auch festgehalten werden, daß bisweilen bereits ein kleines System, also ein System mit wenig Komponenten, sehr mächtig sein kann. Zum Beispiel wird in [Kle96] gezeigt, daß jede rekursiv aufzählbare Sprache durch zwei geeignet kommunizierende endliche Automaten akzeptiert werden kann. Auch mit zwei kooperierenden Kellerautomaten können die rekursiv aufzählbaren Sprachen charakterisiert werden [CVMVMV00].

Alle bisher betrachteten Systeme haben gemeinsam, daß sie ihre Berechnungsaufgabe auf verteilte Art und Weise lösen. Das heißt, die Gesamtaufgabe wird in Teilaufgaben zerlegt und an die unterschiedlichen Komponenten verteilt, die dann diese Teilprobleme bearbeiten. Durch geeignete Kommunikation werden die Teilberechnungen anschließend wieder zur Gesamtberechnung zusammengesetzt. Das hat zur Folge, daß die einzelnen Komponenten in der Regel unterschiedlich strukturiert sind. Beispielsweise wird im Fall kooperierender Kellerautomaten ein Kellerautomat für die Teilaufgabe $A$ anders definiert sein als ein Kellerautomat für die Teilaufgabe $B$.

In dieser Arbeit werden wir nun mit *Zellularautomaten* ein massiv paralleles Berechnungsmodell untersuchen. Im Gegensatz zu mehreren, strukturell unterschiedlichen Automaten besteht ein Zellularautomat aus sehr vielen identischen Automaten, die Zellen genannt werden. Um ein möglichst einfaches System zu erhalten, werden wir in den Zellen einfache Automaten, nämlich deterministische endliche Automaten (DFA), betrachten. Die einzelnen Zellen sind linear angeordnet und jede Zelle wird homogen mit ihrem linken und rechten Nachbarn verbunden. Die Kommunikation zwischen Zellen ist sehr eingeschränkt, denn jede Zelle erhält als Information von anderen Zellen nur den aktuellen Zustand der linken und rechten Nachbarzelle. Ein Zellularautomat arbeitet, indem zu diskreten Zeitschritten eine lokale Überführungsfunktion auf alle Zellen zum gleichen Zeitpunkt angewendet wird.

Zellularautomaten wurden von Ulam und von Neumann Mitte des vergangenen Jahrhunderts erstmals definiert und beschrieben. Die ursprüngliche Motivation war unter anderem, ein theoretisches Modell einer Maschine zu finden, die sich durch eine Berechnung selbst reproduzieren kann. Diese biologische Motivation spiegelt sich auch in einigen Eigenschaften von Zellularautomaten wieder. Ein Grundprinzip von Zellularautomaten ist, daß einfache lokale Regeln ein komplexes globales Verhalten bedingen. Dieses Prinzip läßt sich auch in der Natur wiederfinden. Beispielsweise wird das Wachstum einer Pflanze oder die Kontraktion eines Muskels vermutlich nicht durch eine zentrale Einheit gesteuert, die mit jeder physischen Zelle kommuniziert und entsprechende Impulse zum Wachstum oder zur Kontraktion sendet. Vielmehr bestehen beide Systeme aus vielen, im wesentlichen identischen physischen Zellen, die nur mit einer begrenzten Umgebung oder Nachbarschaft in Kontakt treten können. Die Kooperation zwischen den Zellen ist durch eine lokale Regel beschrieben, die zumindest auf abstrakter Ebene für alle Zellen identisch ist. Ausgehend von bestimmten Zellen, die einen Impuls bekommen, entwickelt sich dann das Gesamtsystem gemäß der lokalen Regeln und man kann ein globales Verhalten beobachten, das heißt, die Pflanze wächst und der Muskel kontrahiert sich. Daher kann das theoretische Modell eines Zellularautomaten durchaus hilfreich sein, um natürliche Phänomene aus der realen Welt zu beschreiben und zu untersuchen. In diesem Bereich gibt es viele Untersuchungen und Anwendungen beispielsweise in der Biologie, der Physik und sogar den Sozialwissenschaften. Dort wird zum Beispiel menschliches Sozialverhalten durch Zellularautomaten modelliert und analysiert [NM93]. Die Fähigkeit von

Zellularautomaten zur Selbstreproduktion wurde in [Cod68] und [Lan84] untersucht. Modellierungen biologischer Fragestellungen wie Räuber-Beute-Ökosysteme und Muster in der Natur finden sich in [vdLH92, CPPR87, CPPRS87]. Weitere genauere Informationen über diese praktischen Aspekte und Anwendungen von Zellularautomaten finden sich in [BMS01], [Wei97] und [Gar95].

Auch aus technischer Sicht sind Zellularautomaten ein interessantes Konzept, denn eine wesentliche Eigenschaft von Zellularautomaten ist ihre Einfachheit. Da alle Zellen identisch aufgebaut und homogen miteinander durch einen einfachen Mechanismus verbunden sind, sollten Zellularautomaten auch einfach auf Hardwareebene zu verwirklichen sein. In [BMS01, Wei97] werden einige Beispiele für Umsetzungen von Zellularautomaten auf Parallelrechner gegeben. Da Zellularautomaten weiterhin ein paralleles Berechnungsmodell sind und Algorithmen auf diesen Automaten bereits parallelisiert sind, ist die Realisierung auf einem Parallelrechner oftmals recht einfach. Einen Überblick über den neueren Stand der hardwaretechnischen Aspekte von Zellularautomaten findet man in [BMS01].

Nicht zuletzt sind Zellularautomaten auch aus theoretischer Sicht interessant. Einerseits kann man ein weiteres paralleles Berechnungsmodell im Vergleich mit sequentiellen Modellen untersuchen. Man kann die Vorteile und Grenzen dieser Form von Parallelität studieren. Andererseits arbeiten Zellularautomaten im Gegensatz zu vielen anderen parallelen Modellen massiv parallel und sind homogen aufgebaut. Daher kann gerade der Vergleich mit anderen parallelen Modellen zum Verstehen paralleler Prozesse beitragen. Benutzt man Zellularautomaten, um formale Sprachen zu akzeptieren, dann ist die generative Mächtigkeit der Automaten trotz ihres einfachen und homogenen Aufbaus recht groß, denn bereits das einfachste Zellularautomatenmodell kann bestimmte kontextsensitive Sprachen akzeptieren.

Zusammenfassend läßt sich sagen, daß Zellularautomaten ein interessantes Modell sowohl aus praktischer als auch aus theoretischer Sicht sind. Daher sind Fragestellungen der Beschreibungskomplexität, nämlich effiziente, kurze Darstellungen von formalen Sprachen durch Zellularautomaten zu finden, obere und untere Schranken bezüglich der Beschreibungsgröße beim Wechsel von einem zellularen Beschreibungssystem in ein anderes zu beweisen und effiziente Minimierungsalgorithmen zu entwerfen, theoretische Fragen von hoher praktischer Bedeutung. Diese Fragen werden in dieser Arbeit behandelt.

Die Beschreibungskomplexität verschiedener Automatenmodelle wird seit über dreißig Jahren untersucht. Ein grundlegendes Ergebnis ist die Arbeit von Meyer und Fischer von 1971 [MF71]. In dieser Arbeit wurde untersucht, wie sich die Größe eines endlichen Automaten, gemessen durch die Anzahl der Zustände, verändert, wenn man einen nichtdeterministischen endlichen Automaten (NFA) in einen deterministischen endlichen Automaten (DFA) umwandelt. Durch die Potenzmengenkonstruktion ergibt sich im allgemeinen eine exponentielle Vergrößerung der Zustandszahl. Meyer und Fischer haben nun nachgewiesen, daß es Sprachen gibt, die diesen exponentiellen Größenzuwachs bei der Beseitigung von Nichtdeterminismus tatsächlich verursachen. In der Folgezeit wurden auch andere Ressourcen in endlichen Automaten und die Auswirkungen auf die Beschreibungsgröße bei der Reduktion dieser Ressourcen betrachtet. Exemplarisch werden wir hier einige untersuchte Ressourcen benennen. Zunächst kann man endliche Automaten mit zweiseitiger Bewegung und das Zusammenspiel mit der Ressource Nichtdeterminismus betrachten. Ergebnisse dazu finden sich in [MF71, SS78, Bir93]. Als Erweiterung des Konzepts des Nichtdeterminismus werden in

[Lei81] alternierende endliche Automaten untersucht. Weiterhin werden eindeutige und in unterschiedlichen Ausprägungen mehrdeutige endliche Automaten in [Sch78, RI89] analysiert. Schließlich wird in [KW80, GKW90] ein Maß für den Nichtdeterminismus eines endlichen Automaten eingeführt und untersucht, wie sich graduelle Reduktionen des Nichtdeterminismus auf die Beschreibungsgröße auswirken. Unäre Sprachen stellen einen Sonderfall dar, da in diesem Fall ein endlicher Automat lediglich „zählen“ muß. Daher können dann Ergebnisse der Zahlentheorie angewendet werden. Grundlegend auf diesem Gebiet ist die Arbeit von Chrobak [Chr86].

Bereits in [MF71] wurde anhand endlicher Automaten und kontextfreier Grammatiken, die eine reguläre Sprache erzeugen, gezeigt, daß der Zuwachs der Beschreibungsgröße beim Wechsel von einem Beschreibungssystem in ein anderes durch keine rekursive Funktion beschrieben werden kann. Solche *nichtrekursiven Tradeoffs* wurden beispielsweise weiterhin zwischen eindeutigen und deterministischen Kellerautomaten [Val76] und zwischen nichtdeterministischen und eindeutigen Kellerautomaten [SS77] bewiesen. Eine wesentliche Erweiterung dieser Ergebnisse findet man in [Her97, Her99]: Dort werden Kellerautomaten mit hierarchisch anwachsendem Grad an Nichtdeterminismus und hierarchisch anwachsender Mehrdeutigkeit untersucht. Das wesentliche Ergebnis ist, daß Kellerautomaten mit mehr Nichtdeterminismus oder Mehrdeutigkeit nichtrekursiv beschränkte Einsparungen gegenüber Kellerautomaten mit weniger Nichtdeterminismus oder Mehrdeutigkeit liefern können. Ergebnisse zur Beschreibungskomplexität der eingangs erwähnten Mehrkopfautomaten und Grammatiksysteme findet man in [Kut03] und [DP91]. Ein genauerer Überblick über die Beschreibungskomplexität unterschiedlicher Automatenmodelle wird in [GKK$^+$02] gegeben.

Die Beschreibungskomplexität von Zellularautomaten und weiteren zellularen Modellen wurde in der Literatur bislang sehr wenig behandelt. In diesem Zusammenhang sind zwei Arbeiten zu nennen: In [GMNP97] werden systolische binäre Baumautomaten hinsichtlich ihrer Beschreibungskomplexität untersucht. Diese Automaten sind ebenfalls ein massiv paralleles Modell. Allerdings liegt dem Modell eine binäre Baumstruktur zugrunde und es müssen mehr als $n$ Prozessoren bereitgestellt werden, um eine Eingabe der Länge $n$ zu akzeptieren. In [Okh02] werden einige Aspekte der Beschreibungskomplexität von Trellisautomaten untersucht. Aber auch diesem Modell liegt keine lineare Struktur zugrunde. Weiterhin ist ebenfalls die Anzahl der benötigten Prozessoren größer als die Länge der Eingabe. Die in beiden Arbeiten erzielten Ergebnisse hängen sehr von der zugrundeliegenden Baum- bzw. Trellisstruktur und der sich verändernden Zahl der zur Berechnung verfügbaren Zellen ab. Im Vergleich dazu besitzen die in dieser Arbeit untersuchten zellularen Modelle eine lineare Struktur, die sich im Laufe der Berechnung auch nicht verändert. Daher sind die erzielten Ergebnisse von strukturellen Eigenschaften des Modells im wesentlichen unabhängig.

Die Arbeit läßt sich grob in zwei Teile gliedern: Im ersten Teil werden wir verschiedene zellulare Modelle mit paralleler und sequentieller Eingabe, mit unterschiedlichen Zeitkomplexitäten und mit unterschiedlichen Kommunikationsstrukturen hinsichtlich ihrer Beschreibungskomplexität untersuchen. Das wesentliche Ergebnis wird sein, daß es zwischen zwei Automatenklassen, deren entsprechende Sprachklassen entweder echt ineinander enthalten oder unvergleichbar sind, nichtrekursive Tradeoffs gibt. Im zweiten Teil werden wir das allgemeine Modell mit unterschiedlichen Einschränkungen betrachten. Hier erhalten wir aus Sicht der Beschreibungskomplexität polynomielle und damit rekursive Tradeoffs beim Wechsel zwi-

schen den verschiedenen Modellen. Im einzelnen gliedert sich die Arbeit wie folgt:

In Kapitel 2 stellen wir grundlegende Definitionen und Ergebnisse aus der Theorie der formalen Sprachen zusammen. Die in dieser Arbeit benötigten Automaten- und Grammatikmodelle der Chomskyhierarchie werden eingeführt. Weiterhin werden die Abschlußeigenschaften und Entscheidbarkeitsfragen für diese Modelle zusammengefaßt. Außerdem werden die notwendigen Konzepte und Ergebnisse aus dem Bereich der Beschreibungskomplexität und Kolmogorowkomplexität zusammengestellt.

Die in dieser Arbeit genauer untersuchten Konzepte „Zellularautomat“ und „iteratives Array“ werden in Kapitel 3 eingeführt. Wir stellen Ergebnisse hinsichtlich der generativen Mächtigkeit und Abschlußeigenschaften dieser Modelle und in späteren Kapiteln benötigte Erweiterungen zusammen.

In Kapitel 4 werden wir die Beschreibungskomplexität von Zellularautomaten untersuchen. Ein wesentliches Ergebnis wird der Nachweis nichtrekursiver Tradeoffs zwischen verschiedenen Zellularautomatenmodellen sein. Weiterhin werden Entscheidbarkeitsfragen und die Frage nach der Existenz eines Pumpinglemmas und eines Minimierungsalgorithmus für Zellularautomaten diskutiert.

Die im vierten Kapitel behandelten Fragen zur Beschreibungskomplexität von Zellularautomaten übertragen wir in Kapitel 5 auf iterative Arrays und erhalten ähnliche Ergebnisse hinsichtlich der Existenz nichtrekursiver Tradeoffs und der Unentscheidbarkeit vieler Entscheidbarkeitsfragen.

In Kapitel 6 werden Zellularautomaten mit einer beschränkten Zellenzahl und zweiseitiger Kommunikation in unterschiedlichen Ausprägungen untersucht. Die erhaltenen Modelle werden hinsichtlich ihrer generativen Mächtigkeit und ihrer Beschreibungskomplexität im Vergleich mit DFA und den unterschiedlichen Varianten beschränkter Zellularautomaten untersucht.

Die Arbeit schließt in Kapitel 7 mit einer Zusammenfassung der erreichten Ergebnisse, einigen offenen Fragen und weiteren Fragestellungen, die in dieser Arbeit nicht behandelt werden konnten.

Die Mehrzahl der in dieser Arbeit zusammengefaßten Ergebnisse ist bereits veröffentlicht worden. Die in Kapitel 4 zusammengestellten nichtrekursiven Tradeoffs und Entscheidbarkeitsfragen für Zellularautomaten wurden in [Mal02] bewiesen. Die Erweiterung und Übertragung dieser Ergebnisse auf iterative Arrays findet man in [Mal04a]. Das Modell eines Zellularautomaten mit beschränkter Zellenzahl und einseitiger Kommunikation wurde in [Mal03b] eingeführt und bezüglich der Beschreibungskomplexität untersucht. Die Erweiterung des Modell hinsichtlich zweiseitiger Kommunikation in unterschiedlicher Ausprägung und die Auswirkung auf die Beschreibungskomplexität der Modelle wurde in [Mal03c, Mal04b] untersucht.

# Kapitel 2

# Grundlagen und Definitionen

In diesem Kapitel stellen wir die allgemeinen Grundlagen, wesentliche Modelle aus der Theorie der formalen Sprachen und Grundlagen der Beschreibungskomplexität zusammen.

## 2.1 Allgemeine Grundlagen

**Definition 2.1.** Mit $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ und $\mathbb{R}$ bezeichnen wir die Mengen der natürlichen, ganzen, rationalen und reellen Zahlen. Wir vereinbaren $0 \in \mathbb{N}$.

Zwei Mengen $M$ und $M'$ heißen *äquivalent*, falls $M = M'$ ist. Die Schreibweise $M \subset M'$ bedeutet, daß $M$ eine echte Teilmenge von $M'$ ist. $M \subseteq M'$ bedeutet, daß entweder $M$ eine echte Teilmenge von $M'$ ist, oder $M$ und $M'$ äquivalent sind. Mit $\setminus$ bezeichnen wir die mengentheoretische *Differenz* und mit $|M|$ die *Kardinalität* der Menge $M$.

Eine Menge $M$ heißt *abzählbar*, falls $M$ endlich ist oder $M$ und $\mathbb{N}$ gleichmächtig sind, das heißt, wenn es eine Bijektion $f : M \to \mathbb{N}$ gibt.

Ein *Alphabet* $\Sigma$ ist eine nichtleere, endliche Menge. Ein *Wort* über dem Alphabet $\Sigma$ ist eine endliche Folge von Symbolen aus $\Sigma$. Mit $\epsilon$ bezeichnen wir das leere Wort. $\Sigma^*$ ist die Menge aller Wörter über dem Alphabet $\Sigma$. Wir definieren $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. Eine Teilmenge $L \subseteq \Sigma^*$ bezeichnen wir als *formale Sprache*. Das *Komplement* einer Sprache $L \subseteq \Sigma^*$ ist $\overline{L} = \Sigma^* \setminus L$. Ist das Alphabet $\Sigma$ nicht explizit angegeben, wird es als das kleinste Alphabet mit $L \subseteq \Sigma^*$ angenommen.

Mit $|w|$ bezeichnen wir die Länge eines Wortes $w$. Mit $|w|_x$ bezeichnen wir die Anzahl des Auftretens des Teilworts $x$ in $w$. Die *Spiegelung* (engl. reversal) eines Wortes $w = w_1 \dots w_n$ ist definiert durch $w^R = w_n \dots w_1$.

Ein Wort $u$ ist ein *Präfix* von $w$, falls es ein $v$ gibt, so daß $w = uv$. Das Präfix heißt *echt*, wenn $u \neq w$ gilt. Ein Wort $u$ ist ein *Suffix* von $w$, falls es ein $v$ gibt, so daß $w = vu$.

Unter einer *Sprachklasse* verstehen wir eine Menge von Sprachen. Zwei Sprachklassen $\mathcal{L}_1$ und $\mathcal{L}_2$ heißen *unvergleichbar*, falls gilt: $\mathcal{L}_1 \setminus \mathcal{L}_2 \neq \emptyset$ und $\mathcal{L}_2 \setminus \mathcal{L}_1 \neq \emptyset$

**Bemerkung 2.2.** In dieser Arbeit unterscheiden wir nicht, ob eine Sprache $L$ das leere Wort $\epsilon$ enthält oder nicht. Das heißt, wir identifizieren die Mengen $L$ und $L \setminus \{\epsilon\}$.

Mit REG, LCF, DCF, CF, CS, REC, RE bezeichnen wir die Sprachklassen der regulären, linear kontextfreien, deterministisch kontextfreien, kontextfreien, kontextsensitiven, rekursiven und rekursiv aufzählbaren Sprachen. Die Definitionen dieser Sprachklassen finden sich in diesem Kapitel. Wir folgen weitgehend der Darstellung in [HU79].

**Definition 2.3.** Es seien Zahlen $m \in \mathbb{N}$ mit $m > 0$ und $a, b \in \mathbb{Z}$ gegeben. $a$ ist kongruent zu $b$ modulo $m$, $a \equiv b \mod m$, wenn $m$ die Differenz $a - b$ teilt.

**Definition 2.4.** Es seien $f, g : \mathbb{N} \to \mathbb{R}$ zwei Funktionen mit $f(n) > 0$ und $g(n) > 0$ für alle $n \in \mathbb{N}$. Dann definieren wir

$$
\begin{aligned}
f(n) = O(g(n)) &\iff \exists C, m > 0 : f(n) \leq C \cdot g(n) \text{ für alle } n \geq m \\
f(n) = \Omega(g(n)) &\iff \exists C, m > 0 : f(n) \geq C \cdot g(n) \text{ für alle } n \geq m \\
f(n) = \Theta(g(n)) &\iff f(n) \in O(g(n)) \text{ und } f(n) \in \Omega(g(n)) \\
f(n) = o(g(n)) &\iff \forall C > 0 : \exists m > 0 : f(n) \leq C \cdot g(n) \text{ für alle } n \geq m
\end{aligned}
$$

Mit log bezeichnen wir die Logarithmusfunktion zur Basis 2.

Im folgenden werden verschiedene Automatenklassen eingeführt. Für einen gegebenen Automaten $A$ bezeichnet $T(A)$ die von dem Automaten akzeptierte Sprache. Zwei Automaten $A_1$ und $A_2$ heißen *äquivalent*, falls sie die gleiche Sprache akzeptieren, das heißt $T(A_1) = T(A_2)$.

## 2.2 Endliche Automaten

Die unterste Stufe der Chomskyhierarchie bilden die regulären Sprachen. Das sind die Sprachen, die von endlichen Automaten akzeptiert werden können. Ein endlicher Automat besteht aus einer endlichen Menge von Zuständen und einer endlichen Zustandskontrolle in Form einer Überführungsfunktion, die jedem Zustand und jedem Eingabesymbol einen Folgezustand zuordnet. Eine Folge von Eingabesymbolen wird akzeptiert, falls von einem bestimmten Startzustand aus unter Anwendung der Überführungsfunktion ein Zustand aus einer vorher festgelegten Menge von akzeptierenden Zuständen erreicht wird. Formal definieren wir folgendes:

**Definition 2.5.** Ein deterministischer endlicher Automat (DFA) ist ein Tupel $(Q, \Sigma, \delta, q_0, F)$, wobei folgendes gilt:

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände,

(2) $\Sigma$ ist das Eingabealphabet,

(3) $q_0 \in Q$ ist der Startzustand,

(4) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände und

(5) $\delta : Q \times \Sigma \to Q$ ist die Überführungsfunktion.

Die Überführungsfunktion $\delta$ wird folgendermaßen zu einer Abbildung $\delta : Q \times \Sigma^* \to Q$ fortgesetzt: $\delta(q, \epsilon) = q$ für alle $q \in Q$ und $\delta(q, xa) = \delta(\delta(q, x), a)$ für alle $q \in Q$, $x \in \Sigma^*$ und $a \in \Sigma$.

$T(M) = \{x \in \Sigma^* \mid \delta(q_0, x) \in F\}$ ist die von einem DFA $M$ akzeptierte Sprache.

Die von DFA akzeptierte Sprachklasse ist die Menge der regulären Sprachen ($\mathrm{REG}$).

Die *Größe* $|M|$ eines DFA $M$ ist als die Anzahl der Zustände von $M$ definiert.

Ist die Funktion $\delta$ dahingehend abgeändert, daß jedem Paar $(q, \sigma) \in Q \times \Sigma$ ein Element der Potenzmenge $2^Q$ von $Q$ zugeordnet wird, dann sprechen wir von einem nichtdeterministischen endlichen Automaten (NFA). $\delta$ wird folgendermaßen zu einer Abbildung $\delta : Q \times \Sigma^* \to 2^Q$ fortgesetzt: $\delta(q, \epsilon) = \{q\}$ für alle $q \in Q$ und $\delta(q, xa) = \bigcup_{p \in \delta(q,x)} \delta(p, a)$ für alle $q \in Q$, $x \in \Sigma^*$ und $a \in \Sigma$. Die von einem NFA $M$ akzeptierte Sprache ist $T(M) = \{x \in \Sigma^* \mid \delta(q_0, x) \cap F \neq \emptyset\}$.

**Definition 2.6.** Es sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Die Nerode-Relation $\equiv_L$ auf $\Sigma^*$ ist folgendermaßen erklärt:

$$x \equiv_L y :\Longleftrightarrow \text{Für alle } z \in \Sigma^* \text{ gilt : } xz \in L \Leftrightarrow yz \in L$$

In [HU79] wird bewiesen, daß $\equiv_L$ eine Äquivalenzrelation ist. Weiterhin ist der Index von $\equiv_L$ ($index(\equiv_L)$), also die Anzahl der verschiedenen von $\equiv_L$ erzeugten Äquivalenzklassen genau dann endlich, wenn $L$ eine reguläre Sprache ist. Außerdem gilt folgender Satz:

**Satz 2.7.** Die minimale Größe eines DFA, der die Sprache $L$ akzeptiert, entspricht dem Index der Nerode-Relation $\equiv_L$.

## 2.3 Kellerautomaten

Wird das Konzept des endlichen Automaten um einen zusätzlichen (im Prinzip unendlichen) Speicher mit Kellerstruktur erweitert, dann sprechen wir von Kellerautomaten. Ein äquivalentes Beschreibungssystem sind die kontextfreien Grammatiken. Beide Modelle werden im folgenden Abschnitt definiert.

**Definition 2.8.** Ein Kellerautomat (PDA) ist ein Tupel $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, wobei folgendes gilt:

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände,

(2) $\Sigma$ ist das Eingabealphabet,

(3) $\Gamma$ ist das Kelleralphabet,

(4) $q_0 \in Q$ ist der Startzustand,

(5) $Z_0 \in \Gamma$ ist das Startkellersymbol,

(6) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände und

(7) $\delta$ ist die Menge der Überführungsregeln; nämlich eine Abbildung von $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ in endliche Teilmengen von $Q \times \Gamma^*$.

Eine *Konfiguration* eines PDA $M$ ist $C = (q, x, w) \in Q \times \Sigma^* \times \Gamma^*$, wobei $q$ der aktuelle Zustand, $x$ die noch zu lesende Eingabe und $w$ der aktuelle Kellerinhalt (mit dem obersten Symbol links) ist. Eine Konfiguration $C = (q_0, x, Z_0)$ wird Startkonfiguration für die Eingabe $x$ genannt. Eine Konfiguration $C = (q, \epsilon, w)$ mit $q \in F$ wird eine akzeptierende Konfiguration genannt. Eine Konfiguration $C'$ ist eine Folgekonfiguration von $C = (q, ax, Zw)$, falls es eine Überführungsregel $(p, w') \in \delta(q, a, Z)$ gibt mit $C' = (p, x, w'w)$.

Zwischen Konfigurationen ist eine *Überführungsrelation* $\vdash^M$ definiert, indem $C \vdash^M C'$ genau dann gilt, wenn $C'$ eine Folgekonfiguration von $C$ ist. Mit $\vdash^M_*$ bezeichnen wir die reflexive und transitive Hülle von $\vdash^M$.

Ein *Schritt* ist ein Paar von Konfigurationen $\mu = (C, C')$ mit $C \vdash^M C'$.

Eine *Berechnung* ist eine Folge von Schritten $\mu = \mu_1 \dots \mu_n$ mit $\mu_i = (C_{i-1}, C_i)$ für $i = 1, \dots, n$. Eine Berechnung $\mu = (C_0, C_1) \dots (C_{n-1}, C_n)$ heißt *akzeptierende Berechnung* für die Eingabe $x$, falls $C_0$ Startkonfiguration für $x$ und $C_n$ akzeptierende Konfiguration ist.

Die Sprache, die von $M$ mit akzeptierenden Zuständen akzeptiert wird, ist die Menge aller Wörter aus $\Sigma^*$, auf denen eine akzeptierende Berechnung von $M$ existiert:

$$T_f(M) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^M_* (q, \epsilon, w) \text{ für ein } q \in F, w \in \Gamma^*\}$$

Die Sprache, die von $M$ mit leerem Keller akzeptiert wird, ist die Menge aller Wörter aus $\Sigma^*$, für die es Berechnungen in $M$ gibt, die in der Startkonfiguration beginnen und in einer Konfiguration enden, in der die gesamte Eingabe gelesen und der Keller geleert ist:

$$T_e(M) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^M_* (q, \epsilon, \epsilon) \text{ für ein } q \in Q\}$$

Die Menge der von PDA akzeptierten Sprachen, die mit akzeptierenden Zuständen oder leerem Keller akzeptiert werden, nennen wir die *kontextfreien Sprachen* ($\mathrm{CF}$).

Ein PDA heißt *deterministisch* (DPDA), falls zu jeder Konfiguration höchstens eine Folgekonfiguration existiert. Das heißt, für alle $q \in Q, a \in \Sigma$ und $Z \in \Gamma$ gilt: $|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$

Die Sprache, die von einem DPDA $M$ akzeptiert wird, ist folgendermaßen definiert:

$$T_f(M) = \{x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^M_* (q, \epsilon, w) \text{ für ein } q \in F, w \in \Gamma^*\}$$

Die von DPDA akzeptierte Sprachklasse ist die Menge der deterministisch kontextfreien Sprachen ($\mathrm{DCF}$).

Als $\epsilon$-Regel bezeichnen wir eine Überführungsregel, in der kein Eingabesymbol gelesen wird. Einen DPDA, der in jeder akzeptierenden Berechnung keine $\epsilon$-Regel anwendet, bezeichnen wir als einen DPDA ohne $\epsilon$-Regeln ($\mathrm{DPDA}_\epsilon$). Die von $\mathrm{DPDA}_\epsilon$ akzeptierte Sprachklasse bezeichen wir mit $\mathrm{DCF}_\epsilon$.

Ein PDA macht eine *Wendung*, wenn für eine Folge von Konfigurationen $(q_1, x_1, w_1)$, $(q_2, x_2, w_2)$ und $(q_3, x_3, w_3)$ gilt, daß die Höhe des Kellers $|w_2|$ echt größer ist als $|w_1|$ und $|w_3|$. Gilt für einen PDA, daß in jeder Berechnung nach einem Schritt, in dem die Kellerhöhe sinkt, nie mehr ein Schritt folgt, in dem die Kellerhöhe wieder steigt, dann macht ein solcher PDA höchstens eine Wendung und wird als 1-turn PDA bezeichnet.

Einige der eingeführten Kellerautomatenmodelle lassen sich durch geeignete Grammatikmodelle charakterisieren. Daher definieren wir zunächst, was unter einer kontextfreien Grammatik zu verstehen ist.

**Definition 2.9.** Eine kontextfreie Grammatik (CFG) ist ein Tupel $(V, \Sigma, P, S)$, wobei $V$ eine endliche Menge von Symbolen, $\Sigma \subseteq V$ das Terminalalphabet, $P \subseteq (V \setminus \Sigma) \times V^*$ die Menge der Produktionen und $S \in V \setminus \Sigma$ das Startsymbol ist. Die Symbole aus $V \setminus \Sigma$ werden Nichtterminale genannt.

Zwischen Wörtern $u, v \in V^*$ ist eine *Ableitungsrelation* $\Rightarrow_G$ erklärt, indem $u \Rightarrow_G v$ genau dann gilt, wenn es eine Produktion $A \to w$ gibt, so daß $u = u_1 A u_2$ und $v = u_1 w u_2$ gilt. Mit $\Rightarrow_G^*$ bezeichnen wir die reflexive und transitive Hülle von $\Rightarrow_G$.

Die von einer CFG erzeugte Sprache ist die Menge aller Wörter, die vom Startsymbol $S$ aus mittels der Relation $\Rightarrow_G$ abzuleiten sind:

$$L(G) = \{x \in \Sigma^* \mid S \Rightarrow_G^* x\}$$

**Definition 2.10.** Eine kontextfreie Grammatik heißt linear kontextfreie Grammatik (LCFG), wenn in allen rechten Seiten von Produktionen höchstens ein Nichtterminal vorkommt. Eine Sprache heißt linear kontextfreie Sprache (LCF), wenn sie von einer LCFG erzeugt wird.

Es sei $k \geq 2$. Eine kontextfreie Grammatik heißt eine $k$-linear kontextfreie Grammatik ($k$-LCFG), wenn das Startsymbol $S$ in keiner rechten Seite einer Produktion auftritt, alle Produktionen für das Startsymbol $S$ von der Form $S \to A_1 \dots A_n$ mit $A_1, \dots, A_n \in V \setminus \{S\}$ und $n \leq k$ sind, und in allen rechten Seiten der übrigen Produktionen höchstens ein Nichtterminal vorkommt. Eine Sprache heißt $k$-linear kontextfreie Sprache ($k$-LCF), wenn sie von einer $k$-LCFG erzeugt wird.

Wir zitieren nun einige grundlegende Ergebnisse zum Zusammenhang zwischen Kellerautomaten und kontextfreien Grammatiken. Zunächst ist für PDA die Akzeptanz mit leerem Keller und die Akzeptanz mit akzeptierenden Zuständen äquivalent:

**Satz 2.11.** [HU79] Zu jeder Sprache $L \in \mathrm{CF}$ existieren PDA $M$ und $M'$, so daß $L = T_e(M) = T_f(M')$.

Weiterhin sind Kellerautomaten und kontextfreie Grammatiken äquivalente Beschreibungsmechanismen für die kontextfreien Sprachen.

**Satz 2.12.** [HU79] Zu jeder CFG $G$ gibt es einen PDA $M$ mit $L(G) = T_e(M)$. Zu jedem PDA $M$ gibt es eine CFG $G$ mit $T_e(M) = L(G)$.

Ein ähnliches Ergebnis ist für die linear kontextfreien Sprachen bekannt, die durch 1-turn PDA und linear kontextfreie Grammatiken beschrieben werden können.

**Satz 2.13.** Zu jeder LCFG $G$ gibt es einen 1-turn PDA $M$ mit $L(G) = T_e(M)$. Zu jedem 1-turn PDA $M$ gibt es eine LCFG $G$ mit $T_e(M) = L(G)$.

## 2.4 Turingmaschinen

Das allgemeinste Modell, das im Rahmen der Chomskyhierarchie betrachtet wird, ist die Turingmaschine. Sie besteht aus einem unendlichen Arbeitsband, das anfangs mit der Eingabe beschriftet ist, und einem Lese- und Schreibkopf, der sich in zwei Richtungen bewegen kann. Weiterhin gibt es eine endliche Zustandskontrolle, die deterministisch oder nichtdeterministisch einem Zustand und einem Bandsymbol einen Folgezustand, ein Ausgabebandsymbol und eine Bewegungsrichtung des Kopfes zuordnet. Eine Eingabe wird akzeptiert, wenn ausgehend von einer Startkonfiguration nach einigen Berechnungsschritten ein akzeptierender Zustand erreicht wird. Da wir in dieser Arbeit fast ausschließlich deterministische Turingmaschinen betrachten werden, verstehen wir unter dem Begriff Turingmaschine eine deterministische Turingmaschine. Anderenfalls werden wir es ausdrücklich erwähnen.

**Definition 2.14.** Eine deterministische Turingmaschine (TM) ist ein Tupel $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, wobei folgendes gilt:

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände,

(2) $\Sigma \subseteq \Gamma \setminus \{B\}$ ist das Eingabealphabet,

(3) $\Gamma$ ist das Bandalphabet,

(4) $q_0 \in Q$ ist der Startzustand,

(5) $B \in \Gamma$ ist das Blanksymbol,

(6) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände und

(7) $\delta$ ist die Menge der Überführungsregeln dargestellt durch eine Abbildung von $Q \times \Gamma$ nach $Q \times \Gamma \times \{L, R\}$.

Eine *Konfiguration* einer TM $M$ ist $C = \gamma_1 q \gamma_2 \in \Gamma^* \times Q \times \Gamma^*$, wobei $q$ der aktuelle Zustand ist und $\gamma_1\gamma_2$ der nichtleere Teil des Bandes beginnend mit dem ersten Nichtblanksymbol von links und endend mit dem am weitesten rechtsstehenden Nichtblanksymbol. Dazwischen können Blanksymbole auf dem Band stehen. Das Symbol unter dem Lesekopf ist das erste Symbol von $\gamma_2$ oder das Blanksymbol, falls $\gamma_2 = \epsilon$.

Zwischen Konfigurationen ist eine *Überführungsrelation* $\vdash^M$ definiert. Für alle $\gamma_1, \gamma_2 \in \Gamma^*$ und $X, Z \in \Gamma$ gilt:

$$
\begin{array}{lcll}
\gamma_1 Z q X \gamma_2 & \vdash^M & \gamma_1 p Z Y \gamma_2 & \text{falls } \delta(q, X) = (p, Y, L) \\
\gamma_1 Z q & \vdash^M & \gamma_1 p Z Y & \text{falls } \delta(q, B) = (p, Y, L) \\
\gamma_1 q X \gamma_2 & \vdash^M & \gamma_1 Y p \gamma_2 & \text{falls } \delta(q, X) = (p, Y, R) \\
\gamma_1 q & \vdash^M & \gamma_1 Y p & \text{falls } \delta(q, B) = (p, Y, R)
\end{array}
$$

Gilt $C \vdash^M C'$, dann heißt $C'$ Folgekonfiguration von $C$. Mit $\vdash^M_*$ bezeichnen wir die reflexive und transitive Hülle von $\vdash^M$. Eine Konfiguration $q_0 x$ wird Startkonfiguration für die Eingabe $x$ genannt. Eine Konfiguration $\gamma_1 p \gamma_2$ mit $p \in F$ wird eine akzeptierende Konfiguration genannt.

Wir sagen, daß eine TM mit einer Konfiguration $C$ anhält, wenn von $C$ aus keine Folgekonfiguration erreicht werden kann. Hält eine TM ausgehend von einer Startkonfiguration mit einer akzeptierenden Konfiguration, wird die Eingabe akzeptiert und ansonsten nicht akzeptiert.

Die von einer TM $M$ akzeptierte Sprache ist

$$T(M) = \{x \in \Sigma^* \mid q_0 x \vdash_*^M \gamma_1 p \gamma_2 \text{ für ein } p \in F, \gamma_1, \gamma_2 \in \Gamma^*\}$$

Die von TM akzeptierte Sprachklasse ist die Klasse der rekursiv aufzählbaren Sprachen (RE). Die Teilmenge der Sprachen aus RE, deren Komplement ebenfalls aus RE ist, wird die Klasse der rekursiven oder entscheidbaren Sprachen (REC) genannt. Mit DCS bezeichnen wir die Sprachklasse der deterministisch kontextsensitiven Sprachen, das sind die Sprachen, die von TM akzeptiert werden, deren Platz linear durch die Länge der Eingabe beschränkt ist.

Es folgt unmittelbar aus der Definition, daß jede rekursive Sprache von einer Turingmaschine akzeptiert werden kann, die auf allen Eingaben hält und entscheidet, ob die Eingabe akzeptiert ist oder nicht. Jede rekursiv aufzählbare Sprache wird von einer Turingmaschine akzeptiert, die zumindest auf den positiven Eingaben, das heißt den Eingaben, die in der Sprache sind, anhält.

Für die späteren Beweise wird es wichtig sein, daß man ohne Beschränkung der Allgemeinheit annehmen darf, daß eine Turingmaschine bestimmte Eigenschaften besitzt. Dazu dient folgendes Lemma:

**Lemma 2.15.** Jede Turingmaschine $M$ kann effektiv in eine äquivalente Turingmaschine $M'$ mit folgenden Eigenschaften modifiziert werden:

(1) $M'$ schreibt nie das Blanksymbol.

(2) In jeder Berechnung in $M'$ werden mindestens zwei Berechnungsschritte ausgeführt.

(3) $M'$ akzeptiert nie nach einer ungeraden Anzahl von Berechnungsschritten.

*Beweis.* Wir folgen dem Beweis aus [Her99]. Alle Eigenschaften lassen sich in der angegebenen Reihenfolge Schritt für Schritt herstellen, ohne bisher erreichte Eigenschaften wieder zu zerstören.

(1) $M'$ schreibt ein neues Blanksymbol $B'$ statt des Blanksymbols $B$ von $M$. Für $B'$ werden die gleichen Regeln angewendet wie für $B$.

(2) Es werden zwei neue Zustände eingeführt und der Start der Berechnung wird um zwei Schritte verzögert.

(3) $M'$ merkt sich in der Zustandsmenge, ob eine gerade oder ungerade Anzahl von Berechnungsschritten ausgeführt wurde. Wird in $M$ nach einer geraden Anzahl von Berechnungsschritten ein akzeptierender Zustand erreicht, dann wird in $M'$ zunächst in einen zusätzlichen, nichtakzeptierenden Zustand gegangen, von dem aus dann in den ursprünglichen akzeptierenden Zustand gewechselt wird. Dadurch wird die akzeptierende Berechnung um einen Berechnungsschritt verlängert.

Damit besitzt $M'$ alle verlangten Eigenschaften. □

Mit dem Begriff der Turingmaschine hängt die Frage zusammen, ob bestimmte Eigenschaften einer Sprache algorithmisch bestimmt, das heißt von einer Turingmaschine errechnet werden können.

**Definition 2.16.** Es sei $S$ eine Teilmenge der rekursiv aufzählbaren Sprachen. Dann nennen wir $S$ eine *Eigenschaft* rekursiv aufzählbarer Sprachen. Eine Eigenschaft $S$ heißt *trivial*, wenn $S$ leer ist oder alle rekursiv aufzählbaren Mengen enthält.

Eine Sprache $L$ besitzt die Eigenschaft $S$, falls $L \in S$ gilt. Sei $L_S$ die Menge $\{<M> |\ T(M) \in S\}$, wobei $<M>$ die Kodierung einer Turingmaschine $M$ ist. Dann heißt eine Eigenschaft $S$

- entscheidbar, falls $L_S$ eine rekursive Sprache ist;
- unentscheidbar, falls $L_S$ keine rekursive Sprache ist;
- semientscheidbar, falls $L_S$ eine rekursiv aufzählbare Sprache ist;
- co-semientscheidbar, falls das Komplement $\overline{L_S}$ eine rekursiv aufzählbare Sprache ist;

**Satz 2.17.** (Satz von Rice) [HU79] Jede nichttriviale Eigenschaft der rekursiv aufzählbaren Mengen ist unentscheidbar.

**Satz 2.18.** (Satz von Rice für rekursiv aufzählbare Indexmengen) [HU79] $L_S$ ist rekursiv aufzählbar $\Longleftrightarrow$

(1) Gilt $L \in S$ und $L \subseteq L'$ für eine rekursiv aufzählbare Sprache $L'$, dann ist auch $L' \in S$.

(2) Ist $L \in S$ eine unendliche Menge, dann gibt es auch eine endliche Teilmenge $L' \subset L$ mit $L' \in S$.

(3) Die Menge der endlichen Sprachen in $S$ ist abzählbar.

Wir betrachten nun die Menge der gültigen und ungültigen Berechnungen einer Turingmaschine. Mit Hilfe dieser Mengen werden wir später Beschreibungskomplexitätsergebnisse sowie Unentscheidbarkeits- und Nichtsemientscheidbarkeitsaussagen beweisen können.

**Definition 2.19.** Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ eine Turingmaschine mit den Eigenschaften aus Lemma 2.15. Eine *gültige Berechnung* der Turingmaschine $M$ auf $w \in \Sigma^*$ ist ein Wort

$$ID_0(w)\#ID_1(w)^R\#ID_2(w)\#ID_3(w)^R\#\ldots\#ID_{2m}(w)\#$$

wobei folgendes gilt:

(1) Für $0 \leq i \leq 2m$ ist $ID_i(w) \in \Gamma^*Q\Gamma^*$ eine Konfiguration von $M$, die nicht mit dem Blanksymbol $B$ endet.

(2) $ID_0(w) = q_0w \in \{q_0\}\Sigma^*$ ist eine Startkonfiguration.

(3) $ID_{2m}(w) \in \Gamma^*F\Gamma^*$ ist eine akzeptierende Konfiguration.

(4) $ID_{i+1}(w)$ ist die Folgekonfiguration von $ID_i(w)$, das heißt $ID_i(w) \stackrel{M}{\vdash} ID_{i+1}(w)$.

Mit $\mathrm{VALC}[M]$ bezeichnen wir die Menge aller gültigen Berechnungen einer Turingmaschine $M$. Die Menge $\mathrm{INVALC}[M]$ der ungültigen Berechnungen einer Turingmaschine $M$ ist das Komplement der Menge $\mathrm{VALC}[M]$ bezüglich des Alphabets $\Gamma \cup Q \cup \{\#\}$.

## 2.5 Abschlußeigenschaften

Da die Kenntnis, ob eine Sprachklasse gegenüber gewissen Operationen abgeschlossen ist oder nicht, ein sehr effizientes Hilfsmittel für Konstruktionen und Beweise ist, werden wir in diesem Abschnitt die wesentlichen Operationen definieren und die Ergebnisse bezüglich der Sprachklassen der Chomskyhierarchie zusammenfassen.

**Definition 2.20.** Eine Sprachklasse $\mathcal{L}$ heißt abgeschlossen unter

- *Vereinigung*, falls für alle $L_1, L_2 \in \mathcal{L}$ gilt: $L_1 \cup L_2 \in \mathcal{L}$
- *Schnitt*, falls für alle $L_1, L_2 \in \mathcal{L}$ gilt: $L_1 \cap L_2 \in \mathcal{L}$
- *Komplement*, falls für alle $L \in \mathcal{L}$ gilt: $\overline{L} \in \mathcal{L}$
- *Homomorphismus*, falls für alle $L \in \mathcal{L}$ und jeden Homomorphismus $h$ gilt: $h(L) \in \mathcal{L}$
- *$\epsilon$-freiem Homomorphismus*, falls für alle $L \in \mathcal{L}$ und jeden $\epsilon$-freien Homomorphismus $h_\epsilon$ gilt: $h_\epsilon(L) \in \mathcal{L}$
- *inversem Homomorphismus*, falls für alle $L \in \mathcal{L}$ und jeden Homomorphismus $h$ gilt: $h^{-1}(L) \in \mathcal{L}$
- *Schnitt mit regulären Mengen*, falls für alle $L \in \mathcal{L}$ und $R \in \mathrm{REG}$ gilt: $L \cap R \in \mathcal{L}$
- *Konkatenation*, falls für alle $L_1, L_2 \in \mathcal{L}$ gilt: $L_1 \cdot L_2 \in \mathcal{L}$
- *Linkskonkatenation mit regulären Mengen*, falls für alle $L \in \mathcal{L}$ und $R \in \mathrm{REG}$ gilt: $R \cdot L \in \mathcal{L}$
- *Rechtskonkatenation mit regulären Mengen*, falls für alle $L \in \mathcal{L}$ und $R \in \mathrm{REG}$ gilt: $L \cdot R \in \mathcal{L}$
- *Kleene-Abschluß*, falls für alle $L \in \mathcal{L}$ gilt: $\bigcup_{i \geq 0} L^i \in \mathcal{L}$
- *Spiegelung*, falls für alle $L \in \mathcal{L}$ gilt: $L^R \in \mathcal{L}$
- *markierter Konkatenation*, falls für alle $L_1, L_2 \in \mathcal{L}$ und ein neues Symbol $c$ gilt: $L_1 c L_2 \in \mathcal{L}$
- *markiertem Kleene-Abschluß*, falls für alle $L \in \mathcal{L}$ und ein neues Symbol $c$ gilt: $\bigcup_{i \geq 0} (Lc)^i \in \mathcal{L}$

Eine Sprachklasse, die unter Homomorphismus, inversem Homomorphismus, Vereinigung, Schnitt mit $\mathrm{REG}$, Konkatenation und Kleene-Abschluß abgeschlossen ist, nennen wir eine *volle AFL* (abstract family of languages).

**Satz 2.21.** Für die Sprachklassen $\mathrm{REG}$, $\mathrm{LCF}$, $\mathrm{DCF}$, $\mathrm{CF}$, $\mathrm{CS}$, $\mathrm{REC}$ und $\mathrm{RE}$ gelten die in Tabelle 2.1 zusammengefaßten Abschlußeigenschaften.

| | REG | LCF | DCF | CF | CS | REC | RE |
|---|---|---|---|---|---|---|---|
| Vereinigung | ja | ja | nein | ja | ja | ja | ja |
| Schnitt | ja | nein | nein | nein | ja | ja | ja |
| Komplement | ja | nein | ja | nein | ja | ja | nein |
| Homomorphismus | ja | ja | nein | ja | nein | nein | ja |
| inv. Homomorphismus | ja | ja | ja | ja | ja | ja | ja |
| Schnitt mit REG | ja | ja | ja | ja | ja | ja | ja |
| Konkatenation | ja | nein | nein | ja | ja | ja | ja |
| Kleene-Abschluß | ja | nein | nein | ja | ja | ja | ja |
| Spiegelung | ja | ja | nein | ja | ja | ja | ja |

Tabelle 2.1: Abschlußeigenschaften der Sprachklassen der Chomskyhierarchie

## 2.6 Entscheidbarkeitsfragen

Eine naheliegende Frage zu einem formalen Automatenmodell ist, wie gut das Modell für praktische Zwecke eingesetzt werden kann. Eine praktische Frage bei der Spezifizierung eines Problems durch Automaten ist zum Beispiel, ob zwei Automatenspezifikationen äquivalent sind, das heißt, ob beide Automaten das gleiche Problem beschreiben. Im folgenden werden solche Entscheidbarkeitsfragen kurz definiert und Ergebnisse bezüglich der Automatenklassen der Chomskyhierarchie zusammengefaßt.

**Definition 2.22.** Es seien $A$ und $A'$ zwei Automaten aus einer Automatenklasse $\mathcal{A}$. Wir betrachten folgende Entscheidbarkeitsfragen:

(1) *Äquivalenz:* Ist $T(A) = T(A')$?

(2) *Inklusion:* Ist $T(A) \subseteq T(A')$?

(3) *Leere:* Ist $T(A) = \emptyset$?

(4) *Universalität:* Ist $T(A) = \Sigma^*$?

(5) *Endlichkeit:* Ist $|T(A)| < \infty$?

(6) *Regularität:* Ist $T(A) \in \mathrm{REG}$?

**Satz 2.23.** Für die Sprachklassen REG, LCF, DCF, CF, CS, REC und RE gelten die in Tabelle 2.2 zusammengefaßten Entscheidbarkeitsfragen.

## 2.7 Beschreibungskomplexität

Die Beschreibungskomplexität formaler Modelle ist ein Teilgebiet der theoretischen Informatik, in dem die Beschreibungsgröße verschiedener Automaten- oder Grammatikmodelle untersucht und miteinander verglichen wird. Eine zentrale Frage ist, wie sich die Beschreibungsgröße einer Sprache verändert, wenn diese Sprache durch verschiedene Beschreibungssysteme

| | REG | LCF | DCF | CF | CS | REC | RE |
|---|---|---|---|---|---|---|---|
| Äquivalenz | ja | nein | ja | nein | nein | nein | nein |
| Inklusion | ja | nein | ja | nein | nein | nein | nein |
| Leere | ja | ja | ja | ja | nein | nein | nein |
| Universalität | ja | nein | nein | nein | nein | nein | nein |
| Endlichkeit | ja | ja | ja | ja | nein | nein | nein |
| Regularität | trivial | nein | ja | nein | nein | nein | nein |

Tabelle 2.2: Entscheidbarkeitsfragen für Sprachklassen der Chomskyhierarchie

beschrieben wird. Man untersucht, wie knapp ein Modell eine formale Sprache im Vergleich mit anderen Modellen beschreiben kann. Ein grundlegendes Ergebnis zu diesen Fragen kommt von Meyer und Fischer [MF71], die folgendes bewiesen haben: Es gibt eine unendliche Folge regulärer Sprachen $(L_n)_{n\geq 1}$, so daß jede Sprache $L_n$ von einem NFA mit $n$ Zuständen akzeptiert wird, aber jeder äquivalente DFA mindestens $2^n$ Zustände benötigt. Andererseits weiß man, daß jeder NFA mit $n$ Zuständen gemäß der Potenzmengenkonstruktion in einen DFA mit höchstens $2^n$ Zuständen umgewandelt werden kann. Also zeigt das Ergebnis von Meyer und Fischer, daß es einen exponentiellen Größenunterschied oder *Tradeoff* zwischen DFA und NFA gibt.

In [MF71] wird außerdem gezeigt, daß der Tradeoff zwischen zwei Beschreibungssystemen unbeschränkt sein kann. Es wird dort bewiesen, daß der Tradeoff zwischen kontextfreien Grammatiken, die eine reguläre Sprache erzeugen, und DFA durch keine rekursive Funktion beschränkt ist. Ein solcher Tradeoff wird nichtrekursiver Tradeoff genannt.

Im folgenden werden wir die Begriffe und Notationen genauer einführen und folgen dabei der Darstellung in [GKK$^+$02].

**Definition 2.24.** Ein *Beschreibungssystem* $D$ ist eine rekursive Menge endlicher Deskriptoren, wobei jeder Deskriptor $A \in D$ eine Sprache $T(A)$ beschreibt. Weiterhin kann jeder Deskriptor $A \in D$ effektiv in eine Turingmaschine $M_A$ umgewandelt werden, die $T(A)$ akzeptiert, das heißt $T(M_A) = T(A)$. Die von einem Beschreibungssystem $D$ beschriebene Sprachklasse ist $T(D) = \{T(A) \mid A \in D\}$. Für jede Sprache $L$ definieren wir $D(L) = \{A \in D \mid T(A) = L\}$.

**Definition 2.25.** Ein *Komplexitätsmaß* für ein Beschreibungssystem $D$ ist eine totale und rekursive Funktion $|\cdot| : D \to \mathbb{N}$. Das heißt, $\{|A| \mid A \in D\}$ ist eine rekursive Menge. Außerdem gilt, daß für ein festes Alphabet die Funktion $|\cdot|$ immer ein endliches Urbild besitzt und die Deskriptoren aus $D$ bezüglich wachsender Komplexität rekursiv aufzählbar sind. Die Forderung nach einem endlichen Urbild stellt sicher, daß es für ein festes Alphabet immer nur endlich viele Deskriptoren einer gewissen Größe gibt. $|A|$ ist die *Komplexität* oder *Größe* eines Deskriptors $A \in D$.

Beim Vergleich zweier Beschreibungssysteme $D_1$ und $D_2$ fordern wir, daß der Schnitt beider beschriebener Sprachklassen $T(D_1) \cap T(D_2)$ nicht leer ist.

**Definition 2.26.** Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(n) \geq n$ heißt eine *obere Schranke* für den Tradeoff zwischen $D_1$ und $D_2$, falls für alle $L \in T(D_1) \cap T(D_2)$ folgendes gilt:

$$\min\{|A| \mid A \in D_2(L)\} \leq f(\min\{|A| \mid A \in D_1(L)\})$$

Eine Funktion $g : \mathbb{N} \to \mathbb{N}$ mit $g(n) \geq n$ heißt eine *untere Schranke* für den Tradeoff zwischen $D_1$ und $D_2$, falls für unendlich viele $L \in T(D_1) \cap T(D_2)$ folgendes gilt:

$$\min\{|A| \mid A \in D_2(L)\} \geq g(\min\{|A| \mid A \in D_1(L)\})$$

Notation:

$$\begin{array}{ccc} D_1 & \longrightarrow & D_2 \\ n & & \leq f(n) \\ n & & \geq g(n) \end{array}$$

Ist keine rekursive Funktion eine obere Schranke für den Tradeoff zwischen zwei Beschreibungssystemen $D_1$ und $D_2$, dann heißt der Tradeoff *nichtrekursiv*. Notation: $D_1 \overset{nonrec}{\longrightarrow} D_2$

Eine obere Schranke $f(n)$ zwischen zwei Beschreibungssystemen $D_1$ und $D_2$ bedeutet folgendes: Wenn eine Sprache $L$ in beiden Beschreibungssystemen beschrieben werden kann, dann gibt es eine Beschreibung $A \in D_1$ der Größe $|A|$ und eine äquivalente Beschreibung $B \in D_2$ der Größe höchstens $f(|A|)$. Eine untere Schranke $g(n)$ zwischen zwei Beschreibungssystemen $D_1$ und $D_2$ bedeutet, daß es unendlich viele Sprachen gibt, die in beiden Beschreibungssystemen beschrieben werden können, und der Wechsel zwischen beiden Beschreibungssystemen führt bei diesen Sprachen immer zu einem Größenzuwachs um mindestens die Funktion $g(n)$.

**Beispiel 2.27.** Ein klassisches Beispiel ist der Tradeoff zwischen deterministischen und nichtdeterministischen endlichen Automaten. Es sei $D_1$ die Menge aller NFA und $D_2$ die Menge aller DFA. Als Komplexitätsmaß $|M|$ eines Automaten $M$ definieren wir die Anzahl der Zustände in $M$. Wir können beobachten, daß $D_1$, $D_2$ und $|\cdot|$ alle oben genannten Anforderungen an ein Beschreibungssystem und im anderen Fall an ein Komplexitätsmaß erfüllen. Durch die Potenzmengenkonstruktion (vgl. [HU79]) kann ein NFA mit $n$ Zuständen in einen DFA mit höchstens $2^n$ Zuständen umgewandelt werden. Daher ist $f(n) = 2^n$ eine obere Schranke für den Tradeoff zwischen NFA und DFA. Meyer und Fischer haben 1971 [MF71] bewiesen, daß es eine unendliche Folge regulärer Sprachen $(L_n)_{n\geq 1}$ gibt, für die folgendes gilt: Jede Sprache $L_n$ wird von einem NFA mit $n$ Zuständen akzeptiert, aber jeder DFA für $L_n$ benötigt mindestens $2^n$ Zustände. Also ist $g(n) = 2^n$ eine untere Schranke für den Tradeoff zwischen NFA und DFA.

Offenbar gilt bei einem nichtrekursiven Tradeoff, daß jede rekursive Funktion eine untere Schranke ist. Im Gegensatz zur Definition einer unteren oder oberen Schranke fällt bei der Definition eines nichtrekursiven Tradeoffs auf, daß dort von keinem bestimmten Komplexitätsmaß die Rede ist. Dies ist aber keine Einschränkung: Bei einem nichtrekursiven Tradeoff impliziert der Wechsel von einem Beschreibungssystem zu einem anderen einen Zuwachs in der Komplexität der Beschreibung, der durch keine rekursive Funktion beschränkt werden kann. Da nach Definition jedes Komplexitätsmaß eine rekursive Funktion ist, wird also der Zuwachs in der Komplexität der Beschreibung nach wie vor durch keine rekursive Funktion beschränkt. Daher sind nichtrekursive Tradeoffs unabhängig von bestimmten (festen) Komplexitätsmaßen.

In [Har79, Har80] wird eine elegante Technik zum Nachweis nichtrekursiver Tradeoffs vorgestellt. Diese Technik wird in folgendem Satz etwas verallgemeinert und hilft in den folgenden Kapiteln beim Nachweis nichtrekursiver Tradeoffs.

**Satz 2.28.** Seien $D_1$ und $D_2$ zwei Beschreibungssysteme. Wenn zu jeder Turingmaschine $M$ eine Sprache $L_M \in T(D_1)$ und eine Beschreibung $A_M \in D_1$ von $L_M$ effektiv konstruiert werden kann, so daß gilt: „$L_M \in T(D_2) \Leftrightarrow T(M)$ ist endlich“, dann ist der Tradeoff zwischen $D_1$ und $D_2$ nichtrekursiv.

*Beweis.* Für einen Widerspruchsbeweis nehmen wir zunächst an, daß der Tradeoff zwischen $D_1$ und $D_2$ rekursiv ist. Dann gibt es eine rekursive Funktion $f$, die eine obere Schranke darstellt. Gibt es also für eine Sprache $L \in T(D_2)$ eine Beschreibung $A \in D_1$ mit $L = T(A)$, dann gibt es eine Beschreibung $B \in D_2$, für die gilt: $T(B) = L$ und $|B| \leq f(|A|)$. Da $f$ und $|\cdot|$ rekursive Funktionen sind, ist der Wert $f(|A|)$ berechenbar. Gemäß den Voraussetzungen an die Funktion $|\cdot|$ können wir nun zu gegebenem $A \in D_1$ alle Beschreibungen $B_1, \ldots, B_n$ in $D_2$ mit $|B_i| \leq f(|A|)$ für $1 \leq i \leq n$ rekursiv aufzählen.

Dann gilt aber:

$$\begin{aligned} T(A) \notin T(D_2) &\iff \forall 1 \leq i \leq n : T(A) \neq T(B_i) \\ &\iff \forall 1 \leq i \leq n : \exists x_i : x_i \in T(A) \setminus T(B_i) \text{ oder } x_i \in T(B_i) \setminus T(A) \end{aligned}$$

Wir wandeln nun alle Beschreibungen $B_1, \ldots, B_n$ effektiv in Turingmaschinen $M_{B_1}, \ldots, M_{B_n}$ um, so daß $T(M_{B_i}) = T(B_i)$ für $1 \leq i \leq n$ gilt. Wir konstruieren eine Turingmaschine, die $A$ und alle Turingmaschinen $M_{B_1}, \ldots, M_{B_n}$ simulieren kann. Für jedes $i$ mit $1 \leq i \leq n$ werden nacheinander alle Eingaben aus $\Sigma^*$ überprüft, bis ein $x_i \in T(A) \setminus T(M_{B_i})$ oder $x_i \in T(M_{B_i}) \setminus T(A)$ gefunden ist. Sobald für jedes $i$ solch ein $x_i$ gefunden ist, hält die Turingmaschine an und akzeptiert die Eingabe. Damit haben wir eine Turingmaschine konstruiert, die anhält und akzeptiert, wenn $T(A) \notin T(D_2)$ ist. Daher ist die Menge $R = \{A \mid A \in D_1, T(A) \notin T(D_2)\}$ rekursiv aufzählbar.

Wir konstruieren nun eine Turingmaschine $M'$, die als Eingabe die Kodierung $<M>$ einer Turingmaschine $M$ bekommt. Anschließend konstruiert $M'$ effektiv die Sprache $L_M$ und eine Beschreibung $A_M \in D_1$ von $L_M$. Dann wird überprüft, ob $A_M \in R$ gilt. Ist dies der Fall, hält $M'$ an und akzeptiert die Eingabe. Nach der Voraussetzung hält $M'$ daher an und akzeptiert die Eingabe, wenn $T(M)$ unendlich ist. Also ist die Menge

$$\{M \mid M \text{ ist Turingmaschine und } T(M) \text{ ist unendlich}\}$$

rekursiv aufzählbar. Das ist aber ein Widerspruch zur Bedingung (2) in Satz 2.18 (Satz von Rice für rekursiv aufzählbare Indexmengen). Also muß der Tradeoff bereits nichtrekursiv gewesen sein. □

**Beispiel 2.29.** Als Beispiel und Anwendung des Satzes untersuchen wir den Tradeoff zwischen den Beschreibungssystemen $D_1$ bestehend aus der Menge aller kontextfreien Grammatiken (CFG) und $D_2$ bestehend aus der Menge aller DFA. Die Existenz eines nichtrekursiven Tradeoffs zwischen diesen beiden Beschreibungssystemen wurde erstmals in [MF71] nachgewiesen. Hier wollen wir nun den nichtrekursiven Tradeoff mittels Satz 2.28 beweisen. Dazu benötigen wir zunächst zwei Ergebnisse aus [HU79]: Zum einen wird dort gezeigt, daß zu jeder Turingmaschine $M$ eine kontextfreie Grammatik für die Sprache $L_M = \text{INVALC}[M]$ effektiv konstruiert werden kann. Andererseits wird dort weiterhin gezeigt, daß $L_M$ genau dann regulär ist, wenn die Turingmaschine $M$ eine endliche Menge akzeptiert. Damit sind alle Voraussetzungen aus Satz 2.28 erfüllt, wir können den Satz anwenden und erhalten somit einen nichtrekursiven Tradeoff zwischen den Beschreibungssystemen CFG und DFA.

## 2.8 Kolmogorowkomplexität

In diesem Abschnitt fassen wir kurz einige Begriffe und Ergebnisse aus dem Bereich der Kolmogorowkomplexität zusammen, die wir zur Anwendung eines Inkompressibilitätsargumentes in einem späteren Kapitel benötigen. Wir folgen der Darstellung in [LV93], einer umfangreichen Einführung in die Theorie der Kolmogorowkomplexität. Außerdem wird dort die Inkompressibilitätsmethode vorgestellt und anhand vieler Beispiele erläutert.

Im Rahmen der Theorie der Kolmogorowkomplexität werden Objekte in der Regel als Wörter aus $\{0,1\}^*$ dargestellt. Es seien zwei Objekte $x, y \in \{0,1\}^*$ gegeben. Intuitiv ist die Kolmogorowkomplexität $C_\phi(x|y)$ definiert als die minimale Größe eines Programms, das $x$ unter Kenntnis von $y$ im Rahmen eines Beschreibungsmechanismus $\phi$ beschreibt, wobei $\phi$ im wesentlichen einer Turingmaschine entspricht. Existiert kein solches Programm, dann ist $C_\phi(x|y) = \infty$.

Ein wesentliches Ergebnis ist der *Invarianz-Satz*, der besagt, daß es einen Beschreibungsmechanismus $\phi_0$ gibt, so daß es zu jedem anderen Beschreibungsmechanismus $\phi$ eine Konstante $c_\phi \geq 0$ gibt mit

$$C_{\phi_0}(x|y) \leq C_\phi(x|y) + c_\phi \text{ für alle } x, y \in \{0,1\}^*$$

Mit anderen Worten ist die Kolmogorowkomplexität eines Objektes bis auf eine additive Konstante vom jeweiligen Beschreibungsmechanismus unabhängig. Daher schreibt man $C(x|y)$ und spricht von der Kolmogorowkomplexität von $x$ abhängig von $y$. Ist $x$ von einem gegebenen $y$ unabhängig, definieren wir $C(x) = C(x|\epsilon)$ sprechen von der Kolmogorowkomplexität von $x$. Da die Kolmogorowkomplexität eines Objektes also vom Beschreibungsmechanismus im wesentlichen unabhängig ist, kann man auch von einer „strukturlosen“ Beschreibungskomplexität sprechen.

Als Beispiel betrachten wir die folgende Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(0) = 1$ und $f(i) = 2^{f(i-1)}$ für $i \geq 1$. Offenbar wächst die Funktion $f$ sehr schnell, das heißt, die Funktionswerte werden schnell sehr groß. Allerdings ist die Beschreibung eines Worts $x = 1^{f(k)}$ mit $k \in \mathbb{N}$ sehr einfach: Der Funktionswert $f(k)$ ist das $k$-fache Potenzieren von 2. Daher ist Größe einer minimalen Beschreibung von $x$ durch die Größe einer minimalen Beschreibung von $k$ und einer additiven Konstante $c$, die nicht von $k$ abhängt zu beschreiben. Damit gilt $C(x) \leq C(k) + c$. Wir haben also ein sehr langes Wort mit einer kleinen Kolmogorowkomplexität gefunden und bezeichnen $x$ daher als *kompressibel*.

Im Gegenzug nennen wir ein Wort $x$ *inkompressibel*, wenn $C(x) \geq |x|$ gilt. Ein fundamentales Ergebnis ist, daß es immer inkompressible Wörter beliebiger Länge gibt, denn es gilt folgender *Inkompressibilitäts-Satz*:

**Satz 2.30.** [LV93] Für alle $n \in \mathbb{N}$ und $y \in \{0,1\}^*$ gibt es ein $x \in \{0,1\}^*$ mit $C(x|y) \geq |x| = n$.

In ähnlicher Weise kann auch bewiesen werden, daß es unendlich viele inkompressible natürliche Zahlen gibt. Das heißt, es gibt unendlich viele $n \in \mathbb{N}$ mit $C(n) \geq \log n$.

**Beispiel 2.31.** Als Beispiel für die Anwendung eines Inkompressibilitätsarguments zeigen wir, daß die Sprache $L = \{0^n 1^n \mid n \geq 1\}$ nicht regulär ist (vgl. [LV93]). Angenommen

die Sprache $L$ ist regulär. Dann gibt es einen DFA $M$ mit $T(M) = L$. Es sei $0^n1^n \in L$. Dann ist der Zustand $q$, der nach Lesen von $0^n$ angenommen wird, zusammen mit einer Beschreibung von $M$ eine Beschreibung der Zahl $n$, denn $n$ ist dann gerade die Anzahl an Einsen, die $M$ lesen muß, um von $q$ aus in einen akzeptierenden Zustand zu gelangen. Die Größe einer Beschreibung von $M$ und $q$ ist offenbar durch eine Konstante $c$ beschränkt, die nicht von $n$ abhängt. Damit gilt $C(n) \leq c + O(1)$. Da es aber unendlich viele inkompressible natürliche Zahlen gibt, können wir ein $n \in \mathbb{N}$ finden mit $C(n) \geq \log n > c + O(1)$. Das ist ein Widerspruch. Daher kann $L$ nicht regulär gewesen sein.

# Kapitel 3

# Zellularautomaten

In diesem Kapitel werden die zellularen Modelle „Zellularautomat“ und „iteratives Array“ eingeführt und es wird erklärt, wie diese Automaten formale Sprachen akzeptieren können. Schließlich werden grundlegende Ergebnisse zusammengefaßt und weitergehende Modelle betrachtet. Die Notationen und Definitionen der beiden Modelle folgen weitgehend der Darstellung in [Kut01a] und [Kut01b]. In dieser Arbeit benutzen wir den Begriff Zellularautomat als Obergriff und Synonym für zellulare Modelle, aber auch als ein paralleles Berechnungsmodell mit paralleler Eingabe, das im nächsten Abschnitt definiert wird.

## 3.1 Definition von Zellularautomaten

Ein Zellularautomat besteht aus einer Menge identischer DFA, die in einer Reihe angeordnet sind. Die Automaten an den beiden Enden sind durch einen Grenzzustand # markiert. Jeder DFA in dieser Reihe wird eine Zelle genannt und ist mit seinem linken und rechten Nachbarn verbunden. Die Arbeitsweise eines Zellularautomaten ist folgendermaßen: Anfangs steht die Eingabe in den Zellen. Von links nach rechts steht jeweils ein Eingabesymbol in jeder Zelle. Daher benötigen wir so viele Zellen, wie die Eingabe lang ist. In diskreten Zeitschritten wird nun eine lokale Überführungsfunktion auf jede Zelle zur gleichen Zeit angewendet. Der nächste Zustand jeder Zelle hängt vom Zustand der Zelle selbst und den Zuständen des linken und rechten Nachbarn ab. Somit sind Zellularautomaten ein paralleles Berechnungsmodell mit paralleler Verarbeitung der Eingabe. Formal definieren wir:

**Definition 3.1.** Ein Zellularautomat, abgekürzt CA (cellular automaton), ist ein Quintupel $(Q, \#, \Sigma, \delta, F)$, wobei folgendes gilt:

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände der Zellen,

(2) $\# \notin Q$ ist der Grenzzustand,

(3) $\Sigma \subseteq Q$ ist das Eingabealphabet,

(4) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände und

(5) $\delta : (Q \cup \{\#\}) \times Q \times (Q \cup \{\#\}) \to Q$ ist die lokale Überführungsfunktion.

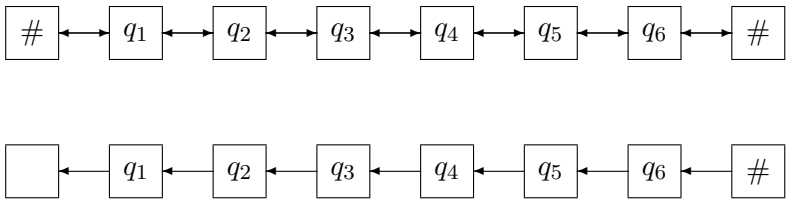



Abbildung 3.1: Ein CA (oben) und ein OCA (unten).

Da in CA jede Zelle mit ihrem linken und rechten Nachbarn verbunden ist, haben wir in diesen Automaten eine beidseitige Kommunikation zwischen den Zellen oder einen zweiseitigen Informationsfluß. Es ist eine naheliegende Beschränkung zu fordern, daß jede Zelle nur von einer Seite Informationen erhält. In diesem Fall ist jede Zelle nur mit ihrem rechten Nachbarn verbunden und die lokale Überführungsfunktion $\delta$ ist dann eine Abbildung von $Q \times (Q \cup \{\#\})$ nach $Q$. Diese Zellularautomaten mit einseitiger Kommunikation oder einseitigem Informationsfluß werden OCA (one-way cellular automata) abgekürzt.

In Abbildung 3.1 ist oben ein Zellularautomat mit zweiseitiger Kommunikation (CA) dargestellt. Unten ist ein Zellularautomat mit einseitiger Kommunikation (OCA) abgebildet. Die linke Randzelle ist im Zustand $q_1 \in Q$ und die rechte Randzelle im Zustand $q_6 \in Q$.

Zur Vereinfachung werden wir im folgenden Zellen mit natürlichen Zahlen bezeichnen.

Eine Konfiguration eines Zellularautomaten zu einem bestimmten Zeitpunkt $t \geq 0$ ist eine Beschreibung des Gesamtzustands des Automaten. Formal definieren wir eine Abbildung $c_t : \{1, \ldots, n\} \to Q$ mit $n \in \mathbb{N}$. Die Anfangskonfiguration zum Zeitpunkt 0 wird durch die Eingabe $w = x_1 \ldots x_n$ definiert: $c_{0,w}(i) = x_i$, $1 \leq i \leq n$.

Während einer Berechnung nimmt ein (O)CA nacheinander verschiedene Konfigurationen an. Die Folgekonfigurationen werden gemäß der globalen Überführungsfunktion $\Delta$ berechnet. Sei $c_t$ mit $t \geq 0$ eine Konfiguration; dann ist die Folgekonfiguration folgendermaßen für CA (oben) und OCA (unten) definiert:

$$
\begin{aligned}
c_{t+1} \;=\;& \Delta(c_t) \Longleftrightarrow \\
& c_{t+1}(1) = \delta(\#, c_t(1), c_t(2)) \\
& c_{t+1}(i) \;= \delta(c_t(i-1), c_t(i), c_t(i+1)), i \in \{2, \ldots, n-1\} \\
& c_{t+1}(n) = \delta(c_t(n-1), c_t(n), \#) \\
c_{t+1} \;=\;& \Delta(c_t) \Longleftrightarrow \\
& c_{t+1}(i) \;= \delta(c_t(i), c_t(i+1)), i \in \{1, \ldots, n-1\} \\
& c_{t+1}(n) = \delta(c_t(n), \#)
\end{aligned}
$$

Wir sagen, daß ein (O)CA eine Eingabe $w$ akzeptiert, falls es einen Zeitpunkt $i$ während der Berechnung gibt, an dem die linke Randzelle einen akzeptierenden Zustand $f \in F$ annimmt.

**Definition 3.2.** Sei $A = (Q, \#, \Sigma, \delta, F)$ ein (O)CA.

(1) Ein Wort $w \in \Sigma^+$ wird von $A$ akzeptiert, falls es einen Zeitpunkt $i \in \mathbb{N}$ gibt mit $c_i(1) \in F$ für die Konfiguration $c_i = \Delta^i(c_{0,w})$.

(2) $T(A) = \{w \in \Sigma^+ \mid w \text{ wird von } A \text{ akzeptiert}\}$ bezeichnet die von $A$ akzeptierte Sprache.

(3) Sei $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, eine rekursive Funktion und $i_w$ der minimale Zeitpunkt, an dem $A$ das Wort $w \in T(A)$ akzeptiert. Falls alle $w \in T(A)$ innerhalb von $i_w \leq t(|w|)$ Zeitschritten akzeptiert werden, dann sagen wir, daß $T(A)$ die Zeitkomplexität $t$ hat.

(4) $\mathcal{L}_t(\mathrm{OCA}) = \{L \mid L \text{ wird von einem OCA mit Zeitkomplexität } t \text{ akzeptiert}\}$
$\mathcal{L}_t(\mathrm{CA}) = \{L \mid L \text{ wird von einem CA mit Zeitkomplexität } t \text{ akzeptiert}\}$

(5) Falls $t(n) = n$, dann sagen wir, daß die Sprachen in Realzeit akzeptiert werden. Die entsprechenden Sprachklassen bezeichnen wir mit $\mathcal{L}_{rt}(\mathrm{OCA})$ und $\mathcal{L}_{rt}(\mathrm{CA})$. Die in Linearzeit akzeptierten Sprachen $\mathcal{L}_{lt}(\mathrm{OCA})$ und $\mathcal{L}_{lt}(\mathrm{CA})$ sind folgendermaßen definiert: $\mathcal{L}_{lt}((\mathrm{O})\mathrm{CA}) = \bigcup_{k\in\mathbb{Q},k\geq 1} \mathcal{L}_{\lceil k\cdot t\rceil}((\mathrm{O})\mathrm{CA})$ mit $t(n) = n$. Die entsprechenden Zellularautomatenklassen bezeichnen wir mit **Realzeit-OCA**, **Realzeit-CA**, **Linearzeit-OCA** und **Linearzeit-CA**. Die Sprachklassen $\mathcal{L}(\mathrm{CA})$ und $\mathcal{L}(\mathrm{OCA})$ umfassen die Sprachen, die von einem CA oder einem OCA in beliebiger Zeit akzeptiert werden können.

**Bemerkung 3.3.** Man kann sich leicht überlegen, daß eine Sprache, die von einem (O)CA in beliebiger Zeit akzeptiert wird, bereits zu einer Sprachklasse $\mathcal{L}_t((\mathrm{O})\mathrm{CA})$ gehört mit $t(n) = O(c^n)$, wobei $c$ eine Konstante ist: Da jeder Zellularautomat nur eine endliche Zustandsmenge besitzt und bei gegebener Eingabe nur eine feste Anzahl von Zellen zur Verfügung steht, werden nach spätestens $O(c^n)$ Zeitschritten die Konfigurationen periodisch. Ist also nach $O(c^n)$ Zeitschritten die Eingabe nicht akzeptiert worden, dann wird sie nie akzeptiert.

**Beispiel 3.4.** Als Beispiel für einen Zellularautomaten, der eine kontextfreie Sprache akzeptiert, betrachten wir einen Realzeit-OCA für die Sprache $\{a^n b^n \mid n \geq 1\}$. Wir skizzieren die Konstruktion: Es werden $a$'s mit $b$'s verglichen. Sobald eine Zelle mit dem Zustand $a$ eine rechte Nachbarzelle mit dem Zustand $b$ hat, geht die Zelle in einen Zustand $m$ über. Die $b$'s werden in jedem Zeitschritt um eine Stelle nach links geschoben. Die letzte Stelle der Eingabe wird mit einem Zustand $e$ für Ende markiert. Sobald eine Zelle mit dem Zustand $m$ eine rechte Nachbarzelle mit dem Zustand $e$ hat, geht die Zelle in einen akzeptierenden Zustand $g$ über. Wird gegen die korrekte Formatierung verstoßen, wird ein Zustand $f$ angenommen. In Abbildung 3.2 finden sich zwei Beispielberechnungen.

## 3.2 Definition von iterativen Arrays

Zellularautomaten sind ein paralleles Berechnungsmodell mit paralleler Eingabe. Wir definieren nun ein paralleles Berechnungsmodell mit sequentieller Eingabe. Dazu betrachten wir wiederum eine Menge identischer Zellen, die in einer Reihe angeordnet sind. Anfangs befinden sich alle Zellen in einem besonderen Ruhezustand $q_0$. Wie bei Zellularautomaten ist jede Zelle mit ihrem linken und rechten Nachbarn verbunden und die lokale Überführungsfunktion wird zu diskreten Zeitschritten synchron auf jede Zelle angewendet. Zusätzlich gibt es eine besondere Zelle, die Kommunikationszelle, die mit der Eingabe kommuniziert. Das heißt, in jedem Zeitschritt wird von dieser Zelle ein Eingabesymbol verarbeitet. Ist die gesamte Eingabe verarbeitet, dann wird ein besonderes Eingabeendesymbol $\triangledown$ gelesen. Formal definieren wir ein iteratives Array wie folgt:

**Definition 3.5.** Ein iteratives Array, abgekürzt IA, ist ein Tupel $(Q, \Sigma, \triangledown, \delta_0, \delta, q_0, F)$, wobei folgendes gilt:

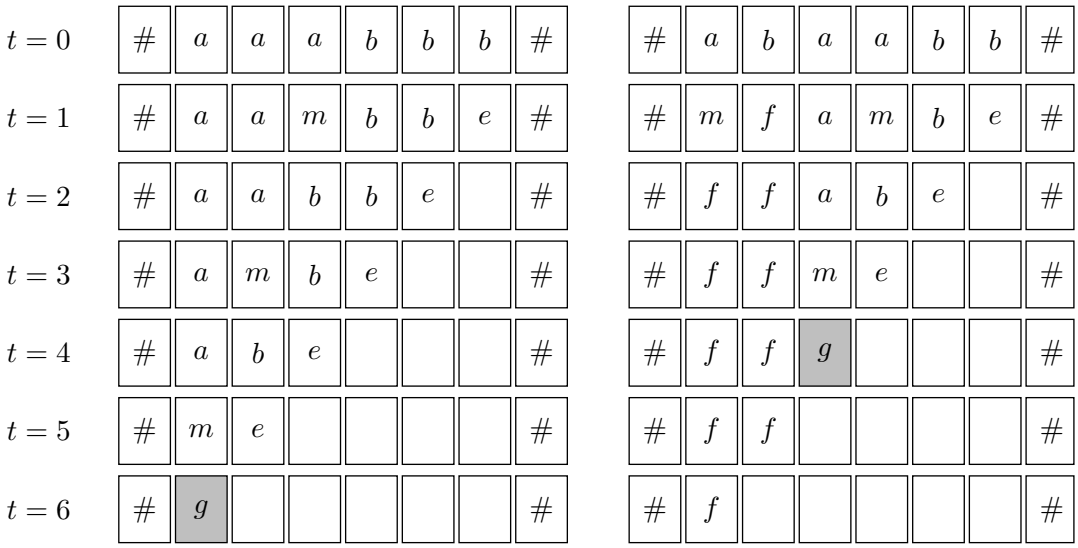



Abbildung 3.2: Beispielberechnungen eines Realzeit-OCA für die Sprache $\{a^n b^n \mid n \geq 1\}$ anhand der Eingaben *aaabbb* und *abaabb*. Im ersten Fall nimmt die erste Zelle nach sechs Zeitschritten einen akzeptierenden Zustand an, im zweiten Fall nimmt die erste Zelle nie einen akzeptierenden Zustand an. Die in den weiß dargestellten Zellen angenommenen Zustände können aufgrund der Realzeitbedingung die Gesamtberechnung nicht mehr beeinflussen und sind daher nicht abgebildet.

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände der Zellen,

(2) $\Sigma$ ist das Eingabealphabet,

(3) $\triangledown \notin \Sigma$ ist das Symbol für das Eingabeende,

(4) $\delta_0 : Q^3 \times (\Sigma \cup \{\triangledown\}) \to Q$ ist die lokale Überführungsfunktion für die mit der Eingabe kommunizierende Zelle, im folgenden Kommunikationszelle genannt.

(5) $\delta : Q^3 \to Q$ ist die lokale Überführungsfunktion für die restlichen, nicht mit der Eingabe kommunizierenden Zellen.

(6) $q_0 \in Q$ ist der Ruhezustand, wobei gilt: $\delta(q_0, q_0, q_0) = q_0$

(7) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.

Eine Konfiguration eines IA zum Zeitpunkt $t \geq 0$ ist ein Paar $(c_t, w_t)$, wobei $w_t \in \Sigma^*$ die verbleibende Eingabe ist und $c_t : \mathbb{Z} \to Q$ eine Funktion ist, die den einzelnen Zellen ihren aktuellen Zustand zuordnet. Die Konfiguration $(c_0, w_0)$ zum Zeitpunkt 0 ist durch die Eingabe $w_0$ und die Abbildung $c_0(i) = q_0$ für $i \in \mathbb{Z}$ definiert. Die globale Überführungsfunktion $\Delta$ wird durch $\delta$ und $\delta_0$ folgendermaßen induziert: Sei $(c_t, w_t)$ für $t \geq 0$ eine Konfiguration.

$$
\begin{aligned}
&(c_{t+1}, w_{t+1}) = \Delta((c_t, w_t)) \iff \\
&\quad c_{t+1}(i) \ = \delta(c_t(i-1), c_t(i), c_t(i+1)), i \in \mathbb{Z} \setminus \{0\} \\
&\quad c_{t+1}(0) = \delta_0(c_t(-1), c_t(0), c_t(1), x),
\end{aligned}
$$

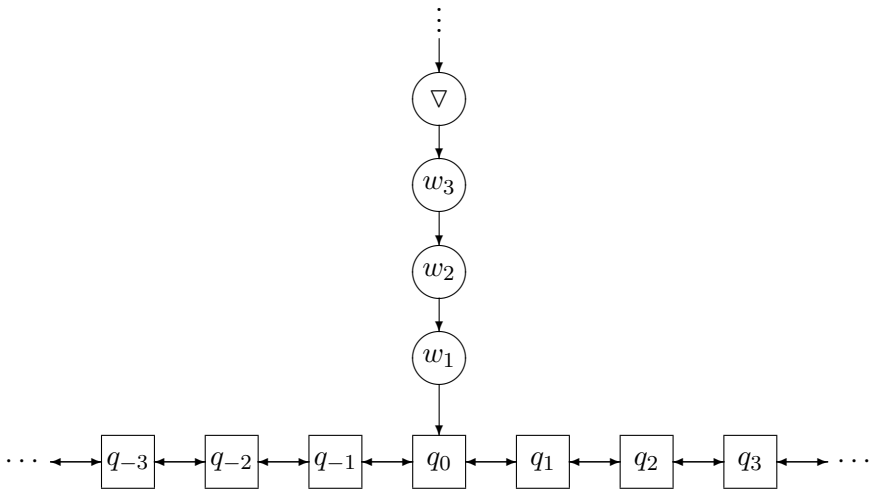



Abbildung 3.3: Ein iteratives Array (IA).

wobei $x = \nabla, w_{t+1} = \epsilon$ (falls $w_t = \epsilon$) und $x = x_1$, $w_{t+1} = x_2 \dots x_n$ (falls $w_t = x_1 \dots x_n$).

Eine Eingabe $w$ wird von einem IA akzeptiert, falls es einen Zeitpunkt $i$ während der Berechnung gibt, so daß die Kommunikationszelle einen akzeptierenden Zustand annimmt.

**Definition 3.6.** Sei $A = (Q, \Sigma, \nabla, \delta_0, \delta, q_0, F)$ ein IA.

(1) Ein Wort $w \in \Sigma^+$ wird von $A$ akzeptiert, falls es einen Zeitpunkt $i \in \mathbb{N}$ gibt mit $c_i(0) \in F$ für die Konfiguration $(c_i, w_i) = \Delta^i((c_0, w_0))$.

(2) $T(A) = \{w \in \Sigma^+ \mid w \text{ wird von } A \text{ akzeptiert}\}$ bezeichnet die von $A$ akzeptierte Sprache.

(3) Sei $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, eine rekursive Funktion und $i_w$ der minimale Zeitpunkt, an dem $A$ das Wort $w \in T(A)$ akzeptiert. Falls alle $w \in T(A)$ innerhalb von $i_w \leq t(|w|)$ Zeitschritten akzeptiert werden, dann sagen wir, daß $T(A)$ die Zeitkomplexität $t$ habe.

(4) $\mathcal{L}_t(\text{IA}) = \{L \mid L \text{ wird von einem IA mit Zeitkomplexität } t \text{ akzeptiert}\}$

(5) Falls $t(n) = n + 1$, dann sagen wir, daß die Sprachen in Realzeit akzeptiert werden. Das ist die kürzeste Zeit, um die gesamte Eingabe zu lesen und das Eingabeende zu erkennen. Die entsprechende Sprachklasse bezeichnen wir mit $\mathcal{L}_{rt}(\text{IA})$. Die in Linearzeit akzeptierten Sprachen $\mathcal{L}_{lt}(\text{IA})$ sind folgendermaßen definiert: $\mathcal{L}_{lt}(\text{IA}) = \bigcup_{k \in \mathbb{Q}, k \geq 1} \mathcal{L}_{\lceil k \cdot t \rceil}(\text{IA})$ mit $t(n) = n + 1$. Die entsprechenden Automatenklassen bezeichnen wir mit **Realzeit-IA** und **Linearzeit-IA**. Mit $\mathcal{L}(\text{IA})$ bezeichnen wir die Klasse der Sprachen, die von IA akzeptiert werden, die keiner Zeitbeschränkung unterliegen und deren Platzbedarf durch die Länge der Eingabe beschränkt ist.

**Beispiel 3.7.** Wir konstruieren ein iteratives Array, das $L = \{a^n c^m b^n \mid n, m \geq 1\}$ akzeptiert. Dazu wird ähnlich zur Konstruktion in [Kut01b] ein binärer Zähler konstruiert, dessen niedrigstwertiges Bit in der Kommunikationszelle lokalisiert ist. Bei Bedarf wird der Zähler nach links vergrößert. Bei einem Eingabesymbol $a$ wird der Zähler um eins erhöht, bei $b$ um

eins verringert und bei einem $c$ nicht verändert. In [Kut01b] wird beobachtet, daß hierfür einseitiger Informationsfluß ausreicht. Wir geben im folgenden die Überführungsfunktionen $\delta_0$ und $\delta$ eines IA an, wobei $\delta_0$ zunächst alle Informationen von links und rechts und $\delta$ alle Informationen von links ignoriert. Ein $\cdot$ bedeutet, daß dieser Übergang nicht definiert werden muß, da bei jeder Eingabe eine solche Situation niemals auftreten kann.

| $\delta_0$ | $a$ | $b$ | $c$ | $\triangledown$ |
|---|---|---|---|---|
| 0 | 1 | $1^-$ | 0 | $g$ |
| $0^+$ | 1 | $1^-$ | 0 | $f$ |
| 1 | $0^+$ | 0 | 1 | $f$ |
| $1^-$ | $0^+$ | 0 | 1 | $f$ |

und

| $\delta$ | 0 | $0^+$ | 1 | $1^-$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | $1^-$ |
| $0^+$ | 0 | $\cdot$ | 0 | $1^-$ |
| 1 | 1 | $0^+$ | 1 | 0 |
| $1^-$ | 1 | $0^+$ | 1 | $\cdot$ |

Wir haben in einem IA beidseitigen Informationsfluß zur Verfügung. Das ermöglicht uns, die Länge des binären Zählers an die tatsächlich benötigte Länge anzupassen: Ist die erste Zelle des Zählers im Zustand $0^+$ und ihr linker Nachbar im Zustand 0, dann muß die Länge um eins erhöht werden. Ist die erste Zelle des Zählers im Zustand 1 und ihr rechter Nachbar im Zustand $1^-$, dann muß die Länge um eins verringert werden. Die erste Zelle des Zählers ist in den folgenden Beispielberechnungen grau hinterlegt.

Ein Realzeit-IA für die Sprache $L$ arbeitet nun folgendermaßen: Die korrekte Formatierung wird von der Kommunikationszelle überprüft. Bei Verstößen dagegen wird die Eingabe verworfen. Bei korrekter Eingabe wird der Zähler bei Lesen von $a$'s erhöht, beim Lesen von $c$'s nicht verändert und beim Lesen von $b$'s verringert. Der Sonderfall, daß mehr $b$'s statt $a$'s gelesen werden, wird wie folgt behandelt: Sobald die Kommunikationszelle die erste Zelle des Zählers ist, der bereits auf Null heruntergezählt wurde, dann wird bei weiteren $b$'s in der Eingabe ein Fehlerzustand $f$ angenommen. Wird das Eingabeendesymbol gelesen, dann wird geprüft, ob die Kommunikationszelle die erste Zelle des Zählers ist und der Zähler auf Null heruntergezählt ist. Ist das der Fall, dann wird ein akzeptierender Zustand $g$ angenommen, ansonsten wird die Eingabe verworfen. In Abbildung 3.4 auf Seite 33 werden einige Beispielberechnungen angegeben. Da die Zellen $1, 2, \ldots$ nicht benötigt werden, sind sie nicht abgebildet. Weiterhin wird in der Kommunikationszelle nicht die Überprüfung der korrekten Formatierung abgebildet.

**Bemerkung 3.8.** Es sei $A$ ein Realzeit-IA. In [Kut01a] wird erklärt, daß das Ergebnis einer Berechnung bei einer verbleibenden Eingabe der Länge $m$ in $A$ lediglich von den Zuständen der Zellen $-m-1, \ldots, 0, \ldots, m+1$ abhängt, da die übrigen Zellen aufgrund der Nachbarschaft das Ergebnis nicht mehr beeinflussen können. Wir bezeichnen daher die Zustände der Zellen $-m-1, \ldots, 0, \ldots, m+1$ mit dem Ausdruck *$m$-Fenster*. Offenbar gibt es in einem Realzeit-IA höchstens $n^{2(m+1)+1}$ verschiedene $m$-Fenster, wobei $n$ die Anzahl der Zustände bezeichnet.

## 3.3 Generative Mächtigkeit

In folgendem Satz fassen wir die bekannten Ergebnisse und Zusammenhänge zwischen CA, IA und Sprachklassen der Chomskyhierarchie zusammen.

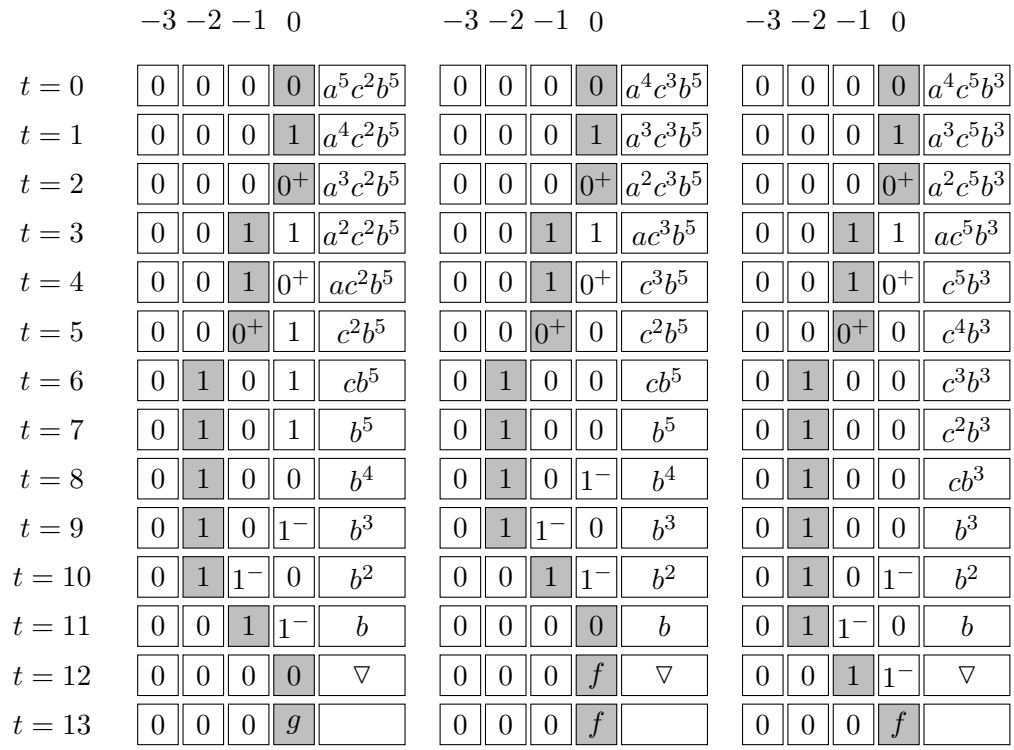



Abbildung 3.4: Berechnungen für die Eingaben $a^5c^2b^5$, $a^4c^3b^5$ und $a^4c^5b^3$ in Beispiel 3.7

**Satz 3.9.** Es gelten die in Abbildung 3.5 dargestellten Inklusionen und Äquivalenzen. Die Sprachklassen $\mathcal{L}_{rt}(\mathrm{OCA})$ und $\mathcal{L}_{rt}(\mathrm{IA})$ sind unvergleichbar.

*Beweis.* Beweise zu den einzelnen Aussagen sind zu finden in [Smi70],[CC84],[Kut01a] und [BKK02]. □

Für spätere Konstruktionen sind die beiden folgenden Sätze besonders wichtig, in denen gezeigt wird, wie Teilklassen der kontextfreien Sprachen konstruktiv von Zellularautomaten akzeptiert werden können.

**Satz 3.10.** [Smi70] Es sei $G$ eine linear kontextfreie Grammatik. Dann kann ein Realzeit-OCA $A$ konstruiert werden, so daß $T(A) = L(G)$ gilt.

**Satz 3.11.** [Kut01a] Es sei $M$ ein $\mathrm{DPDA}_\epsilon$. Dann kann ein Realzeit-IA $A$ konstruiert werden, so daß $T(A) = T(M)$ gilt.

Zur Unvergleichbarkeit zellularer Sprachklassen und kontextfreier Sprachen gibt es folgende Ergebnisse:

**Satz 3.12.** [Ter95] 2-LCF und $\mathcal{L}_{rt}(\mathrm{OCA})$ sind unvergleichbar.

**Satz 3.13.** LCF und $\mathcal{L}_{rt}(\mathrm{IA})$ und DCF und $\mathcal{L}_{rt}(\mathrm{IA})$ sind unvergleichbar.

*Beweis.* In [Col69] wird gezeigt, daß $L = \{ww \mid w \in \Sigma^*\}$ eine Realzeit-IA-Sprache ist. Da $L$ nicht kontextfrei ist, gilt $\mathcal{L}_{rt}(\mathrm{IA}) \setminus \mathrm{LCF} \neq \emptyset$ und $\mathcal{L}_{rt}(\mathrm{IA}) \setminus \mathrm{DCF} \neq \emptyset$. In [Kut01a] wird eine

$$
\begin{array}{ccccccc}
\mathrm{DCS} & = & \mathcal{L}(\mathrm{CA}) & = & \mathcal{L}(\mathrm{IA}) & & \\
 & & \cup\mathrm{I} & & & & \\
\mathrm{CF} & \subset & \mathcal{L}(\mathrm{OCA}) & & \cup\mathrm{I} & & \\
 & & \cup\mathrm{I} & & & & \\
 & & \mathcal{L}_{lt}(\mathrm{CA}) & = & \mathcal{L}_{lt}(\mathrm{IA}) & & \\
 & & \cup\mathrm{I} & & \cup & & \\
\mathrm{DCF} & \subset & \mathcal{L}_{rt}(\mathrm{CA}) & \supset & \mathcal{L}_{rt}(\mathrm{IA}) & \supset & \mathrm{DCF}_{\epsilon} \\
 & & \| & & & & \\
 & & \mathcal{L}_{lt}(\mathrm{OCA})^R & & & & \\
 & & \cup & & & & \\
\mathrm{REG} \subset \mathrm{LCF} & \subset & \mathcal{L}_{rt}(\mathrm{OCA}) & & & &
\end{array}
$$

Abbildung 3.5: Beziehungen zwischen Sprachklassen der Chomskyhierarchie und Sprachklassen, die von CA und IA erzeugt werden.

Sprache $L' \in \mathrm{DCF} \cap \mathrm{LCF}$ konstruiert, von der gezeigt wird, daß sie keine Realzeit-IA-Sprache ist. Also gilt $\mathrm{LCF} \setminus \mathcal{L}_{rt}(\mathrm{IA}) \neq \emptyset$ und $\mathrm{DCF} \setminus \mathcal{L}_{rt}(\mathrm{IA}) \neq \emptyset$. □

Wir werden später sehen, daß die von Zellularautomaten akzeptierten Sprachklassen kein Pumpinglemma besitzen können. Für Realzeit-OCA-Sprachen, die zyklische Wörter enthalten, also Wörter der Form $z = w^k$ mit $w \in \Sigma^*$ und $k \geq 0$, ist es allerdings möglich, ein Pumpinglemma zur Verfügung zu stellen.

**Lemma 3.14.** [Nak99] Sei $L \in \mathcal{L}_{rt}(\mathrm{OCA})$. Dann gibt es ein $n \geq 1$, so daß für jedes Wort $w$ und jede ganze Zahl $k \geq 1$ folgendes gilt: Falls $w^k \in L$ und $k > n^{|w|}$ ist, dann gibt es eine ganze Zahl $m$ mit $1 \leq m \leq n^{|w|}$, so daß $w^{k+j \cdot m} \in L$ für alle $j \geq 1$.

Für spätere Konstruktionen ist die Abgeschlossenheit unter gewissen Operationen hilfreich. Daher fassen wir die für Zellularautomaten bekannten Abschlußeigenschaften in folgendem Satz zusammen:

**Satz 3.15.** Für die Sprachklassen $\mathcal{L}_{rt}(\mathrm{OCA})$, $\mathcal{L}_{rt}(\mathrm{CA})$ und $\mathcal{L}_{rt}(\mathrm{IA})$ gelten die in Tabelle 3.1 zusammengefaßten Abschlußeigenschaften.

*Beweis.* Beweise zu den Aussagen finden sich in [Col69], [Sei79] und [CC84]. □

**Lemma 3.16.** Die Sprachklasse $\mathcal{L}_{rt}(\mathrm{IA})$ ist unter markiertem Kleene-Abschluß abgeschlossen.

*Beweis.* Die Behauptung wird in [Kle99] bewiesen. □

**Bemerkung 3.17.** Die regulären Sprachen sind nach [HU79] unter Quotientenbildung mit beliebigen Mengen abgeschlossen. Diese Abgeschlossenheit ist aber im allgemeinen nicht effektiv in dem Sinn, daß man einen DFA, der den Quotienten $R/M$ einer regulären Menge $R$ mit einer beliebigen Menge $M$ akzeptiert, algorithmisch bestimmen kann, da die Menge $M$ durchaus nicht rekursiv aufzählbar sein kann. Für Einzelheiten sei auf die Diskussion nach Theorem 3.6 in [HU79] verwiesen. Bei allen positiven Abschlußeigenschaften aus Tabelle 3.1 können wir allerdings beobachten, daß diese *effektiv* sind. Das heißt für eine binäre Operation $\circ$: Sind zwei Sprachen einer Sprachklasse durch zwei Automaten $A_1$ und $A_2$ aus einer Automatenklasse $\mathcal{A}$ gegeben, dann kann ein Automat $A$ aus $\mathcal{A}$ *effektiv konstruiert* werden mit $T(A) = T(A_1) \circ T(A_2)$. Für unäre Operationen ist der Begriff analog definiert.

| | $\mathcal{L}_{rt}$(OCA) | $\mathcal{L}_{rt}$(CA) | $\mathcal{L}_{rt}$(IA) |
|---|---|---|---|
| Vereinigung | ja | ja | ja |
| Schnitt | ja | ja | ja |
| Komplement | ja | ja | ja |
| Homomorphismus | nein | nein | nein |
| $\epsilon$-freier Homomorphismus | nein | ? | nein |
| inverser Homomorphismus | ja | ja | ja |
| Linkskonkatenation mit REG | ja | ? | nein |
| Rechtskonkatenation mit REG | ja | ja | ja |
| markierte Konkatenation | ja | ja | ja |
| Spiegelung | ja | ? | nein |

Tabelle 3.1: Abschlußeigenschaften der Sprachklassen $\mathcal{L}_{rt}$(OCA), $\mathcal{L}_{rt}$(CA) und $\mathcal{L}_{rt}$(IA)

## 3.4 Weitere Modelle

Wir beschreiben nun noch einige erweiterte zellulare Modelle, die in dieser Arbeit benötigt werden. Außerdem werden wir den Begriff der Beschleunigung von Zellularautomaten und das Konzept der Signale in Zellularautomaten einführen.

Die Sprachklasse $\mathcal{L}_{rt}$(OCA) ist nicht unter $\epsilon$-freiem Homomorphismus abgeschlossen [Sei79]. In [BKK02] werden Realzeit-OCA untersucht, die im ersten Schritt nichtdeterministisch agieren dürfen. Die erhaltene Automatenklasse wird Realzeit-„One-guess“-OCA genannt und die zugehörige Sprachklasse mit $\mathcal{L}_{rt}$(1G-OCA) bezeichnet. Im weiteren werden wir das Ergebnis verwenden, daß der Abschluß von $\mathcal{L}_{rt}$(OCA) unter $\epsilon$-freiem Homomorphismus äquivalent zu $\mathcal{L}_{rt}$(1G-OCA) ist.

**Satz 3.18.** [BKK02] Eine Sprache $L$ ist genau dann in $\mathcal{L}_{rt}$(1G-OCA) enthalten, wenn es einen $\epsilon$-freien Homomorphismus $h_\epsilon$ und eine Sprache $L' \in \mathcal{L}_{rt}$(OCA) gibt mit $L = h_\epsilon(L')$.

Ein ähnliches Ergebnis ist für iterative Arrays bekannt, die in Realzeit arbeiten und eine nichtdeterministisch agierende Kommunikationszelle besitzen. Man spricht dann von nichtdeterministischen iterativen Arrays (NIA). Die zugehörige Sprachklasse wird mit $\mathcal{L}_{rt}$(NIA) bezeichnet. In analoger Weise charakterisiert der Abschluß von $\mathcal{L}_{rt}$(IA) unter $\epsilon$-freiem Homomorphismus die nichtdeterministische Sprachklasse $\mathcal{L}_{rt}$(NIA).

**Satz 3.19.** [Kle99] Eine Sprache $L$ ist genau dann in $\mathcal{L}_{rt}$(NIA) enthalten, wenn es einen $\epsilon$-freien Homomorphismus $h_\epsilon$ und eine Sprache $L' \in \mathcal{L}_{rt}$(IA) gibt mit $L = h_\epsilon(L')$.

Unter *Beschleunigung* einer Berechnung bei Zellularautomaten verstehen wir folgendes: Wenn ein Automat $A$ eine Eingabe $w$ in Zeit $t(|w|)$ akzeptiert, dann akzeptiert ein „beschleunigter“ Automat $A'$ die Eingabe $w$ in Zeit $t'(|w|)$ und $t'(n) \leq t(n)$ für alle $n \in \mathbb{N}$. Für CA und IA mit beliebigen Zeitkomplexitäten gilt, daß die Berechnungen immer um einen linearen Faktor beschleunigt werden können. Man spricht daher von *linearer Beschleunigung*.

**Satz 3.20.** [Kut01b] Für alle $k \in \mathbb{N}$ mit $k > 0$ existiert zu jedem CA (OCA, IA) $A$ mit Zeitkomplexität $rt + r(n)$, $r : \mathbb{N} \to \mathbb{N}$, ein CA (OCA, IA) $A'$ mit Zeitkomplexität $rt + \left\lfloor \frac{r(n)}{k} \right\rfloor$ und $T(A') = T(A)$.

**Beispiel 3.21.** Durch Anwendung des obigen Satzes wird in [Kut01b] gezeigt, daß Linearzeit-OCA fast bis auf Realzeit beschleunigt werden können. Es gilt für alle $\epsilon \in \mathbb{Q}$ mit $\epsilon > 0$: $\mathcal{L}_{lt}(\text{OCA}) = \mathcal{L}_{(1+\epsilon)\cdot rt}(\text{OCA})$

Wir haben in Beispiel 3.7 gesehen, daß Zähler in Zellularautomaten und iterativen Arrays konstruiert werden können. Wie wir in dieser Arbeit sehen werden, sind Zähler ein wichtiges Hilfsmittel für Konstruktionen in zellularen Modellen. Für weitere Zählerkonstruktionen sei auf [Kut01b] verwiesen. Ein weiteres Konstruktionsmittel sind *Signale*, die in [MT99] und [Kut01b] ausführlich definiert und diskutiert werden. Wir werden in dieser Arbeit den Begriff *Signal* nur in folgendem vereinfachten Sinn verstehen:

Ein Signal wandert mit Geschwindigkeit $v$ von Zelle $i$ nach rechts (links), falls die Zelle $i$ zum Zeitpunkt $t$ im Zustand $q$ ist und für $j \geq 1$ jede Zelle $i + j$ $(i - j)$ zum Zeitpunkt $t + (1/v) \cdot j$ den Zustand $q$ annimmt.

**Beispiel 3.22.** Wir betrachten zwei Beispiele. In Abbildung 3.6 (rechts) findet man ein Signal mit Geschwindigkeit $v = 1$, also mit maximaler Geschwindigkeit, von links nach rechts. In Abbildung 3.6 (links) findet man ein Signal mit Geschwindigkeit $v = 1/2$ von rechts nach links.

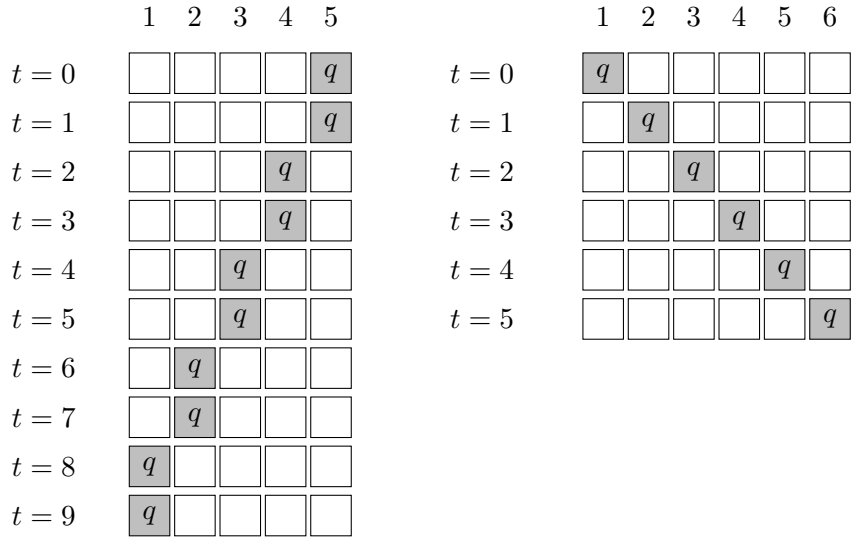



Abbildung 3.6: Signal mit Geschwindigkeit 1/2 von rechts nach links (links) und mit Geschwindigkeit 1 von links nach rechts (rechts)

# Kapitel 4

# Nichtrekursive Tradeoffs in Zellularautomaten

Im vorigen Kapitel haben wir Zellularautomaten und iterative Arrays als parallele Berechnungsmodelle mit paralleler und sequentieller Eingabe definiert und anhand von Beispielen vorgestellt. In diesem Kapitel werden wir nun die Beschreibungskomplexität von Zellularautomaten untersuchen. Wir vergleichen zunächst Zellularautomaten mit sequentiellen Modellen wie beispielsweise endlichen Automaten und Kellerautomaten bezüglich ihrer Beschreibungskomplexität. Das Ergebnis wird sein, daß das parallele Modell im Vergleich zum sequentiellen Modell Größeneinsparungen liefern kann, die durch keine rekursive Funktion beschränkt sind. Wir haben also nichtrekursive Tradeoffs zwischen diesen Modellen. Um die Ergebnisse zu erhalten, werden wir die in Satz 2.28 zusammengefaßte Methode benutzen, die auf eine Technik von Hartmanis zurückgeht. Dazu ist es einerseits nötig, daß man zu einer Turingmaschine $M$ eine Sprache $L_M$ und einen Zellularautomaten, der diese Sprache akzeptiert, konstruieren kann. Andererseits muß gezeigt werden, daß die Sprache $L_M$ nur dann von einem DFA oder PDA akzeptiert werden kann, wenn die ursprüngliche Turingmaschine eine endliche Menge akzeptiert. Hierzu werden wir für $L_M$ die Mengen VALC$[M]$ und INVALC$[M]$ der gültigen und ungültigen Berechnungen einer Turingmaschine $M$ wählen und in Kapitel 4.1 die entsprechenden Behauptungen zeigen. Wir erhalten dann nichtrekursive Tradeoffs zwischen Zellularautomaten und DFA bzw. PDA.

In Kapitel 3.1 wurden Zellularautomaten mit einseitigem und beidseitigem Informationsfluß definiert. Außerdem wurde bezüglich der Zeitkomplexität zwischen Realzeit und Linearzeit unterschieden. Nach Satz 3.9 führt beidseitiger Informationsfluß im Vergleich zu einseitigem Informationsfluß unter Realzeitbedingungen zu einem Anstieg der generativen Mächtigkeit. Ebenso führt in OCA Linearzeit im Vergleich zu Realzeit zu einer größeren generativen Mächtigkeit. Bezüglich der Beschreibungskomplexität erhalten wir in beiden Fällen, daß das mächtigere Modell zu nichtrekursiv beschränkten Einsparungen führen kann. Wir haben nichtrekursive Tradeoffs zwischen Realzeit-CA und Realzeit-OCA und zwischen Linearzeit-OCA und Realzeit-OCA.

In den Kapiteln 4.2 und 4.3 beschäftigen wir uns mit Entscheidbarkeitsfragen für Zellularautomaten. Wie wir bereits in Kapitel 2.6 erwähnt haben, kann man positive Entscheidbarkeitsfragen als ein gutes Zeichen für die praktische Verwendung eines Automatenmodells werten.

Wir werden hier für das allgemeine Modell zunächst nur negative Entscheidbarkeitsergebnisse beweisen können. Die Tatsache, daß die gültigen und ungültigen Berechnungen einer Turingmaschine von Zellularautomaten akzeptiert werden können, führt dazu, daß viele Entscheidbarkeitsfragen für Zellularautomaten auf Entscheidbarkeitsfragen für Turingmaschinen reduziert werden können. Da nach dem Satz von Rice (Satz 2.17) jede nichttriviale Eigenschaft einer Turingmaschine unentscheidbar ist, erhalten wir, daß für Zellularautomaten „fast nichts“ entscheidbar ist. Darüber hinaus können wir zeigen, daß viele Entscheidbarkeitsfragen nicht einmal semientscheidbar sind. Das heißt, es gibt nicht einmal einen Algorithmus, der für positive Eingaben hält und die entsprechenden Fragen beantwortet. Als Konsequenz aus den Unentscheidbarkeits- und Nichtsemientscheidbarkeitsergebnissen erhalten wir, daß es für zellulare Sprachklassen kein Pumpinglemma gibt. Ebenfalls gibt es für Zellularautomaten keinen Minimimierungsalgorithmus, der einen gegebenen Zellularautomaten in einen äquivalenten Zellularautomaten mit einer minimalen Zustandszahl umwandelt.

Nach Satz 3.12 ist bekannt, daß die Klasse der kontextfreien Sprachen und die Klasse der von Realzeit-OCA akzeptierten Sprachen unvergleichbar ist. In Kapitel 4.4 können wir zeigen, daß $\mathcal{L}_{rt}(\text{OCA})$ auch mit vielen anderen Teilklassen der Chomsky-Hierarchie, die zwischen den kontextfreien und rekursiv aufzählbaren Sprachen liegen, unvergleichbar ist.

## 4.1 Nichtrekursive Tradeoffs

Um später nichtrekursive Tradeoffs mit Hilfe von Satz 2.28 zeigen zu können, konstruieren wir zunächst zwei Realzeit-OCA für die gültigen und ungültigen Berechnungen einer Turingmaschine.

**Satz 4.1.** Sei $M$ eine Turingmaschine. Dann können zwei Realzeit-OCA $A_1$ und $A_2$ effektiv konstruiert werden mit $T(A_1) = \text{VALC}[M]$ und $T(A_2) = \text{INVALC}[M]$.

*Beweis.* In [HU79] wird gezeigt, daß INVALC$[M]$ eine kontextfreie Sprache ist. Wir betrachten die Konstruktion nun etwas genauer und zeigen, daß INVALC$[M]$ sogar eine linear kontextfreie Sprache ist. Daher können wir eine linear kontextfreie Grammatik $G$ konstruieren mit $L(G) = \text{INVALC}[M]$. Nach Satz 3.10 kann zu gegebener linear kontextfreier Grammatik $G$ ein Realzeit-OCA $A$ mit $T(A) = L(G)$ konstruiert werden. Also können wir einen Realzeit-OCA $A_2$ konstruieren mit $T(A_2) = \text{INVALC}[M]$. Da $\mathcal{L}_{rt}(\text{OCA})$ effektiv unter Komplementbildung abgeschlossen ist, können wir daher einen Realzeit-OCA $A_1$ konstruieren mit $T(A_1) = \text{VALC}[M]$.

Wir zeigen nun, daß INVALC$[M]$ eine linear kontextfreie Sprache ist.

Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ die zugrundeliegende Turingmaschine. Ein Wort $w$ ist eine ungültige Berechnung, falls eine der folgenden Bedingungen zutrifft:

(1) $w$ hat ein falsches Format, das heißt, $w$ ist nicht von der Form

$$ID_0(w)\#ID_1(w)^R\#ID_2(w)\#\ldots\#ID_{2m}(w)\# \text{ mit } ID_i(w) \in \Gamma^*Q\Gamma^* \text{ für } 0 \leq i \leq 2m.$$

(2) $ID_0(w)$ ist keine Startkonfiguration, das heißt $ID_0(w) \notin \{q_0\}\Sigma^*$.

(3) $ID_{2m}(w)$ ist keine akzeptierende Konfiguration, das heißt $ID_{2m}(w) \notin \Gamma^* F \Gamma^*$.

(4) Der Übergang $ID_i(w) \vdash^M ID_{i+1}(w)$ ist falsch für ein gerades $i$.

(5) Der Übergang $ID_i(w) \vdash^M ID_{i+1}(w)$ ist falsch für ein ungerades $i$.

Offenbar können die Bedingungen (1), (2) und (3) jeweils von einem endlichen Automaten überprüft werden. Wir zeigen jetzt, daß die Bedingungen (4) und (5) jeweils von einem 1-turn-PDA überprüft werden können. Ein 1-turn-PDA $A$, der die Bedingung (4) überprüft, rät zunächst eine Stelle in der Eingabe, der eine gerade Anzahl von #-Zeichen vorausgeht. Die verbleibende Eingabe ist von der Form $ID_i(w)\#z\#z'$ mit $ID_i(w), z \in \Gamma^* Q \Gamma^*$ und $z' \in (\Gamma \cup Q \cup \{\#\})^*$. Es ist nun zu prüfen, ob $z \neq ID_{i+1}(w)^R$ ist. In diesem Fall wird die Eingabe akzeptiert, anderenfalls verworfen. Dazu liest $A$ die Eingabe $ID_i(w)$ ein und schreibt die Folgekonfiguration $ID_{i+1}(w)$ auf den Keller. Sobald ein # gelesen wurde, wird die Eingabe $z$ symbolweise mit dem Keller überprüft. Sind das nächste Eingabesymbol und das oberste Kellersymbol identisch, wird das oberste Kellersymbol vom Keller entfernt und das nächste Eingabesymbol gelesen. Sind beide Symbole nicht identisch, dann heißt das, daß $z \neq ID_{i+1}(w)^R$ ist. Dann liest $A$ den Rest der Eingabe und geht in einen akzeptierenden Zustand.

Seien $p_0, \overline{p}_0, p_1, p_a, p_b, p_c, p_m, p_r$ Zustände mit $\{p_0, \overline{p}_0, p_1, p_a, p_b, p_c, p_m, p_r\} \cap Q = \emptyset$. Weiterhin sei $Z_0$ ein neues Symbol mit $\{Z_0\} \cap (\Gamma \cup Q) = \emptyset$. Wir definieren einen PDA

$$A = \{Q \cup \{p_0, \overline{p}_0, p_1, p_a, p_b, p_c, p_m, p_r\}, \Gamma \cup Q \cup \{\#\}, \Gamma \cup Q \cup \{Z_0\}, \delta', p_0, Z_0, \{p_a\}\}$$

mit den Überführungsregeln

$$\begin{array}{lcll}
\delta'(p_0, \epsilon, Z_0) & = & \{(\overline{p}_0, Z_0), (p_b, Z_0)\} & \\
\delta'(\overline{p}_0, \gamma, Z_0) & = & \{(\overline{p}_0, Z_0)\} & \text{für } \gamma \in \Gamma \cup Q \\
\delta'(\overline{p}_0, \#, Z_0) & = & \{(p_1, Z_0)\} & \\
\delta'(p_1, \gamma, Z_0) & = & \{(p_1, Z_0)\} & \text{für } \gamma \in \Gamma \cup Q \\
\delta'(p_1, \#, Z_0) & = & \{(p_0, Z_0)\} & \\
\delta'(p_b, \gamma, Z) & = & \{(p_b, \gamma Z)\} & \text{für } \gamma \in \Gamma, Z \in \Gamma \cup \{Z_0\} \\
\delta'(p_b, q, Z) & = & \{(q, Z)\} & \text{für } q \in Q, Z \in \Gamma \cup \{Z_0\}.
\end{array}$$

Diese Regeln stellen sicher, daß eine gerade Anzahl von #-Zeichen gelesen wurde, bevor vom Zustand $p_b$ aus nun die Eingabe $ID_i(w)$ solange gelesen und auf den Keller geschrieben wird, bis ein Zustand $q \in Q$ in der Eingabe auftritt. Mit den folgenden Regeln wird dann das Verhalten der Turingmaschine von $q$ aus simuliert.

Seien $\gamma, Z \in \Gamma$ und $q \in Q$. Dann definieren wir folgende Übergänge: ($p \in Q$ und $Y \in \Gamma$)

$$\begin{array}{lcll}
\delta'(q, \gamma, Z) & = & \{(p_c, pYZ)\} & \text{falls } \delta(q, \gamma) = (p, Y, R) \\
\delta'(q, \gamma, Z_0) & = & \{(p_c, pYZ_0)\} & \text{falls } \delta(q, \gamma) = (p, Y, R) \\
\delta'(q, \#, Z) & = & \{(p_m, pYZ)\} & \text{falls } \delta(q, B) = (p, Y, R) \\
\delta'(q, \#, Z_0) & = & \{(p_m, pYZ_0)\} & \text{falls } \delta(q, B) = (p, Y, R) \\
\delta'(q, \gamma, Z) & = & \{(p_c, YZp)\} & \text{falls } \delta(q, \gamma) = (p, Y, L) \\
\delta'(q, \gamma, Z_0) & = & \{(p_c, YBpZ_0)\} & \text{falls } \delta(q, \gamma) = (p, Y, L) \\
\delta'(q, \#, Z) & = & \{(p_m, YZp)\} & \text{falls } \delta(q, B) = (p, Y, L) \\
\delta'(q, \#, Z_0) & = & \{(p_m, YBpZ_0)\} & \text{falls } \delta(q, B) = (p, Y, L)
\end{array}$$

Anschließend dient der Zustand $p_c$ dazu, den Rest der Eingabe bis zum nächsten #-Zeichen auf den Keller zu schreiben. Der Zustand $p_m$ dient zum Vergleich des Kellerinhalts mit der Eingabe. Sobald ein Fehler gefunden ist, wird die Eingabe akzeptiert. Stimmen Eingabe und Kellerinhalt überein, wird ein nichtakzeptierender Zustand $p_r$ angenommen.

$$\begin{array}{lcll}
\delta'(p_c,\gamma,Z) & = & \{(p_c,\gamma Z)\} & \text{für } \gamma\in\Gamma, Z\in\Gamma\cup Q\cup\{Z_0\}\\
\delta'(p_c,\#,Z) & = & \{(p_m,Z)\} & \text{für } Z\in\Gamma\cup Q\cup\{Z_0\}\\
\delta'(p_m,\gamma,Z) & = & \{(p_m,\epsilon)\} & \text{für } \gamma,Z\in\Gamma\cup Q \text{ mit } \gamma=Z\\
\delta'(p_m,\gamma,Z) & = & \{(p_a,Z)\} & \text{für } \gamma\in\Gamma\cup Q\cup\{\#\}, Z\in\Gamma\cup Q\cup\{Z_0\} \text{ mit } \gamma\neq Z\\
\delta'(p_m,\#,Z_0) & = & \{(p_r,Z_0)\} & \\
\delta'(p_a,\gamma,Z) & = & \{(p_a,Z)\} & \text{für } \gamma\in\Gamma\cup Q\cup\{\#\}, Z\in\Gamma\cup Q\cup\{Z_0\}
\end{array}$$

Wir beobachten, daß der Keller nur dann abgebaut werden kann, wenn $A$ im Zustand $p_m$ ist. Von $p_m$ aus kann die Kellerhöhe aber nicht wieder vergrößert werden. Daher ist der konstruierte PDA ein 1-turn-PDA, der die Bedingung (4) überprüft.

Ein 1-turn-PDA $A'$ zur Überprüfung der Bedingung (5) kann ähnlich konstruiert werden. Es ist die Stelle der Eingabe zu raten, der eine ungerade Anzahl von #-Zeichen vorausgeht. Die verbleibende Eingabe ist dann von der Form $ID_i(w)^R\#z\#z'$ mit $ID_i(w), z\in\Gamma^*Q\Gamma^*$ und $z'\in(\Gamma\cup Q\cup\{\#\})^*$. Es ist nun zu prüfen, ob $z\neq ID_{i+1}(w)$ ist. Dazu liest $A'$ die Eingabe $ID_i(w)^R$ und schreibt die Spiegelung der Folgekonfiguration $ID_{i+1}(w)$ auf den Keller. Anschließend wird der Keller mit der Eingabe verglichen. Tritt ein Fehler auf, wird die Eingabe akzeptiert, anderenfalls verworfen. Die formale Konstruktion lautet:

$$A'=\{Q\cup\{p_0,p_1,p_a,p_b,p_c,p_m,p_r\},\Gamma\cup Q\cup\{\#\},\Gamma\cup Q\cup\{Z_0\},\delta',p_0,Z_0,\{p_a\}\}$$

mit den Überführungsregeln

$$\begin{array}{lcll}
\delta'(p_0,\gamma,Z_0) & = & \{(p_0,Z_0)\} & \text{für } \gamma\in\Gamma\cup Q\\
\delta'(p_0,\#,Z_0) & = & \{(p_1,Z_0),(p_b,Z_0)\} & \\
\delta'(p_1,\gamma,Z_0) & = & \{(p_1,Z_0)\} & \text{für } \gamma\in\Gamma\cup Q\\
\delta'(p_1,\#,Z_0) & = & \{(p_0,Z_0)\} & \\
\delta'(p_b,\gamma,Z) & = & \{(p_b,\gamma Z)\} & \text{für } \gamma\in\Gamma, Z\in\Gamma\cup\{Z_0\}\\
\delta'(p_b,q,Z) & = & \{(q,Z)\} & \text{für } q\in Q, Z\in\Gamma\cup\{Z_0\}.
\end{array}$$

Seien $\gamma,\gamma'\in\Gamma$ und $q\in Q$. Dann definieren wir folgende Übergänge: ($p\in Q$ und $Y\in\Gamma$)

$$\begin{array}{lcll}
\delta'(q,\gamma,\gamma') & = & \{(p_c,\gamma Yp)\} & \text{falls } \delta(q,\gamma')=(p,Y,R)\\
\delta'(q,\#,\gamma') & = & \{(p_m,Yp)\} & \text{falls } \delta(q,\gamma')=(p,Y,R)\\
\delta'(q,\gamma,Z_0) & = & \{(p_c,\gamma YpZ_0)\} & \text{falls } \delta(q,B)=(p,Y,R)\\
\delta'(q,\#,Z_0) & = & \{(p_m,pYZ_0)\} & \text{falls } \delta(q,B)=(p,Y,R)\\
\delta'(q,\gamma,\gamma') & = & \{(p_c,p\gamma Y)\} & \text{falls } \delta(q,\gamma')=(p,Y,L)\\
\delta'(q,\#,\gamma') & = & \{(p_m,pBY)\} & \text{falls } \delta(q,\gamma')=(p,Y,L)\\
\delta'(q,\gamma,Z_0) & = & \{(p_c,p\gamma YZ_0)\} & \text{falls } \delta(q,B)=(p,Y,L)\\
\delta'(q,\#,Z_0) & = & \{(p_m,pBYZ_0)\} & \text{falls } \delta(q,B)=(p,Y,L)
\end{array}$$

Weiterhin definieren wir:

$$\begin{array}{lcll}
\delta'(p_c, \gamma, Z) & = & \{(p_c, \gamma Z)\} & \text{für } \gamma \in \Gamma, Z \in \Gamma \cup Q \cup \{Z_0\} \\
\delta'(p_c, \#, Z) & = & \{(p_m, Z)\} & \text{für } Z \in \Gamma \cup Q \cup \{Z_0\} \\
\delta'(p_m, \gamma, Z) & = & \{(p_m, \epsilon)\} & \text{für } \gamma, Z \in \Gamma \cup Q \text{ mit } \gamma = Z \\
\delta'(p_m, \gamma, Z) & = & \{(p_a, Z)\} & \text{für } \gamma \in \Gamma \cup Q \cup \{\#\}, Z \in \Gamma \cup Q \cup \{Z_0\} \text{ mit } \gamma \neq Z \\
\delta'(p_m, \#, Z_0) & = & \{(p_r, Z_0)\} & \\
\delta'(p_a, \gamma, Z) & = & \{(p_a, Z)\} & \text{für } \gamma \in \Gamma \cup Q \cup \{\#\}, Z \in \Gamma \cup Q \cup \{Z_0\}
\end{array}$$

Wie eben beobachten wir, daß der Keller nur von $p_m$ aus abgebaut werden kann, und die Kellerhöhe von dort nicht vergrößert werden kann. Daher ist auch dieser konstruierte PDA ein 1-turn-PDA.

Insgesamt ist INVALC$[M]$ also die Vereinigung von regulären und linear kontextfreien Sprachen, die jeweils effektiv zu konstruieren sind. Daher ist INVALC$[M]$ linear kontextfrei, da LCF effektiv unter Vereinigung abgeschlossen ist. □

Eine Konsequenz aus Satz 4.1 ist, daß sich die rekursiv aufzählbaren Sprachen durch das homomorphe Bild von $\mathcal{L}_{rt}$(OCA) charakterisieren lassen. Diese Tatsache wird später zu einem Unvergleichbarkeitskriterium für $\mathcal{L}_{rt}$(OCA) führen. Wir erhalten also folgenden Charakterisierungssatz:

**Satz 4.2.** Eine Sprache $L$ ist genau dann rekursiv aufzählbar, wenn es einen Homomorphismus $h$ und eine Sprache $L' \in \mathcal{L}_{rt}$(OCA) gibt mit $L = h(L')$.

*Beweis.* Wir haben zwei Richtungen zu zeigen. Für die eine Richtung betrachten wir eine rekursiv aufzählbare Sprache $L$. Dann gibt es eine Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ mit $T(M) = L$ und nach Satz 4.1 einen Realzeit-OCA $A$ mit $T(A) = \text{VALC}[M]$. Offenbar gilt mit $\Lambda = \Gamma \cup Q \cup \{\#\}$:

$$w \in T(M) \Longleftrightarrow \exists w' \in \text{VALC}[M] \text{ und } x \in \Lambda^* : w' = q_0 w \# x$$

Der gesuchte Homomorphismus $h$ muß also $q_0$ und alle Symbole in einem $w' \in \text{VALC}[M]$ ab dem ersten # löschen. Wir werden daher zunächst die zu löschenden Symbole markieren und anschließend mit einem Homomorphismus löschen. Dazu definieren wir $\Lambda'$ als eine Kopie von $\Lambda$ und betrachten folgenden Homomorphismus $h_1 : \Lambda \cup \Lambda' \to \Lambda^*$ mit

$$h_1(a) = a \text{ für alle } a \in \Lambda \text{ und } h_1(a') = a \text{ für alle } a' \in \Lambda'$$

Dann ist die Sprache

$$L' = h_1^{-1}(\text{VALC}[M]) \cap \{q_0'\}\Sigma^*\{\#'\}(\Lambda')^*$$

in $\mathcal{L}_{rt}$(OCA), da die Klasse $\mathcal{L}_{rt}$(OCA) unter inversem Homomorphismus und Schnitt mit regulären Mengen abgeschlossen ist. Wir definieren nun den Homomorphismus $h : \Sigma \cup \Lambda' \to \Sigma^*$ durch

$$h(a) = a \text{ für alle } a \in \Sigma \text{ und } h(a') = \epsilon \text{ für alle } a' \in \Lambda'$$

Damit ist $h(L') = T(M) = L$.

Um die andere Richtung zu zeigen, beobachten wir, daß $L' \in \mathcal{L}_{rt}(\text{OCA})$ eine rekursiv aufzählbare Sprache ist. Dann ist auch $L = h(L')$ eine rekursiv aufzählbare Sprache, da die Klasse der rekursiv aufzählbaren Sprachen unter Homomorphismus abgeschlossen ist. □

Wir beweisen nun weitere notwendige Behauptungen, um Satz 2.28 anwenden zu können und so die gewünschten nichtrekursiven Tradeoffs zu erhalten.

**Lemma 4.3.** Sei $M$ eine Turingmaschine und $\text{L}[M] = \{w^{|w|!} \mid w \in \{\#_0\}\text{VALC}[M]\{\#_1\}\}$.

(1) $\text{INVALC}[M] \in \text{REG} \Leftrightarrow T(M)$ ist endlich

(2) $\text{VALC}[M] \in \text{CF} \Leftrightarrow T(M)$ ist endlich

(3) $\text{L}[M] \in \mathcal{L}_{rt}(\text{OCA}) \Leftrightarrow T(M)$ ist endlich

(4) $\text{L}[M] \in \mathcal{L}_{rt}(\text{CA})$

(5) $\text{L}[M]^R \in \mathcal{L}_{rt}(\text{OCA}) \Leftrightarrow T(M)$ ist endlich

(6) $\text{L}[M]^R \in \mathcal{L}_{lt}(\text{OCA})$

*Beweis.* (2) Die Behauptung wird in Lemma 8.8 in [HU79] bewiesen.

(1) Ist $T(M)$ endlich, dann ist auch die Menge $\text{VALC}[M]$ endlich und damit regulär. Dann ist auch das Komplement $\text{INVALC}[M]$ regulär, da REG unter Komplementbildung abgeschlossen ist. Ist umgekehrt $\text{INVALC}[M]$ regulär, dann ist auch das Komplement $\text{VALC}[M]$ regulär und damit kontextfrei. Nach (2) ist dann $T(M)$ endlich.

(3) „⇐“: Ist $T(M)$ endlich, dann ist die Menge $\text{VALC}[M]$ endlich. Damit ist auch die Menge $\text{L}[M]$ endlich und somit regulär. Da REG eine Teilmenge von $\mathcal{L}_{rt}(\text{OCA})$ ist, folgt damit $\text{L}[M] \in \mathcal{L}_{rt}(\text{OCA})$.

„⇒“: Wir benutzen das Pumpinglemma für zyklische Wörter (Lemma 3.14) und zeigen, daß folgendes gilt:

$$T(M) \text{ ist unendlich } \Rightarrow \text{L}[M] \notin \mathcal{L}_{rt}(\text{OCA})$$

Für einen Widerspruchsbeweis nehmen wir an, daß $\text{L}[M] \in \mathcal{L}_{rt}(\text{OCA})$ gilt.

Es sei $n$ die Konstante aus Lemma 3.14. Da $T(M)$ unendlich ist, können wir ein $w \in \{\#_0\}\text{VALC}[M]\{\#_1\}$ wählen mit $|w|! > n^{|w|}$. Dann gilt $w^{|w|!} \in \text{L}[M]$ und die Voraussetzungen des Lemmas sind erfüllt. Also gibt es ein $m \in \mathbb{N}$ mit $1 \leq m \leq n^{|w|}$, so daß gilt:

$$w^{|w|!+j\cdot m} \in \text{L}[M] \text{ für alle } j \in \mathbb{N}$$

Für $j = 1$ gilt insbesondere: $w^{|w|!+m} \in \text{L}[M]$.

Aus der Abschätzung

$$|w|! + m \leq |w|! + n^{|w|} < 2|w|! < (|w| + 1) \cdot |w|! = (|w| + 1)!$$

folgt, daß $|w|! + m$ keine Fakultät ist. Damit kann es kein $w' \in \{\#_0\}\text{VALC}[M]\{\#_1\}$ geben mit $|w'|! = |w|! + m$. Also gilt $w^{|w|!+m} \notin \text{L}[M]$, was ein Widerspruch ist.

(4) Wir beschreiben nun, wie ein Realzeit-CA konstruiert werden kann, der L[$M$] akzeptiert. Es seien $\#_0$ und $\#_1$ zwei neue Alphabetsymbole mit $\{\#_0, \#_1\} \cap \Sigma = \emptyset$. Wir betrachten die folgenden drei Sprachen $L_1, L_2$ und $L_3$:

$$L_1 = \{wx \mid w \in \{\#_0\}\text{VALC}[M]\{\#_1\}, x \in (\{\#_0\}\Sigma^+\{\#_1\})^+\}$$

$$L_2 = \{w^n \mid w \in \{\#_0\}\Sigma^+\{\#_1\}, n \in \mathbb{N}\}$$

$$L_3 = \{wx \mid w \in \{\#_0\}\Sigma^+\{\#_1\}, x \in (\{\#_0\}\Sigma^+\{\#_1\})^+, |wx|_{\#_0} = |w|!\}$$

Offenbar ergibt der Schnitt von $L_1, L_2$ und $L_3$ gerade L[$M$]. Die Sprache $L_1$ stellt das richtige Format sicher. Die Sprache $L_2$ stellt sicher, daß L[$M$] nur zyklische Wörter enthält und $L_3$ zählt die richtige Anzahl der Wiederholungen. Da $\mathcal{L}_{rt}$(CA) unter Schnittbildung abgeschlossen ist, brauchen wir daher nur zu zeigen, daß $L_1, L_2$ und $L_3$ jeweils von einem Realzeit-CA akzeptiert werden kann.

Nach Satz 4.1 gilt VALC[$M$] $\in \mathcal{L}_{rt}$(OCA). Weiterhin ist $\mathcal{L}_{rt}$(OCA) unter Konkatenation mit regulären Mengen abgeschlossen. Also ist $L_1 \in \mathcal{L}_{rt}(\text{OCA}) \subset \mathcal{L}_{rt}(\text{CA})$.

Wir betrachten nun die Sprache $L$:

$$L = \{x \mid x = x_1\#_0x_2\#_1\#_0x_3\#_1x_4 \Rightarrow x_2 \neq x_3 \text{ mit } x_2, x_3 \in \Sigma^+, \\ x_1, x_4 \in (\{\#_0\}\Sigma^*\{\#_1\})^*\}$$

$L$ enthält gerade die Wörter, die keine aufeinanderfolgenden identischen Teilwörter der Form $\#_0x\#_1$ besitzen. Die Sprache $L_2$ läßt sich daher wie folgt darstellen:

$$L_2 = \overline{L} \cap \left(\{\#_0\}\Sigma^+\{\#_1\}\right)^+$$

Wir zeigen nun, daß $L$ eine linear kontextfreie Sprache ist. Dann ist $L \in$ LCF $\subset \mathcal{L}_{rt}$(OCA). Da $\mathcal{L}_{rt}$(OCA) unter Komplementbildung und Schnitt mit regulären Mengen abgeschlossen ist, folgt daher $L_2 \in \mathcal{L}_{rt}(\text{OCA}) \subset \mathcal{L}_{rt}(\text{CA})$. Wir skizzieren die Konstruktion eines 1-turn-PDA $M$, der die Sprache $L$ akzeptiert:

- Die Eingabe wird gelesen und die korrekte Formatierung überprüft, bis vor einem $\#_0$-Zeichen nichtdeterministisch entschieden wird, die weitere Eingabe auf den Keller zu schreiben und zu prüfen, ob $\#_0x_2\#_1 \neq \#_0x_3\#_1$ gilt.
- Die Eingabe wird nun solange auf den Keller geschrieben, bis ein $\#_1$-Zeichen gelesen ist oder diese Phase nichtdeterministisch verlassen wird. Im ersten Fall wird die Eingabe verworfen.
- Im anderen Fall wird das nächste Eingabesymbol in Form eines besonderen Zustands „gemerkt“ und die weitere Eingabe bis nach dem nächsten $\#_1$-Zeichen wird ignoriert.
- Danach wird zu jedem Eingabesymbol ein Kellersymbol entfernt, bis entweder das Anfangskellersymbol erreicht ist, oder ein $\#_1$-Zeichen gelesen wurde.

  Im ersten Fall wird das nächste Eingabesymbol $\sigma$ mit dem im Zustand gemerkten Eingabesymbol verglichen. Sind beide Symbole voneinander verschieden, dann akzeptiert $M$, sofern die restliche Eingabe $w$ korrekt formatiert ist. Das heißt,

  $$w \in \Sigma^*\{\#_1\}(\{\#_0\}\Sigma^*\{\#_1\})^*, \text{ falls } \sigma \neq \#_1 \text{ und } w \in (\{\#_0\}\Sigma^*\{\#_1\})^*, \text{ falls } \sigma = \#_1$$

Im zweiten Fall gilt $|\#_0 x_2 \#_1| > |\#_0 x_3 \#_1|$ und $M$ akzeptiert, sofern die restliche Eingabe von der Form $(\{\#_0\}\Sigma^*\{\#_1\})^*$ ist.

- In allen anderen Fällen wird die Eingabe verworfen.

Der so konstruierte PDA führt offenbar nur eine Wendung aus. $L$ wird daher von einem 1-turn-PDA akzeptiert und ist also linear kontextfrei.

Wir müssen nun nur noch zeigen, daß $L_3$ von einem Realzeit-CA akzeptiert werden kann. Wir konstruieren einen Realzeit-CA, in dem jede Zelle in vier Teilzellen unterteilt ist. Daher können wir von vier Spuren sprechen.

- In der ersten Spur wird in der ersten Zelle die korrekte Formatierung der Eingabe überprüft. Das heißt, es wird geprüft, ob die Eingabe in der regulären Menge $(\{\#_0\}\Sigma^+\{\#_1\})^+$ enthalten ist. Die Eingabe wird verworfen, sobald ein Fehler festgestellt wird.
- In der zweiten Spur merken wir uns die Anzahl der gelesenen $\#_0$-Zeichen. Das heißt, falls die Eingabe $m$-mal das $\#_0$-Zeichen enthält, dann gibt es einen Zeitpunkt, zu dem die ersten $m$ Zellen mit einem besonderen Symbol \$ gekennzeichnet sind. Dies wird folgendermaßen realisiert: In jedem Zeitschritt wird jedes Eingabesymbol $S \neq \$$ um eine Zelle nach links geschoben, sofern die linke Nachbarzelle nicht das Symbol $\#_0$ oder \$ enthält. In diesem Fall wird das Eingabesymbol $S$ nicht verschoben, sondern in das Symbol \$ umgewandelt, falls $S = \#_0$ ist. Eine Beispielberechnung für die Eingabe $\#_0 a \#_1 \#_0 a \#_1 \#_0 a \#_1$ findet man in Abbildung 4.1. Für eine Eingabe der Länge $n$, die $m$-mal das $\#_0$-Zeichen enthält, ist $n - m$ der Zeitpunkt, an dem die ersten $m$ Zellen mit dem Symbol \$ markiert sind und in der $(m+1)$-ten Zelle das Eingabeendesymbol steht.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $t=0$ | # | $\#_0$ | $a$ | $\#_1$ | $\#_0$ | $a$ | $\#_1$ | $\#_0$ | $a$ | $\#_1$ | # |
| $t=1$ | # | \$ | $\#_1$ | $\#_0$ | $a$ | $\#_1$ | $\#_0$ | $a$ | $\#_1$ | # | |
| $t=2$ | # | \$ | $\#_0$ | $a$ | $\#_1$ | $\#_0$ | $a$ | $\#_1$ | # | | |
| $t=3$ | # | \$ | \$ | $\#_1$ | $\#_0$ | $a$ | $\#_1$ | # | | | |
| $t=4$ | # | \$ | \$ | $\#_0$ | $a$ | $\#_1$ | # | | | | |
| $t=5$ | # | \$ | \$ | \$ | $\#_1$ | # | | | | | |
| $t=6$ | # | \$ | \$ | \$ | # | | | | | | |

Abbildung 4.1: Beispielberechnung für die Eingabe $\#_0 a \#_1 \#_0 a \#_1 \#_0 a \#_1$

- In der dritten Spur wollen wir die Fakultät $n!$ für $n = 1, 2, \ldots$ berechnen. Wir verwenden dazu die Konstruktion von Mazoyer und Terrier [MT99]. In dieser

Konstruktion wird die erste Zelle jeweils zum Zeitpunkt $1, 2, 6, 24, \ldots$ mit einem besonderen Zustand markiert. Die Konstruktion wird nun etwas modifiziert: Auf jeden Berechnungsschritt der Fakultät folgt ein Verschiebungsschritt, der alle Zellen um eine Position nach rechts verschiebt. Im ersten Schritt wird die Fakultät berechnet, im zweiten Schritt werden alle Zellen nach rechts verschoben. Anschließend folgt ein Berechnungsschritt, dann wird wieder verschoben und so weiter. Das Ergebnis dieser Modifikation ist, daß die $n!$-te Zelle von links zum Zeitpunkt $2 \cdot n!$ ($n = 1, 2, \ldots$) mit einem besonderen Symbol markiert werden kann.

- In der vierten Spur wird die $|w|!$-te Zelle von links identifiziert. Die Idee dazu ist folgende: Die Anzahl der Zeichen bis zum ersten $\#_1$-Zeichen definiert $|w|$. Nach jeder Berechnung einer Fakultät in der dritten Spur wird ein Zeichen der Eingabe von rechts nach links und beginnend bei dem ersten $\#_1$-Zeichen „abgestrichen“, das heißt mit einer Markierung versehen. Wird das $\#_0$-Zeichen markiert, dann wurde gerade die $|w|$-te Fakultät berechnet. Diese wurde zum Zeitpunkt $2 \cdot |w|!$ in der $|w|!$-ten Zelle der dritten Spur berechnet. Wir müssen also nur noch sicherstellen, daß das Markieren von $\#_0$ in der $|w|!$-ten Zelle stattfindet. Dazu wird die Eingabe mit Geschwindigkeit $\frac{1}{2}$ nach rechts verschoben, das heißt, in zwei Zeitschritten wird jede Zelle um eine Position nach rechts verschoben. Wird in der dritten Spur eine Fakultät berechnet, also zu den Zeitpunkten $t = 2, 4, 12, 48, \ldots$, wird in der vierten Spur ein Signal $R$ mit maximaler Geschwindigkeit nach rechts losgesendet, das die letzte unmarkierte Zelle, beginnend mit dem ersten $\#_1$-Zeichen, markiert. Zum Zeitpunkt $t = 2 \cdot |w|! - 1$ trägt die $|w|!$-te Zelle das $\#_0$-Zeichen und die rechte Nachbarzelle ist markiert. Im nächsten Zeitschritt kommt aus der dritten Spur die Information, daß eine Fakultät berechnet wurde. Da $\#_0$ das erste Symbol der Eingabe war, kann also die $|w|!$-te Zelle identifiziert werden. Eine Beispielberechnung für $w = \#_0 a \#_1$ findet man in Abbildung 4.2 auf Seite 46.

Wir betrachten nun die Berechnung auf einer Eingabe $wx$ mit $w \in \{\#_0\}\Sigma^+\{\#_1\}$ und $x \in (\{\#_0\}\Sigma^+\{\#_1\})^+$. In der zweiten Spur wird die Anzahl $n$ der $\#_0$-Zeichen gesammelt. Das heißt, zum Zeitpunkt $|wx| - n$ ist der Zustand der $n$-ten Zelle \$, der rechte Nachbar trägt das Eingabeendesymbol und der linke Nachbar ist \$. Zu diesem Zeitpunkt überprüfen wir, ob die vierte Spur markiert ist.

- Ist dies der Fall, dann ist $n = |w|!$ und wir senden ein Signal mit maximaler Geschwindigkeit nach links, um die Eingabe zu akzeptieren, sofern der Rest der Eingabe der korrekten Formatierung entspricht, die in der ersten Spur überprüft wird. Die Markierung der $|w|!$-ten Zelle von links geschieht zum Zeitpunkt $2 \cdot |w|!$. Da $|wx| \geq 3 \cdot |w|!$ gilt, beobachten wir, daß $|wx| - n = |wx| - |w|! \geq 3 \cdot |w|! - |w|! = 2 \cdot |w|!$ ist. Also ist zum Zeitpunkt, an dem die Überprüfung der vierten Spur stattfindet, diese bereits markiert.
- Ist zum Zeitpunkt $|wx| - n$ die vierte Spur nicht markiert, dann ist $n \neq |w|!$ und wir senden ein Signal mit maximaler Geschwindigkeit nach links, um die Eingabe zu verwerfen.

Wir beobachten, daß die Berechnungen in den einzelnen Spuren und die Gesamtberechnung in Realzeit ausgeführt werden können. Also kann ein Realzeit-CA konstruiert

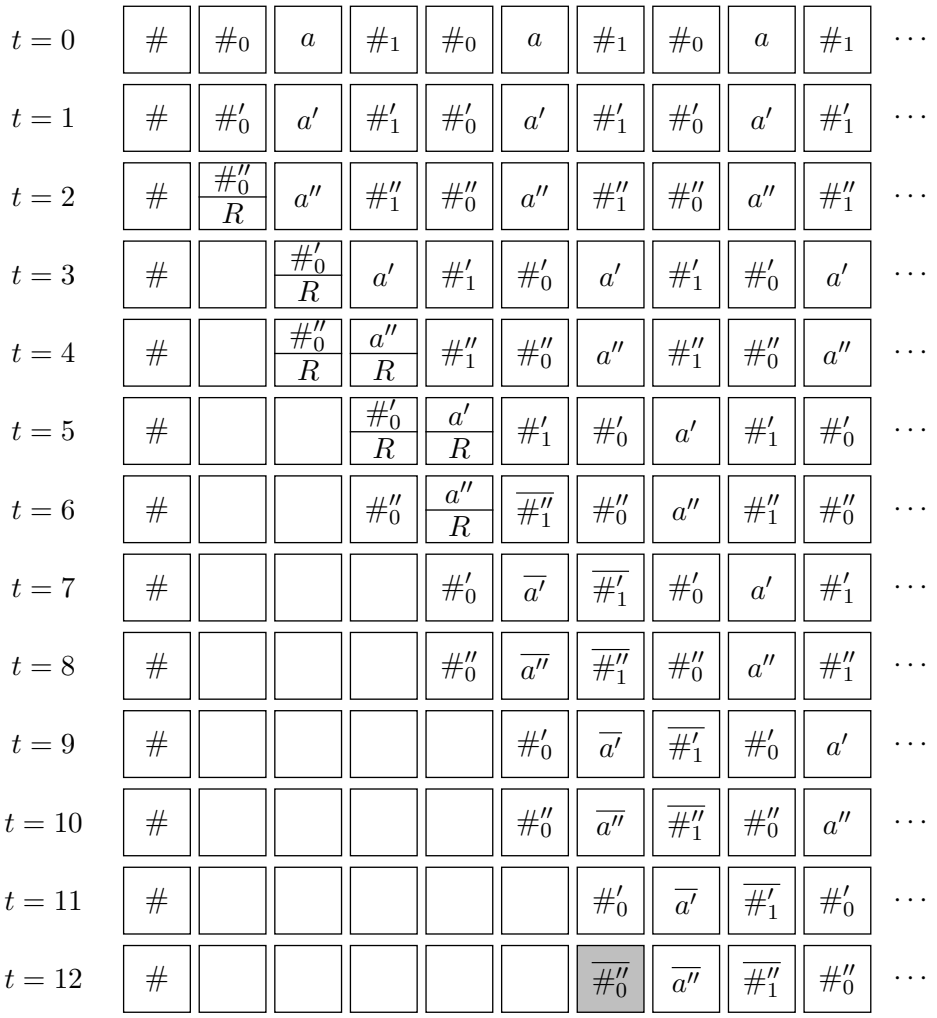



Abbildung 4.2: Beispielberechnung in der vierten Spur für $|w| = 3$

werden, der $L_3$ akzeptiert.

(5) Aus (3) und der Abgeschlossenheit von $\mathcal{L}_{rt}(\text{OCA})$ unter Spiegelung folgt unmittelbar:

$$\text{L}[M]^R \in \mathcal{L}_{rt}(\text{OCA}) \Leftrightarrow \text{L}[M] \in \mathcal{L}_{rt}(\text{OCA}) \Leftrightarrow T(M) \text{ ist endlich}$$

(6) $\text{L}[M] \in \mathcal{L}_{rt}(\text{CA}) = \mathcal{L}_{lt}(\text{OCA})^R$ nach (4) und Satz 3.9. Daher ist $\text{L}[M]^R \in \mathcal{L}_{lt}(\text{OCA})$.

□

Nun sind alle Vorbereitungen getroffen und Satz 2.28 kann angewendet werden. Wir erhalten damit folgende Ergebnisse:

**Satz 4.4.** Es existieren folgende nichtrekursiven Tradeoffs:

(1) Realzeit-OCA $\overset{nonrec}{\longrightarrow}$ DFA

(2) Realzeit-OCA $\overset{nonrec}{\longrightarrow}$ PDA

(3) Realzeit-CA $\overset{nonrec}{\longrightarrow}$ Realzeit-OCA

(4) Linearzeit-OCA $\overset{nonrec}{\longrightarrow}$ Realzeit-OCA

*Beweis.* Wir wenden Satz 2.28 und Lemma 4.3 an: (1) und (2) folgen mit $L_M = \text{INVALC}[M]$ und $L_M = \text{VALC}[M]$. Mit der Setzung $L_M = \text{L}[M]$ und $L_M = \text{L}[M]^R$ folgen die Behauptungen (3) und (4). □

**Korollar 4.5.** Weiterhin existieren folgende nichtrekursiven Tradeoffs:

(1) Realzeit-OCA $\overset{nonrec}{\longrightarrow}$ 1-turn PDA

(2) Realzeit-CA $\overset{nonrec}{\longrightarrow}$ DPDA

(3) OCA $\overset{nonrec}{\longrightarrow}$ PDA

*Beweis.* Nach Lemma 4.3(2) gilt $\text{VALC}[M] \in \text{CF}$ genau dann, wenn $T(M)$ endlich ist. Damit gilt insbesondere $\text{VALC}[M] \in \text{LCF}$ (DCF) genau dann, wenn $T(M)$ endlich ist. Daher können die Behauptungen (1) und (2) ähnlich zu Satz 4.4(2) mit $L_M = \text{VALC}[M]$ bewiesen werden. Da $\mathcal{L}_{rt}(\text{OCA}) \subset \mathcal{L}(\text{OCA})$ gilt, folgt die Behauptung (3) unmittelbar aus Satz 4.4(2). □

Wir erhalten damit nichtrekursive Tradeoffs zwischen parallelen und sequentiellen Modellen, zwischen zweiseitiger und einseitiger Kommunikation und zwischen Linearzeit und Realzeit.

**Bemerkung 4.6.** Nach Satz 3.12 von Terrier sind die Sprachklassen $\mathcal{L}_{rt}(\text{OCA})$ und CF unvergleichbar. Daher ist der unter (2) genannte nichtrekursive Tradeoff zwischen Realzeit-OCA und PDA als Tradeoff zwischen dem Beschreibungssystem $D_1$, bestehend aus der Menge aller Realzeit-OCA, und dem Beschreibungssystem $D_2$, bestehend aus der Menge aller PDA, die eine Realzeit-OCA-Sprache beschreiben, zu verstehen. Im folgenden verstehen wir nichtrekursive Tradeoffs zwischen unvergleichbaren Sprachklassen immer in diesem Sinn.

Wir haben gerade einen nichtrekursiven Tradeoff zwischen Linearzeit-OCA und Realzeit-OCA bewiesen. Für Linearzeit-OCA gilt nach Beispiel 3.21, daß diese fast bis auf Realzeit beschleunigt werden können. In [KK03] werden Automatenklassen betrachtet, deren Zeitkomplexität $t(n)$ zwischen Realzeit und Linearzeit liegt. Diese Automaten und die entsprechenden Sprachklassen werden in [KK03] eingehend untersucht. Insbesondere gibt es zwischen $\mathcal{L}_{rt}(\text{OCA})$ und $\mathcal{L}_{lt}(\text{OCA})$ eine unendliche echte Hierarchie von Sprachfamilien. Für $t(n) = n + \log n$ gilt:

$$\mathcal{L}_{rt}(\text{OCA}) \subset \mathcal{L}_{rt+\log n}(\text{OCA}) \subset \mathcal{L}_{lt}(\text{OCA})$$

Wir verbessern nun den nichtrekursiven Tradeoff zwischen Linearzeit und Realzeit auf einen nichtrekursiven Tradeoff zwischen Realzeit plus $\log n$ und Realzeit. Das Vorgehen ist ähnlich wie in Lemma 4.3(3) und (4). Einerseits können wir wiederum das Pumpinglemma für zyklische Wörter anwenden. Für die zweite Behauptung konstruieren wir einen OCA, der in Realzeit plus $\log n$ arbeitet.

**Lemma 4.7.** Sei $M$ eine Turingmaschine und $\text{L}'[M] = \{w^{2^{2^{|w|}}} \mid w \in \{\#_0\}\text{VALC}[M]\{\#_1\}\}$.

(1) $\text{L}'[M] \in \mathcal{L}_{rt}(\text{OCA}) \Leftrightarrow T(M)$ ist endlich

(2) $\text{L}'[M] \in \mathcal{L}_{rt+\log n}(\text{OCA})$

*Beweis.* Zum Beweis von (1) haben wir zwei Richtungen zu zeigen:

„⇐“: Ist $T(M)$ endlich, dann ist die Menge VALC$[M]$ endlich. Damit ist auch die Menge L$'[M]$ endlich und somit regulär. Da REG eine Teilmenge von $\mathcal{L}_{rt}$(OCA) ist, folgt damit L$'[M] \in \mathcal{L}_{rt}$(OCA).

„⇒“: Wie in Lemma 4.3(3) benutzen wir das Pumpinglemma für zyklische Wörter (Lemma 3.14) und zeigen, daß folgendes gilt:

$$T(M) \text{ ist unendlich } \Rightarrow \mathrm{L}'[M] \notin \mathcal{L}_{rt}(\mathrm{OCA})$$

Wir nehmen an, daß L$'[M] \in \mathcal{L}_{rt}$(OCA) gilt. Sei $n$ die Konstante aus Lemma 3.14. Da $T(M)$ unendlich ist, können wir ein $w \in \{\#_0\}\mathrm{VALC}[M]\{\#_1\}$ wählen mit $2^{2^{|w|}} > n^{|w|}$. Dann gilt $w^{2^{2^{|w|}}} \in \mathrm{L}'[M]$ und die Voraussetzungen des Lemmas sind erfüllt. Also gibt es ein $m \in \mathbb{N}$ mit $1 \leq m \leq n^{|w|}$, so daß gilt:

$$w^{2^{2^{|w|}}+j\cdot m} \in \mathrm{L}'[M] \text{ für alle } j \in \mathbb{N}$$

Für $j = 1$ gilt insbesondere: $w^{2^{2^{|w|}}+m} \in \mathrm{L}'[M]$.

Aus der Abschätzung

$$2^{2^{|w|}} + m \leq 2^{2^{|w|}} + n^{|w|} < 2 \cdot 2^{2^{|w|}} = 2^{2^{|w|}+1} < 2^{2^{|w|}\cdot 2} = 2^{2^{|w|+1}}$$

folgt, daß die Zahl $2^{2^{|w|}} + m$ keine doppelte Zweierpotenz ist. Damit kann es kein $w' \in \{\#_0\}\mathrm{VALC}[M]\{\#_1\}$ geben mit $2^{2^{|w'|}} = 2^{2^{|w|}} + m$. Also gilt $w^{2^{2^{|w|}}+m} \notin \mathrm{L}'[M]$, was ein Widerspruch ist.

Beim Beweis von (2) müssen wir zunächst beachten, daß wir die Konstruktion aus dem Beweis von Lemma 4.3(4) nicht verwenden können, da dort explizit der beidseitige Informationsfluß ausgenutzt wurde. Wir haben hier aber nur einseitigen Informationsfluß und etwas mehr Zeit zur Verfügung. Wir beschreiben nun, wie ein OCA konstruiert werden kann, der L$'[M]$ in Zeit $t(n) = n + \log n$ akzeptiert. Wie im Beweis von Lemma 4.3(4) betrachten wir zwei neue Alphabetsymbole $\#_0$ und $\#_1$ mit $\{\#_0, \#_1\} \cap \Sigma = \emptyset$. Weiterhin seien die Sprachen $L_1, L_2$ und $L_3'$ wie folgt definiert:

$$L_1 = \{wx \mid w \in \{\#_0\}\mathrm{VALC}[M]\{\#_1\}, x \in (\{\#_0\}\Sigma^+\{\#_1\})^+\}$$

$$L_2 = \{w^n \mid w \in \{\#_0\}\Sigma^+\{\#_1\}, n \in \mathbb{N}\}$$

$$L_3' = \{wx \mid w \in \{\#_0\}\Sigma^+\{\#_1\}, x \in (\{\#_0\}\Sigma^+\{\#_1\})^+, |wx|_{\#_0} = 2^{2^{|w|}}\}$$

Wie in Lemma 4.3(4) ist L$'[M]$ offenbar gerade der Schnitt der drei Sprachen $L_1$, $L_2$ und $L_3'$. Da $\mathcal{L}_{rt+\log n}$(OCA) nach [KK03] unter Schnittbildung abgeschlossen ist, haben wir zu zeigen, daß die Sprachen $L_1$, $L_2$ und $L_3'$ jeweils in $\mathcal{L}_{rt+\log n}$(OCA) sind. In Lemma 4.3(4) wurde bereits gezeigt, daß die Sprachen $L_1$ und $L_2$ in $\mathcal{L}_{rt}$(OCA) und damit auch in $\mathcal{L}_{rt+\log n}$(OCA) sind.

Die Konstruktion eines OCA, der die Sprache $L_3'$ in Zeit $t(n) = n + \log n$ akzeptiert, kann folgendermaßen skizziert werden: Jede Zelle ist wiederum in vier Teilzellen oder Spuren unterteilt. In der ersten Spur wird die korrekte Formatierung der Eingabe überprüft. In der

zweiten Spur werden Signale generiert, die zwei Dinge ermöglichen: Zum einen wird in der dritten Spur sichergestellt, daß die Anzahl der $\#_0$-Zeichen in der Eingabe eine doppelte Zweierpotenz ist. Zum anderen wird in der vierten Spur sichergestellt, daß die in der dritten Spur errechnete doppelte Zweierpotenz gerade die $|w|$-te doppelte Zweierpotenz ist. Im folgenden bezeichne $n$ die Länge der Eingabe.

- In der ersten Spur wird die Eingabe zunächst nicht verändert. Lediglich von der rechten Randzelle, die sukzessive von rechts nach links wandert, wird die korrekte Formatierung der Eingabe überprüft. Das heißt, es wird von rechts nach links geprüft, ob die Eingabe in der regulären Menge $(\{\#_1\}\Sigma^+\{\#_0\})^+$ enthalten ist. Sobald ein korrekt formatiertes Suffix der Eingabe erkannt worden ist, wird dies in der entsprechenden Zelle durch einen besonderen Zustand $g$ vermerkt. Dieser Zustand wird im weiteren Verlauf der Berechnung nicht mehr verändert. Nimmt die linke Randzelle nach $n$ Schritten den Zustand $g$ an, dann war die Eingabe korrekt formatiert. War die Eingabe nicht korrekt formatiert, dann kann die linke Randzelle niemals den Zustand $g$ annehmen. Insbesondere gilt ab dem Zeitpunkt $n$: Die linke Randzelle ist genau dann im Zustand $g$, wenn die Eingabe korrekt formatiert ist.

- In der zweiten Spur wird $2^{2^m}$ für $m \geq 1$ berechnet. Das heißt, nach $2^{2^m}+2^m$ Zeitschritten nimmt die $2^{2^m}$-te Zelle von rechts einen bestimmten Zustand an. Dazu betrachten wir zunächst die Konstruktion eines OCA, der die Sprache $\{a^{2^m} \mid m \geq 1\}$ in Zeit $t(n) = n + \log n$ akzeptiert. Diese Konstruktion aus [BKK00b] wird anschließend modifiziert, um die Sprache $\{a^{2^{2^m}} \mid m \geq 1\}$ in Zeit $t(n) = n + \log n$ zu akzeptieren.

  Die Idee der Konstruktion für die Sprache $\{a^{2^m} \mid m \geq 1\}$ ist, einen binären Zähler am rechten Rand zu konstruieren, der in jedem Zeitschritt um eins erhöht wird und um eine Stelle nach links wandert. Dieser binäre Zähler startet mit einem Zeitschritt Verzögerung. Bei Bedarf wird der Zähler rechts um eine Stelle verlängert. Weiterhin wird geprüft, ob der sukzessive durch eine Zelle wandernde Zähler nur aus Einsen besteht, das heißt, ob der gegenwärtige Zählerstand $2^m-1$ für ein $m \geq 1$ ist. Trifft dies zu, dann nimmt die Zelle einen akzeptierenden Zustand + an, sonst einen Fehlerzustand $e$. Die niedrigstwertige Stelle des Zählers erreicht die $n$-te Zelle von rechts nach $n$ Schritten. Nun werden $\log n$ Zeitschritte benötigt, um den Zählerstand zu überprüfen und gegebenenfalls zu akzeptieren. Da bei einem Zählerstand von $2^m - 1$ akzeptiert wird und die Berechnung mit einem Schritt Verzögerung startete, wurde genau die Anzahl $2^m$ für ein $m \geq 1$ erkannt. Eine Beispielberechnung für die Eingabe $a^{16}$ findet man in Abbildung 4.3 auf Seite 50. Die Zustände $\overset{\bullet}{0}$ und $\overset{+}{1}$ kennzeichnen eine Null mit Übertrag und eine Eins, wobei bislang nur Einsen durch die Zelle gewandert sind.

  Nun wollen wir diese Konstruktion modifizieren, um die Sprache $\{a^{2^{2^m}} \mid m \geq 1\}$ zu akzeptieren. Dazu teilen wir die zweite Spur in zwei Teilspuren auf. In der linken Hälfte wird mit der soeben geschilderten Konstruktion ein OCA für die Sprache $\{a^{2^m} \mid m \geq 1\}$ installiert. Während in der linken Hälfte die Länge der Eingabe gezählt wird, wird nun in der rechten Hälfte die Länge des Zählers der linken Hälfte gezählt. Dazu wird ein Zähler wie in der linken Hälfte konstruiert, allerdings wird dieser nicht in jedem Zeitschritt um eins erhöht, sondern nur dann, wenn in der linken Hälfte ein Überlauf stattfindet und der Zähler dort um eine Stelle verlängert werden muß. Es muß dann

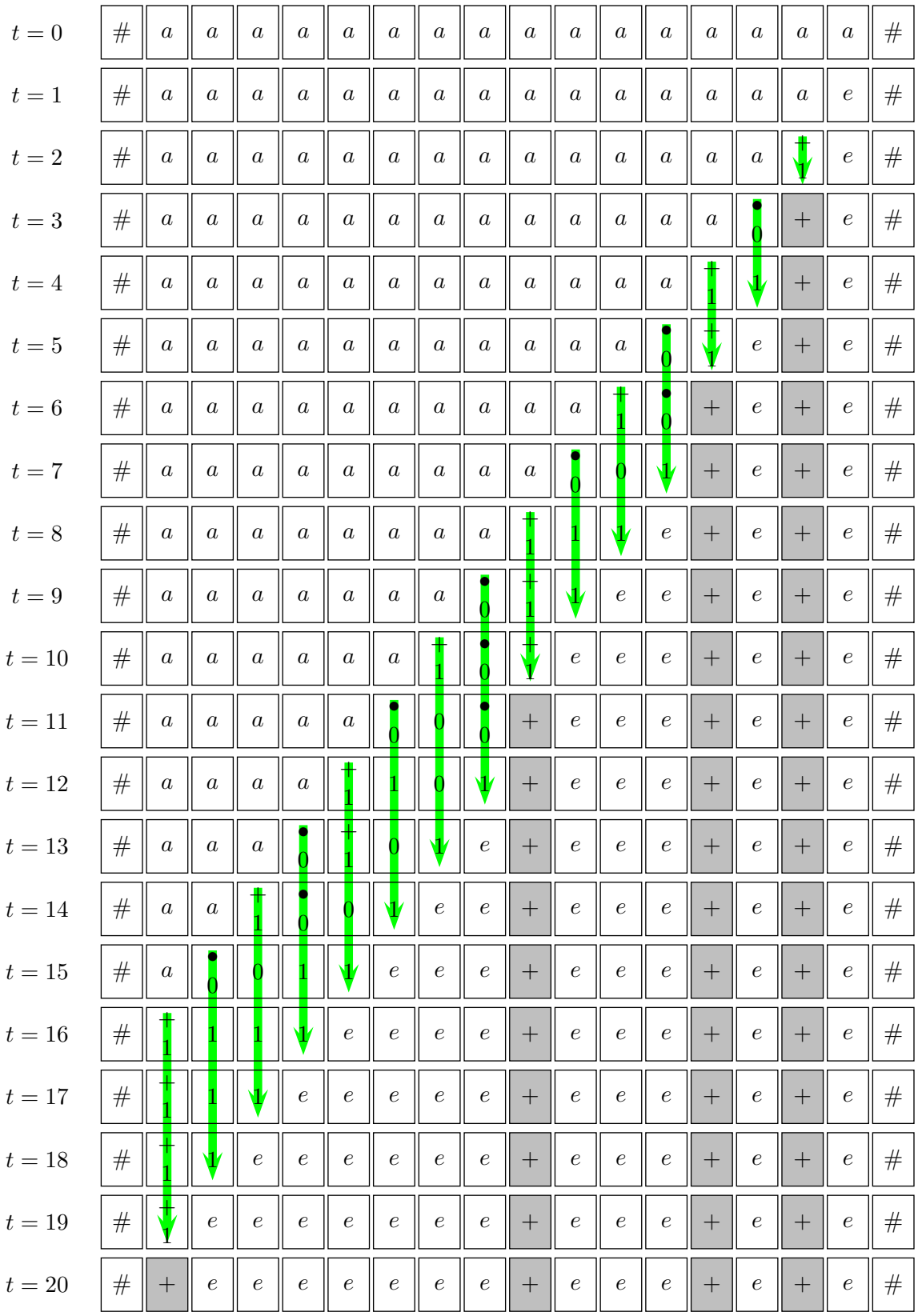



Abbildung 4.3: Ein OCA für die Sprache $\{a^{2^m} \mid m \geq 1\}$ mit der Eingabe $a^{16}$.

das Linkswandern des rechten Zählers um einen Zeitschritt verzögert werden. Dies wird durch ein Aufteilen der rechten Hälfte in zwei Register realisiert: Sukzessive wandern die Informationen durch beide Register und werden so um einen Zeitschritt verlangsamt. Zum Zeitpunkt $2^m + \log 2^m = 2^m + m$ nimmt die linke Hälfte den Zustand + an. Ab diesem Zeitpunkt prüfen wir in der rechten Hälfte, ob der aktuelle Zählerstand nur aus Einsen besteht. Ist dies der Fall, dann haben wir zum Zeitpunkt $2^{2^m} + \log 2^{2^m} + \log\log 2^{2^m} = 2^{2^m} + 2^m + m$ eine Zelle $2^{2^m}$ identifiziert und können diese durch einen besonderen Zustand markieren. Eine Beispielberechnung für die Eingabe $a^{16}$ findet man in Abbildung 4.4 auf Seite 52. Wir beobachten, daß die Zeitkomplexität der Berechnung $n + \log n + \log\log n \leq n + 2\log n$ ist. Nach Satz 3.20 kann die Berechnung allerdings auf $t(n) = n + \log n$ beschleunigt werden.

Also kann zum Zeitpunkt $m + \log m$ in der $m$-ten Zelle von rechts entschieden werden, ob $m$ eine doppelte Zweierpotenz ist oder nicht. Ist $m$ eine doppelte Zweierpotenz, dann werden in der dritten Spur ein Signal $\zeta_2$ und in der vierten Spur ein Signal $\xi$ jeweils mit maximaler Geschwindigkeit nach links gestartet. Ist $m$ keine doppelte Zweierpotenz, dann wird in der dritten Spur ein Signal $\zeta_1$ mit maximaler Geschwindigkeit nach links gestartet. Das heißt, bis zum Zeitpunkt $m+\log m$ werden gerade $m-\lfloor\log\log m\rfloor$ Signale $\zeta_1$ in der dritten Spur und $\lfloor\log\log m\rfloor$ Signale $\zeta_2$ und $\xi$ in der dritten und vierten Spur gestartet.

- In der dritten Spur werden die Signale $\zeta_1$ und $\zeta_2$ verarbeitet. Aufgabe dieser Signale ist es, die Anzahl der $\#_0$-Zeichen zu zählen und sicherzustellen, daß letztlich die Anzahl eine doppelte Zweierpotenz ist. Anfangs steht die Eingabe in der dritten Spur. Sobald nun ein Signal $\zeta_1$ gestartet wurde, läuft dieses solange mit maximaler Geschwindigkeit nach links, bis eine Zelle mit dem Zustand $\#_0$ erreicht wird. Dann wird das Signal gestoppt und die Zelle durch den Zustand $m_1$ markiert. Wurde ein Signal $\zeta_2$ gestartet, so ist das Vorgehen analog. Lediglich eine Zelle mit dem Zustand $\#_0$ wird durch den Zustand $m_2$ markiert.

- In der vierten Spur wird das Signal $\xi$ verarbeitet. Dieses Signal soll sicherstellen, daß $|w|$ doppelte Zweierpotenzen errechnet werden und damit die Anzahl der $\#_0$-Zeichen gerade $2^{2^{|w|}}$ ist. $\xi$ wird bei jeder errechneten doppelten Zweierpotenz gestartet. Daher wird jedes Signal dazu verwendet, ein Symbol des Worts $w$ „abzustreichen“. Sind am Ende der Berechnung alle Symbole des Worts $w$ abgestrichen, dann waren mindestens $2^{2^{|w|}}$ viele $\#_0$-Zeichen in der Eingabe. Da wir aufgrund des einseitigen Informationsflusses das Anfangswort $w$ nicht lokalisieren können, wird $\xi$ alle Teilwörter aus $\{\#_0\}\Sigma^+\{\#_1\}$ und damit auch $w$ sukzessive abstreichen. Dies wird folgendermaßen realisiert: Am Anfang steht die Eingabe auf der vierten Spur. Das Signal $\xi$ wandert mit maximaler Geschwindigkeit nach links ohne die Eingabe zu verändern. Sobald das Signal auf eine Zelle mit dem Zustand $\#_1$ trifft, wird der Zustand $m$ in diese Zelle geschrieben und das Signal wandert weiter nach links. Ist das Signal in der linken Randzelle angekommen, dann sind in der Eingabe alle $\#_1$-Zeichen mit $m$ überschrieben. Sobald das Signal $\xi$ auf eine Zelle trifft, die mit $m$ markiert ist, wird die Zelle mit $\overline{m}$ markiert, die links von dieser Zelle liegende Zelle wird mit $m$ markiert und das Signal wandert weiter nach links.

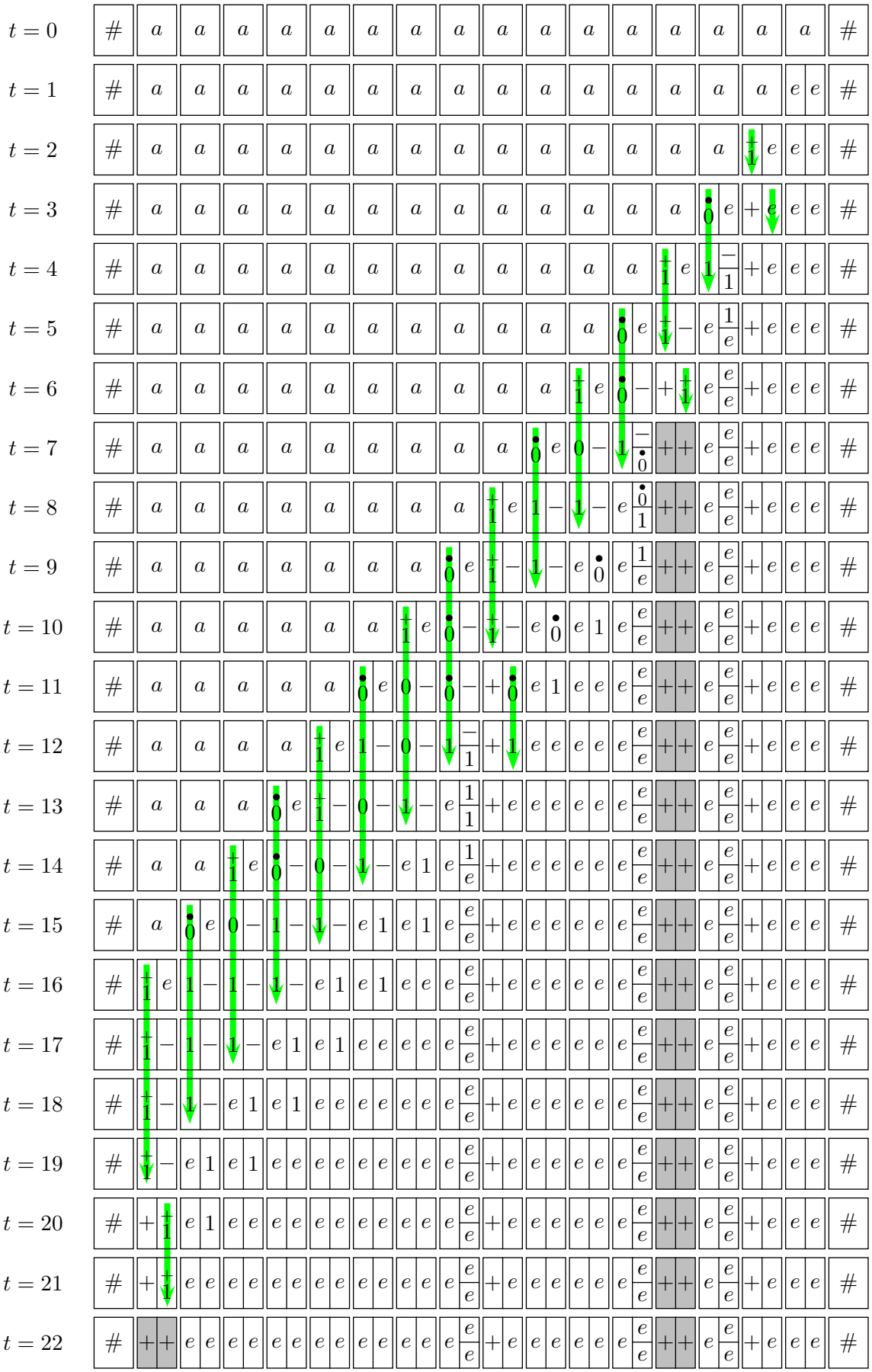



Abbildung 4.4: Ein OCA für die Sprache $\{a^{2^{2^m}} \mid m \geq 1\}$ mit der Eingabe $a^{16}$.

- Jeder Zustand, der in der ersten Spur mit $g$, in der dritten Spur mit $m_2$ und in der vierten Spur mit $m$ markiert ist, wird als ein akzeptierender Zustand definiert.

Wir betrachten nun die Berechnung auf einer Eingabe $wx$ mit $w \in \{\#_0\}\Sigma^+\{\#_1\}$ und $x \in (\{\#_0\}\Sigma^+\{\#_1\})^+$. Wir beobachten folgendes:

- Nach $|wx|$ Zeitschritten ist die erste Spur der linken Randzelle mit $g$ markiert.
- Zum Zeitpunkt $2^{2^{|w|}} + 2^{|w|}$ wurden $2^{2^{|w|}} - |w|$ Signale $\zeta_1$ und $|w|$ Signale $\zeta_2$ und $\xi$ gestartet.

Wir behaupten nun: $|wx|_{\#_0} = 2^{2^{|w|}} \iff$ Nach $|wx| + 2^{|w|}$ Zeitschritten erreichen die Signale $\zeta_2$ und $\xi$ die linke Randzelle und markieren diese mit $m_2$ und im anderen Fall mit $m$.

Die Richtung „$\Rightarrow$“ ist klar nach Definition der beiden Signale. Treffen umgekehrt die beiden Signale zum Zeitpunkt $|wx|+2^{|w|}$ in der linken Randzelle ein, dann stellt das Signal $\zeta_2$ sicher, daß die Anzahl der gelesenen $\#_0$-Zeichen in $wx$ eine doppelte Zweierpotenz ist. Das Signal $\xi$ stellt sicher, daß das zuletzt eingetroffene Signal $\zeta_2$ das $|w|$-te Signal $\zeta_2$ ist. Daher ist die Anzahl der gelesenen $\#_0$-Zeichen gerade die $|w|$-te doppelte Zweierpotenz, also $2^{2^{|w|}}$.

Somit akzeptiert der OCA genau dann, wenn die Anzahl der gelesenen $\#_0$-Zeichen in $wx$ gerade $2^{2^{|w|}}$ ist. Wir müssen nun nur noch sicherstellen, daß der OCA nach spätestens $|wx| + \log|wx|$ Zeitschritten akzeptiert. Sei also $wx \in L_3'$. Nach Konstruktion akzeptiert der OCA nach spätestens $|wx| + 2^{|w|}$ Zeitschritten. Daher verbleibt zu zeigen: $2^{|w|} \leq \log|wx|$.

Es ist $|w| \geq 3$ und $x$ besitzt genau $2^{2^{|w|}} - 1$ $\#_0$-Zeichen. Daher ist $|x| \geq 2^{2^{|w|}} - 1 \geq 2^{2^{|w|}} - |w|$ und damit $|w| + |x| = |wx| \geq 2^{2^{|w|}}$. Also gilt $2^{|w|} \leq \log|wx|$.

Da alle im Beweis verwendeten Konstruktionen effektiv sind, kann also ein OCA mit Zeitkomplexität $t(n) = n + \log n$ effektiv konstruiert werden, der die Sprache $L_3'$ und damit die Sprache $\mathrm{L}'[M]$ akzeptiert. Damit ist (2) und das Lemma bewiesen. □

**Satz 4.8.** Es existiert folgender nichtrekursiver Tradeoff: $(rt + \log n)$-OCA $\overset{nonrec}{\longrightarrow}$ Realzeit-OCA

*Beweis.* Der nichtrekursive Tradeoff folgt unmittelbar aus obigem Lemma unter Anwendung von Satz 2.28. □

**Bemerkung 4.9.** Eine offensichtliche Folgerung aus obigem Satz ist die Existenz eines nichtrekursiven Tradeoffs zwischen Realzeit plus $f(n)$ und Realzeit für $f(n) \in \Omega(\log n)$. Damit impliziert der gerade bewiesene nichtrekursive Tradeoff natürlich auch einen nichtrekursiven Tradeoff zwischen Realzeit und Linearzeit. Somit ist der Beweis der entsprechenden Behauptungen in Lemma 4.3 in gewisser Weise redundant. Allerdings machen beide Beweise auch den Unterschied in den Beweistechniken zwischen beidseitigem und einseitigem Informationsfluß deutlich. Daher können die vorgestellten Beweise beim Beweis weiterer nichtrekursiver Tradeoffs zwischen Zellularautomaten hilfreich sein.

Wir haben in Satz 4.4(1) die Existenz eines nichtrekursiven Tradeoffs zwischen Realzeit-OCA und DFA nachgewiesen. Mit folgendem Beispiel zeigen wir, daß rekursive Tradeoffs von beliebiger Größe nicht nur existieren, sondern explizit konstruiert werden können. Eine ähnliche Konstruktion findet man in [MF71].

**Beispiel 4.10.** Sei $f$ eine rekursive Funktion und $n \in \mathbb{N}$ eine feste natürliche Zahl. Dann gibt es eine reguläre Sprache $L(f, n)$, die von einem Realzeit-OCA mit $O(n)$ Zuständen akzeptiert wird. Aber jeder DFA, der $L(f, n)$ akzeptiert, benötigt $\Omega(f(n))$ Zustände.

*Beweis.* Es sei $f$ eine rekursive Funktion und $n \in \mathbb{N}$ eine feste natürliche Zahl. Dann gibt es eine Turingmaschine $M$ mit unärer Eingabe und Ausgabe, die nur den Wert $f(n)$ berechnet. Daher besteht die Menge $L(f, n) = \text{VALC}[M]$ aus einem Wort $w_n$. Ein DFA, der dieses Wort akzeptiert, benötigt $|w_n| + 1$ Zustände. Also braucht jeder DFA für die Sprache $L(f, n)$ mindestens $\Omega(f(n))$ viele Zustände.

Wir betrachten nun eine Turingmaschine $M'$, die $f(n)$ nicht nur für ein festes $n$ berechnet, sondern für jede Eingabe der Größe $n$. Solch eine Turingmaschine existiert, da $f$ eine rekursive Funktion ist. Nach Satz 4.1 können wir einen Realzeit-OCA $A$ konstruieren, der die Menge VALC$[M]$ akzeptiert. Die Größe von $A$ bezüglich der Länge der Eingabe $n$ ist eine Konstante. Wir modifizieren $A$ nun dahingehend, daß nur die Menge $L(f, n)$ für ein festes $n$ akzeptiert wird. Dazu müssen wir lediglich die Länge der Eingabe $n$ zählen. Also benötigt ein Realzeit-OCA für $L(f, n)$ mit festem $n$ höchstens $O(n)$ Zustände. □

**Bemerkung 4.11.** Ein nichtrekursiver Tradeoff zwischen zwei Beschreibungssystemen $D_1$ und $D_2$ impliziert auch, daß es keinen Algorithmus gibt, der eine Beschreibung $M \in D_1$ in eine Beschreibung $M' \in D_2$ umwandelt. Existierte ein solcher Algorithmus, dann wäre der offenbar durch eine rekursive Funktion beschränkte Platzbedarf des Algorithmus eine rekursive obere Schranke für den Tradeoff. Das heißt, für reguläre und kontextfreie Sprachen gibt es keinen Algorithmus, der einen Realzeit-OCA in einen äquivalenten DFA oder PDA umwandelt. Für Realzeit-OCA-Sprachen gibt es keinen Algorithmus, der einen Realzeit-CA oder einen Linearzeit-OCA in einen äquivalenten Realzeit-OCA umwandelt.

**Bemerkung 4.12.** Unäre Sprachen, die von Realzeit-OCA akzeptiert werden, bilden eine Ausnahme. Es ist bekannt, daß diese Sprachen regulär sind [Sei79]. Darüber hinaus wird in [Sei79] eine Konstruktion vorgestellt, die einen unären Realzeit-OCA mit $n$ Zuständen in einen äquivalenten DFA mit höchstens $n^2$ Zuständen umwandelt. Der nichtrekursive Tradeoff zwischen Realzeit-OCA und DFA reduziert sich also im unären Fall auf einen quadratischen Tradeoff.

Wir können also zusammenfassen, daß es aus Sicht der Beschreibungskomplexität durchaus sinnvoll wäre, reguläre und kontextfreie Sprachen durch Realzeit-OCA zu beschreiben, oder Realzeit-OCA-Sprachen durch Linearzeit-OCA oder Realzeit-CA zu beschreiben, da wir Einsparungen „beliebiger“ Größe erzielen können. Leider muß man aber für die Kürze der Beschreibung einen hohen Preis zahlen, denn das Modell wird sehr unhandlich, da viele Entscheidbarkeitsfragen nicht mehr entscheidbar sind.

## 4.2 Entscheidbarkeitsfragen

In [Sei79] wird durch Reduktionen vom Postschen Korrespondenzproblem gezeigt, daß viele Entscheidbarkeitsfragen für Zellularautomaten unentscheidbar sind. Zum Beispiel wird dies für die Fragen Leere, Endlichkeit, Unendlichkeit und Regularität gezeigt. In [CGS86] wird die Unentscheidbarkeit der Fragen Leere, Universalität und Äquivalenz gezeigt.

Im folgenden nutzen wir das Ergebnis von Satz 4.1 aus: Da die Menge VALC[$M$] von einem Realzeit-OCA akzeptiert werden kann, können wir Unentscheidbarkeitsaussagen über Realzeit-OCA recht einfach auf Unentscheidbarkeitsaussagen über Turingmaschinen zurückführen und erhalten so mit dem Satz von Rice (Satz 2.17) deren Unentscheidbarkeit. Dieses Vorgehen vereinfacht die Beweise und zeigt überdies die Nichtsemientscheidbarkeit der diskutierten Fragen.

Zunächst zeigen wir, daß Leere, Endlichkeit und Unendlichkeit für Turingmaschinen nicht semientscheidbar ist. Später können wir Entscheidbarkeitsfragen für Realzeit-OCA auf diese Fragen zurückführen.

**Lemma 4.13.** Für eine Turingmaschine $M$ sind folgende Fragen nicht semientscheidbar:

(1) Ist $T(M) = \emptyset$?

(2) Ist $T(M)$ endlich?

(3) Ist $T(M)$ unendlich?

*Beweis.* Wir erhalten die Nichtsemientscheidbarkeit der obigen Fragen, indem wir die Verletzung der ersten oder der zweiten Bedingung aus dem Satz von Rice für rekursiv aufzählbare Indexmengen (Satz 2.18) nachweisen.

(1) Sei $L = \emptyset$ und $L' \in \mathrm{RE}$ eine nichtleere Menge. Dann gilt $L \subset L'$. Unter der Annahme, daß (1) semientscheidbar ist, folgt mit Satz 2.18(1) $L' = \emptyset$. Widerspruch.

(2) Sei $L$ eine endliche Menge und $L' \in \mathrm{RE}$ eine unendliche Menge mit $L \subset L'$. Unter der Annahme, daß (2) semientscheidbar ist, folgt mit Satz 2.18(1), daß $L'$ eine endliche Menge ist. Widerspruch.

(3) Sei $L$ eine unendliche Menge. Dann kann es keine endliche Teilmenge $L' \subset L$ geben, die unendlich ist. Also ist die zweite Bedingung aus Satz 2.18 verletzt und (3) somit nicht semientscheidbar.

□

Mit obigem Lemma 4.13 und den Ergebnissen aus Kapitel 4.1 erhalten wir folgende Nichtsemientscheidbarkeitsaussagen:

**Satz 4.14.** Für Realzeit-OCA $A$ und $A'$ sind folgende Fragen nicht semientscheidbar:

(1) Ist $T(A) = \emptyset$?

(2) Ist $T(A) = \Sigma^*$?

(3) Ist $T(A)$ endlich?

(4) Ist $T(A)$ unendlich?

(5) Ist $T(A) = T(A')$?

(6) Ist $T(A) \subseteq T(A')$?

(7) Ist $T(A) \in \mathrm{REG}$?

(8) Ist $T(A) \in \mathrm{CF}$?

*Beweis.* Sei $M$ eine Turingmaschine. Nach Satz 4.1 können zwei Realzeit-OCA $A_1$ und $A_2$ konstruiert werden mit $T(A_1) = \mathrm{VALC}[M]$ und $T(A_2) = \mathrm{INVALC}[M]$.

(1) Es gilt offenbar: $T(M) = \emptyset \Leftrightarrow T(A_1) = \mathrm{VALC}[M] = \emptyset$. Unter der Annahme, daß Leere für Realzeit-OCA semientscheidbar ist, ist Leere dann auch für Turingmaschinen semientscheidbar, was im Widerspruch zu Lemma 4.13(1) steht.

(2) Es gilt: $T(M) = \emptyset \Leftrightarrow \mathrm{VALC}[M] = \emptyset \Leftrightarrow T(A_2) = \mathrm{INVALC}[M] = \Sigma^*$. Unter der Annahme, daß Universalität für Realzeit-OCA semientscheidbar ist, folgt wiederum die Semientscheidbarkeit der Leere für Turingmaschinen, was im Widerspruch zu Lemma 4.13(1) steht.

(3) Es gilt: $T(M)$ ist endlich $\Leftrightarrow \mathrm{VALC}[M]$ ist endlich. Ist die Endlichkeit für Realzeit-OCA semientscheidbar, dann ist sie auch für Turingmaschinen semientscheidbar. Das ist ein Widerspruch zu Lemma 4.13(2).

(4) Es gilt: $T(M)$ ist unendlich $\Leftrightarrow$ $\mathrm{VALC}[M]$ ist unendlich. Ist die Unendlichkeit für Realzeit-OCA semientscheidbar, dann ist sie auch für Turingmaschinen semientscheidbar. Das ist ein Widerspruch zu Lemma 4.13(3).

(5) Sei $A'$ ein Realzeit-OCA mit $T(A') = \Sigma^*$. Ist nun für Realzeit-OCA Äquivalenz semientscheidbar, dann ist die Frage „Ist $T(A) = T(A') = \Sigma^*$?“ semientscheidbar. Das steht im Widerspruch zu (2).

(6) Sei $A$ ein Realzeit-OCA mit $T(A) = \Sigma^*$ und $A'$ ein Realzeit-OCA über dem Alphabet $\Sigma$. Ist nun für Realzeit-OCA Inklusion semientscheidbar, dann ist die Frage „Ist $\Sigma^* = T(A) \subseteq T(A')$?“ äquivalent zu „Ist $T(A') = \Sigma^*$?“ und somit semientscheidbar. Das steht im Widerspruch zu (2).

(7) Es gilt nach Lemma 4.3(1): $\mathrm{INVALC}[M] \in \mathrm{REG} \Leftrightarrow T(M)$ ist endlich. Ist Regularität für Realzeit-OCA semientscheidbar, dann ist auch Endlichkeit für Turingmaschinen semientscheidbar, was ein Widerspruch zu Lemma 4.13(2) ist.

(8) Es gilt nach Lemma 4.3(2): $\mathrm{VALC}[M] \in \mathrm{CF} \Leftrightarrow T(M)$ ist endlich. Ist Kontextfreiheit für Realzeit-OCA semientscheidbar, dann ist auch Endlichkeit für Turingmaschinen semientscheidbar, was ein Widerspruch zu Lemma 4.13(2) ist.

□

Offenbar sind die obigen Aussagen nicht nur auf Realzeit-OCA beschränkt, sondern können in folgender Weise verallgemeinert werden:

**Korollar 4.15.** Obige Fragen sind nicht semientscheidbar sowohl für Automaten $A$ und $A'$ aus einer Automatenklasse, die **Realzeit-OCA** enthält, als auch für Sprachen aus einer Sprachklasse, in der $\mathcal{L}_{rt}(\mathrm{OCA})$ enthalten ist.

**Bemerkung 4.16.** Für zwei Realzeit-OCA $A, A'$ und eine reguläre Sprache $R$ wird in Seidel [Sei79] die Unentscheidbarkeit der folgenden Fragen bewiesen:

- Ist $T(A) \cap T(A') = \emptyset$?
- Ist $T(A) = R$?
- Ist $R \subseteq T(A)$?
- Ist $\overline{T(A)} = \emptyset$?

Die Nichtsemientscheidbarkeit dieser Fragen folgt nun allerdings unmittelbar aus den Aussagen (1) und (2) aus Satz 4.14. Wir wählen $A'$ mit $T(A') = \Sigma^*$, setzen $R = \Sigma^*$ und beachten die Abgeschlossenheit von $\mathcal{L}_{rt}(\text{OCA})$ unter Komplementbildung.

Nach Lemma 4.14(7) und (8) ist es für Realzeit-OCA nicht semientscheidbar, ob der Automat eine Sprache aus einer bestimmten Teilklasse von $\mathcal{L}_{rt}(\text{OCA})$ akzeptiert. In ähnlicher Weise kann man sich fragen, ob man entscheiden kann, ob ein Realzeit-CA eine Sprache aus $\mathcal{L}_{rt}(\text{OCA})$ akzeptiert. Aber auch diese Frage ist nicht semientscheidbar.

**Satz 4.17.** Für einen Realzeit-CA $A$ ist die Frage „Ist $T(A) \in \mathcal{L}_{rt}(\text{OCA})$?“ nicht semientscheidbar.

*Beweis.* Es sei $M$ eine Turingmaschine. Nach Lemma 4.3(4) können wir einen Realzeit-CA $A$ konstruieren, der die Sprache $\text{L}[M]$ akzeptiert. Nach Lemma 4.3(3) gilt: $\text{L}[M] \in \mathcal{L}_{rt}(\text{OCA}) \Leftrightarrow T(M)$ ist endlich. Ist die obige Frage also semientscheidbar, dann ist auch Endlichkeit für Turingmaschinen semientscheidbar, was ein Widerspruch zu Lemma 4.13(2) ist. □

**Korollar 4.18.** Für einen Automaten $A$ aus einer Automatenklasse, die **Realzeit-CA** enthält, ist die Frage „Ist $T(A) \in \mathcal{L}_{rt}(\text{OCA})$?“ nicht semientscheidbar. Für eine Sprache $L \in \mathcal{L}_{rt}(\text{CA})$ und jede Sprachklasse, die $\mathcal{L}_{rt}(\text{CA})$ enthält, ist die Frage „Ist $L \in \mathcal{L}_{rt}(\text{OCA})$?“ nicht semientscheidbar.

Da die rekursiven Sprachen unter Komplement abgeschlossen sind, ist das Komplement einer unentscheidbaren Frage natürlich ebenfalls unentscheidbar. Die rekursiv aufzählbaren Sprachen sind nicht unter Komplement abgeschlossen. Also kann es durchaus sein, daß das Komplement einer nicht semientscheibaren Frage selbst semientscheidbar ist. Wir bezeichnen solch eine Frage dann als co-semientscheidbar. Für Turingmaschinen ist Leere nicht semientscheidbar, die Frage der Nichtleere ist allerdings semientscheidbar (vgl. [HU79]). Also ist Leere für Turingmaschinen nicht semientscheidbar, aber co-semientscheidbar.

**Satz 4.19.** Für Realzeit-OCA $A$ und $A'$ sind folgende Fragen co-semientscheidbar:

(1) Ist $T(A) = \emptyset$?

(2) Ist $T(A) = \Sigma^*$?

(3) Ist $T(A) = T(A')$?

(4) Ist $T(A) \subseteq T(A')$?

Für einen Realzeit-OCA $A$ sind folgende Fragen nicht co-semientscheidbar:

(5) Ist $T(A)$ endlich?

(6) Ist $T(A)$ unendlich?

(7) Ist $T(A) \in \mathrm{REG}$?

(8) Ist $T(A) \in \mathrm{CF}$?

Für einen Realzeit-CA $A$ ist folgende Frage nicht co-semientscheidbar:

(9) Ist $T(A) \in \mathcal{L}_{rt}(\mathrm{OCA})$?

*Beweis.* (1) Nichtleere ist für Turingmaschinen semientscheidbar (vgl. Corollary 2 zu Theorem 8.7 in [HU79]) und daher auch für Realzeit-OCA.

(2) $T(A) \neq \Sigma^* \Leftrightarrow \overline{T(A)} \neq \emptyset$. Da $\mathcal{L}_{rt}(\mathrm{OCA})$ unter Komplementbildung abgeschlossen ist, kann Nichtuniversalität somit auf Nichtleere reduziert werden, und ist damit semientscheidbar.

(3) $T(A) \neq T(A') \Leftrightarrow T(A) \setminus T(A') \neq \emptyset$ oder $T(A') \setminus T(A) \neq \emptyset \Leftrightarrow T(A) \cap \overline{T(A')} \neq \emptyset$ oder $T(A') \cap \overline{T(A)} \neq \emptyset$. Da $\mathcal{L}_{rt}(\mathrm{OCA})$ unter Schnitt- und Komplementbildung abgeschlossen ist, kann Nichtäquivalenz daher auf Nichtleere reduziert werden, und ist also semientscheidbar.

(4) $T(A) \not\subseteq T(A') \Leftrightarrow T(A) \setminus T(A') \neq \emptyset \Leftrightarrow T(A) \cap \overline{T(A')} \neq \emptyset$. Wiederum kann Nichtinklusion auf Nichtleere reduziert werden und ist mithin semientscheidbar.

Endlichkeit (5) und Unendlichkeit (6) sind offenbar nicht co-semientscheidbar.

(7) Es folgt aus Lemma 4.3(1): $\mathrm{INVALC}[M] \notin \mathrm{REG} \Leftrightarrow T(M)$ ist unendlich. Ist Nichtregularität für Realzeit-OCA semientscheidbar, dann ist auch Unendlichkeit für Turingmaschinen semientscheidbar, was ein Widerspruch zu Lemma 4.13(3) ist.

(8) Es folgt aus Lemma 4.3(2): $\mathrm{VALC}[M] \notin \mathrm{CF} \Leftrightarrow T(M)$ ist unendlich. Ist Nichtkontextfreiheit für Realzeit-OCA semientscheidbar, dann ist auch Unendlichkeit für Turingmaschinen semientscheidbar, was ein Widerspruch zu Lemma 4.13(3) ist.

(9) Es folgt aus Lemma 4.3(3): $\mathrm{L}[M] \notin \mathcal{L}_{rt}(\mathrm{OCA}) \Leftrightarrow T(M)$ ist unendlich. Ist das Nichtenthaltensein in $\mathcal{L}_{rt}(\mathrm{OCA})$ für Realzeit-CA semientscheidbar, dann ist auch Unendlichkeit für Turingmaschinen semientscheidbar, was ein Widerspruch zu Lemma 4.13(3) ist.

□

Die Aussagen können wieder folgendermaßen verallgemeinert werden:

**Korollar 4.20.** Die obigen Fragen (1) bis (4) sind co-semientscheidbar für Automatenklassen, deren entsprechende Sprachklasse in REC enthalten ist. Die Fragen (5) bis (8) sind nicht co-semientscheidbar für Automatenklassen, in denen **Realzeit-OCA** enthalten ist. Die Frage (9) ist für Automatenklassen, die **Realzeit-CA** enthalten, nicht co-semientscheidbar.

*Beweis.* Es ist offensichtlich, daß die Fragen (5) bis (9) nicht co-semientscheidbar sind. Sei nun $\mathcal{A}$ eine Automatenklasse mit $T(A) \in \mathrm{REC}$ für alle $A \in \mathcal{A}$. Die Fragen (2) bis (4) können unter Anwendung von Schnitt- und Komplementbildung auf Nichtleere reduziert werden. Da $\mathcal{A}$ eine Sprachklasse erzeugt, die in REC enthalten ist, und REC unter Schnitt- und Komplementbildung abgeschlossen ist, können diese Fragen auf Nichtleere für REC reduziert werden. Nichtleere ist aber für Turingmaschinen semientscheidbar und damit auch für REC. Also sind die Fragen (1) bis (4) co-semientscheidbar. □

Nach Satz 3.15 ist $\mathcal{L}_{rt}(\mathrm{OCA})$ weder unter Homomorphismus noch unter $\epsilon$-freiem Homomorphismus abgeschlosssen. Daher kann man sich fragen, ob das ($\epsilon$-freie) homomorphe Bild einer Sprache aus $\mathcal{L}_{rt}(\mathrm{OCA})$ regulär, kontextfrei oder selbst wieder aus $\mathcal{L}_{rt}(\mathrm{OCA})$ ist. Wir zeigen jetzt, daß diese Fragen nicht semientscheidbar und auch nicht co-semientscheidbar sind.

**Satz 4.21.** Es sei $A$ ein Realzeit-OCA, $h$ ein Homomorphismus und $h_\epsilon$ ein $\epsilon$-freier Homomorphismus. Dann sind folgende Fragen weder semientscheidbar noch co-semientscheidbar.

(1) Ist $h(T(A)) \in \mathrm{REG}$?

(2) Ist $h(T(A)) \in \mathrm{CF}$?

(3) Ist $h(T(A)) \in \mathcal{L}_{rt}(\mathrm{OCA})$?

(4) Ist $h_\epsilon(T(A)) \in \mathrm{REG}$?

(5) Ist $h_\epsilon(T(A)) \in \mathrm{CF}$?

(6) Ist $h_\epsilon(T(A)) \in \mathcal{L}_{rt}(\mathrm{OCA})$?

*Beweis.* Es sei $M$ eine Turingmaschine. Nach Satz 4.2 gibt es einen Realzeit-OCA $A$ und einen Homomorphismus $h$ mit $h(T(A)) = T(M)$. Sind die Fragen (1), (2) und (3) jeweils für Realzeit-OCA semientscheidbar, dann sind diese auch für Turingmaschinen semientscheidbar. Da $\mathcal{L}_{rt}(\mathrm{OCA}) \subseteq \mathcal{L}_{rt}(\mathrm{CA}) \subseteq \mathrm{RE}$ gilt, bekommen wir einen Widerspruch zu Korollar 4.15 und Korollar 4.18.

Die Sprachklasse $\mathcal{L}_{rt}(\mathrm{OCA})$ ist nicht unter $\epsilon$-freiem Homomorphismus abgeschlossen. Nach Satz 3.18 ist der Abschluß von $\mathcal{L}_{rt}(\mathrm{OCA})$ unter $\epsilon$-freiem Homomorphismus durch Realzeit-One-guess-OCA charakterisiert. Es sei also $A'$ ein Realzeit-1G-OCA. Dann gibt es einen Realzeit-OCA $A$ und einen $\epsilon$-freien Homomorphismus $h_\epsilon$, so daß $T(A') = h_\epsilon(T(A))$ gilt. Sind die Fragen (4), (5) und (6) jeweils für Realzeit-OCA semientscheidbar, dann sind sie auch für Realzeit-1G-OCA semientscheidbar. Da nach [BKK02] die Inklusionen $\mathcal{L}_{rt}(\mathrm{OCA}) \subset \mathcal{L}_{rt}(\mathrm{1G\text{-}OCA})$ und $\mathcal{L}_{rt}(\mathrm{CA}) \subseteq \mathcal{L}_{rt}(\mathrm{1G\text{-}OCA})$ gelten, bekommen wir einen Widerspruch zu Korollar 4.15 und Korollar 4.18.

Die Fragen (1) bis (6) sind nicht co-semientscheidbar nach Korollar 4.20. □

Die obigen Ergebnisse werden in folgendem Korollar auf allgemeinere Sprach- und Automatenklassen übertragen.

**Korollar 4.22.** Die Fragen (1), (2), (4) und (5) sind weder semientscheidbar noch co-semientscheidbar sowohl für Automatenklassen, die **Realzeit-OCA** enthalten, als auch für Sprachen aus einer Sprachklasse, in der $\mathcal{L}_{rt}(\mathrm{OCA})$ enthalten ist. Die Fragen (3) und (6) sind weder semientscheidbar noch co-semientscheidbar sowohl für Automatenklassen, die **Realzeit-CA** enthalten, als auch für Sprachen aus einer Sprachklasse, in der $\mathcal{L}_{rt}(\mathrm{CA})$ enthalten ist.

## 4.3 Weitere Ergebnisse

Die eben bewiesenen Ergebnisse können nun angewendet werden, um zu zeigen, daß es kein Pumpinglemma und keinen Minimierungsalgorithmus für Zellularautomaten gibt. Zunächst definieren wir, was wir unter einem Pumpinglemma verstehen.

Nach [Bun95] sagen wir, daß eine Sprachklasse $\mathcal{L}$ ein Pumpinglemma besitzt, falls für $\mathcal{L}$ folgende Eigenschaft gilt: Zu jeder Sprache $L \in \mathcal{L}$ existiert eine aus einer Beschreibung von $L$ algorithmisch zu konstruierende Zahl $n \in \mathbb{N}$ und zu jedem $z \in L$ mit $|z| > n$ gibt es eine Zerlegung $z = uvw$, so daß $|v| \geq 1$ ist und für unendlich viele $i \in \mathbb{N}$ gilt $u'v^iw' \in L$, wobei $u'$ und $w'$ von $u, w$ und $i$ abhängen können.

**Satz 4.23.** $\mathcal{L}_{rt}(\text{OCA})$ und jede Sprachklasse, die $\mathcal{L}_{rt}(\text{OCA})$ enthält, besitzt kein derartiges Pumpinglemma.

*Beweis.* Sei $A = (Q, \#, \Sigma, \delta, F)$ ein Realzeit-OCA. Wir nehmen an, daß es ein Pumpinglemma gibt. Dann sei $n$ die Konstante des Pumpinglemmas. Damit gilt:

$$T(A) \text{ ist unendlich } \iff \exists x \in T(A) : |x| > n$$

Die Richtung „$\Rightarrow$“ ist offensichtlich. Für die andere Richtung betrachten wir ein $x \in T(A)$ mit $|x| > n$. Dann sind die Voraussetzungen des Pumpinglemmas erfüllt, und wir erhalten unendlich viele Wörter in der Sprache durch Pumpen.

Dann kann aber eine Turingmaschine konstruiert werden, die sukzessive alle Wörter aus $\Sigma^*$ der Länge größer als $n$ generiert und den Realzeit-OCA $A$ auf diesen Eingaben simuliert. Gibt es ein $x \in T(A)$ mit $|x| > n$, dann hält die Turingmaschine an. Damit ist die Frage „Ist $T(A)$ unendlich?“ semientscheidbar, was im Widerspruch zu Satz 4.14(4) steht. □

**Korollar 4.24.** Jede Sprachklasse $\mathcal{L}$, die zu einer Turingmaschine $M$ die Menge VALC$[M]$ enthält, besitzt kein Pumpinglemma.

*Beweis.* Für das Ergebnis in obigem Beweis reichte die Nichtsemientscheidbarkeit der Unendlichkeit für Realzeit-OCA aus. Um diese zu zeigen, wurde in Satz 4.14(4) die Tatsache VALC$[M] \in \mathcal{L}_{rt}(\text{OCA})$ ausgenutzt. Ähnliche Überlegungen können daher für alle Sprachklassen $\mathcal{L}$ mit VALC$[M] \in \mathcal{L}$ angestellt werden. □

Im folgenden bezeichnen wir mit $\mathcal{A}$ eine Zellularautomatenklasse mit Zeitkomplexität $t$. Weiterhin gelte **Realzeit-OCA** $\subseteq \mathcal{A}$.

**Satz 4.25.** Für Zellularautomaten aus $\mathcal{A}$ gibt es keinen Minimierungsalgorithmus, der einen beliebigen Zellularautomaten $A \in \mathcal{A}$ in einen Zellularautomaten $A' \in \mathcal{A}$ umwandelt, der $T(A)$ akzeptiert und eine minimale Anzahl von Zuständen besitzt.

*Beweis.* Offenbar benötigt ein minimaler Zellularautomat $A = (Q, \#, \Sigma, \delta, F)$, der $L = \emptyset$ mit Zeitkomplexität $t$ akzeptiert, genau $|\Sigma|$ Zustände und besitzt keine akzeptierenden Zustände. Wir nehmen an, daß es einen Minimierungsalgorithmus gibt. Es sei $A \in \mathcal{A}$ ein beliebiger Zellularautomat. Wir wenden den Minimierungsalgorithmus an und erhalten einen minimalen Zellularautomaten $A' \in \mathcal{A}$. Nun überprüfen wir, ob $A'$ keine akzeptierenden Zustände hat und ob $|Q'| = |\Sigma|$ ist. Trifft beides zu, dann gilt: $T(A') = T(A) = \emptyset$. Falls $|Q'| = |\Sigma|$

und $A'$ akzeptierende Zustände besitzt, dann gibt es mindestens ein Alphabetsymbol, das ein akzeptierender Zustand ist. Dann ist allerdings die akzeptierte Sprache nicht leer. Also ist die Frage „Ist $T(A) = \emptyset$?“ entscheidbar. Das ist ein Widerspruch zu Satz 4.14(1) und Korollar 4.15. □

Neben den in Kapitel 4.2 betrachteten Entscheidbarkeitsfragen, sind für Automaten oftmals Erreichbarkeitsfragen interessant. Zum Beispiel kann man sich fragen, ob ein Zellularautomat im Laufe seiner Berechnung eine bestimmte Konfiguration jemals erreicht, oder ob eine Zelle einen bestimmten Zustand jemals erreicht. Wir werden im folgenden einige Erreichbarkeitsfragen formulieren und deren Entscheidbarkeit oder Unentscheidbarkeit beweisen.

Gegeben sei ein Zellularautomat $A = (Q, \#, \Sigma, \delta, F)$ aus der Automatenklasse $\mathcal{A}$ mit Zeitkomplexität $t$, ein Zustand $q \in Q$ und eine Konfiguration $c \in Q^+$.

(E1) Gibt es eine Eingabe $x \in \Sigma^+$ und einen Zeitpunkt $l \leq t(|x|)$, so daß $\Delta^l(x) = c$ gilt?

(E2) Gibt es eine Eingabe $x \in \Sigma^+$ und einen Zeitpunkt $l \leq t(|x|)$, so daß $c$ ein Präfix von $\Delta^l(x)$ ist?

(E3) Gibt es eine Eingabe $x \in \Sigma^+$ und einen Zeitpunkt $l \leq t(|x|)$, so daß eine Zelle $i$ den Zustand $q$ annimmt?

**Satz 4.26.** Die Frage (E1) ist entscheidbar, die Fragen (E2) und (E3) sind nicht entscheidbar.

*Beweis.* (E1) Ein Algorithmus generiert sukzessive alle Eingaben der Länge $|c|$, simuliert die Berechnung von $A$ auf diesen Eingaben und überprüft, ob die gesuchte Konfiguration bis zum Zeitpunkt $t(|c|)$ in einer Berechnung auftaucht. Sobald eine solche Eingabe gefunden ist, terminiert der Algorithmus und gibt `ja` zurück. Anderenfalls terminiert der Algorithmus und gibt `nein` zurück.

(E2) Wir nehmen an, daß die Frage entscheidbar ist. Wir testen nun für jeden Zustand $q \in F$, ob $c = q$ ein Präfix von $\Delta^l(x)$ ist. Wird die Frage für alle $q \in F$ negativ beantwortet, dann ist $T(A) = \emptyset$, anderenfalls ist $T(A) \neq \emptyset$. Dann ist aber für Zellularautomaten $A \in \mathcal{A}$ entscheidbar, ob die akzeptierte Sprache leer ist. Das ist ein Widerspruch zu Satz 4.14 und Korollar 4.15.

(E3) Wir nehmen an, daß die Frage entscheidbar ist. Wir testen für jeden Zustand $q \in F$, ob die erste Zelle ($i = 1$) den Zustand $q$ annimmt. Wird die Frage für alle $q \in F$ negativ beantwortet, dann ist $T(A) = \emptyset$, anderenfalls ist $T(A) \neq \emptyset$. Wiederum ist dann für Zellularautomaten $A \in \mathcal{A}$ entscheidbar, ob die akzeptierte Sprache leer ist, was im Widerspruch zu Satz 4.14 und Korollar 4.15 steht.

□

**Bemerkung 4.27.** Obige Aussagen gelten insbesondere für Zellularautomaten, die in Realzeit und Linearzeit arbeiten. Nach Bemerkung 3.3 gibt es zu einem Zellularautomaten, der eine Sprache in beliebiger Zeit akzeptiert, einen äquivalenten Zellularautomaten mit Zeitkomplexität $t(n) = O(c^n)$ für eine Konstante $c$. Also gelten obige Aussagen auch für Zellularautomaten ohne explizite Zeitbeschränkung.

**Bemerkung 4.28.** Alle in diesem Kapitel bewiesenen Unentscheidbarkeits- und Nichtsemientscheidbarkeitsaussagen gelten für alle in Kapitel 3 eingeführten Zellularautomatenklassen. Also für **Realzeit-OCA**, **Realzeit-CA**, **Linearzeit-OCA**, **Linearzeit-CA**, **OCA**, **CA**, **Realzeit-1G-OCA** und OCA, die in Realzeit plus log arbeiten. Für alle diese Automatenklassen gibt es außerdem kein Pumpinglemma und keinen Minimierungsalgorithmus. Weiterhin sind einige Erreichbarkeitsfragen nicht entscheidbar. Mit anderen Worten: Für Zellularautomaten ist fast nichts entscheidbar und semientscheidbar. Eine Ausnahme bilden wiederum Realzeit-OCA, die unäre Sprachen akzeptieren. Da diese Automaten algorithmisch in DFA umgewandelt werden können, reduzieren sich die Fragen auf Entscheidbarkeitsfragen für DFA und reguläre Sprachen und sind somit oftmals entscheidbar.

## 4.4 Unvergleichbarkeit

Nach Satz 4.2 lassen sich die rekursiv aufzählbaren Sprachen durch das homomorphe Bild von $\mathcal{L}_{rt}(\text{OCA})$ charakterisieren. Daher können wir folgendes Unvergleichbarkeitskriterium herleiten:

**Satz 4.29.** $\mathcal{L}_{rt}(\text{OCA})$ ist unvergleichbar mit jeder Sprachklasse $\mathcal{L}$ für die gilt: 2-LCF $\subseteq \mathcal{L} \subseteq \mathcal{L}' \subset$ RE und $\mathcal{L}'$ ist abgeschlossen unter Homomorphismus.

*Beweis.* Nach Satz 3.12 ist 2-LCF nicht in $\mathcal{L}_{rt}(\text{OCA})$ enthalten. Daher ist 2-LCF$\setminus\mathcal{L}_{rt}(\text{OCA}) \neq \emptyset$. Damit ist $\mathcal{L} \setminus \mathcal{L}_{rt}(\text{OCA}) \neq \emptyset$.

Wir nehmen nun an, daß $\mathcal{L}_{rt}(\text{OCA}) \subseteq \mathcal{L}$ gilt. Dann gilt $h(\mathcal{L}_{rt}(\text{OCA})) \subseteq h(\mathcal{L}) \subseteq \mathcal{L}'$. Aus Satz 4.2 folgt dann, daß RE $\subseteq \mathcal{L}'$ gilt. Das ist aber ein Widerspruch zu der Annahme, daß $\mathcal{L}'$ eine echte Teilmenge von RE ist. Also gilt: $\mathcal{L}_{rt}(\text{OCA}) \setminus \mathcal{L} \neq \emptyset$. □

**Korollar 4.30.** $\mathcal{L}_{rt}(\text{OCA})$ ist unvergleichbar mit jeder Sprachklasse $\mathcal{L}$, für die folgendes gilt: CF $\subseteq \mathcal{L} \subset$ RE und $\mathcal{L}$ ist abgeschlossen unter Homomorphismus.

**Korollar 4.31.** $\mathcal{L}_{rt}(\text{OCA})$ ist mit folgenden Sprachklassen $\mathcal{L}$ unvergleichbar:

(1) $\mathcal{L} = \text{CF}$

(2) $\mathcal{L} = k$-LCF für $k \geq 2$

(3) 2-LCF $\subseteq \mathcal{L} \subseteq$ CF

(4) Jede volle AFL $\mathcal{L}$ mit LCF $\subseteq \mathcal{L} \subset$ RE.

*Beweis.* (1), (2) und (3) sind unmittelbar klar. Zum Beweis von (4) beachten wir, daß $\mathcal{L}$ insbesondere unter Konkatenation abgeschlossen ist und somit 2-LCF enthält. □

Eine Anwendung dieses Kriteriums zeigt, daß $\mathcal{L}_{rt}(\text{OCA})$ mit vielen bekannten und gut untersuchten Sprachklassen unvergleichbar ist. Beispielsweise mit den Sprachklassen, die von folgenden Formalismen erzeugt werden. Wir werden die einzelnen Formalismen hier nicht einführen, sondern verweisen auf die entsprechende Literatur.

(1) Indexgrammatiken $I$ (siehe [DPS97]): Es gilt CF $\subset \mathcal{L}(I) \subset$ RE und $\mathcal{L}(I)$ ist abgeschlossen unter Homomorphismus.

(2) Verschiedene Grammatiken mit gesteuerter Ersetzung (siehe [DPS97]):

- Matrixgrammatiken ohne *appearance checking* $\lambda M$, $M$: Es gilt CF $\subset \mathcal{L}(M) \subset \mathcal{L}(\lambda M) \subset$ RE und $\mathcal{L}(\lambda M)$ ist abgeschlossen unter Homomorphismus.
- Vektorgrammatiken ohne löschende Regeln $uV$: Es gilt CF $\subset \mathcal{L}(uV) \subset \mathcal{L}(\lambda M)$.
- Geordnete Grammatiken $\lambda O$, $O$: Es gilt CF $\subset \mathcal{L}(O) \subset \mathcal{L}(\lambda O) \subset$ RE und $\mathcal{L}(\lambda O)$ ist abgeschlossen unter Homomorphismus.
- Regulär kontrollierte Grammatiken ohne *appearance checking* $\lambda rC$, $rC$: Es gilt CF $\subset \mathcal{L}(rC) \subseteq \mathcal{L}(\lambda rC) \subseteq \mathcal{L}(\lambda M)$.

(3) Verschiedene Lindenmayersysteme (siehe [KRS97]) wie beispielsweise ET0L. Es gilt CF $\subset \mathcal{L}$(ET0L) $\subset$ RE und $\mathcal{L}$(ET0L) ist abgeschlossen unter Homomorphismus.

(4) Verschiedene Grammatiksysteme (siehe [DPR97])

- Viele *CD grammar systems*, also Grammatiksysteme mit sequentieller Arbeitsweise. In Theorem 3.1 und Theorem 3.2 in [DPR97] wird gezeigt, daß viele CD grammar systems Sprachklassen erzeugen, die CF enthalten und selbst enthalten sind in $\mathcal{L}(M)$, $\mathcal{L}(\lambda M)$ oder ET0L. Mit den Ergebnissen aus (2) und (3) folgt dann die Unvergleichbarkeit.
- Einige CD grammar systems mit *teams*: Die erzeugten Sprachklassen sind $\mathcal{L}(M)$ oder $\mathcal{L}(\lambda M)$. (Theorem 3.5 in [DPR97])

(5) Verschiedene *contextual grammars* (siehe [EPR97]), z.B. $\mathrm{TC}_c$. Es gilt CF $\subset \mathcal{L}(\mathrm{TC}_c) \subset$ RE und $\mathcal{L}(\mathrm{TC}_c)$ ist abgeschlossen unter Homomorphismus.

(6) Verschiedene Sprachklassen aus dem Gebiet des *membrane computing* (siehe [Pău02]).

Insbesondere folgt die Unvergleichbarkeit mit allen Teilmengen der genannten Sprachklassen, die 2-LCF enthalten.

## 4.5 Zusammenfassung

Wir haben in diesem Kapitel die Beschreibungskomplexität von Zellularautomaten untersucht und nichtrekursive Tradeoffs zwischen Realzeit-OCA und verschiedenen sequentiellen Modellen bewiesen. Weitere nichtrekursive Tradeoffs existieren zwischen Linearzeit-OCA und Realzeit-OCA sowie zwischen Realzeit-CA und Realzeit-OCA. Im Vergleich mit der Sprachhierarchie aus Satz 3.9 haben wir nichtrekursive Tradeoffs zwischen allen Beschreibungsmechanismen nachgewiesen, deren Sprachklassen echt ineinander enthalten sind. Die Tradeoffs zwischen anderen Zellularautomatenklassen, z.B. zwischen Linearzeit-CA und Realzeit-CA sind derzeit unbekannt. Um beispielsweise einen nichtrekursiven Tradeoff zwischen Linearzeit-CA und Realzeit-CA mit Satz 2.28 nachzuweisen, müßte man eine Sprache $L_M \in \mathcal{L}_{lt}(\mathrm{CA})$ finden, die nur dann in $\mathcal{L}_{rt}(\mathrm{CA})$ ist, falls $T(M)$ endlich ist. Da bislang keine Methoden bekannt sind, um zu zeigen, daß eine Sprache aus $\mathcal{L}_{lt}(\mathrm{CA})$ nicht in $\mathcal{L}_{rt}(\mathrm{CA})$ ist, und darüber hinaus noch nicht einmal klar ist, ob eine der Inklusionen oberhalb von $\mathcal{L}_{rt}(\mathrm{CA})$ echt ist, können wir die in Satz 2.28 zusammengefaßte Methode daher nicht verwenden.

Wir haben weiterhin Entscheidbarkeitsfragen für Zellularautomaten untersucht. Hier war das Ergebnis, daß sehr viele Fragen für Zellularautomaten unentscheidbar und auch nicht semientscheidbar sind. Auch konnten wir nachweisen, daß es für Zellularautomaten und den entsprechenden Sprachklassen keine Werkzeuge wie beispielsweise ein Pumpinglemma oder einen Minimierungsalgorithmus gibt. Diese Ergebnisse sind natürlich nicht befriedigend, wenn man an die praktische Verwendbarkeit von Zellularautomaten denkt. Denn manche praktische Aufgaben, wie zum Beispiel das Minimieren eines Automaten oder die Überprüfung, ob zwei Automaten die gleiche Sprache akzeptieren, können nicht algorithmisch gelöst werden, sondern müssen aufgrund der Unentscheidbarkeitsergebnisse immer *ad hoc* erledigt werden.

Daher ist es zunächst naheliegend, ähnliche Modelle und deren Beschreibungskomplexität zu untersuchen. Weiterhin sind auch geeignete abgeschwächte Modelle von Zellularautomaten interessant, um „handlichere“ Automatenklassen mit rekursiv beschränkten Größeneinsparungen zu erhalten. Diese Untersuchungen werden in den folgenden Kapiteln unternommen.

# Kapitel 5

# Nichtrekursive Tradeoffs in iterativen Arrays

Im vorigen Kapitel haben mir mit Zellularautomaten ein paralleles Modell mit paralleler Eingabe betrachtet. Als Abschwächung untersuchen wir hier mit iterativen Arrays ein paralleles Modell mit sequentieller Eingabe. Aus Sicht der Beschreibungskomplexität ergeben sich ähnliche Ergebnisse wie bei Zellularautomaten. Wir haben nichtrekursive Tradeoffs zwischen Realzeit-IA und sequentiellen Modellen und zwischen Linearzeit-IA und Realzeit-IA. Bezüglich der generativen Mächtigkeit sind $\mathcal{L}_{rt}(\text{OCA})$ und $\mathcal{L}_{rt}(\text{IA})$ unvergleichbar. Wir erhalten in diesem Kapitel das Ergebnis, daß beide Sprachklassen auch aus Sicht der Beschreibungskomplexität unvergleichbar sind, da nichtrekursive Tradeoffs zwischen Realzeit-IA und Realzeit-OCA und umgekehrt zwischen Realzeit-OCA und Realzeit-IA existieren. Die Ergebnisse lassen sich ähnlich wie im vorigen Kapitel mit Satz 2.28 beweisen, und wir zeigen dazu in Kapitel 5.2, unter welchen Bedingungen die gültigen und ungültigen Berechnungen einer Turingmaschine und Variationen dieser Sprachen von Realzeit-IA und Linearzeit-IA akzeptiert werden können. Einige notwendige Vorbereitungen werden in Kapitel 5.1 zusammengefaßt. In den Kapiteln 5.3 und 5.4 ergänzen wir die Ergebnisse, die für Zellularautomaten bekannt sind: Auch für iterative Arrays sind viele Entscheidbarkeitsfragen unentscheidbar und nicht semientscheidbar. Ebenfalls gibt es für iterative Arrays keinen Minimierungsalgorithmus und kein Pumpinglemma für die entsprechenden Sprachklassen. Schließlich läßt sich auch für $\mathcal{L}_{rt}(\text{IA})$ ein Unvergleichbarkeitskriterium formulieren.

## 5.1 Technische Vorbereitungen

Für eine spätere Konstruktion wird es wichtig sein, daß ein Automat die Zustände, die im ersten Berechnungsschritt angenommen werden, in späteren Berechnungsschritten nicht mehr erreicht. Diese Modifikation wird in folgendem Lemma anhand einer Turingmaschine durchgeführt, kann aber in analoger Weise auf Kellerautomaten und endliche Automaten übertragen werden.

**Lemma 5.1.** Es sei $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ eine Turingmaschine und $Q_1 \subseteq Q$ die Menge der

Zustände, die im ersten Berechnungsschritt von $M$ erreicht werden können. Das heißt,

$$Q_1 = \{q \in Q \mid \exists \gamma_1, \gamma_2 \in \Gamma, S \in \{L, R\} : \delta(q_0, \gamma_1) = (q, \gamma_2, S)\}$$

Dann kann eine äquivalente Turingmaschine $M'$ mit folgenden Eigenschaften konstruiert werden:

- Alle Zustände $q \in Q_1$ werden nur im ersten Berechnungsschritt erreicht.
- Der Startzustand $q_0$ wird nie wieder erreicht.

*Beweis.* Wir definieren eine Turingmaschine $M' = (Q \cup Q' \cup \{\overline{q}_0\}, \Sigma, \Gamma, \delta', \overline{q}_0, B, F \cup F')$, wobei $Q' = \{q' \mid q \in Q\}$ eine Kopie von $Q$ und $\overline{q}_0 \notin Q \cup Q'$ ein neuer Zustand ist. Für $p, q \in Q$, $\gamma_1, \gamma_2 \in \Gamma$ und $S \in \{L, R\}$ definieren wir

$$\begin{array}{lcll} \delta'(\overline{q}_0, \gamma_1) & = & (p, \gamma_2, S) & \text{falls } \delta(q_0, \gamma_1) = (p, \gamma_2, S) \\ \delta'(q, \gamma_1) & = & (p', \gamma_2, S) & \text{falls } \delta(q, \gamma_1) = (p, \gamma_2, S) \\ \delta'(q', \gamma_1) & = & (p', \gamma_2, S) & \text{falls } \delta(q, \gamma_1) = (p, \gamma_2, S) \end{array}$$

Offenbar ist $M'$ eine äquivalente deterministische Turingmaschine mit den verlangten Eigenschaften. □

Nach Satz 3.11 gilt $\mathrm{DCF}_\epsilon \subset \mathcal{L}_{rt}(\mathrm{IA})$. Für spätere Konstrukionen zeigen wir den Abschluß von $\mathrm{DCF}_\epsilon$ unter markierter Konkatenation und markiertem Kleene-Abschluß und als Vorbereitung dazu folgendes Lemma:

**Lemma 5.2.** Es sei $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ ein $\mathrm{DPDA}_\epsilon$. Dann kann ein äquivalenter $\mathrm{DPDA}_\epsilon$ $M'$ konstruiert werden, dessen Keller niemals vollständig geleert wird.

*Beweis.* Nach der Vorbemerkung zu Lemma 5.1 können wir ohne Beschränkung der Allgemeinheit annehmen, daß die Zustände in $M$, die im ersten Berechnungsschritt erreicht werden, in späteren Berechnungsschritten nicht wieder erreicht werden können.

Sei $Z_0'$ ein neues Kellersymbol mit $Z_0' \notin \Gamma$. Für $q \in Q$, $\sigma \in \Sigma$ und $\gamma \in \Gamma^*$ ersetzen wir jede Transition $(q, \gamma) \in \delta(q_0, \sigma, Z_0)$ in $M$ durch die Transition $(q, \gamma Z_0') \in \delta(q_0, \sigma, Z_0)$. Da die Zustände $q \in Q$ aus den ersetzten Transitionen in späteren Berechnungsschritten nicht wieder erreicht werden können, wird das neue Kellersymbol $Z_0'$ daher nur im ersten Berechnungsschritt auf den Keller geschrieben. Da es keine Transition gibt, die $Z_0'$ vom Keller entfernt, wird der Keller also niemals vollständig geleert. □

**Lemma 5.3.** $\mathrm{DCF}_\epsilon$ ist abgeschlossen unter markierter Konkatenation und markiertem Kleene-Abschluß.

*Beweis.* Seien $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_0^1, Z_0^1, F_1)$ und $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_0^2, Z_0^2, F_2)$ zwei $\mathrm{DPDA}_\epsilon$. Wir können ohne Einschränkung annehmen, daß $Q_1 \cap Q_2 = \emptyset$ und $\Gamma_1 \cap \Gamma_2 = \emptyset$ gilt. Wir können weiterhin gemäß Lemma 5.2 annehmen, daß der Keller von $M_1$ niemals vollständig geleert wird. Sei \$ nun das Markierungssymbol mit $\$ \notin \Sigma_1 \cup \Sigma_2$. Wir definieren nun $M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2 \cup \{\$\}, \Gamma_1 \cup \Gamma_2, \delta, q_0^1, Z_0^1, F_2)$, wobei die Transitionen wie folgt definiert sind:

$$\begin{array}{lcll} \delta(q, \sigma, Z) & = & \delta_1(q, \sigma, Z) & \text{für } q \in Q_1, \sigma \in \Sigma_1, Z \in \Gamma_1 \\ \delta(q, \$, Z) & = & \{(q_0^2, Z_0^2 Z)\} & \text{für } q \in F_1, Z \in \Gamma_1 \\ \delta(q, \sigma, Z) & = & \delta_2(q, \sigma, Z) & \text{für } q \in Q_2, \sigma \in \Sigma_2, Z \in \Gamma_2 \end{array}$$

Die Disjunktheit der beiden Zustandsmengen und der beiden Kelleralphabete stellt sicher, daß vor dem Lesen des Markierungssymbols nur Wörter aus $T(M_1)$ akzeptiert werden und danach nur Wörter aus $T(M_2)$ akzeptiert werden. Die Forderung, daß der Keller von $M_1$ niemals vollständig geleert wird, stellt sicher, daß die Berechnung nicht blockiert wird, falls $M_1$ ursprünglich gleichzeitig in einen akzeptierenden Zustand übergeht und den Keller vollständig leert. Daher gilt: $T(M) = T(M_1)\$T(M_2)$

Die Konstruktion für den markierten Kleene-Abschluß ist ähnlich. Gegeben sei ein DPDA$_\epsilon$ $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_0^1, Z_0^1, F_1)$, dessen Keller niemals vollständig geleert wird. Wir können weiterhin gemäß Lemma 5.1 annehmen, daß der Startzustand während einer Berechnung in $M_1$ nicht wieder angenommen wird. Wir definieren nun $M = (Q_1, \Sigma_1 \cup \{\$\}, \Gamma_1, \delta, q_0^1, Z_0^1, \{q_0^1\})$. Die Transitionen sind wie folgt definiert:

$$\begin{aligned} \delta(q, \sigma, Z) &= \delta_1(q, \sigma, Z) \qquad \text{für } q \in Q_1, \sigma \in \Sigma_1, Z \in \Gamma_1 \\ \delta(q, \$, Z) &= \{(q_0^1, Z_0^1 Z)\} \quad \text{für } q \in F_1, Z \in \Gamma_1 \end{aligned}$$

Wiederum stellt die Forderung nach Vermeidung einer vollständigen Kellerentleerung sicher, daß eine Berechnung nicht blockiert wird, falls $M_1$ ursprünglich gleichzeitig in einen akzeptierenden Zustand übergeht und den Keller vollständig leert. Daher ist $T(M) = (T(M_1)\$)^*$. □

Das folgende technische Lemma zeigt, daß $\mathcal{L}_{rt}(\text{OCA})$ unter einer ähnlichen Operation wie markiertem Kleene-Abschluß abgeschlossen ist.

**Lemma 5.4.** Es sei $A = (Q, \#, \Sigma, \delta, F)$ ein Realzeit-OCA und $\$ \notin \Sigma$ ein Markierungssymbol. Dann gilt: $(\{\$\}T(A))^*\{\$\} \in \mathcal{L}_{rt}(\text{OCA})$

*Beweis.* Ein Realzeit-OCA für $(\{\$\}T(A))^*\{\$\}$ kann folgendermaßen skizziert werden:

(1) Auf allen Teileingaben der Form $\{\$\}\Sigma^*\{\$\}$ wird der gegebene Realzeit-OCA $A$ simuliert, wobei jedes Markierungssymbol \$ als Randsymbol # aufgefaßt wird.

(2) Jede Zelle $i$, die \$ als Eingabe hat, wird im ersten Zeitschritt besonders gekennzeichnet. Gleichzeitig wird ein Signal $L$ mit maximaler Geschwindigkeit nach links gestartet.

(3) In jedem Zeitschritt prüft eine \$-Zelle, ob ihre rechte Nachbarzelle einen akzeptierenden Zustand aus $F$ angenommen hat. Ist das der Fall, wird ein Zustand $q_f$ angenommen, der nicht mehr verlassen wird und alle ankommenden $L$-Signale stoppt. Erreicht die rechte Nachbarzelle bis zum Eintreffen des Signals $L$ keinen Zustand aus $F$, dann ist die entsprechende Teileingabe nicht aus $T(A)\{\$\}$. Daher wird ein Fehlerzustand $q_e$ angenommen, der ebenfalls nicht mehr verlassen wird und alle ankommenden $L$-Signale stoppt.

(4) Ist die Eingabe der letzten Zelle \$, dann wird ein Signal $E$ mit maximaler Geschwindigkeit nach links gestartet, das überprüft, ob alle anderen Zellen, deren Eingabe \$ war, in einem Zustand $q_f$ sind. Ist das der Fall, dann sind alle Teileingaben in $\{\$\}T(A)\{\$\}$ und die Gesamteingabe somit aus $(\{\$\}T(A))^*\{\$\}$. Daher wird ein akzeptierender Zustand angenommen. In allen anderen Fällen wird die Eingabe verworfen.

□

## 5.2 Nichtrekursive Tradeoffs

Wie in Kapitel 4.1 zeigen wir zunächst, daß die gültigen und ungültigen Berechnungen einer Turingmaschine von Realzeit-IA akzeptiert werden.

**Satz 5.5.** Sei $M$ eine Turingmaschine. Dann können zwei Realzeit-IA $A_1$ und $A_2$ effektiv konstruiert werden mit $T(A_1) = \text{VALC}[M]$ und $T(A_2) = \text{INVALC}[M]$.

*Beweis.* Nach [HU79] gilt: $\text{VALC}[M] = L_1 \cap L_2$, wobei

$$
\begin{aligned}
L_1 &= L_3^*(\{\epsilon\} \cup \Gamma^* F \Gamma^* \{\#\}) \\
L_2 &= \{q_0\}\Sigma^*\{\#\}L_4^*(\{\epsilon\} \cup \Gamma^* F \Gamma^* \{\#\}) \\
L_3 &= \{y\#z^R\# \mid y \vdash^{\underline{M}} z\} \\
L_4 &= \{y^R\#z\# \mid y \vdash^{\underline{M}} z\}
\end{aligned}
$$

Wir konstruieren zu einer Turingmaschine $M$ zwei $\text{DPDA}_\epsilon$, die jeweils $L_1$ und $L_2$ akzeptieren. Da nach Satz 3.11 zu jedem $\text{DPDA}_\epsilon$ ein äquivalentes Realzeit-IA konstruiert werden kann und die Sprachklasse $\mathcal{L}_{rt}(\text{IA})$ effektiv unter Durchschnitt abgeschlossen ist, können wir also ein Realzeit-IA $A_1$ konstruieren mit $T(A_1) = \text{VALC}[M]$. Weiterhin ist $\mathcal{L}_{rt}(\text{IA})$ effektiv unter Komplementbildung abgeschlossen. Also kann auch ein Realzeit-IA $A_2$ konstruiert werden mit $T(A_2) = \text{INVALC}[M]$.

Wir zeigen zunächst, daß die Sprache $L_3$ von einem $\text{DPDA}_\epsilon$ akzeptiert wird. In [HU79] wird ein Kellerautomat konstruiert, der $L_3$ mit leerem Keller akzeptiert. Wir modifizieren nun diese Konstruktion dahingehend, daß der Kellerautomat deterministisch arbeitet, keine $\epsilon$-Regeln benötigt und mit akzeptierenden Zuständen akzeptiert.

Die Idee ist folgende: Der $\text{DPDA}_\epsilon$ liest die Eingabe $y$ bis zum Sonderzeichen #. Dabei wird überprüft, ob $y$ von der Form $\Gamma^* Q \Gamma^*$ ist. Gleichzeitig wird $z^R$ auf den Keller geschrieben, wobei $z$ die Folgekonfiguration von $y$ in $M$ ist, das heißt $y \vdash^{\underline{M}} z$. Anschließend wird die Eingabe mit dem Kellerinhalt abgeprüft. Tritt ein Fehler auf, dann wird die Eingabe verworfen. Anderenfalls wird die Eingabe akzeptiert.

Dazu sei $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ die zugrundeliegende Turingmaschine. Wir betrachten Zustände $p_0, p_1, p_a, p_m$ mit $\{p_0, p_1, p_a, p_m\} \cap Q = \emptyset$. Weiterhin sei $Z_0$ ein neues Symbol mit $\{Z_0\} \cap (\Gamma \cup Q) = \emptyset$. Wir definieren einen Kellerautomat

$$A = \{Q \cup \{p_0, p_1, p_a, p_m\}, \Gamma \cup Q \cup \{\#\}, \Gamma \cup Q \cup \{Z_0\}, \delta', p_0, Z_0, \{p_a\}\}$$

mit den Überführungsregeln

$$
\begin{aligned}
\delta'(p_0, \gamma, Z) &= \{(p_0, \gamma Z)\} \quad \text{für } \gamma \in \Gamma, Z \in \Gamma \cup \{Z_0\} \\
\delta'(p_0, q, Z) &= \{(q, Z)\} \quad \text{für } q \in Q, Z \in \Gamma \cup \{Z_0\}
\end{aligned}
$$

Diese Regeln stellen sicher, daß $A$ die Eingabe $y$ bis zum ersten Symbol aus $Q$ liest. Gleichzeitig wird die Eingabe auf den Keller geschrieben. Mit den folgenden Regeln wird nun das Verhalten der Turingmaschine von $q$ aus simuliert. Wir merken an, daß die folgenden Regeln identisch zur Konstruktion in Satz 4.1 sind.

Seien $\gamma, Z \in \Gamma$ und $q \in Q$. Dann definieren wir folgende Übergänge:

$$\begin{array}{lcll}
\delta'(q,\gamma,Z) & = & \{(p_1, pYZ)\} & \text{falls } \delta(q,\gamma) = (p,Y,R) \\
\delta'(q,\gamma,Z_0) & = & \{(p_1, pYZ_0)\} & \text{falls } \delta(q,\gamma) = (p,Y,R) \\
\delta'(q,\#,Z) & = & \{(p_m, pYZ)\} & \text{falls } \delta(q,B) = (p,Y,R) \\
\delta'(q,\#,Z_0) & = & \{(p_m, pYZ_0)\} & \text{falls } \delta(q,B) = (p,Y,R) \\
\delta'(q,\gamma,Z) & = & \{(p_1, YZp)\} & \text{falls } \delta(q,\gamma) = (p,Y,L) \\
\delta'(q,\gamma,Z_0) & = & \{(p_1, YBpZ_0)\} & \text{falls } \delta(q,\gamma) = (p,Y,L) \\
\delta'(q,\#,Z) & = & \{(p_m, YZp)\} & \text{falls } \delta(q,B) = (p,Y,L) \\
\delta'(q,\#,Z_0) & = & \{(p_m, YBpZ_0)\} & \text{falls } \delta(q,B) = (p,Y,L)
\end{array}$$

Anschließend dient der Zustand $p_1$ dazu, den Rest der Eingabe bis zum nächsten #-Zeichen auf den Keller zu schreiben. Danach wird in den Zustand $p_m$ gewechselt, der zum Vergleich des Kellerinhalts mit der Eingabe dient. Sobald ein Fehler gefunden ist, wird die Eingabe verworfen, da für diesen Fall kein Übergang definiert ist. Wird die gesamte Resteingabe ohne Fehler mit dem gesamten Keller abgeprüft, dann landet der Kellerautomat schließlich in einem akzeptierenden Zustand $p_a$.

$$\begin{array}{lcll}
\delta'(p_1,\gamma,Z) & = & \{(p_1, \gamma Z)\} & \text{für } \gamma \in \Gamma, Z \in \Gamma \cup Q \cup \{Z_0\} \\
\delta'(p_1,\#,Z) & = & \{(p_m, Z)\} & \text{für } Z \in \Gamma \cup Q \cup \{Z_0\} \\
\delta'(p_m,\gamma,Z) & = & \{(p_m, \epsilon)\} & \text{für } \gamma, Z \in \Gamma \cup Q \text{ mit } \gamma = Z \\
\delta'(p_m,\#,Z_0) & = & \{(p_a, Z_0)\} &
\end{array}$$

Wir können beobachten, daß der Kellerautomat $A$ deterministisch arbeitet und keine $\epsilon$-Regeln benötigt. Also kann $L_3$ von einem DPDA$_\epsilon$ akzeptiert werden.

Die Konstruktion eines DPDA$_\epsilon$ für die Sprache $L_4$ ist analog zur obigen Konstruktion; lediglich die Regeln zur Simulation der Turingmaschine müssen folgendermaßen ersetzt werden: Es seien $\gamma, \gamma' \in \Gamma$ und $q \in Q$.

$$\begin{array}{lcll}
\delta'(q,\gamma,\gamma') & = & \{(p_1, \gamma Y p)\} & \text{falls } \delta(q,\gamma') = (p,Y,R) \\
\delta'(q,\#,\gamma') & = & \{(p_m, Yp)\} & \text{falls } \delta(q,\gamma') = (p,Y,R) \\
\delta'(q,\gamma,Z_0) & = & \{(p_1, \gamma Y p Z_0)\} & \text{falls } \delta(q,B) = (p,Y,R) \\
\delta'(q,\#,Z_0) & = & \{(p_m, pYZ_0)\} & \text{falls } \delta(q,B) = (p,Y,R) \\
\delta'(q,\gamma,\gamma') & = & \{(p_1, p\gamma Y)\} & \text{falls } \delta(q,\gamma') = (p,Y,L) \\
\delta'(q,\#,\gamma') & = & \{(p_m, pBY)\} & \text{falls } \delta(q,\gamma') = (p,Y,L) \\
\delta'(q,\gamma,Z_0) & = & \{(p_1, p\gamma Y Z_0)\} & \text{falls } \delta(q,B) = (p,Y,L) \\
\delta'(q,\#,Z_0) & = & \{(p_m, pBY Z_0)\} & \text{falls } \delta(q,B) = (p,Y,L)
\end{array}$$

Damit sind also die Sprachen $L_3$ und $L_4$ in DCF$_\epsilon$. Dann sind auch die Sprachen $L_3^*$ und $L_4^*$ in DCF$_\epsilon$, da DCF$_\epsilon$ nach Lemma 5.3 effektiv unter markiertem Kleene-Abschluß abgeschlossen ist und jedes zweite #-Zeichen als Markierungssymbol aufgefaßt werden kann. Letzteres kann mit der endlichen Zustandskontrolle eines DPDA$_\epsilon$ gesteuert werden.

Wir wollen weiterhin zeigen, daß die Rechtskonkatenation der regulären Sprache $R = (\{\epsilon\} \cup \Gamma^* F \Gamma^* \{\#\})$ mit den Sprachen $L_3$ und $L_4$ ebenfalls von einem DPDA$_\epsilon$ verwirklicht werden kann. Dazu führen wir in den Zustandsmengen der DPDA$_\epsilon$ für $L_3^*$ und $L_4^*$ jeweils eine zweite

Komponente ein. In der ersten Komponente wird der $\text{DPDA}_\epsilon$ für $L_3^*$ bzw. $L_4^*$ simuliert. Wir beobachten, daß in der ersten Komponente nach jedem zweiten #-Zeichen ein akzeptierender Zustand angenommen wird, sofern die Eingabe in der Sprache ist. Daher wird dann in der zweiten Komponente (ab jedem zweiten #-Zeichen) überprüft, ob die folgende Eingabe bis zum nächsten #-Zeichen die Form $(\{\epsilon\} \cup \Gamma^* F \Gamma^* \{\#\})$ besitzt. Ist das der Fall, wird ein akzeptierender Zustand angenommen. Ist zu diesem Zeitpunkt bereits die gesamte Eingabe gelesen, dann wird die Eingabe somit akzeptiert. Ist die gesamte Eingabe noch nicht gelesen, dann wird in der zweiten Komponente die folgende Eingabe bis zum nächsten #-Zeichen ignoriert. Danach wird wiederum überprüft, ob die restliche Eingabe die Form $(\{\epsilon\} \cup \Gamma^* F \Gamma^* \{\#\})$ besitzt. Offenbar führt die beschriebene Konstruktion keine $\epsilon$-Regeln und keinen Nichtdeterminismus ein. Daher sind die Sprachen $L_3^* R$ und $L_4^* R$ in $\text{DCF}_\epsilon$.

Abschließend beobachten wir, daß auch die Sprache $\{q_0\}\Sigma^*\{\#\}L_4^* R$ von einem $\text{DPDA}_\epsilon$ akzeptiert werden kann, da $\text{DCF}_\epsilon$ nach Lemma 5.3 unter markierter Konkatenation abgeschlossen ist und das erste #-Zeichen als Markierungssymbol aufgefaßt werden kann.

Damit können also zwei $\text{DPDA}_\epsilon$ effektiv konstruiert werden, die jeweils $L_1$ und $L_2$ akzeptieren. Damit ist der Satz bewiesen. □

Aus beweistechnischen Gründen wollen wir die Anzahl der gültigen Berechnungen einer Turingmaschine vergrößern. Dazu erweitern wir eine beliebige Turingmaschine $M$ dahingehend, daß mit einem akzeptierten Wort der Länge $n$ auch alle anderen Eingaben der Länge $n$ akzeptiert werden. Für die modifizierte Turingmaschine $M'$ gilt dann: $T(M)$ ist genau dann endlich, wenn $T(M')$ endlich ist. Ebenfalls gilt für die dazugehörigen gültigen Berechnungen: $\text{VALC}[M]$ ist genau dann endlich, wenn $\text{VALC}[M']$ endlich ist. Zunächst beweisen wir, daß die gewünschte Modifikation möglich ist.

**Lemma 5.6.** Es sei $M$ eine Turingmaschine, die eine Sprache $L \subseteq \Sigma^*$ akzeptiert. Dann kann $M$ in eine deterministische Turingmaschine $M'$ umgewandelt werden, die eine Sprache $L' \subseteq (\Sigma')^*$ akzeptiert, für die folgendes gilt:

(1) $\Sigma \subseteq \Sigma'$ and $|\Sigma'| \geq 2$

(2) Für alle $y \in (\Sigma')^*$ gilt: $y \in L' \Leftrightarrow \exists x \in L$ mit $|x| = |y|$

*Beweis.* Falls $\Sigma$ ein einelementiges Alphabet ist, definieren wir $\Sigma' = \Sigma \cup \{a\}$, wobei $a \notin \Sigma$ gilt. Ist $|\Sigma| \geq 2$, definieren wir $\Sigma' = \Sigma$.

Wir konstruieren eine nichtdeterministische Turingmaschine $M_0$, die folgendermaßen arbeitet:

- $M_0$ liest die Eingabe $y \in (\Sigma')^*$ von links nach rechts. Nach jedem gelesenen Eingabesymbol bestimmt $M_0$ nichtdeterministisch ein Symbol $\sigma \in \Sigma$ und schreibt dieses auf das Band. Nachdem die gesamte Eingabe gelesen ist, steht also eine „geratene“ Eingabe $x \in \Sigma^*$ der Länge $|y|$ auf dem Band.
- Dann kehrt der Kopf an das linke Ende der Eingabe zurück und $M_0$ simuliert eine Berechnung der gegebenen Turingmaschine $M$ auf der Eingabe $x$. Wird $x$ von $M$ akzeptiert, dann akzeptiert $M_0$ die Eingabe $y$. Ansonsten wird die Eingabe verworfen.
- Schließlich wird die nichtdeterministische Turingmaschine $M_0$ gemäß der Konstruktion aus Theorem 7.3 in [HU79] in eine deterministische Turingmaschine $M'$ umgewandelt.

Offenbar besitzt $M'$ die geforderten Eigenschaften. □

Wir betrachten im folgenden Wörter aus $\text{VALC}[M']^R$ und beobachten, daß jedes Wort $x \in \text{VALC}[M']^R$ mit einem Teilwort $\#yq_0$ endet, wobei $\#$ das Trennungssymbol, $y \in \Sigma^*$ die Spiegelung der Eingabe und $q_0$ der Startzustand von $M'$ ist. Wir definieren eine Abbildung $\pi : \text{VALC}[M']^R \to \Sigma^*$, die jedes Wort $x$ aus $\text{VALC}[M']^R$ auf die ursprüngliche Eingabe projiziert, indem wir $\pi(x) = y^R$ setzen.

Wir definieren nun zu jeder Turingmaschine $M$ eine Sprache $\text{L}'[M]$, so daß $\text{L}'[M]$ genau dann von einem Realzeit-IA akzeptiert werden kann, wenn $M$ eine endliche Sprache akzeptiert. In [Kut01a] wird von einer ähnlichen Sprache gezeigt, daß diese nicht in $\mathcal{L}_{rt}(\text{IA})$ ist. Es sei also $M$ eine Turingmaschine, die eine Sprache $L \subseteq \Sigma^*$ akzeptiert. Nach Lemma 5.6 kann $M$ in eine Turingmaschine $M'$ mit den im Lemma 5.6 geforderten Eigenschaften umgewandelt werden. Weiterhin seien \$ und & zwei neue Alphabetsymbole mit $\{\$, \&\} \cap \Sigma' = \emptyset$. Dann definieren wir

$$\text{L}'[M] = \{\$x_k\$ \dots \$x_1 \& y_1\$ \dots \$y_k\$ \mid x_i^R = y_i z_i \text{ mit } y_i, z_i \in (\Sigma')^* \text{ und } x_i \in \text{VALC}[M']^R\}$$

Die notwendigen Voraussetzungen, um später Satz 2.28 anwenden zu können, fassen wir in dem folgenden Lemma zusammen.

**Lemma 5.7.** Es sei $M$ eine Turingmaschine und $\text{L}[M] = \{w^{|w|!} \mid w \in \{\#_0\}\text{VALC}[M]\{\#_1\}\}$. Dann gilt

(1) $\text{INVALC}[M] \in \text{REG} \Leftrightarrow T(M)$ ist endlich

(2) $\text{VALC}[M] \in \text{CF} \Leftrightarrow T(M)$ ist endlich

(3) $\text{L}'[M] \in \mathcal{L}_{rt}(\text{IA}) \Leftrightarrow T(M)$ ist endlich

(4) $\text{L}'[M] \in \mathcal{L}_{rt}(\text{OCA})$

(5) $\text{L}[M] \in \mathcal{L}_{rt}(\text{OCA}) \Leftrightarrow T(M)$ ist endlich

(6) $\text{L}[M] \in \mathcal{L}_{rt}(\text{IA})$

*Beweis.* Die Behauptungen (1), (2) und (5) wurden bereits in Lemma 4.3 bewiesen.

(3) „⇒“: Wir benutzen eine ähnliche Vorgehensweise wie in [Kut01a] und zeigen, daß $\text{L}'[M] \notin \mathcal{L}_{rt}(\text{IA})$ gilt, falls $T(M)$ unendlich ist. Für einen Widerspruchsbeweis nehmen wir an, daß $\text{L}'[M]$ von einem Realzeit-IA $A = (Q, \Sigma, \triangledown, \delta_0, \delta, q_0, F)$ akzeptiert wird. Aufgrund der Definition der Sprache $\text{L}'[M]$ und der Konstruktion der Turingmaschine $M'$ können wir folgendes beobachten: Ist $w \in \text{VALC}[M']^R$ und $|\pi(w)| = k$, dann gibt es $|\Sigma'|^k$ viele Wörter $x \in \text{VALC}[M']^R$ mit paarweise verschiedenen Eingaben der Länge $k$, das heißt $|\pi(x)| = k$. Daher führt jedes Wort $w \in \text{VALC}[M']^R$ mit $|\pi(w)| = k$ zu einer Menge $V(k) \subseteq \text{VALC}[M']^R$ und $|V(k)| = |\Sigma'|^k$.

Zu einem beliebigen $k \in \mathbb{N}$ mit $k > 0$ betrachten wir zwei verschiedene Präfixe $w$ und $w'$ von $\text{L}'[M]$ der Form $\$x_k\$ \dots \$x_1\&$, wobei $x_j \in V(k)$ gilt. Dann gibt es mindestens

ein $1 \leq j \leq k$, so daß $x_j \neq x'_j$ ist. Es bezeichne $q_0$ den Anfangszustand von $M'$. Dann gilt:

$$w\$^{j-1}q_0\pi(x_j)\$^{k-j+1} \in \mathrm{L}'[M] \text{ und } w'\$^{j-1}q_0\pi(x_j)\$^{k-j+1} \notin \mathrm{L}'[M]$$

Offenbar gibt es $\left(|\Sigma'|^k\right)^k \geq 2^{k^2}$ verschiedene Präfixe der obigen Form. Nach Bemerkung 3.8 kann $A$ höchstens $|Q|^{2(2k+2)+1} = |Q|^{4k+5}$ verschiedene $(2k+1)$-Fenster unterscheiden.

Da $T(M)$ unendlich ist, können wir ein $k \in \mathbb{N}$ und eine Menge $V(k)$ finden, so daß $2^{k^2} > |Q|^{4k+5}$ ist. Daher nimmt $A$ ein identisches $(2k+1)$-Fenster beim Verarbeiten zweier verschiedener Präfixe $w$ und $w'$ an. Damit gilt:

$$w\$^{j-1}q_0\pi(x_j)\$^{k-j+1} \in \mathrm{L}'[M] \Leftrightarrow w'\$^{j-1}q_0\pi(x_j)\$^{k-j+1} \in \mathrm{L}'[M]$$

Das ist ein Widerspruch und die Behauptung ist bewiesen.

„$\Leftarrow$“: Ist $T(M)$ endlich, dann sind auch die Mengen $\mathrm{VALC}[M]^R$ und $\mathrm{VALC}[M']^R$ endlich. Wir betrachten nun eine Menge $\Phi$ bestehend aus Symbolen, so daß $|\Phi| = |\mathrm{VALC}[M']^R|$ gilt. Dann können wir jedes $w \in \mathrm{VALC}[M']^R$ eindeutig mit einem Symbol $\phi_w \in \Phi$ identifizieren.

Wir skizzieren jetzt die Konstruktion eines DPDA $A$, der die Sprache $\mathrm{L}'[M]$ akzeptiert.

- Zunächst liest $A$ die Eingabe bis zum Symbol &. In der endlichen Zustandskontrolle von $A$ wird das korrekte Format der Eingabe überprüft. Außerdem werden nacheinander die Teileingaben $w \in \{\$\}\mathrm{VALC}[M']^R$ identifiziert und die entsprechende Symbole $\phi_w \in \Phi$ auf den Keller geschrieben. Dies kann durch die endliche Zustandskontrolle realisiert werden, da $\mathrm{VALC}[M']^R$ endlich ist.
- Sobald das Trennungssymbol & gelesen ist, entfernt $A$ das oberste Kellersymbol $\phi_w$ vom Keller und überprüft in der endlichen Zustandskontrolle, ob die folgende Eingabe bis zum nächsten \$-Zeichen ein Präfix von $w^R$ ist. Dieses Verhalten wird wiederholt, bis alle Symbole aus $\Phi$ vom Keller entfernt sind, und die entsprechenden Eingaben korrekt gelesen wurden.
- Nachdem das letzte Symbol aus $\Phi$ entfernt und die entsprechende Eingabe korrekt überprüft wurde, wird ein akzeptierender Zustand angenommen.

Wir können beobachten, daß $A$ gerade $\mathrm{L}'[M]$ akzeptiert. Offenbar arbeitet $A$ deterministisch und benötigt keine $\epsilon$-Regeln. Nach Satz 3.11 gilt $\mathrm{DCF}_\epsilon \subset \mathcal{L}_{rt}(\mathrm{IA})$. Daher ist $\mathrm{L}'[M] \in \mathcal{L}_{rt}(\mathrm{IA})$.

(4) Wir zeigen, daß sich $\mathrm{L}'[M]$ durch den Schnitt zweier Sprachen $L_1, L_2 \in \mathcal{L}_{rt}(\mathrm{OCA})$ darstellen läßt. Wir definieren

$$L_1 = \{\$x_k\$ \ldots \$x_1\&y_1\$ \ldots \$y_k\$ \mid x_i^R = y_i z_i \text{ mit } y_i, z_i \in (\Sigma')^*\}$$
$$L_2 = \{w \mid w \in (\{\$\}\mathrm{VALC}[M']^R)^*\{\&\}((\Sigma')^*\{\$\})^*\}$$

Es wird in [Kut01a] beobachtet, daß $L_1$ von einer linear kontextfreien Grammatik erzeugt werden kann. Nach Satz 3.10 ist dann $L_1 \in \mathcal{L}_{rt}(\mathrm{OCA})$. In Satz 4.1 wurde gezeigt, daß $\mathrm{VALC}[M'] \in \mathcal{L}_{rt}(\mathrm{OCA})$ gilt. Da $\mathcal{L}_{rt}(\mathrm{OCA})$ unter Spiegelung abgeschlossen

ist, folgt $\text{VALC}[M']^R \in \mathcal{L}_{rt}(\text{OCA})$. Weiterhin beobachten wird, daß nach Lemma 5.4 $(\{\$\}\text{VALC}[M']^R)^*\{\&\} \in \mathcal{L}_{rt}(\text{OCA})$ gilt. Außerdem ist $\mathcal{L}_{rt}(\text{OCA})$ unter Konkatenation mit REG abgeschlossen. Daraus folgt $L_2 \in \mathcal{L}_{rt}(\text{OCA})$. Offenbar gilt $\text{L}'[M] = L_1 \cap L_2$. Da $\mathcal{L}_{rt}(\text{OCA})$ unter Schnitt abgeschlossen ist, erhalten wir also $\text{L}'[M] \in \mathcal{L}_{rt}(\text{OCA})$ und damit (4).

(6) Wir beschreiben nun, wie ein Realzeit-IA für die Sprache $\text{L}[M]$ konstruiert werden kann. Zunächst seien $\#_0, \#_1$ neue Alphabetsymbole mit $\{\#_0, \#_1\} \cap \Sigma = \emptyset$. Wir betrachten die drei folgenden Sprachen

$$L_1 = \{w \mid w \in (\{\#_0\}\text{VALC}[M]\{\#_1\})^*\}$$
$$L_2 = \{w^n \mid w \in \{\#_0\}\Sigma^*\{\#_1\}, n \geq 2 \text{ und } n \text{ ist gerade}\}$$
$$L_3 = \{wx \mid w \in \{\#_0\}\Sigma^*\{\#_1\}, x \in (\{\#_0\}\Sigma^*\{\#_1\})^* \text{ und } |wx|_{\#_0} = |w|!\}$$

Offenbar läßt sich $\text{L}[M]$ durch den Schnitt der drei Sprachen darstellen. Die Sprache $L_1$ stellt sicher, daß alle Teilwörter aus $\{\#_0\}\text{VALC}[M]\{\#_1\}$ sind. Die Sprache $L_2$ stellt sicher, daß $\text{L}[M]$ nur zyklische Wörter enthält und in $L_3$ wird die richtige Anzahl der Wiederholungen gezählt. Da $\mathcal{L}_{rt}(\text{IA})$ unter Schnittbildung abgeschlossen ist, müssen wir nur noch zeigen, daß $L_1$, $L_2$ und $L_3$ jeweils von einem Realzeit-IA akzeptiert werden können.

Nach Satz 5.5 ist $\text{VALC}[M] \in \mathcal{L}_{rt}(\text{IA})$ und damit gilt auch $\{\#_0\}\text{VALC}[M] \in \mathcal{L}_{rt}(\text{IA})$. Nach Lemma 3.16 ist $\mathcal{L}_{rt}(\text{IA})$ unter markiertem Kleene-Abschluß abgeschlossen; daher gilt $L_1 \in \mathcal{L}_{rt}(\text{IA})$, wobei $\#_1$ als Markierungssymbol dient.

In [Col69] wird gezeigt, daß die Sprache $\{ww \mid w \in \Sigma^*\}$ von einem Realzeit-IA akzeptiert werden kann. Durch eine kleine Modifikation der Konstruktion erhalten wir daher

$$L = \{ww \mid w \in \{\#_0\}\Sigma^*\{\#_1\}\} \in \mathcal{L}_{rt}(\text{IA})$$

Wir können weiterhin beobachten, daß $L^* \in \mathcal{L}_{rt}(\text{IA})$ gilt, da $L^*$ in gewisser Weise als markierter Kleene-Abschluß aufgefaßt werden kann, indem wir jedes zweite $\#_1$-Zeichen als Markierungssymbol interpretieren. $\mathcal{L}_{rt}(\text{IA})$ ist unter markierter Konkatenation abgeschlossen; daher gilt $\{\#_0\}\Sigma^*\{\#_1\}L^* \in \mathcal{L}_{rt}(\text{IA})$. Schließlich ist $\mathcal{L}_{rt}(\text{IA})$ unter Rechtskonkatenation mit regulären Sprachen abgeschlossen. Also gilt

$$\{\#_0\}\Sigma^*\{\#_1\}L^*\{\#_0\}\Sigma^*\{\#_1\} \in \mathcal{L}_{rt}(\text{IA})$$

Wir können nun $L_2$ folgendermaßen darstellen:

$$L_2 = L^* \cap \{\#_0\}\Sigma^*\{\#_1\}L^*\{\#_0\}\Sigma^*\{\#_1\}$$

Da $\mathcal{L}_{rt}(\text{IA})$ unter Schnittbildung abgeschlossen ist, folgt $L_2 \in \mathcal{L}_{rt}(\text{IA})$.

Wir müssen nun noch zeigen, daß auch $L_3$ von einem Realzeit-IA akzeptiert werden kann. Dazu konstruieren wir ein iteratives Array, in dem jede Zelle in vier Teilzellen (Spuren) unterteilt ist.

- In der ersten Spur überprüfen wir, ob die Eingabe korrekt formatiert, also ein Wort aus $(\{\#_0\}\Sigma^*\{\#_1\})^*$ ist.

- In der zweiten Spur berechnen wir die Fakultätsfunktion gemäß der Konstruktion von Mazoyer und Terrier aus [MT99]. Diese Konstruktion stellt sicher, daß die Kommunikationszelle zu jedem Zeitpunkt, an dem eine Fakultät errechnet wird, also zu den Zeitpunkten $t = 1, 2, 6, 24, \ldots$, einen bestimmten Zustand annimmt.
- In der dritten und vierten Spur installieren wir binäre Zähler, die mit `Zähler1` und `Zähler2` bezeichnet werden. Ähnlich wie in Beispiel 3.7 befindet sich das niedrigstwertige Bit in der Kommunikationszelle. Die Anzahl der Zellen, die für den binären Zähler benötigt werden, hängt von der Größe der zu speichernden Zahl ab. Wie in Beispiel 3.7 kann auch hier die erste Zelle eines Zählers aufgrund des beidseitigen Informationsflusses besonders markiert werden.

  Nun wird die Eingabe gelesen. In jedem Zeitschritt bis zum ersten $\#_1$-Zeichen in der Eingabe wird `Zähler1` um eins erhöht. Sobald in der zweiten Spur eine Fakultät errechnet wird, wird `Zähler1` um eins verringert. Wir können beobachten, daß `Zähler1` genau zum Zeitpunkt $|w|!$ auf Null heruntergezählt wird. Bis zu diesem Zeitpunkt wird `Zähler2` in der vierten Spur in jedem Zeitschritt um eins erhöht. Wird ein $\#_0$-Zeichen gelesen, dann wird `Zähler2` um eins verringert. Beide Zählerstände werden nicht verändert, wenn sie zum gleichen Zeitpunkt um eins erhöht und verringert werden sollen. Ist `Zähler2` schließlich bis auf Null heruntergezählt, dann wurden bis dahin genau $|w|!$ viele $\#_0$-Zeichen gelesen. Ist die verbleibende Eingabe aus $\Sigma^*\{\#_1\}$, dann wird die Eingabe akzeptiert und anderenfalls verworfen.

  Eine Beispielberechnung in der dritten und vierten Spur für die Eingabe $u = (\#_0 a \#_1)^6$ ist in Abbildung 5.1 dargestellt. Die erste Zelle des binären Zählers ist grau hinterlegt. In der dritten Spur wird zum Zeitpunkt $t = 1$ und $t = 2$ der Zählerstand nicht verändert, da zu den Zeitpunkten jeweils eine Fakultät in der zweiten Spur berechnet wird. In der vierten Spur wird zum Zeitpunkt $t = 1$ und $t = 4$ der Zählerstand nicht verändert, da zu den Zeitenpunkten jeweils ein $\#_0$-Zeichen gelesen wird.

Wir können beobachten, daß die Berechnungen in den einzelnen Spuren in Realzeit ausgeführt werden können. Also können wir ein Realzeit-IA für $L_3$ konstruieren. Damit ist $\mathrm{L}[M] \in \mathcal{L}_{rt}(\mathrm{IA})$ und (6) bewiesen.

□

Nun sind alle Vorbereitungen für Satz 2.28 erfüllt und wir erhalten nichtrekursive Tradeoffs zwischen Realzeit-IA und sequentiellen Modellen, zwischen Realzeit-CA und Realzeit-IA, zwischen Linearzeit-IA und Realzeit-IA und zwischen Realzeit-OCA und Realzeit-IA und umgekehrt zwischen Realzeit-IA und Realzeit-OCA.

**Satz 5.8.**

(1) Realzeit-IA $\overset{nonrec}{\longrightarrow}$ DFA

(2) Realzeit-IA $\overset{nonrec}{\longrightarrow}$ $\mathrm{DPDA}_\epsilon$, Realzeit-IA $\overset{nonrec}{\longrightarrow}$ PDA

(3) Linearzeit-IA $\overset{nonrec}{\longrightarrow}$ Realzeit-IA

| | $-3$ | $-2$ | $-1$ | $0$ | | $-3$ | $-2$ | $-1$ | $0$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| $t=0$ | 0 | 0 | 0 | 0 | $(\#_0a\#_1)^6$ | 0 | 0 | 0 | 0 | $(\#_0a\#_1)^6$ |
| $t=1$ | 0 | 0 | 0 | 0 | $a\#_1(\#_0a\#_1)^5$ | 0 | 0 | 0 | 0 | $a\#_1(\#_0a\#_1)^5$ |
| $t=2$ | 0 | 0 | 0 | 0 | $\#_1(\#_0a\#_1)^5$ | 0 | 0 | 0 | 1 | $\#_1(\#_0a\#_1)^5$ |
| $t=3$ | 0 | 0 | 0 | 1 | $(\#_0a\#_1)^5$ | 0 | 0 | 0 | $0^+$ | $(\#_0a\#_1)^5$ |
| $t=4$ | 0 | 0 | 0 | 1 | $a\#_1(\#_0a\#_1)^4$ | 0 | 0 | 1 | 0 | $a\#_1(\#_0a\#_1)^4$ |
| $t=5$ | 0 | 0 | 0 | 1 | $\#_1(\#_0a\#_1)^4$ | 0 | 0 | 1 | 1 | $\#_1(\#_0a\#_1)^4$ |
| $t=6$ | 0 | 0 | 0 | 0 | $(\#_0a\#_1)^4$ | 0 | 0 | 1 | $0^+$ | $(\#_0a\#_1)^4$ |
| $t=7$ | 0 | 0 | 0 | 0 | $a\#_1(\#_0a\#_1)^3$ | 0 | 0 | $0^+$ | $1^-$ | $a\#_1(\#_0a\#_1)^3$ |
| $t=8$ | 0 | 0 | 0 | 0 | $\#_1(\#_0a\#_1)^3$ | 0 | 1 | $1^-$ | 1 | $\#_1(\#_0a\#_1)^3$ |
| $t=9$ | 0 | 0 | 0 | 0 | $(\#_0a\#_1)^3$ | 0 | 0 | 1 | 1 | $(\#_0a\#_1)^3$ |
| $t=10$ | 0 | 0 | 0 | 0 | $a\#_1(\#_0a\#_1)^2$ | 0 | 0 | 1 | 0 | $a\#_1(\#_0a\#_1)^2$ |
| $t=11$ | 0 | 0 | 0 | 0 | $\#_1(\#_0a\#_1)^2$ | 0 | 0 | 1 | 0 | $\#_1(\#_0a\#_1)^2$ |
| $t=12$ | 0 | 0 | 0 | 0 | $\#_0a\#_1\#_0a\#_1$ | 0 | 0 | 1 | 0 | $\#_0a\#_1\#_0a\#_1$ |
| $t=13$ | 0 | 0 | 0 | 0 | $a\#_1\#_0a\#_1$ | 0 | 0 | 1 | $1^-$ | $a\#_1\#_0a\#_1$ |
| $t=14$ | 0 | 0 | 0 | 0 | $\#_1\#_0a\#_1$ | 0 | 0 | 0 | 1 | $\#_1\#_0a\#_1$ |
| $t=15$ | 0 | 0 | 0 | 0 | $\#_0a\#_1$ | 0 | 0 | 0 | 1 | $\#_0a\#_1$ |
| $t=16$ | 0 | 0 | 0 | 0 | $a\#_1$ | 0 | 0 | 0 | 0 | $a\#_1$ |
| $t=17$ | 0 | 0 | 0 | 0 | $\#_1$ | 0 | 0 | 0 | 0 | $\#_1$ |
| $t=18$ | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | |

Abbildung 5.1: Beispielberechnung in der dritten und vierten Spur für die Eingabe $u = (\#_0a\#_1)^6$

(4) Realzeit-CA $\overset{nonrec}{\longrightarrow}$ Realzeit-IA

(5) Realzeit-OCA $\overset{nonrec}{\longrightarrow}$ Realzeit-IA

(6) Realzeit-IA $\overset{nonrec}{\longrightarrow}$ Realzeit-OCA

*Beweis.* Wir wenden Theorem 2.28 und Lemma 5.7 an. (1) und (2) folgen mit $L_M = \mathrm{INVALC}[M]$ und $L_M = \mathrm{VALC}[M]$. Für (3), (4) und (5) setzen wir $L_M = \mathrm{L}'[M]$ und beobachten, daß folgendes gilt: $L_M \in \mathcal{L}_{rt}(\mathrm{OCA}) \subset \mathcal{L}_{lt}(\mathrm{CA}) = \mathcal{L}_{lt}(\mathrm{IA})$ [Sei79]. Schließlich kann (6) mit der Setzung $L_M = \mathrm{L}[M]$ gezeigt werden. □

## 5.3 Entscheidbarkeitsfragen

In Kapitel 4.2 wurde gezeigt, daß viele Entscheidbarkeitsfragen für Realzeit-OCA unentscheidbar, nicht semientscheidbar und auch nicht co-semientscheidbar sind. Wesentlich für

die Ergebnisse ist die Tatsache, daß die Menge VALC[$M$] der gültigen Berechnungen einer Turingmaschine von einem Realzeit-OCA akzeptiert werden kann. Damit können viele Entscheidbarkeitsfragen für Realzeit-OCA auf Entscheidbarkeitsfragen für Turingmaschinen reduziert werden. Aufgrund der Sätze von Rice sind aber sehr viele Entscheidbarkeitsfragen für Turingmaschinen unentscheidbar und nicht semientscheidbar. In Satz 5.5 wurde bewiesen, daß VALC[$M$] auch von einem Realzeit-IA akzeptiert werden kann. Daher können wir viele Entscheidbarkeitsaussagen über Realzeit-IA nahezu genauso wie die entsprechenden Entscheidbarkeitsaussagen über Realzeit-OCA beweisen. Im folgenden werden wir die Entscheidbarkeitsaussagen zusammenstellen und nur dann einen Beweis erbringen, wenn die Vorgehensweise unterschiedlich zu der in Kapitel 4.2 ist.

Es sei darauf hingewiesen, daß einige Unentscheidbarkeitsaussagen zuerst von Seidel in [Sei79] durch Reduktionen vom Postschen Korrespondenzproblem gezeigt wurden. Das in Kapitel 4.2 und hier vorgestellte Vorgehen vereinfacht die Beweise und macht außerdem Aussagen über die Nichtsemientscheidbarkeit.

**Satz 5.9.** Für Realzeit-IA $A$ und $A'$ sind folgende Fragen nicht semientscheidbar, aber co-semientscheidbar:

(1) Ist $T(A) = \emptyset$?

(2) Ist $T(A) = \Sigma^*$?

(3) Ist $T(A) = T(A')$?

(4) Ist $T(A) \subseteq T(A')$?

Für einen Realzeit-IA $A$ sind folgende Fragen weder semientscheidbar noch co-semientscheidbar:

(5) Ist $T(A)$ endlich?

(6) Ist $T(A)$ unendlich?

(7) Ist $T(A) \in \mathrm{REG}$?

(8) Ist $T(A) \in \mathrm{CF}$?

Für einen Realzeit-CA oder Realzeit-OCA $A$ oder einen Realzeit-IA $A'$ sind folgende Fragen weder semientscheidbar noch co-semientscheidbar:

(9) Ist $T(A) \in \mathcal{L}_{rt}(\mathrm{IA})$?

(10) Ist $T(A') \in \mathcal{L}_{rt}(\mathrm{OCA})$?

*Beweis.* Der Beweis von (1)-(8) ist analog zu den Beweisen von Satz 4.14 und Satz 4.19. Der Beweis von (9) und (10) ist ähnlich zum Beweis von Satz 4.17, wenn man die Aussagen (3) und (4) und im anderen Fall (5) und (6) aus Lemma 5.7 benutzt. □

**Korollar 5.10.** Die Fragen (1) bis (9) sind nicht semientscheidbar sowohl für Automaten $A$ und $A'$ aus einer Automatenklasse, die **Realzeit-IA** enthält, als auch für Sprachen aus einer Sprachklasse, in der $\mathcal{L}_{rt}(\mathrm{IA})$ enthalten ist. Die Fragen (1) bis (4) sind co-semientscheidbar für Automatenklassen, deren entsprechende Sprachklasse in REC enthalten ist. Die Fragen (5) bis (9) sind nicht co-semientscheidbar für Automatenklassen, in denen **Realzeit-IA** enthalten ist. Die Frage (10) ist weder semientscheidbar noch co-semientscheidbar sowohl für Automaten $A$ und $A'$ aus einer Automatenklasse, die **Realzeit-OCA** enthält, als auch für Sprachen aus einer Sprachklasse, in der $\mathcal{L}_{rt}(\mathrm{OCA})$ enthalten ist.

Ähnlich zu Satz 4.2 lassen sich die rekursiv aufzählbaren Sprachen durch das homomorphe Bild von $\mathcal{L}_{rt}(\mathrm{IA})$ charakterisieren. Als Folgerung daraus kann man Unentscheidbarkeitsaussagen über das homomorphe Bild von Realzeit-IA-Sprachen beweisen.

**Satz 5.11.** Eine Sprache $L$ ist genau dann rekursiv aufzählbar, wenn es einen Homomorphismus $h$ und eine Sprache $L' \in \mathcal{L}_{rt}(\mathrm{IA})$ gibt mit $L = h(L')$.

*Beweis.* Der Beweis von Satz 4.2 nutzt aus, daß $\mathrm{VALC}[M]$ von einem Realzeit-OCA akzeptiert wird und die Sprachklasse $\mathcal{L}_{rt}(\mathrm{OCA})$ unter inversem Homomorphismus und Schnitt mit regulären Mengen abgeschlossen ist. Da aber $\mathcal{L}_{rt}(\mathrm{IA})$ ebenfalls unter den genannten Operationen abgeschlossen ist und $\mathrm{VALC}[M] \in \mathcal{L}_{rt}(\mathrm{IA})$ ist, kann die obige Charakterisierung analog bewiesen werden. □

**Satz 5.12.** Es seien $A$ ein Realzeit-IA, $A'$ ein Realzeit-OCA, $h$ ein Homomorphismus und $h_\epsilon$ ein $\epsilon$-freier Homomorphismus. Dann sind folgende Fragen weder semientscheidbar noch co-semientscheidbar.

(1) Ist $h(T(A)) \in \mathrm{REG}$?

(2) Ist $h(T(A)) \in \mathrm{CF}$?

(3) Ist $h(T(A)) \in \mathcal{L}_{rt}(\mathrm{IA})$?

(4) Ist $h(T(A)) \in \mathcal{L}_{rt}(\mathrm{OCA})$?

(5) Ist $h(T(A')) \in \mathcal{L}_{rt}(\mathrm{IA})$?

(6) Ist $h_\epsilon(T(A)) \in \mathrm{REG}$?

(7) Ist $h_\epsilon(T(A)) \in \mathrm{CF}$?

(8) Ist $h_\epsilon(T(A)) \in \mathcal{L}_{rt}(\mathrm{IA})$?

(9) Ist $h_\epsilon(T(A)) \in \mathcal{L}_{rt}(\mathrm{OCA})$?

(10) Ist $h_\epsilon(T(A')) \in \mathcal{L}_{rt}(\mathrm{IA})$?

*Beweis.* Der Beweis von (1) bis (5) ist analog zum Beweis von Satz 4.21 unter Verwendung von Satz 5.11 und Korollar 5.10.

Zum Beweis von (6) bis (10) beachten wir, daß nach Satz 3.19 der Abschluß von $\mathcal{L}_{rt}(\mathrm{IA})$ unter $\epsilon$-freiem Homomorphismus durch Realzeit-NIA charakterisiert ist. Sind die Fragen (6) bis (10)

für Realzeit-IA oder Realzeit-OCA semientscheidbar, dann sind sie auch für Realzeit-NIA semientscheidbar. Da $\mathcal{L}_{rt}(\text{IA}) \subset \mathcal{L}_{rt}(\text{NIA})$ und $\mathcal{L}_{rt}(\text{OCA}) \subset \mathcal{L}_{rt}(\text{NIA})$ [Kut01b], bekommen wir einen Widerspruch zu Korollar 5.10. □

Die Entscheidbarkeitsfragen für Realzeit-OCA, Realzeit-CA und Realzeit-IA sind in folgender Tabelle zusammengefaßt.

| Eigenschaft | Realzeit-OCA | | Realzeit-CA | | Realzeit-IA | |
|---|---|---|---|---|---|---|
| | semient. | co-semi. | semient. | co-semi. | semient. | co-semi. |
| Leere | nein | ja | nein | ja | nein | ja |
| Universalität | nein | ja | nein | ja | nein | ja |
| Äquivalenz | nein | ja | nein | ja | nein | ja |
| Inklusion | nein | ja | nein | ja | nein | ja |
| Endlichkeit | nein | nein | nein | nein | nein | nein |
| Unendlichkeit | nein | nein | nein | nein | nein | nein |
| Regularität | nein | nein | nein | nein | nein | nein |
| Kontextfreiheit | nein | nein | nein | nein | nein | nein |
| $T(A) \in \mathcal{L}_{rt}(\text{OCA})$ | trivial | trivial | nein | nein | nein | nein |
| $T(A) \in \mathcal{L}_{rt}(\text{IA})$ | nein | nein | nein | nein | trivial | trivial |
| $h(T(A)) \in \text{REG}$ | nein | nein | nein | nein | nein | nein |
| $h(T(A)) \in \text{CF}$ | nein | nein | nein | nein | nein | nein |
| $h(T(A)) \in \mathcal{L}_{rt}(\text{OCA})$ | nein | nein | nein | nein | nein | nein |
| $h(T(A)) \in \mathcal{L}_{rt}(\text{IA})$ | nein | nein | nein | nein | nein | nein |
| $h_\epsilon(T(A)) \in \text{REG}$ | nein | nein | nein | nein | nein | nein |
| $h_\epsilon(T(A)) \in \text{CF}$ | nein | nein | nein | nein | nein | nein |
| $h_\epsilon(T(A)) \in \mathcal{L}_{rt}(\text{OCA})$ | nein | nein | nein | nein | nein | nein |
| $h_\epsilon(T(A)) \in \mathcal{L}_{rt}(\text{IA})$ | nein | nein | nein | nein | nein | nein |

Tabelle 5.1: Zusammenfassung der Entscheidbarkeitsfragen

Da $\text{VALC}[M] \in \mathcal{L}_{rt}(\text{IA})$ gilt, folgt aus Korollar 4.24 unmittelbar folgender Satz:

**Satz 5.13.** $\mathcal{L}_{rt}(\text{IA})$ und jede Sprachklasse, die $\mathcal{L}_{rt}(\text{IA})$ enthält, besitzt kein Pumpinglemma.

Im folgenden bezeichnen wir mit $\mathcal{A}$ eine Automatenklasse iterativer Arrays mit Zeitkomplexität $t$. Weiterhin gelte **Realzeit-IA** $\subseteq \mathcal{A}$.

**Satz 5.14.** Für Automaten aus $\mathcal{A}$ gibt es keinen Minimierungsalgorithmus, der einen beliebigen Automaten $A \in \mathcal{A}$ in einen Automaten $A' \in \mathcal{A}$ umwandelt, der $T(A)$ akzeptiert und eine minimale Anzahl von Zuständen besitzt.

*Beweis.* Der Beweis ist analog zum Beweis von Satz 4.25, da auch für $\mathcal{A}$ Leere unentscheidbar ist und ein minimaler Automat $A = (Q, \Sigma, \triangledown, \delta_0, \delta, q_0, F)$, der $L = \emptyset$ mit Zeitkomplexität $t$ akzeptiert, genau einen Zustand besitzt, der nichtakzeptierend ist. □

Analog zu Zellularautomaten betrachten wir nun folgende Erreichbarkeitsfrage für iterative Arrays:

Gegeben sei ein iteratives Array $A = (Q, \Sigma, \triangledown, \delta_0, \delta, q_0, F)$ aus der Automatenklasse $\mathcal{A}$ mit Zeitkomplexität $t$ und ein Zustand $q \in Q$.

(E1) Gibt es eine Eingabe $x \in \Sigma^+$ und einen Zeitpunkt $l \leq t(|x|)$, so daß eine Zelle $i$ den Zustand $q$ annimmt?

**Satz 5.15.** Die Frage (E1) ist nicht entscheidbar.

*Beweis.* Die Nichtentscheidbarkeit der Frage (E1) ist analog zu Satz 4.26 zu beweisen, da auch für iterative Arrays Leere unentscheidbar ist. □

**Bemerkung 5.16.** Alle in diesem Kapitel bewiesenen Unentscheidbarkeits- und Nichtsemientscheidbarkeitsaussagen gelten für alle hier eingeführten Automatenklassen iterativer Arrays. Also für **Realzeit-IA**, **Linearzeit-IA**, **IA** und **Realzeit-NIA**. Für alle diese Automatenklassen gibt es außerdem kein Pumpinglemma und keinen Minimierungsalgorithmus. Also gilt auch für iterative Arrays, daß fast keine Fragen entscheidbar oder semientscheidbar sind.

## 5.4 Unvergleichbarkeit

Nach Satz 3.13 ist $\mathcal{L}_{rt}(\text{IA})$ mit den Sprachklassen LCF und DCF unvergleichbar. Weiterhin läßt sich nach Satz 5.11 die Menge der rekursiv aufzählbaren Sprachen durch das homomorphe Bild von $\mathcal{L}_{rt}(\text{IA})$ charakterisieren. Daher läßt sich ähnlich zu Satz 4.29 folgendes Unvergleichbarkeitskriterium herleiten:

**Satz 5.17.** $\mathcal{L}_{rt}(\text{IA})$ ist unvergleichbar mit jeder Sprachklasse $\mathcal{L}$, für die gilt:

- $\text{LCF} \subseteq \mathcal{L} \subseteq \mathcal{L}' \subset \text{RE}$ und $\mathcal{L}'$ ist abgeschlossen unter Homomorphismus.
- $\text{DCF} \subseteq \mathcal{L} \subseteq \mathcal{L}' \subset \text{RE}$ und $\mathcal{L}'$ ist abgeschlossen unter Homomorphismus.

Damit ist $\mathcal{L}_{rt}(\text{IA})$ insbesondere mit allen Sprachklassen unvergleichbar, die in Kapitel 4.4 genannt werden. Außerdem kann obiges Kriterium nützlich sein, um die Unvergleichbarkeit mit Formalismen, die LCF oder DCF enthalten, zu beweisen.

## 5.5 Zusammenfassung

Wir haben in diesem Kapitel die für Zellularautomaten bekannten Ergebnisse bezüglich ihrer Beschreibungskomplexität und Entscheidbarkeitsfragen auf iterative Arrays übertragen. Wir haben auch für iterative Arrays das Ergebnis, daß im Vergleich mit Satz 3.9 nichtrekursive Tradeoffs zwischen allen Beschreibungsmechanismen existieren, deren Sprachklassen entweder echt ineinander enthalten oder unvergleichbar sind. Wie in Kapitel 4 dargelegt wurde, können wir derzeit keine Aussagen zu Tradeoffs zwischen Modellen machen, wenn deren Sprachklassen bezüglich der generativen Mächtigkeit nicht voneinander getrennt sind. Zum Beispiel ist bislang ungeklärt, ob $\mathcal{L}_{lt}(\text{IA})$ echt in $\mathcal{L}(\text{IA})$ enthalten ist. Daher ist auch unbekannt, ob es zwischen IA und Linearzeit-IA einen nichtrekursiven Tradeoff gibt oder nicht.

Wir haben ebenfalls kaum entscheidbare oder semientscheidbare Fragen für iterative Arrays und auch keine Werkzeuge wie ein Pumpinglemma oder einen Minimierungsalgorithmus. Also ist die Abschwächung von paralleler Eingabe auf sequentielle Eingabe hinsichtlich eines handlicheren Modells wenig erfolgreich. Zwar ist $\mathcal{L}_{rt}(\mathrm{IA})$ echt in $\mathcal{L}_{rt}(\mathrm{CA})$ enthalten und es existiert ein nichtrekursiver Tradeoff zwischen Realzeit-CA und Realzeit-IA. Aber Realzeit-IA sind immer noch mächtig genug, um die gültigen und ungültigen Berechnungen einer Turingmaschine zu akzeptieren. Dies führt zu nichtrekursiven Tradeoffs im Vergleich mit sequentiellen Modellen und zu den bewiesenen negativen Entscheidbarkeitsergebnissen. Wir werden daher im folgenden Kapitel eine weitere Beschränkung einführen, um ein handlicheres Modell zu erhalten.

# Kapitel 6

# Zellularautomaten mit beschränkter Zellenzahl

Wir haben in Kapitel 4 und Kapitel 5 zwei parallele Berechnungsmodelle mit paralleler und sequentieller Eingabe untersucht. Beiden Modellen ist gemeinsam, daß die Anzahl der für die Berechnung verfügbaren Zellen abhängig von der Länge der Eingabe ist und daher beliebig groß werden kann. Dies führt zu einem Anstieg der generativen Mächtigkeit, zur Existenz von nichtrekursiven Tradeoffs und zur Unentscheidbarkeit vieler Entscheidbarkeitsfragen. Bei Zellularautomaten wird gefordert, daß so viele Zellen bereitgestellt werden, wie die Eingabe lang ist. Diese Forderung nach unbegrenzten Ressourcen ist sicherlich nicht sehr realistisch aus praktischer Sicht. Außerdem führt die Forderung zu den bereits dargelegten Nachteilen bezüglich der Handlichkeit des Modells. Wir werden daher in diesem Kapitel Zellularautomaten dahingehend beschränken, daß nur noch eine konstante Zellenzahl zur Verfügung steht. Diese Beschränkung hat große Auswirkungen auf die generative Mächtigkeit des Modells, denn es werden dann lediglich die regulären Sprachen akzeptiert. Allerdings haben wir aber damit ein Automatenmodell mit beschränkter Parallelität für die regulären Sprachen, und wir können die Beschreibungskomplexität dieses Modells im Vergleich mit endlichen Automaten untersuchen. In Zellularautomaten führt zweiseitige Kommunikation im Vergleich mit einseitiger Kommunikation zu einem Anstieg der generativen Mächtigkeit. Wir werden in diesem Kapitel eingeschränkte Modelle mit unterschiedlichen zweiseitigen Kommunikationsstrukturen und die Auswirkungen auf die Beschreibungskomplexität der Modelle untersuchen.

Wir messen die Beschreibungskomplexität der eingeschränkten Zellularautomaten, indem wir die Anzahl der Zustände zählen. Es wird sich herausstellen, daß der Größenzuwachs zwischen DFA und Zellularautomaten mit beschränkter Zellenzahl durch ein Polynom beschränkt werden kann. Damit kann ein eingeschränkter Zellularautomat in polynomieller Zeit in einen DFA umgewandelt werden. Somit sind alle Entscheidbarkeitsfragen, die für DFA in polynomieller Zeit gelöst werden können, auch für eingeschränkte Zellularautomaten in polynomieller Zeit lösbar. Im Gegensatz zu Zellularautomaten oder iterativen Arrays kann das eingeschränkte Modell bezüglich der Anzahl der Zustände minimiert werden. Allerdings ist im Gegensatz zu DFA zur Zeit nicht bekannt, ob es einen effizienten Minimierungsalgorithmus gibt.

Wir werden im nächsten Abschnitt das eingeschränkte Modell definieren und zunächst nur mit einseitigem Informationsfluß ausstatten. Modelle mit minimaler, maximaler und graduell

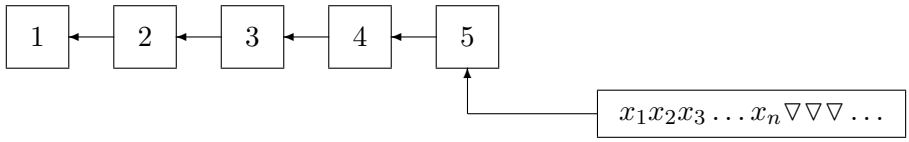



Abbildung 6.1: Ein 5-Zellen Zellularautomat mit einseitiger Kommunikation (5C-OCA)

steigender zweiseitiger Kommunikation werden in den Abschnitten 6.2 und 6.3 betrachtet. Im Abschnitt 6.4 untersuchen wir die Beschreibungskomplexität zwischen eingeschränkten Zellularautomaten mit einer unterschiedlichen Zellenzahl. Fragen zur Minimierung der eingeschränkten Modelle werden in Abschnitt 6.5 behandelt.

## 6.1 Zellularautomaten mit einseitigem Informationsfluß

Für ein $k \geq 1$ definieren und untersuchen wir in diesem Abschnitt Zellularautomaten mit beschränkter Zellenzahl $k$ und einseitiger Kommunikation. Ein solcher Automat besteht aus $k$ Zellen, die jeweils mit ihrem rechten Nachbar verbunden sind. Die rechte Randzelle ist die Kommunikationszelle für die Eingabe. Ein Beispiel für $k = 5$ findet man in Abbildung 6.1. Zellularautomaten mit beschränkter Zellenzahl arbeiten ähnlich wie das unbeschränkte Modell. Der nächste Zustand jeder Zelle hängt vom Zustand der Zelle selbst und vom Zustand ihres rechten Nachbarn ab. Die Überführungsfunktion wird synchron in jeder Zelle zum gleichen Zeitpunkt angewendet. Die Eingabe im eingeschränkten Modell wird ähnlich wie in iterativen Arrays verarbeitet. Anfangs sind alle Zellen im Ruhezustand $q_0$. In jedem Zeitschritt wird von der Kommunikationszelle ein Eingabesymbol gelesen und verarbeitet. Alle anderen Zellen arbeiten wie eben beschrieben. Sobald die erste Zelle einen akzeptierenden Zustand annimmt, wird die Eingabe akzeptiert. Die minimale Zeit, um die Eingabe zu lesen und alle Informationen von der Kommunikationszelle in die erste Zelle zu senden, ist die Länge der Eingabe plus die Anzahl der Zellen $k$. Daher verarbeitet die Kommunikationszelle ein besonderes Eingabeendesymbol $\triangledown$, sobald die Eingabe gelesen wurde.

### 6.1.1 Definition

Für die formale Definition eines Zellularautomaten mit beschränkter Zellenzahl $k$ passen wir das unbeschränkte Modell aus Kapitel 3 an.

**Definition 6.1.** Ein Zellularautomat mit beschränkter Zellenzahl $k$ und einseitigem Informationsfluß, abgekürzt $k$C-OCA ($k$ cells one-way CA), ist ein Tupel $(Q, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$, wobei folgendes gilt:

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände der Zellen,

(2) $\Sigma$ ist das Eingabealphabet,

(3) $\triangledown \notin Q \cup \Sigma$ ist das Symbol für das Eingabeende,

(4) $k$ ist die Anzahl der Zellen,

(5) $\delta_r : Q \times (\Sigma \cup \{\triangledown\}) \to Q$ ist die lokale Überführungsfunktion für die Kommunikationszelle,

(6) $\delta : Q \times Q \to Q$ ist die lokale Überführungsfunktion für die restlichen Zellen,

(7) $q_0 \in Q$ ist der Ruhezustand, für den folgendes gilt: $\delta_r(q_0, \triangledown) = q_0$ und $\delta(q_0, q_0) = q_0$

(8) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.

**Bemerkung 6.2.** Falls $k = 1$ ist, dann entfällt die Funktion $\delta$. Wir haben dann einen DFA mit der Akzeptanzbedingung eines $k$C-OCA. Das heißt, die Eingabe wird akzeptiert, sobald ein akzeptierender Zustand erreicht wird.

Eine Konfiguration eines $k$C-OCA zum Zeitpunkt $t \geq 0$ ist ein Paar $(c_t, w_t)$, wobei $w_t \in \Sigma^*$ die verbleibende Eingabe ist und $c_t$ eine Beschreibung der Zustände der $k$ Zellen ist. Formal ist $c_t$ eine Abbildung $c_t : \{1, \ldots, k\} \to Q$. Die Konfiguration $(c_0, w_0)$ zum Zeitpunkt 0 ist durch die Eingabe $w_0$ und die Abbildung $c_0(i) = q_0$ für $1 \leq i \leq k$ definiert. Die globale Überführungsfunktion $\Delta$ wird durch $\delta$ und $\delta_r$ folgendermaßen induziert: Sei $(c_t, w_t)$ für $t \geq 0$ eine Konfiguration. Dann ist:

$$\begin{aligned}(c_{t+1}, w_{t+1}) \;=\; & \Delta(c_t, w_t) \Longleftrightarrow \\ & c_{t+1}(i) = \delta(c_t(i), c_t(i+1)), i \in \{1, \ldots, k-1\} \\ & c_{t+1}(k) = \delta_r(c_t(k), x)\end{aligned}$$

wobei $x = \triangledown$, $w_{t+1} = \epsilon$ (falls $w_t = \epsilon$) und $x = x_1$, $w_{t+1} = x_2 \ldots x_n$ (falls $w_t = x_1 x_2 \ldots x_n$).

Eine Eingabe $w$ wird von einem $k$C-OCA akzeptiert, falls es einen Zeitpunkt $i$ während der Berechnung gibt, so daß die erste Zelle einen akzeptierenden Zustand annimmt.

**Definition 6.3.** Sei $A = (Q, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$ ein $k$C-OCA.

(1) Ein Wort $w \in \Sigma^+$ wird von $A$ akzeptiert, falls es einen Zeitpunkt $i \in \mathbb{N}$ gibt mit $c_i(1) \in F$ für die Konfiguration $(c_i, w_i) = \Delta^i(c_0, w_0)$.

(2) $T(A) = \{w \in \Sigma^+ \mid w \text{ wird von } A \text{ akzeptiert}\}$ bezeichnet die von $A$ akzeptierte Sprache.

(3) Der Automat $A$ akzeptiert in Realzeit, falls alle $w \in T(A)$ innerhalb von $|w|+k$ Zeitschritten akzeptiert werden. $\mathcal{L}(k\text{C-OCA}) = \{L \mid L \text{ wird von einem Realzeit-}k\text{C-OCA akzeptiert}\}$.

Als Maß für die Beschreibungskomplexität eines $k$C-OCA zählen wir die Anzahl der Zustände. Für einen $k$C-OCA $A = (Q, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$ definieren wir daher: $|A| = |Q|$

**Bemerkung 6.4.** In diesem Kapitel betrachten wir ausschließlich Zellularautomaten mit beschränkter Zellenzahl, die in Realzeit arbeiten. Daher werden wir die Begriffe „$k$C-OCA“ und „Realzeit-$k$C-OCA“ synonym verwenden.

**Beispiel 6.5.** Als Beispiel betrachten wir die Sprache

$$L_{n,k} = \{a^m \mid m \geq n^k\}$$

und präsentieren die Konstruktion für $n = 2$ und $k = 4$. Die Idee besteht darin, einen $n$-nären Zähler in den $k$ Zellen zu konstruieren. Dabei steht der Zustand + für einen Übertrag des Zählers. Sobald die erste Zelle den akzeptierenden Zustand + annimmt, wurden mindestens $n^k$ Eingabesymbole gelesen und die Eingabe wird daher akzeptiert. Wir konstruieren einen 4C-OCA $A = (\{0, 1, +\}, \{a\}, 0, \triangledown, 4, \delta_r, \delta, \{+\})$, wobei $\delta_r$ und $\delta$ wie folgt definiert sind:

| $\delta$ | 0 | 1 | + |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | + |
| + | 0 | 0 | · |

und

| $\delta_r$ | $a$ | $\triangledown$ |
|---|---|---|
| 0 | 1 | 0 |
| 1 | + | 1 |
| + | 1 | 1 |

Die Arbeitsweise des Automaten können wir an zwei Beispielen in Abbildung 6.2 auf Seite 85 sehen. In der linken Spalte wird die Eingabe $u = a^{20}$ verarbeitet. Nach $19 \leq |u| + k = 24$ Schritten nimmt die erste Zelle den akzeptierenden Zustand + an und akzeptiert somit die Eingabe. In der rechten Spalte bekommen wir als Eingabe $u = a^8$. Wir können beobachten, daß die erste Zelle niemals den akzeptierenden Zustand + annehmen kann. Wir sagen in einem solchen Fall, daß die Berechnung blockiert ist.

### 6.1.2 Umwandlung eines $k$C-OCA in einen DFA

Wir möchten nun zeigen, daß die Beschränkung auf eine feste Zellenzahl die generative Mächtigkeit des Modells auf die regulären Sprachen reduziert. Dazu geben wir zunächst eine Konstruktion an, die einen $k$C-OCA in einen DFA umwandelt. Diese Konstruktion liefert dann eine obere Schranke.

**Lemma 6.6.** Jeder $k$C-OCA $A$ mit $n$ Zuständen kann in einen DFA $M$ umgewandelt werden, so daß $T(M) = T(A)$ und $|M| \leq n^k - n^{k-1} + 1$ gilt.

*Beweis.* Ein DFA akzeptiert eine Eingabe $w$ genau dann, wenn der DFA nach $|w|$ Zeitschritten einen akzeptierenden Zustand annimmt. Die Akzeptierung bei einem $k$C-OCA ist nach Definition folgendermaßen: Eine Eingabe $w$ wird akzeptiert, sobald die erste Zelle einen akzeptierenden Zustand annimmt. Dies kann einerseits zu einem Zeitpunkt $t < |w|$ geschehen, also bevor die gesamte Eingabe gelesen ist. Andererseits kann auch erst zu einem Zeitpunkt $t$ mit $|w| \leq t \leq |w| + k$ akzeptiert werden, also nachdem die gesamte Eingabe gelesen ist. Bei der Konstruktion eines DFA müssen wir nun diese beiden Fälle berücksichtigen. Die Konstruktion kann in drei Schritte eingeteilt werden: Zunächst konstruieren wir das Kreuzprodukt der $k$ Zellen und erhalten so einen DFA, der ein Präfix von $w\triangledown^k$ akzeptiert, falls $w$ von dem gegebenen $k$C-OCA akzeptiert wird. Im nächsten Schritt wird dieser DFA folgendermaßen modifiziert: Sei $t$ der Zeitpunkt der Akzeptierung. Ist $t < |w|$, dann wird die restliche Eingabe in einer akzeptierenden Schleife verarbeitet. Ist $|w| < t \leq |w| + k$, dann wird die Menge der akzeptierenden Zustände geeignet erweitert, um die Eingabe zu akzeptieren. Im letzten Schritt werden einige akzeptierende Zustände zu einem akzeptierenden Zustand zusammengefaßt.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $a^{20}$ |
| 0 | 0 | 0 | 1 | $a^{19}$ |
| 0 | 0 | 0 | + | $a^{18}$ |
| 0 | 0 | 1 | 1 | $a^{17}$ |
| 0 | 0 | 1 | + | $a^{16}$ |
| 0 | 0 | + | 1 | $a^{15}$ |
| 0 | 1 | 0 | + | $a^{14}$ |
| 0 | 1 | 1 | 1 | $a^{13}$ |
| 0 | 1 | 1 | + | $a^{12}$ |
| 0 | 1 | + | 1 | $a^{11}$ |
| 0 | + | 0 | + | $a^{10}$ |
| 1 | 0 | 1 | 1 | $a^9$ |
| 1 | 0 | 1 | + | $a^8$ |
| 1 | 0 | + | 1 | $a^7$ |
| 1 | 1 | 0 | + | $a^6$ |
| 1 | 1 | 1 | 1 | $a^5$ |
| 1 | 1 | 1 | + | $a^4$ |
| 1 | 1 | + | 1 | $a^3$ |
| 1 | + | 0 | + | $a^2$ |
| + | 0 | 1 | 1 | $a$ |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $a^8$ |
| 0 | 0 | 0 | 1 | $a^7$ |
| 0 | 0 | 0 | + | $a^6$ |
| 0 | 0 | 1 | 1 | $a^5$ |
| 0 | 0 | 1 | + | $a^4$ |
| 0 | 0 | + | 1 | $a^3$ |
| 0 | 1 | 0 | + | $a^2$ |
| 0 | 1 | 1 | 1 | $a$ |
| 0 | 1 | 1 | + | $\nabla$ |
| 0 | 1 | + | 1 | $\nabla$ |
| 0 | + | 0 | 1 | $\nabla$ |
| 1 | 0 | 0 | 1 | $\nabla$ |
| 1 | 0 | 0 | 1 | $\nabla$ |

Abbildung 6.2: Berechnungen für die Eingaben $u = a^{20}$ und $u = a^8$ in Beispiel 6.5

1. Schritt:

Es sei $A = (Q, \Sigma, q_0, \nabla, k, \delta_r, \delta, F)$ der gegebene $k$C-OCA. Wir konstruieren nun einen DFA $M_1$ als das Kreuzprodukt der $k$ Zellen wie folgt: $M_1 = (Q_1, \Sigma_1, \delta_1, q_0', F_1)$ mit

$$\begin{aligned} Q_1 &= Q^k, \\ \Sigma_1 &= \Sigma \cup \{\nabla\}, \\ q_0' &= (q_0, q_0, \ldots, q_0), \\ F_1 &= F \times Q^{k-1} \end{aligned}$$

Für $q_i, q_i' \in Q$ $(1 \leq i \leq k)$ und $\sigma \in \Sigma_1$ definieren wir

$$\delta_1((q_1, q_2, \ldots, q_k), \sigma) = (q_1', q_2', \ldots, q_k'),$$

wobei $q_1' = \delta(q_1, q_2), q_2' = \delta(q_2, q_3), \ldots, q_{k-1}' = \delta(q_{k-1}, q_k)$ und $q_k' = \delta_r(q_k, \sigma)$ ist.

Sei nun eine Eingabe $w = w_1 \ldots w_n$ gegeben. Wir schreiben $w\nabla^k = w_1 w_2 \ldots w_n w_{n+1} \ldots w_{n+k}$, wobei $w_{n+l} = \nabla$ für $1 \leq l \leq k$ ist. Für $1 \leq i \leq n+k$ und $1 \leq j \leq k$ behaupten wir folgendes:

$$c_i(j) = q \Leftrightarrow \delta_1(q_0', w_1 w_2 \ldots w_i) = (q_1, q_2, \ldots, q_k) \text{ und } q_j = q.$$

Diese Behauptung kann durch eine Induktion nach $i$ bewiesen werden, wobei die beiden Fälle $j < k$ und $j = k$ zu unterscheiden sind.

Dann gilt:

$$\begin{aligned} w \in T(A) &\Leftrightarrow \exists i \leq |w| + k : c_i(1) \in F \\ &\Leftrightarrow \exists \overline{w} : \overline{w} \text{ ist ein Präfix von } w\nabla^k \text{ und } \delta_1(q_0', \overline{w}) = (q_1, q_2, \ldots, q_k) \text{ mit } q_1 \in F \\ &\Leftrightarrow \exists \overline{w} : \overline{w} \text{ ist ein Präfix von } w\nabla^k \text{ und } \overline{w} \in T(M_1) \end{aligned}$$

2. Schritt:

In diesem Schritt werden wir den DFA $M_1$ dahingehend modifizieren, daß nunmehr nicht nur die Präfixe von $w\nabla^k$ akzeptiert werden, sondern das vollständige Wort $w$, sofern $w \in T(A)$ ist. Dazu definieren wir einen DFA $M_2 = (Q_1, \Sigma, \delta_2, q_0', F_2)$ mit folgenden Eigenschaften:

(1) $\delta_2(q, \sigma) = \delta_1(q, \sigma)$ für $q \notin F_1$ und $\sigma \in \Sigma$

(2) $\delta_2(q, \sigma) = q$ für $q \in F_1$ und $\sigma \in \Sigma$

(3) $F_2 = F_1 \cup F_1'$ mit $F_1' = \{q \in (Q \setminus F) \times Q^{k-1} \mid \exists 1 \leq l \leq k : \delta_1(q, \nabla^l) \in F_1\}$

Sei $t$ der Zeitpunkt, an dem der $k$C-OCA akzeptiert. Dann stellt die Eigenschaft (2) sicher, daß die komplette Eingabe gelesen und akzeptiert wird, falls $t < |w|$. Die Eigenschaft (3) stellt sicher, daß $M_2$ bereits nach $|w|$ Zeitschritten akzeptiert, falls der $k$C-OCA erst nach mehr als $|w|$ Schritten akzeptiert. Zum formalen Beweis benötigen wir zunächst die folgende technische Aussage.

*Behauptung* 6.7. Es sei $q \in Q_1$ und $w \in \Sigma^*$. Falls $\delta_1(q, w') \notin F_1$ für alle echten Präfixe $w'$ von $w$ gilt, dann ist $\delta_2(q, w) = \delta_1(q, w)$. Falls $\delta_2(q, w') \notin F_1$ für alle echten Präfixe $w'$ von $w$ gilt, dann ist $\delta_1(q, w) = \delta_2(q, w)$.

Die Behauptung wird durch eine Induktion nach $|w|$ bewiesen. Wir beweisen nur die erste Aussage; der Beweis der zweiten Aussage verläuft analog.

Induktionsverankerung: Für $|w| = 1$ ist das leere Wort $\epsilon$ das einzige echte Präfix von $w$. Nach Voraussetzung ist $\delta_1(q, \epsilon) = q \notin F_1$. Nach Eigenschaft (1) ist dann $\delta_2(q, w) = \delta_1(q, w)$.

Induktionsschritt: Die Behauptung sei für $|w| = n$ bewiesen. Nach Voraussetzung gilt für die echten Präfixe $P = \{\epsilon, w_1, w_1w_2, \ldots, w_1 \ldots w_n\}$ von $w_1 \ldots w_{n+1}$: $\delta_1(q, w') \notin F_1$ für $w' \in P$. Dann ist

$$\begin{aligned} \delta_2(q, w_1 \ldots w_n w_{n+1}) &= \delta_2(\delta_2(q, w_1 \ldots w_n), w_{n+1}) \\ &= \delta_2(\delta_1(q, w_1 \ldots w_n), w_{n+1}) \text{ (nach Induktionsvoraussetzung)} \\ &= \delta_1(\delta_1(q, w_1 \ldots w_n), w_{n+1}) \text{ (nach (1), da } \delta_1(q, w_1 \ldots w_n) \notin F_1) \\ &= \delta_1(q, w_1 \ldots w_n w_{n+1}) \end{aligned}$$

Damit ist die Behauptung bewiesen.

Wir zeigen nun: Es gibt ein Wort $\overline{w} \in T(M_1)$ und $\overline{w}$ ist ein Präfix von $w\nabla^k \Leftrightarrow w \in T(M_2)$

„⇒“: Gegeben ist ein Wort $\overline{w}$, für das gilt: $\overline{w} \in T(M_1)$ und $\overline{w}$ ist ein Präfix von $w\nabla^k$. O.B.d.A. können wir annehmen, daß $\overline{w}$ das kürzeste Präfix von $w\nabla^k$ ist, so daß $\overline{w} \in T(M_1)$ ist. Wir haben zwei Fälle zu unterscheiden:

1. $\overline{w} = w_1 \dots w_i$ mit $i \leq n \Rightarrow \delta_1(q_0', \overline{w}) \in F_1$
   $\Rightarrow \delta_2(q_0', \overline{w}) \in F_1$ (Behauptung 6.7)
   $\Rightarrow \delta_2(q_0', w) \in F_1$ (Eigenschaft (2))
   $\Rightarrow w \in T(M_2)$

2. $\overline{w} = w\nabla^l$ mit $1 \leq l \leq k \Rightarrow \delta_1(q_0', w) = q \notin F_1$ und $\delta_1(q, \nabla^l) \in F_1$
   $\Rightarrow \delta_2(q_0', w) = q \in F_1'$ (Behauptung 6.7 und (3))
   $\Rightarrow w \in T(M_2)$

„⇐“: Ist $w \in T(M_2)$, dann gilt $\delta_2(q_0', w) \in F_1$ oder $\delta_2(q_0', w) \in F_1'$.

1. $\delta_2(q_0', w) \in F_1 \Rightarrow$ es gibt ein kürzestes Präfix $\overline{w}$ von $w$, so daß $\delta_2(q_0', \overline{w}) = q \in F_1$
   $\Rightarrow \delta_1(q_0', \overline{w}) \in F_1$ (Behauptung 6.7)
   $\Rightarrow \overline{w} \in T(M_1)$ und $\overline{w}$ ist ein Präfix von $w$ und daher von $w\nabla^k$

2. $\delta_2(q_0', w) \in F_1' \Rightarrow \exists q \notin F_1, l \leq k : \delta_2(q_0', w) = q$ und $\delta_2(q, \nabla^l) \in F_1$ ($l$ ist minimal)
   $\Rightarrow \delta_1(q_0', w) = q$ und $\delta_1(q, \nabla^l) \in F_1$ (Behauptung 6.7)
   $\Rightarrow \delta_1(q_0', w\nabla^l) = \delta_1(q_0', \overline{w}) \in F_1$ ($\overline{w} = w\nabla^l$)
   $\Rightarrow \overline{w} \in T(M_1)$ und $\overline{w}$ ist ein Präfix von $w\nabla^k$

Zusammen mit den Überlegungen aus dem 1. Schritt erhalten wir:

$$w \in T(A) \Leftrightarrow \exists \overline{w} : \overline{w} \text{ ist ein Präfix von } w\nabla^k \text{ und } \overline{w} \in T(M_1) \Leftrightarrow w \in T(M_2)$$

Damit ist $T(A) = T(M_2)$.

3. Schritt:

In diesem Schritt wird die Anzahl der Zustände reduziert, da einige Zustände zusammengefaßt werden können. Da es keine Transitionen von einem akzeptierenden Zustand in $F_1$ zu einem nichtakzeptierenden Zustand gibt, können wir alle akzeptierenden Zustände in $F_1$ durch einen einzigen akzeptierenden Zustand $f$ ersetzen. Formal definieren wir:

$M_3 = ((Q_1 \setminus F_1) \cup \{f\}, \Sigma, \delta_3, q_0', \{f\})$ und $\delta_3$ ist für $q \in Q_1 \setminus F_1$ und $\sigma \in \Sigma$ folgendermaßen definiert:

$$\begin{aligned} \delta_3(q, \sigma) &= \begin{cases} \delta_2(q, \sigma) & \text{falls } \delta_2(q, \sigma) \notin F_1 \\ f & \text{sonst} \end{cases} \\ \delta_3(f, \sigma) &= f \end{aligned}$$

Offenbar ist $T(M_3) = T(M_2) = T(A)$.

Die Anzahl der erreichbaren Zustände in $M_3$ kann nun folgendermaßen abgeschätzt werden: Wir setzen $n = |Q|$ und erhalten, da $|F| \geq 1$ angenommen werden kann:

$$|(Q_1 \setminus F_1) \cup \{f\}| = n^k - |F_1| + 1 = n^k - |F|n^{k-1} + 1 \leq n^k - n^{k-1} + 1$$

Damit haben wir zu einem gegebenen $k$C-OCA mit $n$ Zuständen einen äquivalenten DFA mit höchstens $n^k - n^{k-1} + 1$ Zuständen konstruiert. □

Wir zeigen nun mit Hilfe der Familie $L_{n,k}$ unärer Sprachen, daß die soeben in Lemma 6.6 bewiesene obere Schranke in der Größenordnung scharf ist. Für $n \geq 1$ und $k \geq 1$ sei

$$L_{n,k} = \{a^m \mid m \geq n^k\}$$

**Lemma 6.8.** Jeder DFA für die Sprache $L_{n,k}$ benötigt mindestens $n^k + 1$ Zustände.

*Beweis.* Nach Satz 2.7 entspricht der Index der Nerode-Relation $\equiv_L$ einer Sprache $L$ der Anzahl der Zustände eines minimalen DFA, der $L$ akzeptiert. Wir verwenden nun die Nerode-Relation $\equiv_{L_{n,k}}$ auf $L_{n,k}$ und zeigen, daß der Index von $\equiv_{L_{n,k}}$ größer oder gleich $n^k + 1$ ist. Für $x, y \in \Sigma^*$ ist $\equiv_{L_{n,k}}$ folgendermaßen definiert:

$$x \equiv_{L_{n,k}} y :\Longleftrightarrow xz \in L_{n,k} \Leftrightarrow yz \in L_{n,k} \text{ für alle } z \in \Sigma^*$$

Es seien $i, j \in \mathbb{N}$ mit $0 \leq i < j \leq n^k$. Dann ist $a^i a^{n^k-i-1} = a^{n^k-1} \notin L_{n,k}$ und $a^j a^{n^k-i-1} = a^{n^k+j-i-1} \in L_{n,k}$, da $j - i - 1 \geq 0$. Daraus folgt $a^i \not\equiv_{L_{n,k}} a^j$ und daher haben wir mindestens $n^k + 1$ paarweise verschiedene Äquivalenzklassen. Daher gilt: $\text{index}(\equiv_{L_{n,k}}) \geq n^k + 1$. □

**Lemma 6.9.** $L_{n,k}$ kann von einem $k$C-OCA mit $n + 1$ Zuständen akzeptiert werden und jeder $k$C-OCA für die Sprache $L_{n,k}$ benötigt mindestens $n + 1$ Zustände.

*Beweis.* Das Beispiel 6.5 auf Seite 84 zeigt, daß es einen $k$C-OCA mit $n + 1$ Zuständen gibt, der $L_{n,k}$ akzeptiert.

Wir zeigen nun, daß jeder $k$C-OCA, der $L_{n,k}$ akzeptiert, mindestens $n+1$ Zustände benötigt. Jeder $k$C-OCA $A$ muß mindestens $n^k + 1$ verschiedene Konfigurationen (inklusive der Startkonfiguration $(q_0, \ldots, q_0)$) innerhalb der ersten $n^k$ Zeitschritte annehmen. Werden weniger als $n^k + 1$ verschiedene Konfigurationen angenommen, dann wird innerhalb der ersten $n^k$ Zeitschritte mindestens eine Konfiguration zweimal angenommen. Seien $t_1$ und $t_2$ diese beiden Zeitpunkte mit $t_1 < t_2$. Da $a^{t_2} a^{n^k-t_2} \in L_{n,k}$ ist und nach Lesen von $a^{t_1}$ und $a^{t_2}$ die gleiche Konfiguration angenommen wird, gilt auch $a^{t_1} a^{n^k-t_2} \in L_{n,k}$. Das ist aber ein Widerspruch, da $n^k - t_2 + t_1 < n^k$ ist.

Wir nehmen nun an, daß der $k$C-OCA $A$ höchstens $n$ Zustände besitzt. Da $A$ aber $k$ Zellen besitzt, kann $A$ daher nur höchstens $n^k$ verschiedene Konfigurationen annehmen. Das ist aber ein Widerspruch, da nach den obigen Überlegungen mindestens $n^k + 1$ Konfigurationen unterschieden werden müssen. Also muß $A$ mindestens $n+1$ Zustände haben, damit mindestens $n^k + 1 \leq (n + 1)^k$ verschiedene Konfigurationen unterschieden werden können. Damit folgt die Behauptung. □

Wir fassen die Ergebnisse der Lemmata 6.6, 6.8 und 6.9 zusammen:

| $k$C-OCA | $\longrightarrow$ | DFA |
|---|---|---|
| $n$ | | $\leq n^k - n^{k-1} + 1 = O(n^k)$ |
| $n$ | | $\geq (n-1)^k + 1 = \Omega(n^k)$ |

Obwohl die obere Schranke nur in der Größenordnung scharf ist, läßt sich trotzdem eine echte Hierarchie bezüglich der Anzahl der Zustände zeigen. Jede Sprache, die von einem $k$C-OCA

mit $n$ Zuständen akzeptiert werden kann, wird offenbar auch von einem $k$C-OCA mit $n+1$ Zuständen akzeptiert. Aber es gibt eine unendliche Sprachfolge $L_n$, so daß jede Sprache $L_n$ von einem $k$C-OCA mit $n$ Zuständen akzeptiert wird, aber von keinem $k$C-OCA mit weniger als $n$ Zuständen.

**Korollar 6.10.** Für $n \geq 1$ und $k \geq 1$ gilt: Es gibt eine Sprache $L(n,k)$, die von einem $k$C-OCA mit $n+1$ Zuständen akzeptiert wird. Aber gibt es keinen $k$C-OCA für $L(n,k)$ mit weniger als $n+1$ Zuständen.

*Beweis.* Wir setzen $L(n,k) = L_{n,k}$ und erhalten die Behauptung unmittelbar aus Lemma 6.9. □

### 6.1.3 Umwandlung eines DFA in einen $k$C-OCA

Bei der Simulation eines DFA durch einen $k$C-OCA ist wiederum zu beachten, daß die beiden Automatenmodelle unterschiedliche Akzeptanzbedingungen besitzen. Wenn wir einen gegebenen DFA in der Kommunikationszelle eines $k$C-OCA simulieren, dann müssen wir sicherstellen, daß ein akzeptierender Zustand erst dann angenommen wird, wenn die gesamte Eingabe gelesen ist. Dazu ist ein zusätzlicher Zustand notwendig.

**Lemma 6.11.** Jeder DFA $M$ mit $n$ Zuständen kann in einen $k$C-OCA $A$ umgewandelt werden, so daß $T(A) = T(M)$ und $|A| = n+1$ gilt.

*Beweis.* Sei $M = (Q, \Sigma, \delta, q_0.F)$ ein DFA mit $n$ Zuständen. Wir konstruieren einen $k$C-OCA, der in der Kommunikationszelle den DFA $M$ simuliert. Nachdem die Eingabe gelesen ist, senden wir einen akzeptierenden Zustand mit maximaler Geschwindigkeit nach links, falls die Eingabe in $T(M)$ ist. Ansonsten wird die Berechnung blockiert.

Sei nun $g \notin Q$ ein neuer Zustand und $Q' = Q \cup \{g\}$. Wir definieren nun einen $k$C-OCA $A = (Q', \Sigma, q_0, \triangledown, k, \delta'_r, \delta', \{g\})$, wobei für $\delta'_r$ und $\delta'$ folgendes gilt:

$$\begin{aligned}
\delta'_r(q,\sigma) &= \delta(q,\sigma) \text{ für } \sigma \in \Sigma \text{ und } q \in Q \\
\delta'_r(f,\triangledown) &= g \text{ für } f \in F \\
\delta'_r(p,\triangledown) &= p \text{ für } p \in Q' \setminus F \\
\delta'(p,q) &= q \text{ für } p \in Q' \text{ und } q \in Q'
\end{aligned}$$

Durch eine einfache Induktion nach $i$ kann man zeigen:

$$\delta(q_0, u_1 u_2 \ldots u_i) = q \Leftrightarrow c_i(k) = q$$

Damit gilt:

$$\begin{aligned}
u \in T(M) &\Leftrightarrow \delta(q_0,u) \in F \Leftrightarrow c_{|u|}(k) \in F \text{ und } w_{|u|} = \epsilon \Leftrightarrow \\
&\Leftrightarrow c_{|u|+1}(k) = g \Leftrightarrow c_{|u|+k}(1) = g \Leftrightarrow u \in T(A)
\end{aligned}$$

Damit ist $T(A) = T(M)$ und $|A| = n+1$ und die Behauptung daher bewiesen. □

Wir zeigen nun, daß die soeben vorgestellte Konstruktion optimal ist. Das heißt, in manchen Fällen benötigt ein paralles Berechnungsmodell sogar mehr Zustände als ein sequentielles Modell. Zum Beweis der unteren Schranke betrachten wir eine feste Primzahl $p \geq 2$ und die Sprache

$$L_p = \{a^n \mid n \geq 0 \text{ und } n \equiv 1 \bmod p\}$$

**Lemma 6.12.** Jeder $k$C-OCA für die Sprache $L_p$ benötigt mindestens $p+1$ Zustände.

*Beweis.* Gegeben ist ein $k$C-OCA $A = (Q, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$, der die Sprache $L_p$ mit $n$ Zuständen akzeptiert. Für $1 \leq i \leq k$ definieren wir Projektionen $\pi_i : Q^k \times \Sigma^* \to Q^{k-i+1}$ durch $\pi_i((q_1, q_2, \ldots, q_k), w) = (q_i, q_{i+1}, \ldots, q_k)$.

Wir betrachten nun die Folgen $s_i = (\pi_i(\Delta^t(c_0, a^t)))_{t \geq 0}$, die die Entwicklung der Zustände der letzten $k-i+1$ Zellen des $k$C-OCA $A$ unter entsprechenden Eingaben $a^t$ darstellen. Da $A$ eine unäre Eingabe verarbeitet und außerdem nur einseitiger Informationsfluß möglich ist, werden die Folgen $s_i$ ab einem Zeitpunkt periodisch. Da $A$ gerade $n$ Zustände und $k$ Zellen besitzt, sind zwei Elemente innerhalb der ersten $n^{k-i+1}+1$ Elemente der Folge $s_i$ identisch. Mit $l_i$ bezeichnen wir die Länge der Periode zwischen dem ersten Auftreten zweier identischer Elemente in $s_i$. Wir setzen $p_k = l_k$. Offenbar gilt: $l_k = p_k \leq n$

Nun ist $l_{k-1}$ offenbar ein Vielfaches von $l_k$. Das heißt, $l_{k-1} = p_{k-1}p_k$, wobei $1 \leq p_{k-1} \leq n$ gilt, da $|A| = n$. Mit der gleichen Argumentation erhalten wir: $l_{k-2} = p_{k-2}p_{k-1}p_k$ mit $1 \leq p_{k-2} \leq n$ Allgemein haben wir dann: $l_i = p_i p_{i+1} \ldots p_k$ mit $1 \leq p_j \leq n$ für $1 \leq i \leq k$ und $i \leq j \leq k$. Dann ist $l_1 = p_1 p_2 \ldots p_{k-1} p_k$ die Länge der „Periode von $A$“, da $\Delta^{l_1}(c, a^m) = (c, a^{m-l_1})$ für jede Konfiguration $(c, a^m)$ mit $m \geq l_1$ gilt.

Annahme 1: $n < p$

Daraus folgt unmittelbar, daß $p$ nicht $l_1 = p_1 p_2 \ldots p_k$ teilt, da $p_i \leq n < p$ für $1 \leq i \leq k$ gilt und $p$ eine Primzahl ist. Wir wählen jetzt eine ganze Zahl $t'$ so, daß $t'p + 1 > n^k + 1$ gilt. Nun ist $a^{t'p+1} \in L_p$. Also ist $\Delta^{t'p+1}(c_0, a^{t'p+1}\triangledown^k) = (c', \triangledown^k)$ und es gibt eine ganze Zahl $k'$ mit $1 \leq k' \leq k$, so daß $\Delta^{k'}(c', \triangledown^{k'}) = (c'', \epsilon)$ und $c''(1) \in F$ gilt. Wir betrachten nun die Eingabe $w = a^{t'p+1+l_1}$. Dann ist $\Delta^{t'p+1}(c_0, w\triangledown^k) = (c', a^{l_1}\triangledown^k)$, $\Delta^{l_1}(c', a^{l_1}\triangledown^k) = (c', \triangledown^k)$ und $\Delta^{k'}(c', \triangledown^{k'}) = (c'', \epsilon)$. Dann ist $w \in L_p$ und daher gilt: $t'p + 1 + l_1 = t''p + 1$ mit einem $t'' \geq 1$. Also ist $t'p + l_1$ ein Vielfaches von $p$. Daher teilt $p$ die Zahl $l_1 = p_1 p_2 \ldots p_k$, was ein Widerspruch ist.

Annahme 2: $n = p$

Wir beobachten, daß es mindestens eine Zelle $j$ gibt, die alle $p$ Zustände unter der Eingabe $a^m$ $(m \geq n^k + 1)$ annimmt. Anderenfalls ist $p_i < n = p$ für $1 \leq i \leq k$ und es folgt, daß $p$ nicht die Zahl $l_1 = p_1 p_2 \ldots p_k$ teilt. Wie eben können wir dann schließen, daß $p$ doch die Zahl $l_1 = p_1 p_2 \ldots p_k$ teilt und erhalten einen Widerspruch.

Die Sprache $L_p$ ist so konstruiert, daß erst nach Lesen der gesamten Eingabe entschieden werden kann, ob die Eingabe akzeptiert wird oder nicht. Daher kann bei einer Eingabe $a^m\triangledown^k$ $(m \geq 1)$ die erste Zelle frühestens zum Zeitpunkt $m+k$ einen akzeptierenden Zustand annehmen. Sei nun $a^m \in L_p$ $(m \geq n^k + 1)$ gegeben. Nachdem das Eingabeendesymbol $\triangledown$ das erste Mal gelesen wurde, muß die Information, daß somit die gesamte Eingabe gelesen worden ist, zur ersten Zelle transportiert werden. Diese Information passiert die Zelle $j$ zum Zeitpunkt

$m+k-j+1$. Da die Information in Form eines Zustands transportiert wird, bezeichnen wir mit $q \in Q$ diesen Zustand, den die Zelle $j$ zum Zeitpunkt $m+k-j+1$ annimmt. Also gilt: $\Delta^{m+k-j+1}(c_0, a^m \triangledown^k) = (c, \triangledown^{j-1})$ mit $c(j) = q$ und $\Delta^{j-1}(c, \triangledown^{j-1}) = (c', \epsilon)$ mit $c'(1) \in F$.

Wir definieren nun eine Projektion $\pi : Q^k \to Q^j$ durch $\pi(q_1, q_2, \ldots, q_k) = (q_1, q_2, \ldots, q_{j-1}, q_j)$. Wir können beobachten, daß der Zustand $q$ in der Zelle $j$ zu einem akzeptierenden Zustand in der ersten Zelle nach $j-1$ Zeitschritten führt, und zwar unabhängig von der verbleibenden Eingabe. Daher führt jede Konfiguration $d \in Q^k$ mit $\pi(d) = \pi(c)$ zu einem akzeptierenden Zustand in der ersten Zelle nach $j-1$ Zeitschritten unabhängig von den Zuständen der Zellen $j+1, \ldots, k$ und der verbleibenden Eingabe. Da nun $s_1$ periodisch ist, in $A$ nur einseitiger Informationsfluß möglich ist und die Zelle $j$ alle Zustände in $Q$ annimmt, gibt es eine ganze Zahl $m' \leq n^k$ mit $\Delta^{m'}(c_0, a^{m'}) = (d, \epsilon)$ und $\pi(d) = \pi(c)$. Sei nun eine ganze Zahl $m'' \geq j$ so gewählt, daß $m'+m''-1$ kein Vielfaches von $p$ ist. Dann gilt: $\Delta^{m'}(c_0, a^{m'+m''}) = (d, a^{m''})$ und $\Delta^{j-1}(d, a^{m''}) = (d', a^{m''-j+1})$ mit $d'(1) \in F$. Also ist $a^{m'+m''} \in L_p$ und daher ist $m'+m''-1$ ein Vielfaches von $p$. Das ist ein Widerspruch zur vorherigen Annahme.

Da also beide Annahmen zu einem Widerspruch führen, benötigt jeder $k$C-OCA für die Sprache $L_p$ mindestens $n \geq p+1$ Zustände. □

**Lemma 6.13.** Jeder DFA für die Sprache $L_p$ benötigt mindestens $p$ Zustände.

*Beweis.* Wie im Beweis von Lemma 6.8 benutzen wir Satz 2.7 und die Nerode-Relation $\equiv_{L_p}$ auf $L_p$. Es seien $i, j \in \mathbb{N}$ mit $0 \leq i < j \leq p-1$. Dann ist $a^i a^{p-i+1} = a^{p+1} \in L_p$ und $a^j a^{p-i+1} = a^{p+1+j-i} \notin L_p$, da $0 < j-i < p$ gilt. Also ist $a^i \not\equiv_{L_p} a^j$ und wir erhalten $\text{index}(\equiv_{L_p}) \geq p$. □

Da es unendlich viele Primzahlen gibt, ist $g(n) = n+1$ eine untere Schranke für den Tradeoff zwischen DFA und $k$C-OCA. Wir haben also:

| DFA | $\longrightarrow$ | $k$C-OCA |
|---|---|---|
| $n$ | | $\leq n+1$ |
| $n$ | | $\geq n+1$ |

Diese Überlegungen zeigen, daß es Sprachen gibt, denen ein paralleles Berechnungsmodell nicht hilft, um die Beschreibungsgröße im Vergleich mit einem sequentiellen Modell zu reduzieren. Es sollte erwähnt werden, daß das obige Ergebnis nicht von der speziellen Anzahl $k$ der Zellen eines $k$C-OCA abhängt. Daher sind die Sprachen $L_p$ für $k$C-OCA inhärent sequentiell oder schwer zu parallelisieren, da jede Erhöhung der Parallelität in Form von zusätzlichen Zellen nicht die Beschreibungsgröße reduzieren kann.

In der Konstruktion in Lemma 6.11 wurde ein zusätzlicher Zustand $g$ eingeführt, der sicherstellt, ob die gesamte Eingabe gelesen wurde oder nicht. Die untere Schranke zeigt, daß dieser zusätzliche Zustand in manchen Fällen notwendig ist. Also haben wir das Ergebnis, daß in manchen Fällen zusätzlicher Aufwand notwendig ist, um ein Array von DFA zu verwalten im Gegensatz zu einem einzelnen DFA.

## 6.2 Zellularautomaten mit zweiseitigem Informationsfluß

In diesem Abschnitt führen wir zweiseitigen Informationsfluß in $k$C-OCA ein. Im unbeschränkten Fall führt zweiseitige Kommunikation gegenüber einseitiger Kommunikation zu

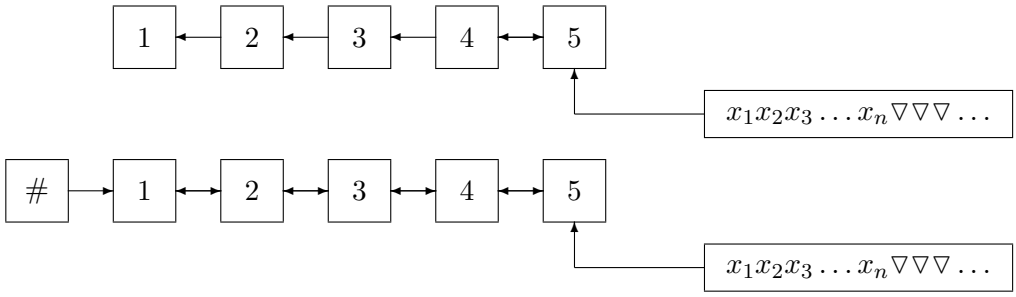



Abbildung 6.3: Ein 5-Zellen Zellularautomat mit zweiseitiger Kommunikation, 5C-CA (unten), und ein 5-Zellen Zellularautomat mit einseitiger Kommunikation und zweiseitiger Kommunikationszelle, 5C-OCA$_t$ (oben).

einer größeren generativen Mächtigkeit und zu nichtrekursiven Tradeoffs. Wir betrachten hier mit $k$C-CA die eingeschränkte Variante eines Realzeit-CA, das heißt, jede Zelle kann mit ihrem linken und rechten Nachbarn kommunizieren. Die linke Randzelle besitzt dann als linken Nachbarn einen besonderen Grenzzustand. $k$C-CA sind ein eingeschränktes Modell mit maximalem zweiseitigem Informationsfluß. Wir werden außerdem mit $k$C-OCA$_t$ ein Modell mit minimalem zweiseitigem Informationsfluß betrachten. Da sich die Kommunikationszelle von allen anderen Zellen bereits unterscheidet, werden wir dort zusätzlich Kommunikation mit der linken Nachbarzelle erlauben. Alle anderen Zellen bleiben mit einseitiger Kommunikation ausgestattet. Im nächsten Abschnitt werden wir die beiden Modelle formal definieren. Beispiele für einen 5C-CA und 5C-OCA$_t$ finden sich in Abbildung 6.3.

### 6.2.1 Definition

Die Definition eines $k$C-OCA wird in naheliegender Weise auf beschränkte Modelle mit zweiseitigem Informationsfluß erweitert.

**Definition 6.14.** Ein Zellularautomat mit beschränkter Zellenzahl $k$ und zweiseitigem Informationsfluß, abgekürzt $k$C-CA ($k$ cells CA), ist ein Tupel $(Q, \#, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$, wobei folgendes gilt:

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände der Zellen,

(2) $\# \notin Q \cup \Sigma$ ist der Grenzzustand,

(3) $\Sigma$ ist das Eingabealphabet,

(4) $\triangledown \notin Q \cup \Sigma$ ist das Symbol für das Eingabeende,

(5) $k$ ist die Anzahl der Zellen,

(6) $\delta_r : (Q \cup \{\#\}) \times Q \times (\Sigma \cup \{\triangledown\}) \to Q$ ist die lokale Überführungsfunktion für die Kommunikationszelle,

(7) $\delta : (Q \cup \{\#\}) \times Q \times Q \to Q$ ist die lokale Überführungsfunktion für die restlichen Zellen,

(8) $q_0 \in Q$ ist der Ruhezustand, für den folgendes gilt: $\delta_r(q_0, q_0, \triangledown) = q_0$, $\delta_r(\#, q_0, \triangledown) = q_0$, $\delta(q_0, q_0, q_0) = q_0$ und $\delta(\#, q_0, q_0) = q_0$

(9) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.

Ein Zellularautomat mit beschränkter Zellenzahl, einseitigem Informationsfluß und zweiseitiger Kommunikationszelle, abgekürzt $k$C-OCA$_t$ ($k$ cells OCA with two-way communication cell), ist wie ein $k$C-CA definiert. Lediglich die Funktion $\delta$ ist folgendermaßen beschränkt:

(7′) $\delta : Q \times Q \to Q$ ist die Überführungsfunktion für die restlichen Zellen. Es gilt: $\delta(q_0, q_0) = q_0$

**Bemerkung 6.15.** Für den Fall $k = 1$ entfällt auch hier die Funktion $\delta$ und wir erhalten einen DFA mit veränderter Akzeptanzbedingung.

Die Arbeitsweise von $k$C-OCA$_t$ und $k$C-CA ist nahezu identisch zur Arbeitsweise von $k$C-OCA, denn wie in $k$C-OCA hängt der nächste Zustand jeder Zelle vom Zustand der Zelle selbst und ihres rechten Nachbarn ab. Zusätzlich hängt der nächste Zustand der Kommunikationszelle in $k$C-OCA$_t$ und der nächste Zustand aller Zellen in $k$C-CA vom Zustand der linken Nachbarzelle ab.

Die Definition einer Konfiguration eines $k$C-OCA$_t$ oder $k$C-CA ist analog zur Definition bei $k$C-OCA (s. S. 83). Die globale Überführungsfunktion $\Delta$ wird durch $\delta$ und $\delta_r$ folgendermaßen induziert: Sei $(c_t, w_t)$ für $t \geq 0$ eine Konfiguration.

Für $k$C-OCA$_t$:

$$\begin{aligned}(c_{t+1}, w_{t+1}) \;=\; & \Delta(c_t, w_t) \Longleftrightarrow \\ & c_{t+1}(i) = \delta(c_t(i), c_t(i+1)), i \in \{1, \ldots, k-1\} \\ & c_{t+1}(k) = \delta_r(c_t(k-1), c_t(k), x)\end{aligned}$$

Für $k$C-CA:

$$\begin{aligned}(c_{t+1}, w_{t+1}) \;=\; & \Delta(c_t, w_t) \Longleftrightarrow \\ & c_{t+1}(1) = \delta(\#, c_t(1), c_t(2)) \\ & c_{t+1}(i) = \delta(c_t(i-1), c_t(i), c_t(i+1)), i \in \{2, \ldots, k-1\} \\ & c_{t+1}(k) = \delta_r(c_t(k-1), c_t(k), x),\end{aligned}$$

wobei $x = \triangledown$, $w_{t+1} = \epsilon$ (falls $w_t = \epsilon$) und $x = x_1$, $w_{t+1} = x_2 \ldots x_n$ (falls $w_t = x_1 x_2 \ldots x_n$).

Eine Eingabe $w$ wird von einem $k$C-CA oder $k$C-OCA$_t$ akzeptiert, falls es einen Zeitpunkt $i$ während der Berechnung gibt, so daß die erste Zelle einen akzeptierenden Zustand annimmt. Die Sprachklassen $\mathcal{L}(k\text{C-OCA}_t)$ und $\mathcal{L}(k\text{C-CA})$ sind analog zu Definition 6.3 definiert.

### 6.2.2 Umwandlung eines $k$C-OCA$_t$ ($k$C-CA) in einen DFA

Als erstes Ergebnis ist festzuhalten, daß zweiseitiger Informationsfluß die generative Mächtigkeit nicht erhöht.

**Lemma 6.16.** Jeder $k$C-OCA$_t$ ($k$C-CA) $A$ mit $n$ Zuständen kann in einen DFA $M$ umgewandelt werden, so daß $T(M) = T(A)$ und $|M| \leq n^k - n^{k-1} + 1$ gilt.

*Beweis.* Die Konstruktion für $k$C-OCA$_t$ und $k$C-CA ist im wesentlichen identisch zur Konstruktion für $k$C-OCA aus Lemma 6.6. Lediglich die Überführungsfunktion $\delta_1$ muß an die entsprechende Nachbarschaft angepaßt werden. Das heißt, für $q_i, q_i' \in Q$ $(1 \leq i \leq k)$ und $\sigma \in \Sigma_1$ definieren wir

$$\delta_1((q_1, q_2, \ldots, q_k), \sigma) = (q_1', q_2', \ldots, q_k'),$$

wobei $q_1' = \delta(q_1, q_2), q_2' = \delta(q_2, q_3), \ldots, q_{k-1}' = \delta(q_{k-1}, q_k)$ und $q_k' = \delta_r(q_{k-1}, q_k, \sigma)$ ($k$C-OCA$_t$) und $q_1' = \delta(\#, q_1, q_2), q_2' = \delta(q_1, q_2, q_3), \ldots, q_{k-1}' = \delta(q_{k-2}, q_{k-1}, q_k)$ und $q_k' = \delta_r(q_{k-1}, q_k, \sigma)$ für $k$C-CA. Durch diese Anpassungen wird offenbar die Anzahl der Zustände nicht verändert. □

Auch für $k$C-CA und $k$C-OCA$_t$ ist diese obere Schranke asymptotisch scharf. Daher ist die Konstruktion im wesentlichen optimal.

**Lemma 6.17.** $L_{n,k}$ kann von einem $k$C-OCA$_t$ ($k$C-CA) mit $n+1$ Zuständen akzeptiert werden und jeder $k$C-OCA$_t$ ($k$C-CA) für die Sprache $L_{n,k}$ benötigt mindestens $n+1$ Zustände.

*Beweis.* Der Beweis für $k$C-OCA$_t$ und $k$C-CA ist identisch zum Beweis für $k$C-OCA aus Lemma 6.9, denn die Fähigkeit den Zustand des linken Nachbarn zu sehen, verringert offenbar nicht die Anzahl der zu unterscheidenden Konfigurationen. □

### 6.2.3 Umwandlung eines DFA in einen $k$C-OCA$_t$ ($k$C-CA)

Wir haben in Lemma 6.12 bewiesen, daß die Konstruktion aus Lemma 6.11 für $k$C-OCA im allgemeinen nicht verbessert werden kann. In $k$C-OCA$_t$ und $k$C-CA steht der Kommunikationszelle der aktuelle Zustand ihres linken Nachbarn zusätzlich zur Verfügung. Daher können wir in diesen Fällen einen gegebenen DFA in den letzten beiden Zellen eines $k$C-OCA$_t$ simulieren. Die dadurch erzielte Ersparnis bezüglich der Anzahl der Zustände ist quadratisch.

**Lemma 6.18.** Jeder DFA $M$ mit $n$ Zuständen kann in einen äquivalenten $k$C-OCA$_t$ ($k$C-CA) $A$ mit $k \geq 2$ umgewandelt werden. Beschreibt $M$ eine Sprache aus $\Sigma^*$, dann gilt für die Anzahl der Zustände in $A$: $|A| \leq \lceil \sqrt{n} \rceil (|\Sigma| + 1) + 2|\Sigma| + 2 = O(\sqrt{n})$

*Beweis.* Es sei $M = (Q, \Sigma, \delta, 0, F)$ ein DFA mit $n$ Zuständen.

Wir konstruieren einen $k$C-OCA$_t$, der den DFA $M$ in den beiden letzten Zellen simuliert. Die Zustandsmenge $Q$ wird durch zwei Symbole über einem $\lceil \sqrt{n} \rceil$-nären Alphabet kodiert. Diese zweistellige Kodierung wird dann benutzt, um die erste Stelle des aktuellen Zustands von $M$ in der vorletzten Zelle und die zweite Stelle in der letzten Zelle zu berechnen. Nachdem die gesamte Eingabe gelesen ist, wird überprüft, ob ein akzeptierender Zustand von $M$ in den beiden letzten Zellen errechnet wurde. Ist dies der Fall, dann wird ein akzeptierender

Zustand mit maximaler Geschwindigkeit nach links gesendet. Ansonsten wird die Berechnung blockiert.

Zunächst eine technische Vorbereitungen:

- Es sei $Q = \{0, 1, 2, \ldots, n-1\}$ die Zustandsmenge des gegebenen DFA und $m = \lceil \sqrt{n} \rceil - 1$. Wir definieren die $m$-näre Kodierung von $Q$ wie folgt:

  $$Q_c = \{00, 01, 02, \ldots, 0m, 10, 11, 12, \ldots, 1m, \ldots, m0, m1, m2, \ldots, mm\}$$

  Falls $|Q_c| > |Q|$ ist, dann entfernen wir einige Elemente aus $Q_c$, damit $|Q_c| = |Q|$ ist.

- $\phi : Q \to Q_c$ ist eine bijektive Funktion zwischen $Q$ und $Q_c$, die Zustände aus $Q$ und $Q_c$ eindeutig zuordnet. Außerdem gelte $\phi(0) = 00$.

- Die Abbildungen $[\cdot]_1, [\cdot]_2 : Q \to \{0, 1, 2, \ldots, m\}$ definieren wir als Projektionen von Zuständen $q \in Q$ auf die erste bzw. zweite Stelle ihrer Kodierungen.
  Z.B.: $q = 1, \phi(q) = 01, [q]_1 = 0, [q]_2 = 1$.

- Nun definieren wir die Zustandsmenge $Q' = (\{0, 1, 2, \ldots, m\} \times (\Sigma \cup \{h\})) \cup \{0, g\} \cup \Sigma \cup \Sigma'$, wobei $\Sigma'$ eine Kopie von $\Sigma$ ist und $\{g, h\} \cap (\{0, 1, 2, \ldots, m\} \cup \Sigma \cup \Sigma') = \emptyset$ gilt.

- $\psi : \Sigma \to \Sigma'$ ist eine Bijektion, die durch $\psi(\sigma) = \sigma'$ für alle $\sigma \in \Sigma$ definiert ist.

Wir definieren nun den $k$C-OCA$_t$ $A = (Q', \#, \Sigma, 0, \triangledown, k, \delta'_r, \delta', \{g\})$. Die Überführungsfunktionen definieren wir für $i, j \in \{0, 1, 2, \ldots, m\}$, $\sigma, \tau, \tau_l, \tau_r \in \Sigma$ und $\tau', \tau'_r \in \Sigma'$ wie folgt:

$$\begin{aligned}
\delta'_r(0, 0, \sigma) &= \sigma && (6.1)\\
\delta'_r(0, \tau, \sigma) &= \psi(\sigma) && (6.2)\\
\delta'_r((i, \tau_l), \tau', \sigma) &= ([\delta(0, \tau_l)]_2, \sigma) && (6.3)\\
\delta'_r((j, \tau_l), (i, \tau), \sigma) &= ([\delta(\phi^{-1}(j, i), \tau_l)]_2, \sigma) && (6.4)\\
\delta'_r(0, \tau, \triangledown) &= \begin{cases} g & \text{falls } \delta(0, \tau) \in F \\ \tau & \text{sonst} \end{cases} && (6.5)\\
\delta'_r((i, \tau_l), \tau', \triangledown) &= ([\delta(\phi^{-1}(i, 0), \tau_l)]_2, h) && (6.6)\\
\delta'_r((j, \tau_l), (i, \tau), \triangledown) &= ([\delta(\phi^{-1}(j, i), \tau_l)]_2, h) && (6.7)\\
\delta'_r(q_l, q, \triangledown) &= q \text{ für } q_l \in Q' \text{ und } q \in \{g\} \cup (\{0, \ldots, m\} \times \{h\}) && (6.8)\\
\delta'(0, \tau) &= (0, \tau) && (6.9)\\
\delta'((0, \tau), \tau'_r) &= ([\delta(0, \tau)]_1, \psi^{-1}(\tau'_r)) && (6.10)\\
\delta'((i, \tau), (j, \tau_r)) &= ([\delta(\phi^{-1}(i, j), \tau)]_1, \tau_r) && (6.11)\\
\delta'((i, \tau), (j, h)) &= \begin{cases} g & \text{falls } \delta(\phi^{-1}(i, j), \tau) \in F \\ (i, \tau) & \text{sonst} \end{cases} && (6.12)\\
\delta'(q, g) &= g;\ q \in Q' && (6.13)\\
\delta'(g, q) &= q;\ q \in Q' && (6.14)\\
\delta'(0, (i, \tau_r)) &= (i, \tau_r) && (6.15)
\end{aligned}$$

Die übrigen Transitionen sind nicht definiert. Anstatt hier einen formalen Beweis zu geben, beschreiben wir im folgenden die Arbeitsweise eines $k$C-OCA$_t$. Außerdem verweisen wir auf das folgende Beispiel 6.19.

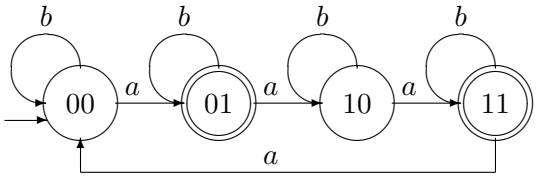
Die beiden letzten Zellen werden in den ersten beiden Zeitschritten initialisiert. Dazu dienen die Regeln (6.1), (6.2) und (6.9). Danach steht eine Kodierung des Startzustandes in der linken Komponente der Zellen und die ersten beiden Eingabesymbole in der rechten Komponente. Ein Zustand $\sigma'$ in der Kommunikationszelle entspricht einem Zustand $(0, \sigma)$. In jedem nächsten Zeitschritt wird nun ein Eingabesymbol gelesen und in die rechte Komponente der Kommunikationszelle geschoben, deren rechte Komponente bereits in die vorletzte Zelle geschrieben wurde. Gleichzeitig wird der nächste Zustand des DFA errechnet und in die linken Komponenten geschrieben. Dazu dienen die Regeln (6.4) und (6.11), sowie (6.3) und (6.10) für die Zustände $\sigma'$. Wird das Eingabeendesymbol zum ersten Mal gelesen, wird die Kommunikationszelle mit einem besonderen Symbol $h$ markiert (Regeln (6.6) und (6.7)). Im nächsten Schritt verarbeitet die vorletzte Zelle das letzte Eingabesymbol, das in der rechten Komponente gespeichert war (Regel (6.12)). Wird ein akzeptierender Zustand von $M$ errechnet, dann wird ein akzeptierender Zustand $g$ mit maximaler Geschwindigkeit nach links gesendet. Anderenfalls wird die Berechnung blockiert. Besteht die Eingabe nur aus einem Symbol, dann muß die Kommunikationszelle bereits nach zwei Zeitschritten entscheiden, ob die Eingabe akzeptiert wird oder nicht. Diesen Fall deckt die Regel (6.5) ab. Die Regeln (6.8), (6.13) und (6.14) dienen zur Verarbeitung des Zustands $g$. Die Regel (6.15) dient zur Initialisierung der restlichen Zellen.

Die Anzahl der Zustände in $Q'$ kann folgendermaßen abgeschätzt werden:

$$|Q'| \leq \lceil \sqrt{n} \rceil (|\Sigma| + 1) + 2|\Sigma| + 2 = O(\sqrt{n})$$

Die Konstruktion für $k$C-CA ist im wesentlichen identisch: In allen Zellen außer der Kommunikationszelle werden Informationen von links einfach ignoriert. □

**Beispiel 6.19.** Wir betrachten folgenden DFA $M$. Die Arbeitsweise eines 4C-OCA$_t$ auf den Eingaben $b$, $ab$, $a$, $aababa$ und $aaab$ ist in den Abbildungen 6.4, 6.6 und 6.5 dargestellt.



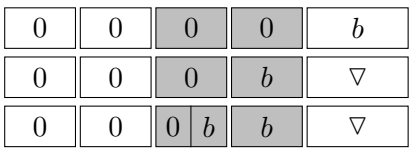




Abbildung 6.4: Berechnung in Beispiel 6.19 für die Eingabe $b$

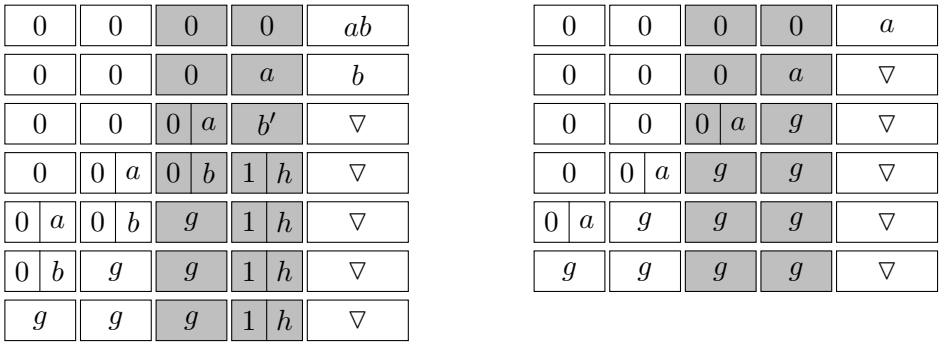



Abbildung 6.5: Berechnung in Beispiel 6.19 für die Eingaben $ab$ und $a$

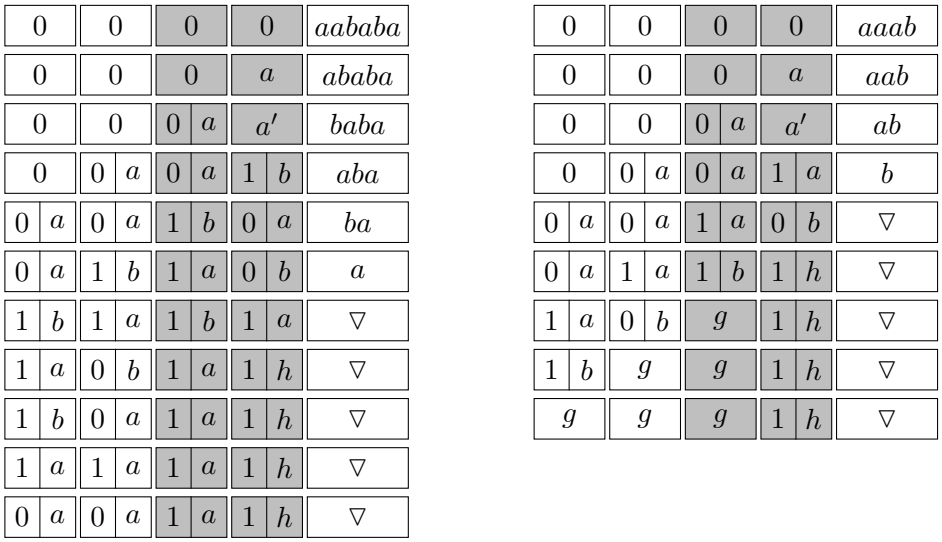



Abbildung 6.6: Berechnung in Beispiel 6.19 für die Eingaben $aababa$ und $aaab$

Da in der gerade vorgestellten Konstruktion gerade die letzten beiden Zellen genutzt wurden und die restlichen Zellen ungenutzt blieben, stellt sich natürlich unmittelbar die Frage, ob die Konstruktion optimal ist oder verbessert werden kann. Im Fall minimaler zweiseitiger Kommunikation können wir im nächsten Lemma zeigen, daß die Konstruktion aus Lemma 6.18 fast optimal ist, denn wir erhalten eine asymptotisch fast scharfe obere Schranke. Dieses Ergebnis ist für $k$C-OCA$_t$ plausibel, da nur die letzten beiden Zellen durch zweiseitige Kommunikation erreicht werden können. Wenn wir aber einen $k$C-CA betrachten, könnte man sich zunächst vorstellen, die Zustandsmenge des gegebenen DFA $k$-stellig mit Symbolen über einem $\lceil \sqrt[k]{n} \rceil$-nären Alphabet zu kodieren und die Simulation dann in allen $k$ Zellen durchzuführen. Mit ähnlichen Überlegungen wie für $k$C-OCA$_t$ können wir zeigen, daß das nicht immer möglich ist, denn wir erhalten eine untere Schranke für $k$C-CA der Größenordnung $\Omega(\sqrt[3]{n/\log n})$.

**Lemma 6.20.** Es gibt eine unendliche Folge von Sprachen $(L_n)_{n\in\mathbb{N}}$, so daß jede Sprache $L_n$ von einem DFA mit $n+1$ Zuständen akzeptiert wird. Jeder $k$C-OCA$_t$ für $L_n$ benötigt allerdings mindestens $\Omega(\sqrt{n/\log n})$ Zustände. Jeder $k$C-CA für $L_n$ benötigt mindestens $\Omega(\sqrt[3]{n/\log n})$ Zustände.

*Beweis.* Wir zeigen, daß es eine von $k$ abhängige Konstante $c$ gibt, so daß für jedes $n \in \mathbb{N}$ mit $n \geq c$ gilt: Es gibt eine einelementige Sprache $L_n = \{x\}$ der Länge $|x| = n$, die von einem DFA mit $n+1$ Zuständen akzeptiert wird, und jeder $k$C-OCA$_t$ für $L_n$ benötigt mindestens $m$ Zustände, wobei gilt:

$$m + 1 \geq \sqrt{\frac{n}{(\log n)(|\Sigma| + 1)}}$$

Im folgenden werden wir ein Inkompressibilitätsargument verwenden. Allgemeine Informationen über Kolmogorowkomplexität und die Inkompressibilitätsmethode finden sich in Kapitel 2.8 und [LV93].

Es sei $L_n$ eine einelementige Menge der Länge $n$ und $A$ ein $k$C-OCA$_t$, der die Sprache $L_n$ akzeptiert. Dann ist $C(L_n|n)$ als die minimale Größe eines Programms definiert, das die Sprache $L_n$ beschreibt unter Kenntnis der Länge $n$. Die Größe dieser minimalen Beschreibung ist offenbar kleiner oder gleich der Größe irgendeiner Beschreibung von $L_n$. Wir betrachten hier eine Beschreibung von $L_n$ durch einen $k$C-OCA$_t$ $A$, der gerade die Sprache $L_n$ akzeptiert. Die Größe dieser Beschreibung läßt sich abschätzen durch die Größe einer Kodierung $cod(A)$ von $A$ und die Größe $|P|$ eines Programms $P$, das beschreibt, wie ein $k$C-OCA$_t$ kodiert ist und wie ein $k$C-OCA$_t$ eine Sprache akzeptiert. Also ist $C(L_n|n) \leq |cod(A)| + |P|$. Die Größe des Programms $P$ hängt offenbar nicht von der Sprache $L_n$, dem Automaten $A$ und der Länge $n$ ab. Eine Kodierung $cod(A)$ von $A$ besteht aus Kodierungen $cod(\delta)$ und $cod(\delta_r)$ der Überführungsfunktionen, einer Kodierung $cod(F)$ der akzeptierenden Zustände und einer Kodierung $cod(k)$ von $k$.

Wir wählen nun ein $n \in \mathbb{N}$ mit $|P| \leq (1/8)n$ und $\log k \leq (1/8)n$. Weiterhin wählen wir ein inkompressibles Wort $x$ der Länge $n$ über einem Alphabet mit mindestens zwei Symbolen, und für $x$ gelte $C(x|n) \geq n$. Ein solches Wort $x$ existiert nach Satz 2.30. Wir betrachten nun die Sprache $L_n = \{x\}$, für die $C(L_n|n) \geq n$ gilt.

Für einen Widerspruchsbeweis nehmen wir an, daß es einen $k$C-OCA$_t$ $A$ gibt, der die Sprache $L_n$ mit $m$ Zuständen akzeptiert, wobei gilt:

$$m + 1 < \sqrt{\frac{n}{(\log n)(|\Sigma| + 1)}}$$

Zunächst benötigen wir die folgenden beiden Ungleichungen:

$$(m+1)^2 \log m \leq (1/3)(m+1)^2(|\Sigma| + 1) \log m, \tag{6.16}$$

$$(m+1) \log m \leq (1/6)(m+1)^2(|\Sigma| + 1) \log m \tag{6.17}$$

Es ist $|\Sigma| \geq 2$. Daraus folgt $(1/3)(|\Sigma| + 1) \geq 1$ und damit (6.16). Da $m \geq 1$ und damit $(1/2)(m+1) \geq 1$ ist, erhalten wir $(1/6)(m+1)(|\Sigma| + 1) \geq 1$ und damit (6.17).

Dann können wir $|cod(A)| + |P|$ folgendermaßen abschätzen:

$$\begin{aligned}
|cod(A)| + |P| &\leq \underbrace{\log m^{m^2}}_{|cod(\delta)|} + \underbrace{\log m^{(m+1)m(|\Sigma|+1)}}_{|cod(\delta_r)|} + \underbrace{m \log m}_{|cod(F)|} + \underbrace{\log k}_{|cod(k)|} + |P| \\
&\leq m^2 \log m + (m+1)m(|\Sigma|+1)\log m + m \log m + \frac{1}{4}n \\
&\leq (m+1)^2 \log m + (m+1)^2(|\Sigma|+1)\log m + (m+1)\log m + \frac{1}{4}n \\
&\leq \left(\frac{1}{3} + 1 + \frac{1}{6}\right)(m+1)^2(|\Sigma|+1)\log m + \frac{1}{4}n \\
&= \frac{3}{2}(m+1)^2(|\Sigma|+1)\log m + \frac{1}{4}n \\
&< \frac{3}{2}\frac{n}{(\log n)(|\Sigma|+1)}(|\Sigma|+1)\log\left(\left(\frac{n}{(\log n)(|\Sigma|+1)}\right)^{1/2}\right) + \frac{1}{4}n \\
&= \frac{3}{4}\frac{n}{\log n}\log\left(\frac{n}{(\log n)(|\Sigma|+1)}\right) + \frac{1}{4}n \\
&= \frac{3}{4}\frac{n}{\log n}\left(\log n - \log\log n - \log(|\Sigma|+1)\right) + \frac{1}{4}n \\
&= \frac{3}{4}n\left(1 - \frac{\log\log n}{\log n} - \frac{\log(|\Sigma|+1)}{\log n}\right) + \frac{1}{4}n \\
&\leq \frac{3}{4}n + \frac{1}{4}n = n
\end{aligned}$$

Damit gilt $C(L_n|n) \leq |cod(A)| + |P| < n$. Das ist ein Widerspruch zur Inkompressibilität von $L_n$, da $C(L_n|n) \geq n$ gilt.

Für $k$C-CA ist die Vorgehensweise ähnlich. Zum Widerspruchbeweis nehmen wir an, daß es einen $k$C-OCA$_t$ $A$ gibt, der die Sprache $L_n$ mit $m$ Zuständen akzeptiert, wobei gilt:

$$m + 1 < \sqrt[3]{\frac{n}{(\log n)(|\Sigma|+1)}}$$

Die folgenden Ungleichungen folgen wie vorhin unmittelbar aus $|\Sigma| \geq 2$ und $m \geq 1$.

$$\begin{aligned}
(m+1)^3 \log m &\leq (1/3)(m+1)^3(|\Sigma|+1)\log m, \\
(m+1)\log m &\leq (1/6)(m+1)^3(|\Sigma|+1)\log m, \\
(m+1)^2(|\Sigma|+1)\log m &\leq (m+1)^3(|\Sigma|+1)\log m
\end{aligned}$$

Wiederum können wir $|cod(A)| + |P|$ abschätzen:

$$\begin{aligned}
|cod(A)| + |P| &\leq \underbrace{\log m^{(m+1)m^2}}_{|cod(\delta)|} + \underbrace{\log m^{(m+1)m(|\Sigma|+1)}}_{|cod(\delta_r)|} + \underbrace{m \log m}_{|cod(F)|} + \underbrace{\log k}_{|cod(k)|} + |P| \\
&\leq (m+1)m^2 \log m + (m+1)m(|\Sigma|+1)\log m + m \log m + \frac{1}{4}n
\end{aligned}$$

$$
\begin{aligned}
&\leq (m+1)^3 \log m + (m+1)^3(|\Sigma|+1)\log m + (m+1)\log m + \frac{1}{4}n \\
&\leq \left(\frac{1}{3}+1+\frac{1}{6}\right)(m+1)^3(|\Sigma|+1)\log m + \frac{1}{4}n \\
&= \frac{3}{2}(m+1)^3(|\Sigma|+1)\log m + \frac{1}{4}n \\
&< \frac{3}{2}\frac{n}{(\log n)(|\Sigma|+1)}(|\Sigma|+1)\log\left(\left(\frac{n}{(\log n)(|\Sigma|+1)}\right)^{1/3}\right) + \frac{1}{4}n \\
&= \frac{1}{2}\frac{n}{\log n}\log\left(\frac{n}{(\log n)(|\Sigma|+1)}\right) + \frac{1}{4}n \\
&= \frac{1}{2}\frac{n}{\log n}\left(\log n - \log\log n - \log(|\Sigma|+1)\right) + \frac{1}{4}n \\
&= \frac{1}{2}n\left(1 - \frac{\log\log n}{\log n} - \frac{\log(|\Sigma|+1)}{\log n}\right) + \frac{1}{4}n \\
&\leq \frac{1}{2}n + \frac{1}{4}n < n
\end{aligned}
$$

Damit gilt $C(L_n|n) \leq |cod(A)| + |P| < n$. Das ist ein Widerspruch zu $C(L_n|n) \geq n$. □

**Bemerkung 6.21.** Wie in Lemma 6.12 gilt auch hier, daß die untere Schranke nicht von der Anzahl der Zellen abhängt. Das heißt, daß es zu jeder vorgegebenen Anzahl von Zellen reguläre Sprachen gibt, so daß im Vergleich mit einem DFA ein $k$C-CA höchstens kubische Einsparungen erreichen kann. Also gibt es auch für $k$C-CA und $k$C-OCA$_t$ inhärent sequentielle Sprachen. Es ist derzeit offen, ob die Konstruktion für die obere Schranke aus Lemma 6.18 für $k$C-CA auf $O(\sqrt[3]{n})$ verbessert werden kann. Ebenfalls ist offen, ob alternativ der Beweis der unteren Schranke für $k$C-CA auf $\Omega(\sqrt{n})$ verbessert werden kann.

### 6.2.4 Umwandlungen zwischen $k$C-OCA, $k$C-OCA$_t$ und $k$C-CA

Wir haben mit $k$C-OCA, $k$C-OCA$_t$ und $k$C-CA bislang drei eingeschränkte Zellularautomatenmodelle betrachtet, die mit keiner, minimaler und maximaler zweiseitiger Kommunikation versehen sind. Alle Modelle beschreiben die regulären Sprachen. Aus Sicht der Beschreibungskomplexität ist es nun interessant zu untersuchen, wie sich die Anzahl der Zustände verändert, wenn die Anzahl der Zellen mit zweiseitiger Kommunikation reduziert oder erweitert wird. In diesem Abschnitt werden wir daher obere und untere Schranken bei den Umwandlungen zwischen $k$C-OCA, $k$C-OCA$_t$ und $k$C-CA untersuchen.

Zunächst kann ein Modell mit weniger zweiseitiger Kommunikation in ein Modell mit mehr zweiseitiger Kommunikation eingebettet werden, ohne die Anzahl der Zustände zu erhöhen.

**Lemma 6.22.** Jeder $k$C-OCA mit $n$ Zuständen kann in einen äquivalenten $k$C-CA oder $k$C-OCA$_t$ mit $n$ Zuständen umgewandelt werden. Jeder $k$C-OCA$_t$ mit $n$ Zuständen kann in einen äquivalenten $k$C-CA mit $n$ Zuständen umgewandelt werden. Diese oberen Schranken sind scharf.

*Beweis.* Offenbar kann ein $k$C-CA oder $k$C-OCA$_t$ einen $k$C-OCA simulieren, ohne die Anzahl der Zustände zu erhöhen. Ebenso kann ein $k$C-CA einen $k$C-OCA$_t$ simulieren. In Lemma 6.9 wird gezeigt, daß jeder $k$C-OCA, $k$C-OCA$_t$ oder $k$C-CA für die Sprache $L_{n,k}$ mindestens $n+1$

Zustände benötigt, da $n^k + 1$ Konfigurationen mit $k$ Zellen unterschieden werden müssen. Damit sind die Behauptungen bewiesen. □

Bei der Reduzierung von Zellen mit zweiseitiger Kommunikation ergeben sich folgende oberen Schranken:

**Lemma 6.23.**

(1) Jeder $k$C-CA mit $n$ Zuständen kann in einen $k$C-OCA mit höchstens $O(n^k)$ Zuständen umgewandelt werden.

(2) Jeder $k$C-OCA$_t$ mit $n$ Zuständen kann in einen $k$C-OCA mit höchstens $n^2 + n$ Zuständen umgewandelt werden.

(3) Jeder $k$C-CA mit $n$ Zuständen kann in einen $k$C-OCA$_t$ mit höchstens $n^{\lceil k/2 \rceil} + 3$ Zuständen umgewandelt werden.

*Beweis.* (1) Ein $k$C-CA mit $n$ Zuständen kann nach Lemma 6.6 in einen DFA mit $O(n^k)$ Zuständen umgewandelt werden. Dieser DFA kann dann mit Lemma 6.11 in einen $k$C-OCA mit $O(n^k)$ Zuständen umgewandelt werden.

(2) Es sei $A$ ein $k$C-OCA$_t$ mit $n$ Zuständen. Wir konstruieren einen $k$C-OCA $A'$, der im wesentlichen identisch zu $A$ ist mit folgender Ausnahme: Da die Kommunikationszelle keine Informationen von links bekommen kann, muß zusätzlich die vorletzte Zelle in der Kommunikationszelle simuliert werden. Daher betrachten wir dort das Kreuzprodukt zweier Zellen. Eine Skizze der Konstruktion findet man in Abbildung 6.7. $A'$ arbeitet nun wie der gegebene Automat $A$; nur in der ersten Komponente der Kommunikationszelle wird zusätzlich die vorletzte Zelle von $A$ simuliert. Formal lautet die Konstruktion: Es sei $A = (Q, \#, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$ der gegebene $k$C-OCA$_t$. Wir definieren einen $k$C-OCA $A' = (Q', \Sigma, q_0, \triangledown, k, \delta'_r, \delta', F)$, wobei $Q' = Q \cup Q \times Q$ ist und für alle $q, q', q'' \in Q$ und $\sigma \in \Sigma$ gilt:

$$\begin{aligned}
\delta'_r(q_0, \sigma) &= (q_0, \delta_r(q_0, q_0, \sigma)) \\
\delta'_r((q, q'), \sigma) &= (\delta(q, q'), \delta_r(q, q', \sigma)) \\
\delta'(q, q') &= \delta(q, q') \\
\delta'(q, (q', q'')) &= \delta(q, q'')
\end{aligned}$$

Nach den obigen Überlegungen kann man leicht einsehen, daß $T(A') = T(A)$ gilt. Offenbar ist $|A'| \leq n^2 + n$.

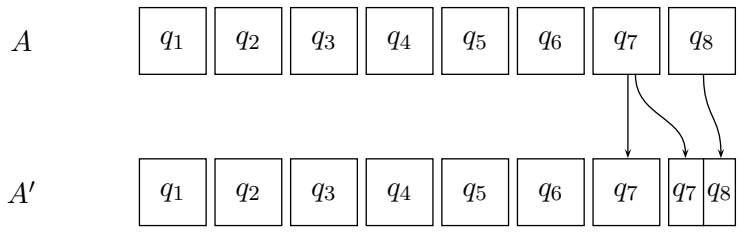



Abbildung 6.7: Simulieren eines $k$C-OCA$_t$ durch einen $k$C-OCA

(3) Es sei $A$ ein $k$C-CA mit $n$ Zuständen. Ein $k$C-OCA$_t$ $A'$ muß nun $k$ Zellen in den beiden letzten Zellen simulieren. Daher betrachten wir das Kreuzprodukt von $\lceil k/2 \rceil$ Zellen von $A$ in jeder Zelle in $A'$. In der Kommunikationszelle von $A'$ werden die letzten $\lceil k/2 \rceil$ Zellen simuliert und in der vorletzten Zelle die ersten $k - \lceil k/2 \rceil$ Zellen. Eine Skizze der Konstruktion findet man in Abbildung 6.8. Die beiden letzten Zellen von $A'$ simulieren also den gegebenen $k$C-CA $A$. Um eine korrekte Akzeptierung zu gewährleisten, müssen wir zwei Fälle unterscheiden: Sobald die linke Randzelle in $A$, die in der ersten Teilzelle der vorletzten Zelle in $A$ simuliert wird, einen akzeptierenden Zustand aus $A$ annimmt, wird mit maximaler Geschwindigkeit ein akzeptierender Zustand $g$ nach links gesendet. Sobald die Kommunikationszelle zum ersten Mal das Eingabeendesymbol liest, wird von $A'$ geprüft, ob die aktuelle Konfiguration von $A$, die ja in den beiden letzten Zellen kodiert ist, in den nächsten $k$ Zeitschritten zu einer Akzeptierung in $A$ führt. Ist dies der Fall, dann wird mit maximaler Geschwindigkeit ein akzeptierender Zustand $g$ nach links gesendet. Anderenfalls wird die Berechnung blockiert.

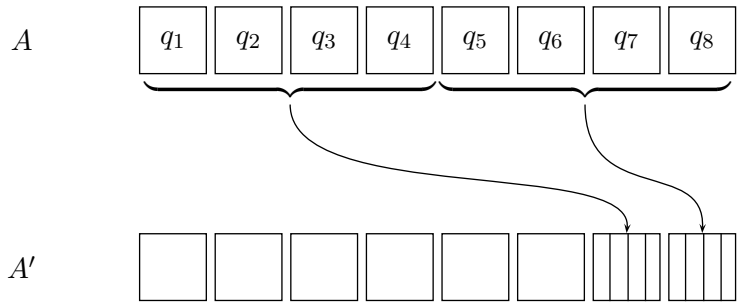



Abbildung 6.8: Simulieren eines $k$C-CA durch einen $k$C-OCA$_t$

Es sei $A = (Q, \#, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$ der gegebene $k$C-CA und $l = \lceil k/2 \rceil$. Wir definieren einen $k$C-OCA$_t$ $A' = (Q', \#, \Sigma, q_0, \triangledown, k, \delta'_r, \delta', \{g\})$, wobei $Q' = Q^l \cup \{b, g, q_0\}$ ist. Im folgenden bezeichnen wir mit $\pi_i : Q^l \to Q$ $(1 \leq i \leq l)$ die Projektion auf die $i$-te Kompenente eines Vektors aus $Q^l$.

Zunächst betrachten wir die Konstruktion für gerades $k$. In diesem Fall ist $k/2$ eine ganze Zahl und wir können $k/2$ Zellen aus $A$ jeweils in der vorletzten Zelle und letzten Zelle in $A'$ simulieren. Im folgenden seien $p, q, q' \in Q^l$ und $\sigma \in \Sigma$. Die Initialisierung von $A'$ mit Vektoren aus $Q^l$ geschieht mit folgenden Regeln:

$$\delta'_r(q_0, q_0, \sigma) = q \text{ mit } \pi_l(q) = \delta_r(q_0, q_0, \sigma) \text{ und } \pi_j(q) = q_0 \text{ für } j < l \tag{6.18}$$

$$\delta'(q_0, q) = p \text{ mit } \pi_l(p) = \delta(q_0, q_0, \pi_1(q)) \text{ und } \pi_j(p) = q_0 \text{ für } j < l \tag{6.19}$$

Falls in der ersten Teilzelle der vorletzten Zelle ein akzeptierender Zustand aus $F$ erreicht wird, dann wurde in der ersten Zelle des zu simulierenden Automaten $A$ ein akzeptierender Zustand erreicht und die Eingabe dort akzeptiert. Daher senden wir in diesem Fall einen akzeptierenden Zustand $g$ mit maximaler Geschwindigkeit nach links. Dazu dienen folgende

Regeln:

$$\delta'_r(q,q',\sigma) = g \text{ falls } \pi_1(q) \in F \tag{6.20}$$
$$\delta'_r(q,g,\sigma) = g \tag{6.21}$$
$$\delta'_r(q,g,\triangledown) = g \tag{6.22}$$
$$\delta'(q,g) = g \tag{6.23}$$

Wird in der ersten Teilzelle der vorletzten Zelle kein akzeptierender Zustand aus $F$ erreicht, dann wird die Berechnung von $A$ mit folgenden Regeln simuliert:

$$\delta'_r(q,q',\sigma) = p \text{ mit } \begin{cases} \pi_1(p)=\delta(\pi_l(q),\pi_1(q'),\pi_2(q')) \\ \pi_j(p)=\delta(\pi_{j-1}(q'),\pi_j(q'),\pi_{j+1}(q')) \text{ für } 2 \le j < l \\ \pi_l(p)=\delta_r(\pi_{l-1}(q'),\pi_l(q'),\sigma) \end{cases} \tag{6.24}$$

$$\delta'(q,q') = p \text{ mit } \begin{cases} \pi_1(p)=\delta(\#,\pi_1(q),\pi_2(q)) \\ \pi_j(p)=\delta(\pi_{j-1}(q),\pi_j(q),\pi_{j+1}(q)) \text{ für } 2 \le j < l \\ \pi_l(p)=\delta(\pi_{l-1}(q),\pi_l(q),\pi_1(q')) \end{cases} \tag{6.25}$$

Ist die komplette Eingabe gelesen, das heißt $\triangledown$ ist das nächste Eingabesymbol, dann haben wir einerseits zu prüfen, ob in der ersten Teilzelle der vorletzten Zelle ein akzeptierender Zustand aus $F$ errechnet wurde. Ist das der Fall, wird der akzeptierende Zustand $g$ mit maximaler Geschwindigkeit nach links geschickt. Andererseits müssen wir prüfen, ob die aktuelle Konfiguration $c = (\pi_1(q),\ldots,\pi_l(q),\pi_1(q'),\ldots,\pi_l(q'))$, die in den beiden letzten Zellen kodiert ist, in den nächsten $k$ Schritten zu einer Akzeptierung in $A$ führt. Trifft das zu, senden wir ebenfalls den akzeptierenden Zustand $g$ mit maximaler Geschwindigkeit nach links. Treffen beide Fälle nicht zu, dann wird die Eingabe in $A$ nicht akzeptiert. Daher gehen wir in einen nichtakzeptierenden Zustand $b$ über, der mit maximaler Geschwindigkeit nach links geschickt wird und sicherstellt, daß die Eingabe nicht akzeptiert wird. Im einzelnen haben wir folgende Regeln:

$$\delta'_r(q,q',\triangledown) = \begin{cases} g & \text{falls } \exists 1 \le j \le k : c_j(1) \in F \text{ für } (c_j,w_j) = \Delta^j(c,\triangledown^k) \\ & \text{oder } \pi_1(q) \in F \\ b & \text{sonst} \end{cases} \tag{6.26}$$
$$\delta'_r(q,b,\sigma) = b \tag{6.27}$$
$$\delta'_r(q,b,\triangledown) = b \tag{6.28}$$
$$\delta'(q,b) = b \tag{6.29}$$

Wir betrachten nun die Konstruktion für ungerades $k$. In diesem Fall werden in der letzten Zelle $\lceil k/2 \rceil$ Zellen aus $A$ simuliert und in der vorletzten Zelle $k - \lceil k/2 \rceil = \lfloor k/2 \rfloor$ Zellen aus $A$. Also steht in der vorletzten Zelle in $A'$ eine Teilzelle zuviel zur Verfügung. Daher werden wir die erste Teilzelle in der vorletzten Zelle in $A'$ immer mit $q_0$ belegen und die erste Zelle aus $A$ in der zweiten Teilzelle der vorletzten Zelle in $A'$ simulieren. Die Regeln (6.20), (6.25) und (6.26) müssen daher für $p,q,q' \in Q^l$, $\sigma \in \Sigma$ und die Konfiguration $c = (\pi_2(q),\ldots,\pi_l(q),\pi_1(q'),\ldots,\pi_l(q'))$ folgendermaßen angepaßt werden:

$$\delta'_r(q,q',\sigma) = g \text{ falls } \pi_2(q) \in F \tag{6.30}$$

$$\delta'(q,q') = p \text{ mit } \begin{cases} \pi_1(p)=q_0 \\ \pi_2(p)=\delta(\#,\pi_2(q),\pi_3(q)) \\ \pi_j(p)=\delta(\pi_{j-1}(q),\pi_j(q),\pi_{j+1}(q)) \text{ für } 3 \le j < l \\ \pi_l(p)=\delta(\pi_{l-1}(q),\pi_l(q),\pi_1(q')) \end{cases} \tag{6.31}$$

$$\delta'_r(q,q',\triangledown) = \begin{cases} g & \text{falls } \exists 1 \le j \le k : c_j(1) \in F \text{ für } (c_j,w_j) = \Delta^j(c,\triangledown^k) \\ & \text{oder } \pi_2(q) \in F \\ b & \text{sonst} \end{cases} \tag{6.32}$$

Insgesamt erhalten wir $T(A') = T(A)$. Für die Anzahl der Zustände in $A'$ gilt: $|A'| \le n^{\lceil k/2 \rceil}+3$. □

Wir werden nun zeigen, daß die in Lemma 6.23 bewiesenen oberen Schranken asymptotisch scharf sind. Dazu betrachten wir die Sprachen $L'_p = \{a^n \mid n \ge 0 \text{ und } n \equiv 0 \mod p\}$, wobei $p$ eine Primzahl ist. Ein ähnlicher Beweis wie in Lemma 6.12 zeigt, daß jeder $k$C-OCA für $L'_p$ mindestens $p$ Zustände benötigt. Als Anwendung einer allgemeinen Konstruktion erhalten wir einen $k$C-CA für $L'_p$ mit $O(\sqrt[k]{p})$ Zuständen und bekommen damit einen polynomiellen Größenzuwachs vom Grad $k$ zwischen $k$C-CA und $k$C-OCA. Die restlichen unteren Schranken lassen sich dann aus diesem Ergebnis ableiten.

**Lemma 6.24.** Jeder $k$C-OCA für $L'_p$ benötigt mindestens $p$ Zustände.

*Beweis.* Wir haben in Lemma 6.12 gezeigt, daß jeder $k$C-OCA für die Sprache $L_p = L'_p \cdot \{a\}$ mindestens $p+1$ Zustände benötigt. Der erste Teil des Beweises wird nun etwas modifiziert und wir können zeigen, daß jeder $k$C-OCA für $L'_p$ mindestens $p$ Zustände benötigt.

Es sei $A$ ein $k$C-OCA, der die Sprache $L'_p$ mit $n$ Zuständen akzeptiert. Wiederum betrachten wir die Folgen $s_i = (\pi_i(\Delta^t(c_0,a^t)))_{t\ge 0}$, die die Entwicklung der Zustände der letzten $k-i+1$ Zellen des $k$C-OCA $A$ unter entsprechenden Eingaben $a^t$ darstellen. Mit $l_i$ bezeichnen wir die Länge der Periode zwischen dem ersten Auftreten zweier identischer Elemente in $s_i$. Es gilt: $l_1 = p_1 p_2 \dots p_k$ mit $1 \le p_i \le n$ für $1 \le i \le k$ und $l_1$ ist die Länge der „Periode von $A$“. Das heißt, $\Delta^{l_1}(c,a^m) = (c,a^{m-l_1})$ für jede Konfiguration $(c,a^m)$ mit $m \ge l_1$.

Wir nehmen an, daß $n < p$ ist. Dann folgt unmittelbar, daß $p$ nicht $l_1 = p_1 p_2 \dots p_k$ teilt, da $p_i \le n < p$ für $1 \le i \le k$ gilt und $p$ eine Primzahl ist. Wir wählen jetzt eine ganze Zahl $t'$ so, daß $t'p > n^k + 1$ gilt. Nun ist $a^{t'p} \in L'_p$. Also ist $\Delta^{t'p}(c_0,a^{t'p}\triangledown^k) = (c',\triangledown^k)$ und es gibt eine ganze Zahl $k'$ mit $1 \le k' \le k$, so daß $\Delta^{k'}(c',\triangledown^{k'}) = (c'',\epsilon)$ und $c''(1) \in F$ gilt. Wir betrachten nun die Eingabe $w = a^{t'p+l_1}$. Dann ist $\Delta^{t'p}(c_0,w\triangledown^k) = (c',a^{l_1}\triangledown^k)$, $\Delta^{l_1}(c',a^{l_1}\triangledown^k) = (c',\triangledown^k)$ und $\Delta^{k'}(c',\triangledown^{k'}) = (c'',\epsilon)$. Dann ist $w \in L'_p$ und daher gilt: $t'p+l_1 = t''p$ mit einem $t'' \ge 1$. Also ist $t'p + l_1$ ein Vielfaches von $p$. Daher teilt $p$ die Zahl $l_1 = p_1 p_2 \dots p_k$, was ein Widerspruch ist. □

Wir wissen aus Lemma 6.16 und Lemma 6.17, daß ein $k$C-CA gegenüber einem DFA polynomielle Einsparungen vom Grad $k$ erzielen kann. Andererseits haben wir in Lemma 6.20 auch gezeigt, daß es Sprachen über einem Alphabet mit mindestens zwei Symbolen gibt, die schwer zu parallelisieren sind. Hier betrachten wir nun unäre Sprachen, also Sprachen,

die über einem Alphabet mit einem Symbol definiert sind. Da unäre DFA eine besonders einfache Struktur besitzen, können wir zeigen, daß $k$C-CA im unären Fall DFA oftmals mit polynomiellen Einsparungen vom Grad $k$ simulieren können.

In einem unären DFA besitzt jeder Zustand höchstens eine Auskante. Daher kann ein unärer DFA auch nur höchstens eine Schleife besitzen. Wir bezeichnen mit $n_1$ die Länge der Vorperiode, das heißt, $n_1$ ist die Anzahl der Zustände, die sich außerhalb der Schleife befinden. Mit $n_2$ wird die Länge der Periode, also die Anzahl der Zustände innerhalb der Schleife bezeichnet. In folgendem Beispiel 6.25 ist ein unärer DFA mit Vorperiode $n_1 = 3$ und der Periode $n_2 = 6$ abgebildet.

**Beispiel 6.25.**

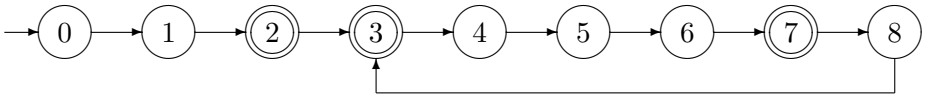



Abbildung 6.9: Ein unärer DFA mit Vorperiode $n_1 = 3$ und Periode $n_2 = 6$.

**Satz 6.26.** Für $k \geq 1$ gilt: Jeder unäre DFA mit Vorperiode $n_1 \geq 0$ und Periode $n_2 \geq 2k - 1$ kann in einen äquivalenten $k$C-CA mit $O(\sqrt[k]{n_1} + \sqrt[k]{n_2} + |F|)$ Zuständen umgewandelt werden.

*Beweis.* Es sei $M = (Q, \Sigma, q_0, \delta, F)$ ein unärer DFA mit Vorperiode $n_1$ und Periode $n_2$.

Zunächst skizzieren wir die grobe Idee der Konstruktion. Wir definieren $l_1 = \lceil \sqrt[k]{n_1} \rceil$ und $l_2 = \lceil \sqrt[k]{n_2} \rceil$. Um die Anzahl der Zustände außerhalb der Schleife zu zählen, starten wir einen $l_1$-nären Zähler und prüfen in jedem Zeitschritt von rechts nach links sukzessive, ob die $l_1$-näre Kodierung von $n_1$ gezählt wurde. Ist dies der Fall, senden wir ein Signal nach rechts, das einen $l_2$-nären Zähler initialisiert. Ist das Signal an der Kommunikationszelle angekommen, zählen wir die Anzahl der Zustände in der Schleife, indem wir den Zähler starten und in jedem Zeitschritt von rechts nach links sukzessive prüfen, ob die $l_2$-näre Kodierung von $n_2$ gezählt wurde. Ist das der Fall, wird wiederum ein Signal nach rechts gesendet, das den Zähler geeignet initialisiert und das nächste Zählen von $n_2$ beginnt. Sobald die Kommunikationszelle das Eingabeendesymbol liest, überprüfen wir, ob in den Zellen ein akzeptierender Zustand kodiert ist und akzeptieren gegebenenfalls die Eingabe.

Es sei $A_1$ bzw. $A_2$ ein $k$C-OCA, der einen $l_1$-nären bzw. $l_2$-nären Zähler simuliert. Eine Konstruktion findet man in Beispiel 6.5. Mit $d_t^1(j)$ bzw. $d_t^2(j)$ bezeichnen wir den Zustand der $j$-ten Zelle in $A_1$ bzw. $A_2$, nachdem die Eingabe $a^t$ gelesen wurde. Der zu konstruierende $k$C-CA $A$ ist in zwei Spuren aufgeteilt. In der ersten Spur werden Zähler realisiert. Die zweite Spur dient zum Senden von Signalen. Mit $c_t^1(j)$ bzw. $c_t^2(j)$ bezeichnen wir den Zustand der ersten bzw. zweiten Spur der $j$-ten Zelle des $k$C-CA $A$ zum Zeitpunkt $t$.

1. Schritt:

Wir starten in der ersten Spur einen $l_1$-nären Zähler und wollen überprüfen, ob die $l_1$-näre Kodierung von $n_1$ bereits gezählt wurde. Dazu starten wir in jedem Zeitschritt in der zweiten

Spur ein Signal $L$ in der Kommunikationszelle, das sukzessive überprüft, ob die $l_1$-näre Kodierung von $n_1$ in den Zellen vorhanden ist. Formal überprüfen wir die folgenden Gleichungen:

$$\begin{aligned} c_t^1(k) &= d_{n_1}^1(k) \\ c_{t+1}^1(k-1) &= d_{n_1+1}^1(k-1) \\ \dots &\quad \dots \\ c_{t+k-2}^1(2) &= d_{n_1+k-2}^1(2) \\ c_{t+k-1}^1(1) &= d_{n_1+k-1}^1(1) \end{aligned}$$

Falls eine dieser Gleichungen nicht gilt, wird das Signal gestoppt. Gelten alle Gleichungen, dann erreicht das Signal $L$ zum Zeitpunkt $n_1 + k - 1$ die linke Randzelle. Wir starten dann ein Signal $R$ mit maximaler Geschwindigkeit von links nach rechts. Eine Beispielberechnung für $k = 3$, $n_1 = 3$ und $l_1 = 2$ findet man in Abbildung 6.10

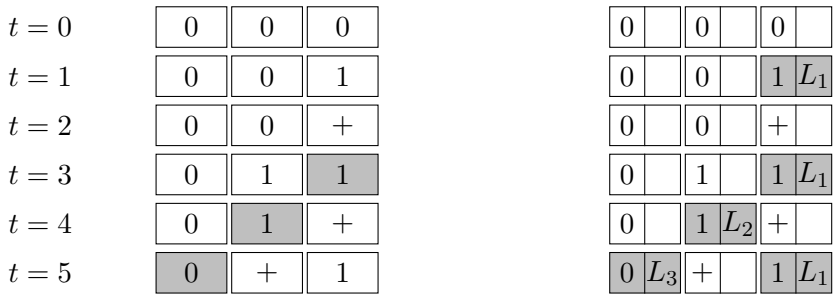



Abbildung 6.10: Binäre Kodierung von $n_1 = 3$ (links) und Zustandsdiagramm für den 1. Schritt (rechts)

2. Schritt:

Das Signal $R$ erreicht die Kommunikationszelle zum Zeitpunkt $n_1 + k - 1 + k = n_1 + 2k - 1$. Wir haben zu diesem Zeitpunkt also bereits $2k - 1$ Zustände der Periode gezählt. Daher wird das Signal $R$ den Zähler mit der $l_2$-nären Kodierung von $2k - 1$ initialisieren. Das geschieht folgendermaßen: Sei $a_1 a_2 \dots a_k$ die $l_2$-näre Kodierung von $2k - 1$. Das Signal $R$ wandert mit maximaler Geschwindigkeit nach rechts, das heißt, nach $j$ Zeitschritten wird die $j$-te Zelle erreicht. Sobald das Signal $R$ die $j$-te Zelle erreicht, ändern wir den Zustand dahingehend, daß in $k - j$ Zeitschritten der Zustand $a_j$ angenommen wird. Für $1 \leq j \leq k$ gilt also: Nach $j + k - j = k$ Zeitschritten ist die $j$-te Zelle im Zustand $a_j$. Für eine Beispielberechnung betrachten wir $k = 3$ und $l_2 = 2$: Dann ist $2k - 1 = 5$ und die binäre Kodierung ist $a_1 a_2 a_3 = 0 + 1$. Eine Beispielberechnung findet man in Abbildung 6.11.

3. Schritt:

Sobald das Signal $R$ die Kommunikationszelle erreicht, ist der $l_2$-näre Zähler mit $2k - 1$ initialisiert. Wir starten den Zähler und müssen wiederum überprüfen, ob die $l_2$-näre Kodierung von $n_2$ gezählt wurde. Ähnlich dem Vorgehen im 1. Schritt starten wir in jedem Zeitschritt ein Signal $L'$, das sukzessive überprüft, ob in den Zellen die $l_2$-näre Kodierung von $n_2$ steht.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $t=5$ | 0 | $L_3$ | + | | 1 | $L_1$ | | | | | |
| $t=6$ | $0_2$ | $R_1$ | 0 | | + | | | | | | |
| $t=7$ | $0_1$ | $R_1$ | $+_1$ | $R_2$ | 1 | $L_1$ | | | | | |
| $t=8$ | $0'$ | | $+'$ | | $1'$ | | | | | | |
| $t=9$ | $1'$ | | $0'$ | | $+'$ | | | | | | |

Abbildung 6.11: Initialisierung mit der binären Kodierung von $2k-1=5$ zum Zeitpunkt $t = n_1 + 2k - 1 = 8$

Das heißt, es werden folgende Gleichungen überprüft:

$$\begin{aligned} c^1_t(k) &= d^2_{n_2}(k) \\ c^1_{t+1}(k-1) &= d^2_{n_2+1}(k-1) \\ \dots &\quad \dots \\ c^1_{t+k-2}(2) &= d^2_{n_2+k-2}(2) \\ c^1_{t+k-1}(1) &= d^2_{n_2+k-1}(1) \end{aligned}$$

Falls eine dieser Gleichungen nicht gilt, wird das Signal gestoppt. Gelten alle Gleichungen, dann erreicht das Signal $L'$ die linke Randzelle, nachdem $n_2 + k - 1$ gezählt wurde. Wir starten dann ein Signal $R'$ mit maximaler Geschwindigkeit von links nach rechts, das den Zähler mit der $l_2$-nären Kodierung von $2k-1$ initialisiert. Die Konstruktion hier ist ähnlich zur Konstruktion im 2. Schritt. Eine Beispielberechnung findet man in Abbildung 6.12.

| | | | |
|---|---|---|---|
| $t=5$ | 0 | + | 1 |
| $t=6$ | 1 | 0 | + |
| $t=7$ | 1 | 1 | 1 |
| $t=8$ | 1 | 1 | + |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $t=8$ | $0'$ | | $+'$ | | $1'$ | | | | | | |
| $t=9$ | $1'$ | | $0'$ | | $+'$ | $L'_1$ | | | | | |
| $t=10$ | $1'$ | | $1'$ | $L'_2$ | $1'$ | | | | | | |
| $t=11$ | $1'$ | $L'_3$ | $1'$ | | $+'$ | $L'_1$ | | | | | |
| $t=12$ | $0'_2$ | $R'_1$ | $+'$ | | $1'$ | | | | | | |
| $t=13$ | $0'_1$ | $R'_1$ | $+'_1$ | $R'_2$ | $+'$ | $L'_1$ | | | | | |
| $t=14$ | $0'$ | | $+'$ | | $1'$ | | | | | | |

Abbildung 6.12: Binäre Kodierung von $n_2 = 6$ (links) und Zustandsdiagramm für den 3. Schritt (rechts)

4. Schritt:

Wir sind bislang davon ausgegangen, daß die Eingabe ausreichend lang war, also das Eingabeendesymbols $\triangledown$ noch nicht gelesen wurde. Wird das Symbol $\triangledown$ gelesen, dann müssen wir von rechts nach links überprüfen, ob in den Zellen ein akzeptierender Zustand kodiert ist. Im folgenden identifizieren wir jeden Zustand $f \in F$ mit der minimalen Anzahl $t \geq 0$, so daß nach

Lesen der Eingabe $a^t$ der Zustand $f$ in $M$ erreicht wird. Wir haben zwei Fälle zu unterscheiden. Zunächst nehmen wir an, daß der Zustand der Kommunikationszelle zum Zeitpunkt des erstmaligen Lesens des Eingabeendesymbols ein Zustand des $l_1$-nären Zählers für die Vorperiode ist. Diese Situation kann spätestens bis zum Zeitpunkt $n_1+2k-1$ eintreten. Sei $F_1 \subseteq F$ die Teilmenge der akzeptierenden Zustände in $M$, die nach weniger als $n_1+2k$ Schritten in $M$ angenommen werden können. Also ist $F_1 = \{f \mid 0 \leq f \leq n_1+2k-1 \text{ und } f \in T(M)\}$

Für jedes $f \in F_1$ betrachten wir die entsprechende Kodierung $cod(f) \in Q^k$ von rechts nach links, die aus den Zuständen der Diagonalen beginnend zum Zeitpunkt $f$ in der Kommunikationszelle und endend zum Zeitpunkt $f+k$ in der linken Randzelle besteht.

$$cod(f) = (c^1_f(k), c^1_{f+1}(k-1), \cdots, c^1_{f+k-1}(2), c^1_{f+k}(1))$$

Wir konstruieren nun einen DFA, der alle Kodierungen aus $F_1$ akzeptiert. Beim Lesen von $\triangledown$ wird nun dieser DFA simuliert, und in der linken Randzelle wird dann gegebenenfalls akzeptiert. Im zweiten Fall ist der Zustand der Kommunikationszelle beim erstmaligen Lesen des Eingabeendesymbols ein Element des $l_2$-nären Zählers. In diesem Fall betrachten wir die Menge $F_2 = \{f \mid n_1+2k \leq f \leq n_1+n_2+2k-1 \text{ und } f \in T(M)\}$ der akzeptierenden Zustände in der Periode, die von dem $l_2$-nären Zähler erreicht werden. Auch zu dieser Menge betrachten wir die entsprechenden Kodierungen von rechts nach links und konstruieren einen DFA, der alle Kodierungen aus $F_2$ akzeptiert. Analog zum ersten Fall wird beim Lesen von $\triangledown$ dieser DFA simuliert und in der linken Randzelle gegebenenfalls akzeptiert.

Wir haben nun noch die Anzahl der Zustände zu zählen, die für die Konstruktion benötigt werden. Die $l_1$-nären und $l_2$-nären Zähler können mit $l_1+1$ bzw. $l_2+1$ Zuständen realisiert werden. Kombiniert mit den Zählern benötigen die Signale $L$ und $L'$ jeweils $(l_1+1)k$ bzw. $(l_2+1)k$ Zustände und die Signale $R$ und $R'$ $(l_1+1)(k-1)^2$ bzw. $(l_2+1)(k-1)^2$ Zustände. Für die Überprüfung der akzeptierenden Zustände brauchen wir schließlich höchstens $|F_1|k+1+|F_2|k+1 \leq 2|F|k$ Zustände. Insgesamt ergibt sich also:

$$\begin{aligned} |A| &\leq (l_1+1)(1+k+(k-1)^2)+(l_2+1)(1+k+(k-1)^2)+2|F|k = \\ &= (l_1+l_2+2)(k^2-k+2)+2|F|k = O(l_1+l_2+|F|) \end{aligned}$$

□

**Beispiel 6.25.** (Fortsetzung)

Wir betrachten nun die obige Konstruktion für den unären DFA aus Beispiel 6.25 und $k=3$. Dann ist $l_1 = \lceil \sqrt[3]{3} \rceil = 2$ und $l_2 = \lceil \sqrt[3]{6} \rceil = 2$. In der ersten Spalte in Abbildung 6.14 ist ein binärer Zähler dargestellt, aus dem die binären Kodierungen von rechts nach links diagonal abgelesen werden können. Die binäre Kodierung von $n_1 = 3$ ist $(1,1,0)$ und die binäre Kodierung von $n_2 = 6$ ist $(+,1,1)$. Außerdem kann die binäre Kodierung von $2k-1 = 5$ waagrecht von links nach rechts abgelesen werden: $(0,+,1)$. Die Mengen $F_1$ und $F_2$ ergeben sich folgendermaßen:

$$\begin{aligned} F_1 &= \{f \mid 0 \leq f \leq 8 \text{ und } f \in T(M)\} = \{2,3,7\} \\ F_2 &= \{f \mid 9 \leq f \leq 14 \text{ und } f \in T(M)\} = \{9,13\} \end{aligned}$$

Die Kodierungen von rechts nach links können aus der zweiten Spalte in Abbildung 6.14 abgelesen werden: $cod(2) = (+,1,0)$, $cod(3) = (1,1,0)$, $cod(7) = (1,+',1')$, $cod(9) = (+',1',1')$

und $cod(13) = (+', +', 1')$. Ein DFA für $F_1$ bzw. $F_2$ ist in Abbildung 6.13 dargestellt. Beispielberechnungen für die Eingaben $a^2$, $a^9$ und $a^{13}$ finden sich in der dritten und vierten Spalte von Abbildung 6.14.

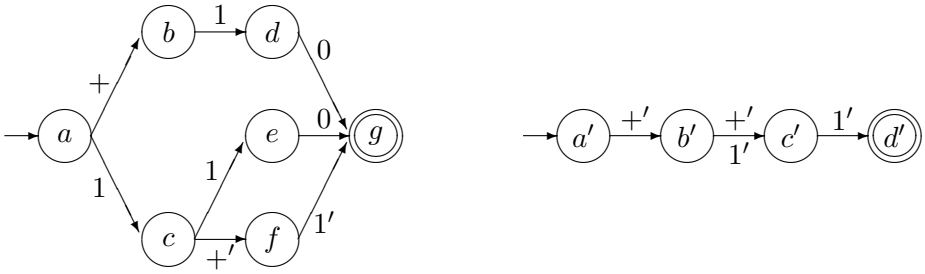



Abbildung 6.13: DFA für die Kodierungen der Mengen $F_1$ bzw. $F_2$

**Korollar 6.27.** Jede Sprache $L'_p$ kann von einem $k$C-CA mit $O(\sqrt[k]{p})$ Zuständen akzeptiert werden, sofern $p \geq 2k-1$ gilt.

*Beweis.* Zunächst konstruieren wir einen DFA $M_p$, der die Sprache $L'_p$ mit $p$ Zuständen akzeptiert. $M_p = (\{0, 1, \ldots, p-1\}, \{a\}, \delta, 0, \{0\})$ mit $\delta(i, a) = (i+1) \mod p$. $M_p$ akzeptiert offenbar die Sprache $L'_p$. Wir beobachten, daß $M_p$ keine Vorperiode und nur einen akzeptierenden Zustand besitzt. Also ist $n_1 = 0$, $n_2 = p$ und $|F| = 1$. Da $p \geq 2k-1$ ist, kann nach Satz 6.26 dann ein $k$C-CA für $L'_p$ mit $O(\sqrt[k]{n_1} + \sqrt[k]{n_2} + |F|) = O(\sqrt[k]{p} + 1) = O(\sqrt[k]{p})$ konstruiert werden. □

**Bemerkung 6.28.** Wir betrachten die Konstruktion im vierten Schritt aus Satz 6.26 für den DFA $M_p$ etwas genauer. Wir hatten dort jeweils einen DFA konstruiert, der alle Kodierungen aus $F_1$ bzw. $F_2$ akzeptiert. Da $M_p$ nur einen akzeptierenden Zustand besitzt, wird in obiger Konstruktion nur ein DFA $M$ konstruiert, der genau eine Kodierung aus $F_1$ oder $F_2$ akzeptiert. Damit gibt es also gerade einen akzeptierenden Zustand $f$ in $M$. Nach Lesen des Eingabeendesymbols wird im $k$C-CA $A$ nun dieser DFA $M$ simuliert. Für eine Eingabe $w$ können wir folgendes beobachten:

(1) Ist $w \in T(M_p)$, dann nimmt die linke Randzelle des $k$C-CA $A$ den akzeptierenden Zustand $f$ genau zum Zeitpunkt $|w| + k$ an.

(2) Weiterhin hat keine andere Zelle bis zum Zeitpunkt $|w|+k$ den Zustand $f$ angenommen.

(3) Ist $w \notin T(M_p)$, dann nimmt keine Zelle des $k$C-CA $A$ den Zustand $f$ an.

**Bemerkung 6.29.** Wenn im folgenden obiges Korollar zum Beweis unterer Schranken herangezogen wird, gehen wir implizit davon aus, daß es zu einer gegebenen Konstante $k \geq 1$ immer unendlich viele Primzahlen $p \geq 2k-1$ und somit auch unendliche viele Sprachen $L'_p$ gibt.

**Lemma 6.30.** Jeder $k$C-OCA$_t$ für $L'_p$ benötigt mindestens $\Omega(\sqrt{p})$ Zustände.

*Beweis.* Für einen Widerspruchsbeweis nehmen wir an, daß $L'_p$ von einem $k$C-OCA$_t$ mit $n = o(\sqrt{p})$ Zuständen akzeptiert wird. Aufgrund der Konstruktion in Lemma 6.23(2) gibt es dann einen $k$C-OCA, der $L'_p$ mit $n^2 + n = o(p)$ Zuständen akzeptiert. Das ist ein Widerspruch zu Lemma 6.24. □

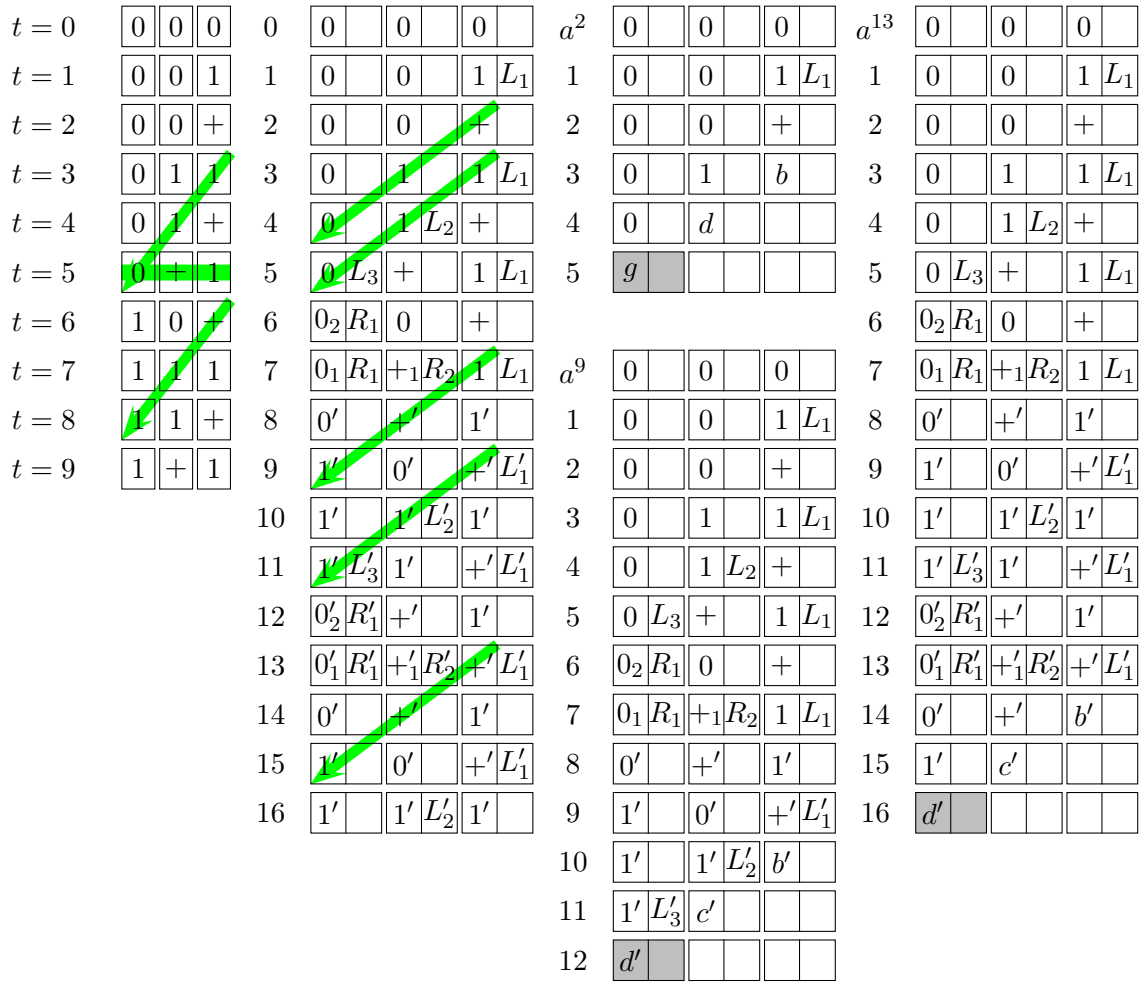



Abbildung 6.14: Beispielberechnungen zu Beispiel 6.25

**Satz 6.31.** Die folgende Tabelle faßt die bisher bewiesenen oberen und unteren Schranken für $k \geq 2$ zusammen. Ein Eintrag in der Spalte $A$ und der Zeile $B$ nennt die oberen und unteren Schranken, wenn Typ-$A$ Automaten in Typ-$B$ Automaten konvertiert werden.

| | DFA | $k$C-OCA | $k$C-OCA$_t$ | $k$C-CA |
|---|---|---|---|---|
| DFA | —— | $O(n^k)$ $(a)$<br>$\Omega(n^k)$ | $O(n^k)$ $(a)$<br>$\Omega(n^k)$ | $O(n^k)$ $(a)$<br>$\Omega(n^k)$ |
| $k$C-OCA | $\leq n+1$ $(b)$<br>$\geq n+1$ | —— | $\leq n^2+n$ $(c)$<br>$\Omega(n^2)$ | $O(n^k)$ $(d)$<br>$\Omega(n^k)$ |
| $k$C-OCA$_t$ | $O(\sqrt{n})$ $(e)$<br>$\Omega(\sqrt{n/\log n})$ | $\leq n$ $(f)$<br>$\geq n$ | —— | $\leq n^{\lceil k/2 \rceil}+3$ $(g)$<br>$\Omega(n^{k/2})$ |
| $k$C-CA | $O(\sqrt{n})$ $(e)$<br>$\Omega(\sqrt[3]{n/\log n})$ | $\leq n$ $(f)$<br>$\geq n$ | $\leq n$ $(f)$<br>$\geq n$ | —— |

*Beweis.*

(a) Die oberen und unteren Schranken folgen aus den Lemmata 6.6, 6.9, 6.16 und 6.17.

(b) Die Behauptung folgt aus Lemma 6.11 und Lemma 6.12.

(c) Die obere Schranke folgt aus Lemma 6.23(2). Nach Korollar 6.27 gibt es einen DFA für $L'_p$ mit $p$ Zuständen. Durch Anwendung von Lemma 6.18 können wir einen $k$C-OCA$_t$ konstruieren, der $L'_p$ mit $n = O(\sqrt{p})$ Zuständen akzeptiert. Ein äquivalenter $k$C-OCA benötigt nach Lemma 6.24 $p = \Omega(n^2)$ Zustände.

(d) Die obere Schranke folgt aus Lemma 6.23(1). Die untere Schranke folgt aus Korollar 6.27 und Lemma 6.24.

(e) Die oberen und unteren Schranken folgen aus Lemma 6.18 und Lemma 6.20.

(f) Die Behauptung folgt aus Lemma 6.22.

(g) Die obere Schranke folgt aus Lemma 6.23(3). Die untere Schranke folgt aus Korollar 6.27 und Lemma 6.30.

□

## 6.3 Zellularautomaten mit graduellem zweiseitigem Informationsfluß

Wir haben bislang Modelle betrachtet, die entweder keine zweiseitige Kommunikation gestatten ($k$C-OCA), oder minimalen bzw. maximalen zweiseitigen Informationsfluß erlauben ($k$C-OCA$_t$ bzw. $k$C-CA). In diesem Abschnitt werden wir zweiseitigen Informationsfluß als eine quantifizierbare Ressource auffassen und Zellularautomaten mit beschränkter Zellenzahl betrachten, die schrittweise mit mehr Zellen ausgestattet werden, die zweiseitige Kommunikation erlauben. Wir sprechen also im folgenden von Zellularautomaten mit beschränkter Zellenzahl $k$ und zweiseitigem Informationsfluß vom Grad $l$, wenn die letzten $l' \leq l$ Zellen, also höchstens die letzten $l$ Zellen, mit beiden Nachbarn kommunizieren können.

### 6.3.1 Definition

**Definition 6.32.** Ein Zellularautomat mit beschränkter Zellenzahl $k$ und eingeschränktem zweiseitigem Informationsfluß vom Grad $l$ ($1 \leq l \leq k$), abgekürzt $k$C-OCA($l$), ist ein Tupel $(Q, \#, \Sigma, q_0, \triangledown, k, l, \delta_r, \delta_t, \delta, F)$, wobei folgendes gilt:

(1) $Q \neq \emptyset$ ist die endliche Menge der Zustände der Zellen,

(2) $\# \notin Q \cup \Sigma$ ist der Grenzzustand,

(3) $\Sigma$ ist das Eingabealphabet,

(4) $\triangledown \notin Q \cup \Sigma$ ist das Symbol für das Eingabeende,

(5) $k$ ist die Anzahl der Zellen,

(6) $l$ ist die maximale Anzahl der Zellen mit zweiseitigem Informationsfluß, wobei die letzten $1 \leq l' \leq l$ Zellen mit beidseitiger Kommunikation ausgestattet sind,

(7) $\delta_r : (Q \cup \{\#\}) \times Q \times (\Sigma \cup \{\triangledown\}) \to Q$ ist die lokale Überführungsfunktion für die Kommunikationszelle,

(8) $\delta_t : (Q \cup \{\#\}) \times Q \times Q \to Q$ ist die lokale Überführungsfunktion für die restlichen Zellen mit zweiseitigem Informationsfluß, das heißt für die Zellen $k - l' + 1$ bis $k - 1$,

(9) $\delta : Q \times Q \to Q$ ist die lokale Überführungsfunktion für die restlichen Zellen mit einseitigem Informationsfluß, das heißt für die Zellen 1 bis $k - l'$,

(10) $q_0 \in Q$ ist der Ruhezustand, für den folgendes gilt: $\delta_r(q_0, q_0, \triangledown) = q_0$, $\delta_r(\#, q_0, \triangledown) = q_0$, $\delta_t(q_0, q_0, q_0) = q_0$, $\delta_t(\#, q_0, q_0) = q_0$ und $\delta(q_0, q_0) = q_0$

(11) $F \subseteq Q$ ist die Menge der akzeptierenden Zustände.

**Bemerkung 6.33.** Gemäß der Definition können in einem $k$C-OCA($l$) auch weniger als $l$ Zellen mit zweiseitiger Kommunikation vorhanden sein. Wenn wir in den folgenden Aussagen einen $k$C-OCA($l$) gegeben haben, dann gehen wir implizit davon aus, daß in dem $k$C-OCA($l$) auch tatsächlich $l$ Zellen mit zweiseitiger Kommunikation vorhanden sind. Sind nämlich nur $l' < l$ Zellen mit zweiseitiger Kommunikation vorhanden, dann können wir diesen Automaten auch als $k$C-OCA($l'$) auffassen und betrachten die Aussage für einen gegebenen $k$C-OCA($l'$).

**Bemerkung 6.34.** Für den Fall $k = 1$ entfallen die Funktionen $\delta$ und $\delta_t$. Die Funktion $\delta_r$ reduziert sich zu einer Abbildung $\delta_r : \{\#\} \times Q \times (\Sigma \cup \{\triangledown\}) \to Q$ und wir erhalten wieder einen DFA mit veränderter Akzeptanzbedingung. Ist $l' = 1$, dann haben wir außer der Kommunikationszelle keine weiteren Zellen mit zweiseitiger Kommunikation. Also entfällt die Funktion $\delta_t$ und wir schreiben $A = (Q, \#, \Sigma, q_0, \triangledown, k, 1, \delta_r, \delta, F)$. Ist $l' = k$, dann haben wir keine Zellen mit einseitiger Kommunikation. Daher entfällt die Funktion $\delta$ und wir schreiben $A = (Q, \#, \Sigma, q_0, \triangledown, k, \delta_r, \delta_t, F)$.

Die Arbeitsweise eines $k$C-OCA($l$) ist wiederum nahezu identisch zur Arbeitsweise eines $k$C-OCA: Der nächste Zustand jeder Zelle hängt vom Zustand der Zelle selbst und ihres rechten Nachbarn ab. Zusätzlich hängt der nächste Zustand der letzten $l$ Zellen vom Zustand der linken Nachbarzelle ab.

Die Definition einer Konfiguration eines $k$C-OCA($l$) ist analog zur Definition bei $k$C-OCA (s. S. 83). Im folgenden nehmen wir gemäß Bemerkung 6.33 an, daß in dem gegebenen $k$C-OCA($l$) tatsächlich $l$ Zellen mit zweiseitiger Kommunikation vorhanden sind. Für $l = 1$ bzw. $l = k$ entfallen gemäß Bemerkung 6.34 die Funktionen $\delta_t$ bzw. $\delta$. Die globale Überführungsfunktion $\Delta$ wird durch $\delta$, $\delta_t$ und $\delta_r$ folgendermaßen induziert: Sei $(c_t, w_t)$ für $t \geq 0$ eine Konfiguration.

$$
\begin{aligned}
(c_{t+1}, w_{t+1}) \;=\; & \Delta(c_t, w_t) \Longleftrightarrow \\
& c_{t+1}(i) = \delta(c_t(i), c_t(i+1)), i \in \{1, \ldots, k-l\} \\
& c_{t+1}(i) = \delta_t(c_t(i-1), c_t(i), c_t(i+1)), i \in \{k-l+1, \ldots, k-1\} \\
& c_{t+1}(k) = \delta_r(c_t(k-1), c_t(k), x)
\end{aligned}
$$

wobei $x = \triangledown$, $w_{t+1} = \epsilon$ (falls $w_t = \epsilon$) und $x = x_1$, $w_{t+1} = x_2 \dots x_n$ (falls $w_t = x_1 x_2 \dots x_n$).

Eine Eingabe $w$ wird von einem $k$C-OCA($l$) akzeptiert, falls es einen Zeitpunkt $i$ während der Berechnung gibt, so daß die erste Zelle einen akzeptierenden Zustand annimmt. Die Sprachklassen $\mathcal{L}(k\text{C-OCA}(l))$ sind analog zu Definition 6.3 definiert.

**Bemerkung 6.35.** Wir haben bislang einen $k$C-OCA($l$) nur für $1 \leq l \leq k$ definiert. Der verbleibende Fall $l = 0$, das heißt, es gibt keinen zweiseitigen Informationsfluß, kann auf das bereits eingeführte Modell reduziert werden, indem wir einen $k$C-OCA(0) mit einem $k$C-OCA identifizieren.

### 6.3.2 Umwandlung eines $k$C-OCA($l$) in einen DFA

Durch ähnliche Beweise wie in den vorhergehenden Abschnitten kann gezeigt werden, daß ein $k$C-OCA($l$) in einen DFA mit polynomiellem Zuwachs vom Grad $k$ umgewandelt werden kann. Weiterhin ist diese obere Schranke asymptotisch scharf.

**Lemma 6.36.** Jeder $k$C-OCA($l$) ($0 \leq l \leq k$) $A$ mit $n$ Zuständen kann in einen DFA $M$ umgewandelt werden, so daß $T(M) = T(A)$ und $|M| \leq n^k - n^{k-1} + 1$ gilt.

*Beweis.* Für $l = 0$ folgt die Behauptung aus Lemma 6.6. Für $1 \leq l \leq k$ ist die Konstruktion wiederum im wesentlichen identisch zur Konstruktion für $k$C-OCA aus Lemma 6.6. Lediglich die Überführungsfunktion $\delta_1$ muß an die entsprechende Nachbarschaft angepaßt werden. Das heißt, für $q_i, q_i' \in Q$ ($1 \leq i \leq k$) und $\sigma \in \Sigma_1$ definieren wir

$$\delta_1((q_1, q_2, \dots, q_k), \sigma) = (q_1', q_2', \dots, q_k'),$$

wobei gilt:

- $q_1' = \delta(q_1, q_2), q_2' = \delta(q_2, q_3), \dots, q_{k-l}' = \delta(q_{k-l-1}, q_{k-l})$,
- $q_{k-l+1}' = \delta_t(q_{k-l}, q_{k-l+1}, q_{k-l+2}), \dots, q_{k-1}' = \delta_t(q_{k-2}, q_{k-1}, q_k)$ und
- $q_k' = \delta_r(q_{k-1}, q_k, \sigma)$.

Durch diese Anpassungen wird offenbar die Anzahl der Zustände nicht verändert. □

**Lemma 6.37.** $L_{n,k}$ kann von einem $k$C-OCA($l$) ($0 \leq l \leq k$) mit $n+1$ Zuständen akzeptiert werden und jeder $k$C-OCA($l$) ($0 \leq l \leq k$) für die Sprache $L_{n,k}$ benötigt mindestens $n+1$ Zustände.

*Beweis.* Der Beweis ist wiederum identisch zum Beweis für $k$C-OCA aus Lemma 6.9, denn die Fähigkeit manche Zustände des linken Nachbarn zu sehen, verringert offenbar nicht die Anzahl der zu unterscheidenden Konfigurationen. □

### 6.3.3 Umwandlung eines DFA in einen $k$C-OCA($l$)

In Lemma 6.18 wurde gezeigt, wie ein DFA in den beiden letzten Zellen eines $k$C-OCA$_t$ bzw. $k$C-CA simuliert werden kann. Eine identische Konstruktion ist für $k$C-OCA($l$) möglich, sofern $k \geq 2$ und $l \geq 1$ gilt. Dann steht nämlich mindestens eine Zelle mit zweiseitigem Informationsfluß zur Verfügung und der DFA kann in den beiden letzten Zellen simuliert werden. Damit erhalten wir das folgende Lemma:

**Lemma 6.38.** Jeder DFA $M$ mit $n$ Zuständen kann in einen äquivalenten $k$C-OCA($l$) $A$ mit $k \geq 2$ und $1 \leq l \leq k$ umgewandelt werden. Beschreibt $M$ eine Sprache aus $\Sigma^*$, dann gilt für die Anzahl der Zustände in $A$: $|A| \leq \lceil \sqrt{n} \rceil (|\Sigma| + 1) + 2|\Sigma| + 2 = O(\sqrt{n})$

Auch die untere Schranke kann mit einem ähnlichen Inkompressibilitätsargument wie in Lemma 6.20 bewiesen werden.

**Lemma 6.39.** Es gibt eine unendliche Folge von Sprachen $(L_n)_{n\in\mathbb{N}}$, so daß jede Sprache $L_n$ von einem DFA mit $n+1$ Zuständen akzeptiert wird. Jeder $k$C-OCA($l$) ($1 \leq l \leq k$) für $L_n$ benötigt allerdings mindestens $\Omega(\sqrt[3]{n/\log n})$ Zustände.

*Beweis.* Wir verwenden ein ähnliches Argument wie in Lemma 6.20.

Dazu wählen wir ein $n \in \mathbb{N}$ mit $|P| \leq (1/8)n$ und $\log k + \log l \leq (1/8)n$. Nach Satz 2.30 gibt es ein inkompressibles Wort $x$ der Länge $n$ mit $C(x|n) \geq n$. Wir setzen $L_n = \{x\}$. Zum Widerspruchbeweis nehmen wir an, daß es einen $k$C-OCA($l$) $A$ gibt, der die Sprache $L_n$ mit $m$ Zuständen akzeptiert, wobei gilt:

$$m + 1 < \sqrt[3]{\frac{n}{(\log n)(|\Sigma| + 1)}}$$

Die folgenden Ungleichungen folgen wie in Lemma 6.20 unmittelbar aus $|\Sigma| \geq 2$ und $m \geq 1$.

$$\begin{aligned}
(m+1)^3 \log m &\leq (1/3)(m+1)^3(|\Sigma|+1)\log m, \\
(m+1)\log m &\leq (1/6)(m+1)^3(|\Sigma|+1)\log m, \\
(m+2)m^2 &\leq (m+1)^3
\end{aligned}$$

Dann gilt folgende Abschätzung

$$\begin{aligned}
|cod(A)| &\leq \underbrace{\log m^{m^2}}_{|cod(\delta)|} + \underbrace{\log m^{(m+1)m^2}}_{|cod(\delta_t)|} + \underbrace{\log m^{(m+1)m(|\Sigma|+1)}}_{|cod(\delta_r)|} + \underbrace{m \log m}_{|cod(F)|} + \underbrace{\log k}_{|cod(k)|} + \underbrace{\log l}_{|cod(l)|} \\
&\leq m^2 \log m + (m+1)m^2 \log m + (m+1)m(|\Sigma|+1)\log m + m \log m + \frac{1}{8}n \\
&\leq (m+1)^3 \log m + (m+1)^3(|\Sigma|+1)\log m + (m+1)\log m + \frac{1}{8}n \\
&\leq \left(\frac{1}{3} + 1 + \frac{1}{6}\right)(m+1)^3(|\Sigma|+1)\log m + \frac{1}{8}n \\
&\leq \frac{1}{2}n + \frac{1}{8}n = \frac{5}{8}n
\end{aligned}$$

gemäß analogen Überlegungen wie im Beweis von Lemma 6.20. Damit gilt

$$C(L_n|n) \leq |cod(A)| + |P| \leq \frac{5}{8}n + \frac{1}{8}n = \frac{3}{4}n < n$$

Das ist ein Widerspruch zu $C(L_n|n) \geq n$. □

### 6.3.4 Umwandlungen zwischen $k$C-OCA($l$) und $k$C-OCA($m$)

In diesem Abschnitt untersuchen wir die Auswirkungen auf die Beschreibungskomplexität, wenn die Anzahl der Zellen mit zweiseitiger Kommunikation erhöht oder reduziert wird. Zunächst läßt sich jeder Automat in einen anderen Automat mit einer größeren Anzahl an Zellen mit zweiseitiger Kommunikation einbetten, ohne die Anzahl der Zustände zu erhöhen.

**Lemma 6.40.** Für $0 \leq l < m \leq k$ gilt: Jeder $k$C-OCA($l$) mit $n$ Zuständen kann in einen äquivalenten $k$C-OCA($m$) mit $n$ Zuständen umgewandelt werden. Diese obere Schranke ist scharf.

*Beweis.* Aus der Definition folgt unmittelbar, daß ein $k$C-OCA($m$) einen $k$C-OCA($l$) simulieren kann, ohne die Anzahl der Zustände zu erhöhen. In Lemma 6.37 wird gezeigt, daß jeder $k$C-OCA($l$) oder $k$C-OCA($m$) für die Sprache $L_{n,k}$ mindestens $n+1$ Zustände benötigt, da $n^k+1$ Konfigurationen mit $k$ Zellen unterschieden werden müssen. Damit ist die Behauptung bewiesen. □

Bei der kompletten Beseitigung von zweiseitiger Kommunikation ergibt sich folgende obere Schranke:

**Lemma 6.41.** Für $1 \leq l \leq k$ gilt: Jeder $k$C-OCA($l$) mit $n$ Zuständen kann in einen äquivalenten $k$C-OCA mit höchstens $2n^{l+1}$ Zuständen umgewandelt werden.

*Beweis.* Wir betrachten eine ähnliche Konstruktion wie in Lemma 6.23. Da alle Zellen in einem $k$C-OCA keine Informationen von links bekommen können, wurde dort der linke Nachbar der Kommunikationszelle zusätzlich in der Kommunikationszelle simuliert. Hier ist das Vorgehen ähnlich: Zunächst simuliert jede Zelle mit beidseitiger Kommunikation zusätzlich ihren linken Nachbarn. Dies ist aber nicht ausreichend, falls der simulierte linke Nachbar ebenfalls eine Zelle mit beidseitiger Kommunikation ist, dessen Zustände dann wiederum von dessen linkem Nachbarn abhängen können. Daher muß jede Zelle mit beidseitiger Kommunikation sämtliche Zellen links von ihr simulieren, von denen sie Informationen erhalten kann. Das sind alle Zellen mit beidseitiger Kommunikation und zusätzlich die erste Zelle von rechts mit einseitiger Kommunikation.

Es sei also $A$ ein $k$C-OCA($l$) mit Zustandsmenge $Q$. Wir konstruieren einen $k$C-OCA $A'$ mit folgender Zustandsmenge $Q' = Q^{l+1} \cup Q^l \cup \ldots \cup Q^2 \cup Q$. Dann werden in der Kommunikationszelle der ursprüngliche Zustand und zusätzlich die Zustände der nächsten $l$ linken Nachbarzellen simuliert. In der vorletzten Zelle werden der ursprüngliche Zustand und die Zustände der nächsten $l-1$ linken Nachbarzellen simuliert. Dies wird für die nächsten Zellen mit zweiseitiger Kommunikation analog fortgeführt. In der $l$-ten Zelle von rechts werden schließlich der ursprüngliche Zustand und der Zustand der linken Nachbarzelle simuliert. In den restlichen Zellen mit einseitiger Kommunikation wird weiterhin der ursprüngliche Zustand simuliert. Wir verzichten auf eine formale Definition des $k$C-OCA $A'$ und verweisen stattdessen auf die Abbildung 6.15 auf Seite 116. Dort findet man ein Zustandsdiagramm für die Simulation eines 6C-OCA(3) durch einen 6C-OCA.

Man kann leicht einsehen, daß $T(A') = T(A)$ gilt. Die Zustandsmenge kann folgendermaßen abgeschätzt werden, wobei $n = |Q|$ ist:

$$|Q'| = \sum_{i=1}^{l+1} n^i = \frac{n^{l+2}-1}{n-1} - 1 \leq \frac{n^{l+2}-1}{n-1} \leq \frac{n^{l+2}}{n-1} = \frac{n}{n-1} n^{l+1} \leq 2n^{l+1}$$

□

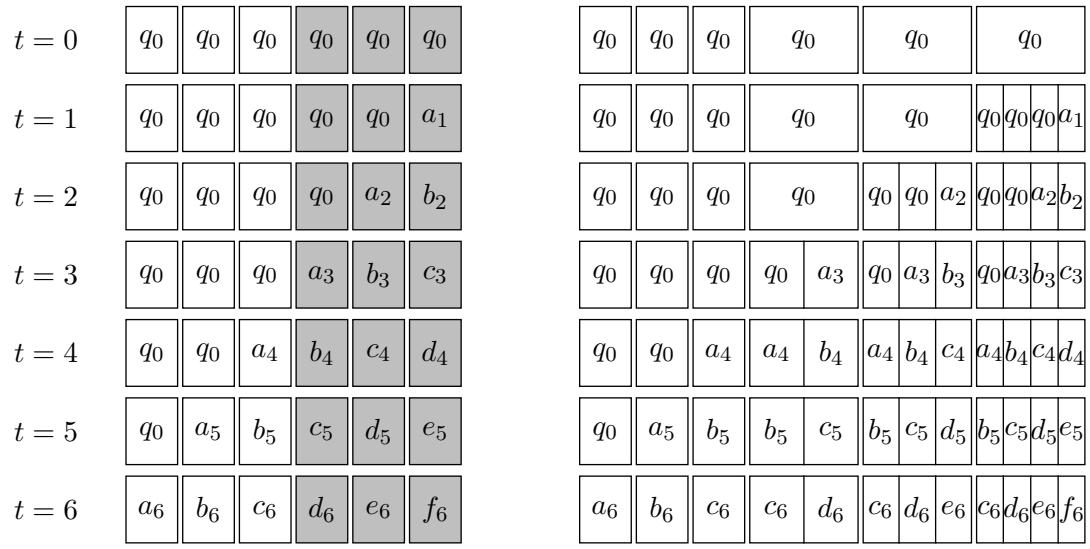



Abbildung 6.15: Simulieren eines $k$C-OCA($l$) durch einen $k$C-OCA

Zum Beweis der unteren Schranke zeigen wir, wie die Sprache $L'_p$ von einem $k$C-OCA($l$) mit möglichst wenigen Zuständen akzeptiert werden kann. In Korollar 6.27 hatten wir das Ergebnis, daß ein $k$C-CA für $L'_p$ gerade $O(\sqrt[k]{p})$ Zustände benötigt. Dazu wurde ausgenutzt, daß ein DFA für die unäre Sprache $L'_p$ nur aus einem periodischen Teil der Länge $p$ und einem akzeptierenden Zustand besteht. Das periodische Zählen der Länge $p$ konnte auf die $k$ Zellen verteilt werden, so daß in jeder Zelle nur $O(\sqrt[k]{p})$ Zustände benötigt werden. Mit ähnlichen Überlegungen kann man auch einen $k$C-OCA($k$) für $L'_p$ mit $O(\sqrt[k]{p})$ Zuständen konstruieren. In einem $k$C-OCA($l$) kann man das periodische Zählen der Länge $p$ auf die letzten $l+1$ Zellen verteilen. Daher werden in jeder Zelle dann $O(p^{1/(l+1)})$ Zustände benötigt.

**Lemma 6.42.** Für $1 \leq l < k$ und $p \geq 2k-1$ gilt:

(1) Die Sprache $L'_p$ wird von einem $k$C-OCA($k$) mit $O(p^{1/k})$ Zuständen akzeptiert.

(2) Die Sprache $L'_p$ wird von einem $k$C-OCA($l$) mit $O(p^{1/(l+1)})$ Zuständen akzeptiert.

(3) Jeder $k$C-OCA($l$) für die Sprache $L'_p$ benötigt mindestens $\frac{1}{2}p^{1/(l+1)}$ Zustände.

*Beweis.* Wir betrachten zunächst den Fall $l = k$. Nach Korollar 6.27 kann die Sprache $L'_p$ von einem $k$C-CA $A = (Q, \#, \Sigma, q_0, \triangledown, k, \delta_r, \delta, F)$ mit $O(p^{1/k})$ Zuständen akzeptiert werden. Wir simulieren nun den $k$C-CA $A$ durch einen $k$C-OCA($k$) $A' = (Q, \#, \Sigma, q_0, \triangledown, k, \delta'_r, \delta'_t, F)$, indem wir $\delta'_r$ mit $\delta_r$ und $\delta'_t$ mit $\delta$ identifizieren. Offenbar gilt $T(A') = T(A)$ und $|A'| = O(p^{1/k})$ und damit (1).

Im Fall $1 \leq l < k$ betrachten wir zunächst einen $(l+1)$C-CA $A = (Q, \#, \Sigma, q_0, \triangledown, l+1, \delta_r, \delta, F)$, der die Sprache $L'_p$ mit $O(p^{1/(l+1)})$ Zuständen akzeptiert. Ein solcher $(l+1)$C-CA existiert nach Korollar 6.27. Wir betten nun diesen $(l+1)$C-CA folgendermaßen in einen $k$C-OCA($l$) $A' = (Q, \#, \Sigma, q_0, \triangledown, k, l, \delta'_r, \delta'_t, \delta', F)$ ein: Die $l$ Zellen in $A'$ mit zweiseitiger Kommunikation verhalten sich wie die letzten $l$ Zellen in $A$. Das heißt, wir identifizieren $\delta'_r$ mit $\delta_r$ und $\delta'_t$ mit

$\delta$. Die verbleibenden $k-l$ Zellen mit einseitiger Kommunikation verhalten sich wie die erste Zelle in $A$. Das heißt, $\delta'(q,q') = \delta(\#,q,q')$ für alle $q,q' \in Q$. Also wird insbesondere in der $(l+1)$-ten Zelle von rechts in $A'$ die erste Zelle aus $A$ simuliert. Damit wird der gegebene $(l+1)$C-CA $A$ in den letzten $l+1$ Zellen in $A'$ simuliert. In den ersten $k-l-1$ Zellen in $A'$ werden irgendwelche Zustände angenommen. Eine Skizze der Konstruktion findet man in Abbildung 6.16.

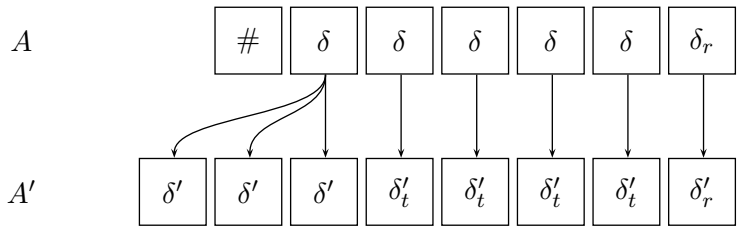



Abbildung 6.16: Einbetten eines $(l+1)$C-CA $A$ in einen $k$C-OCA$(l)$ $A'$

Nach Bemerkung 6.28 zu Korollar 6.27 können wir annehmen, daß die erste Zelle in $A$ bei einer Eingabe $w$ zum Zeitpunkt $|w|+l+1$ genau dann einen bestimmten akzeptierenden Zustand $f$ annimmt, falls $w \in L'_p$ ist. Weiterhin können wir voraussetzen, daß ausschließlich die erste Zelle diesen Zustand annehmen kann.

1.Fall: $w \in L'_p$

Da $A'$ den $(l+1)$C-CA $A$ simuliert und daher die $(l+1)$-te Zelle von rechts den Zustand $f$ zum Zeitpunkt $|w|+l+1$ annimmt, müssen wir diesen Zustand $f$ in den verbleibenden $k-l-1$ Zeitschritten in die erste Zelle von $A'$ befördern. Da bislang keine andere Zelle in $A$ und somit auch in $A'$ den Zustand $f$ angenommen hat, können wir $\delta'$ wie folgt ergänzen: $\delta'(q,f) = f$ für alle $q \in Q$. Damit erreicht der Zustand $f$ nach $k-l-1$ Zeitschritten die erste Zelle in $A'$ und die Eingabe wird akzeptiert.

2.Fall: $w \notin L'_p$

Nach Bemerkung 6.28 nimmt keine Zelle in $A$ in diesem Fall den akzeptierenden Zustand $f$ an. Damit nimmt auch keine in Zelle in $A'$ den Zustand $f$ an, und die Eingabe wird daher nicht akzeptiert.

Damit gibt es einen $k$C-OCA$(l)$ $A'$, der $L'_p$ mit $|Q| = O(p^{1/(l+1)})$ Zuständen akzeptiert und daraus folgt (2).

Zum Beweis von (3) nehmen wir für einen Widerspruchsbeweis an, daß es einen $k$C-OCA$(l)$ für die Sprache $L'_p$ mit $n < \frac{1}{2}p^{1/(l+1)}$ Zuständen gibt. Nach Lemma 6.41 kann dann ein $k$C-OCA für $L'_p$ mit $m \leq 2n^{l+1}$ Zuständen konstruiert werden. Nun gilt

$$m \leq 2n^{l+1} < 2\left(\frac{1}{2}p^{\frac{1}{l+1}}\right)^{l+1} = \frac{1}{2^l}p \leq \frac{1}{2}p < p$$

Also gibt es einen $k$C-OCA für die Sprache $L'_p$ mit weniger als $p$ Zuständen. Das ist ein Widerspruch zu Lemma 6.24.

□

Eine obere Schranke für die Reduktion von $m$ Zellen mit zweiseitiger Kommunikation auf $l$ Zellen liefert das folgende Lemma.

**Lemma 6.43.** Für $1 \leq l < m \leq k$ gilt: Jeder $k$C-OCA($m$) mit $n$ Zuständen kann in einen äquivalenten $k$C-OCA($l$) mit höchstens $2n^{m-l+1} + n = O(n^{m-l+1})$ Zuständen umgewandelt werden.

*Beweis.* Sei $A$ ein $k$C-OCA($m$) mit $n$ Zuständen. Um einen $k$C-OCA($l$) $A'$ zu konstruieren, gehen wir ähnlich wie in Lemma 6.41 vor. Wir teilen die $k$ Zellen in $A'$ in drei Teile auf:

- In den ersten $k-m$ Zellen werden die ersten $k-m$ Zellen aus $A$ simuliert.
- In den letzten $l$ Zellen werden die letzten $l$ Zellen aus $A$ simuliert.
- In den verbleibenden $m-l$ Zellen $k-m+1, k-m+2, \ldots, k-m+m-l = k-l$ mit einseitiger Kommunikation werden die entsprechenden Zellen mit beidseitiger Kommunikation aus $A$ simuliert. Wie in Lemma 6.41 werden dazu in jeder Zelle, zusätzlich zum Zustand der Zelle selbst, die Zustände aller linken Nachbarn mit zweiseitiger Kommunikation simuliert.

Es sei $Q$ die Zustandsmenge des $k$C-OCA($m$) $A$. Die Zustandsmenge in $A'$ ist dann

$$Q^{m-l+1} \cup Q^{m-l} \cup \ldots \cup Q^2 \cup Q \cup Q',$$

wobei $Q' \cap Q = \emptyset$ und $|Q'| = |Q|$ ist. In den letzten $l$ Zellen in $A'$ werden die letzten $l$ Zellen aus $A$ mit Zuständen aus $Q'$ simuliert. In der $(k-l)$-ten Zelle in $A'$, also der ersten Zelle mit einseitiger Kommunikation, die zweiseitige Kommunikation simulieren muß, simulieren wir dann den ursprünglichen Zustand aus $A$ und zusätzlich die Zustände der nächsten $m-l$ linken Nachbarzellen. Diese Konstruktion wird für die nächsten $m-l-1$ Zellen analog fortgeführt. Die $(k-m+1)$-te Zelle simuliert dann den ursprünglichen Zustand aus $A$ und zusätzlich den Zustand der linken Nachbarzelle. In den verbleibenden Zellen mit einseitiger Kommunikation werden die entsprechenden Zellen aus $A$ simuliert. Eine Skizze der Konstruktion findet man in Abbildung 6.17.

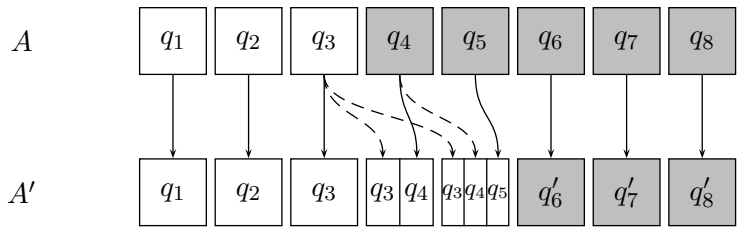



Abbildung 6.17: Simulieren eines $k$C-OCA($m$) $A$ durch einen $k$C-OCA($l$) $A'$

Wir verzichten auch hier auf eine formale Definition des $k$C-OCA $A'$ und verweisen auf die Abbildung 6.18 auf Seite 119. Dort findet man ein Zustandsdiagramm für die Simulation eines 6C-OCA(4) durch einen 6C-OCA(2).

Wiederum gilt offenbar $T(A') = T(A)$. Die Zustandsmenge kann folgendermaßen abgeschätzt werden:

$$|Q'| = \sum_{i=1}^{m-l+1} n^i + n = \frac{n^{m-l+2}-1}{n-1} - 1 + n \leq 2n^{m-l+1} + n$$

□

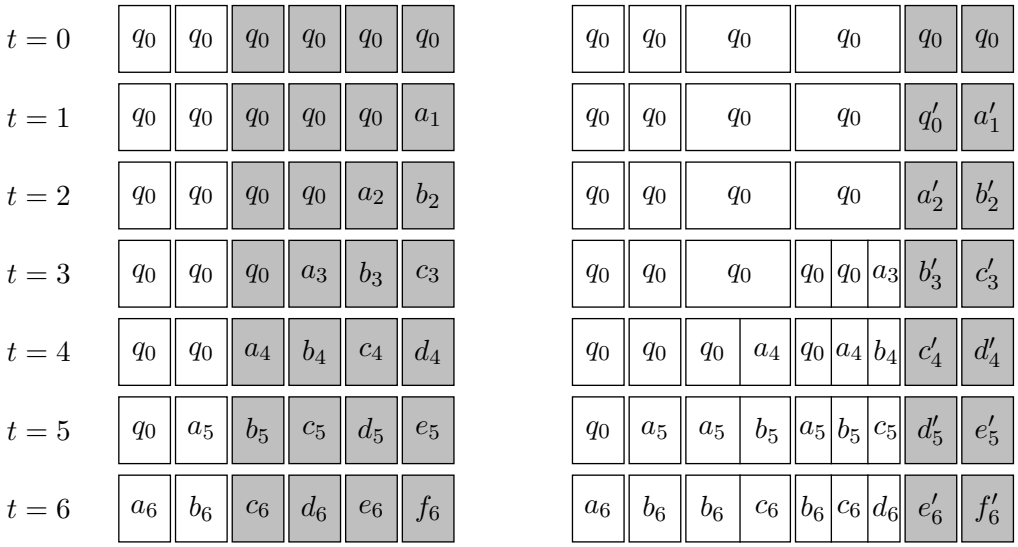



Abbildung 6.18: Simulieren eines 6C-OCA(4) durch einen 6C-OCA(2)

**Satz 6.44.** Die folgende Tabelle faßt die oberen und unteren Schranken für $k \geq 2$ zusammen. Ein Eintrag in der Spalte $A$ und der Zeile $B$ nennt die oberen und unteren Schranken, wenn Typ-$A$ Automaten in Typ-$B$ Automaten umgewandelt werden. Es gilt: $1 \leq l < m < k$.

| | DFA | $k$C-OCA($l$) | $k$C-OCA($m$) | $k$C-OCA($k$) |
|---|---|---|---|---|
| DFA | —— | $O(n^k)$ $(a)$<br>$\Omega(n^k)$ | $O(n^k)$ $(a)$<br>$\Omega(n^k)$ | $O(n^k)$ $(a)$<br>$\Omega(n^k)$ |
| $k$C-OCA | $\leq n+1$ $(b)$<br>$\geq n+1$ | $\leq 2n^{l+1}$ $(c)$<br>$\Omega(n^{l+1})$ | $\leq 2n^{m+1}$ $(c)$<br>$\Omega(n^{m+1})$ | $O(n^k)$ $(d)$<br>$\Omega(n^k)$ |
| $k$C-OCA($l$) | $O(\sqrt{n})$ $(e)$<br>$\Omega(\sqrt[3]{n/\log n})$ | —— | $O(n^{m-l+1})$ $(f)$<br>$\Omega(n^{\frac{m+1}{l+1}})$ | $O(n^{k-l+1})$ $(g)$<br>$\Omega(n^{\frac{k}{l+1}})$ |
| $k$C-OCA($m$) | $O(\sqrt{n})$ $(e)$<br>$\Omega(\sqrt[3]{n/\log n})$ | $\leq n$ $(h)$<br>$\geq n$ | —— | $O(n^{k-m+1})$ $(g)$<br>$\Omega(n^{\frac{k}{m+1}})$ |
| $k$C-OCA($k$) | $O(\sqrt{n})$ $(e)$<br>$\Omega(\sqrt[3]{n/\log n})$ | $\leq n$ $(h)$<br>$\geq n$ | $\leq n$ $(h)$<br>$\geq n$ | —— |

*Beweis.*

(a) Die oberen und unteren Schranken folgen aus den Lemmata 6.36 und 6.37.

(b) Die oberen und unteren Schranken wurden bereits in Satz 6.31(b) bewiesen.

(c) Die obere Schranke folgt aus Lemma 6.41. Nach Lemma 6.42(2) gibt es einen $k$C-OCA($l$) für die Sprache $L'_p$ mit $O(p^{1/(l+1)})$ Zuständen. Nach Lemma 6.24 benötigt jeder $k$C-OCA für $L'_p$ mindestens $p$ Zustände. Wir setzen $n = cp^{1/(l+1)}$ für eine Konstante $c > 0$. Dann ist $p = c'n^{l+1}$ und damit folgt die Behauptung. Identische Überlegungen ergeben die Schranken für $k$C-OCA($m$).

(d) Die oberen und unteren Schranken folgen aus Satz 6.31(d), indem wir jeden $k$C-OCA($k$) als $k$C-CA auffassen.

(e) Die oberen und unteren Schranken folgen aus Lemma 6.38 und Lemma 6.39.

(f) Die obere Schranke folgt aus Lemma 6.43. Nach Lemma 6.42(2) gibt es einen $k$C-OCA($m$) für die Sprache $L'_p$ mit $O(p^{1/(m+1)})$ Zuständen. Nach Lemma 6.42(3) benötigt jeder $k$C-OCA($l$) für $L'_p$ mindestens $t \geq \frac{1}{2}p^{1/(l+1)}$ Zustände. Wir setzen $n = cp^{1/(m+1)}$ für eine Konstante $c > 0$. Dann ist $p = c'n^{m+1}$ für eine weitere Konstante $c' > 0$. Damit gilt für eine weitere Konstante $c'' > 0$:

$$t \geq \frac{1}{2}p^{\frac{1}{l+1}} = \frac{1}{2}(c'n^{m+1})^{\frac{1}{l+1}} = c''n^{\frac{m+1}{l+1}} = \Omega(n^{\frac{m+1}{l+1}})$$

(g) Die obere Schranke folgt aus Lemma 6.43. Die untere Schranke läßt sich analog zu (f) beweisen. Wir wissen nach Lemma 6.42(1), daß es einen $k$C-OCA($k$) für die Sprache $L'_p$ mit $O(p^{1/k})$ Zuständen gibt. Nach Lemma 6.42(3) benötigt jeder $k$C-OCA($m$) für $L'_p$ mindestens $\frac{1}{2}p^{1/(m+1)}$ Zustände. Nach analogen Umformungen erhalten wir die Behauptung.

(h) Die Behauptung folgt aus Lemma 6.40.

□

**Bemerkung 6.45.** In obiger Tabelle fällt der große Unterschied zwischen oberer und unterer Schranke bei der Reduktion von zweiseitiger Kommunikation auf. Es ist gegenwärtig nicht bekannt, ob die obere Schranke verbessert werden kann, oder eine größere untere Schranke bewiesen werden kann. Eine naheliegende Idee zur Verbesserung der oberen Schranke bei der Reduktion von $m$ Zellen auf $l$ Zellen mit beidseitiger Kommunikation ist, in den letzten $l$ Zellen das Kreuzprodukt von jeweils $\lceil m/l \rceil$ Zellen zu betrachten. Bei diesem Vorgehen stellen sich zwei Probleme ein: Zum einen kann es passieren, daß $l \cdot \lceil m/l \rceil > m$ ist. Dann müssen die $m$ Zellen aber geeignet unter den $l$ Zellen aufgeteilt werden, was vermutlich zusätzlichen Aufwand in Form von zusätzlichen Zuständen erfordert. Andererseits führt die Betrachtung des Kreuzproduktes dazu, daß mehrere Zellen des gegebenen Automaten in einer Zelle des neuen Automaten zusammengefaßt werden. Insbesondere wird die erste Zelle, in der die Akzeptierung der Eingabe entschieden wird, dann in einer $n$-ten Zelle mit $n > 1$ simuliert. Auch hier ist vermutlich zusätzlicher Verwaltungsaufwand notwendig, um die erste Zelle des gegebenen Automaten auch in der ersten Zelle des neuen Automaten simulieren zu können.

## 6.4 Untersuchung der Zellenzahl

Aus Kapitel 4 wissen wir von der Existenz eines nichtrekursiven Tradeoff zwischen DFA und Realzeit-OCA. Da $k$C-OCA in DFA umgewandelt werden können, haben wir daher auch einen nichtrekursiven Tradeoff zwischen $k$C-OCA und Realzeit-OCA. Wenn wir also von einer festen Zellenzahl zu einer Zellenzahl übergehen, die der Länge der Eingabe entspricht, dann kann es passieren, daß sich der Größenzuwachs nicht mehr durch eine rekursive Funktion beschränken läßt.

In diesem Abschnitt werden wir kurz untersuchen, wie sich die Beschreibungsgröße zwischen $k$C-OCA und $k'$C-OCA verändern kann, wenn $k'$ größer als $k$ ist. Welche Einsparungen sind möglich, wenn mehr Zellen zur Verfügung gestellt werden? Wie groß können die Einsparungen sein, wenn ein $k$C-OCA in einen $k'C$-OCA mit $k' > k$ eingebettet wird? Oder gibt es Fälle, in denen zusätzliche Ressourcen die benötigte Zustandszahl nicht verringern können?

Die letzte Frage läßt sich mit dem Ergebnis aus Lemma 6.12 einfach beantworten. Wir hatten dort gezeigt, daß jeder $k$C-OCA für die Sprache $L_p$ mindestens $p+1$ Zustände haben muß. Dies galt unabhängig von der jeweiligen Zellenzahl $k$. Also hilft in diesem Fall keine Vergrößerung der Zellenzahl, um die Anzahl der Zustände zu verringern.

Die ersten beiden Fragen lassen sich leider nicht so einfach beantworten, da wir gegenwärtig nicht wissen, ob ein $k$C-OCA mit $n$ Zuständen in einen $(k+1)$C-OCA, der ebenfalls $n$ Zustände hat, eingebettet werden kann.

Ein naheliegendes Vorgehen, um einen $k$C-OCA in einen $(k+1)$C-OCA einzubetten und die Anzahl der Zustände nicht zu erhöhen, ist folgendes: Wir nehmen in dem $(k+1)$C-OCA einfach die Überführungsfunktion des $k$C-OCA und senden akzeptierende Zustände nach links zur linken Randzelle. Leider kann dieses Vorgehen fehlschlagen. Dazu betrachten wir die Konstruktion für die Sprache $L_{n,k}$ aus Beispiel 6.5. Wir beobachten, daß ein $k$C-OCA $A$, der $L_{n,k}$ akzeptiert, und ein $(k+1)$C-OCA, der $L_{n,k+1}$ akzeptiert, identische Überführungsfunktionen $\delta_r$ und $\delta$ besitzen. Wir wollen nun die Sprache $L_{n,k}$ mit einem $(k+1)$C-OCA $A'$ mit $n$ Zuständen akzeptieren. Wenn wir als Überführungsfunktionen von $A'$ die Überführungsfunktionen von $A$ definieren, dann ist $T(A') \neq L_{n,k}$.

Obwohl wir also nicht wissen, ob jeder $k$C-OCA mit $n$ Zuständen in einen $(k+1)$C-OCA mit $n$ Zuständen umgewandelt werden kann, können wir trotzdem zeigen, daß es Sprachen gibt, die von einem $(k+1)$C-OCA mit $n$ Zuständen akzeptiert werden, aber von keinem $k$C-OCA mit $n$ Zuständen. Mit anderen Worten: $k+1$ ist die minimale Zellenzahl, so daß diese Sprachen von einem eingeschränkten OCA mit $n$ Zuständen akzeptiert werden können. Dazu benötigen wir zunächst folgendes Lemma:

**Lemma 6.46.** Für $n \geq 3$ und $1 \leq k \leq n$ gilt: $(n+1)^k \leq n^{k+1}$ und $(n+1)^{k-i} \leq n^{k+1-i}$ für $0 \leq i \leq k$.

*Beweis.* Die erste Behauptung wird durch eine Induktion nach $n$ bewiesen:

Induktionsverankerung: Es sei $n = 3$. Dann gilt für $k = 1$: $(3+1)^1 = 4 \leq 3^{1+1} = 9$, für $k = 2$: $(3+1)^2 = 16 \leq 3^{2+1} = 27$ und für $k = 3$: $(3+1)^3 = 64 \leq 3^{3+1} = 81$.

Induktionsschritt: Zu zeigen ist folgendes: $(n+2)^k \leq (n+1)^{k+1}$

Gemäß der Formel $(x+y)^k = \sum_{i=0}^{k} \binom{k}{i} x^{k-i}y^i$ können wir $(n+2)^k = ((n+1)+1)^k$ und $(n+1)^{k+1}$ folgendermaßen schreiben:

$$\begin{aligned}(n+2)^k &= (n+1)^k + k(n+1)^{k-1} + \binom{k}{2}(n+1)^{k-2} + \ldots + \binom{k}{k-1}(n+1) + 1 + 0\\ (n+1)^{k+1} &= n^{k+1} + (k+1)n^k + \binom{k+1}{2}n^{k-1} + \ldots + \binom{k+1}{k-1}n^2 + (k+1)n + 1\end{aligned}$$

Da $\binom{k}{i} \leq \binom{k+1}{i}$ für $0 \leq i \leq k$ gilt, können wir unter Ausnutzung der Induktionsannahme zeigen, daß jeder Summand der oberen Gleichung kleiner oder gleich dem entsprechenden Summand der unteren Gleichung ist. Damit ist $(n+2)^k \leq (n+1)^{k+1}$ und die erste Ungleichung ist bewiesen. Zum Beweis der zweiten Ungleichung beobachten wir:

$$(n+1)^k \leq n^{k+1} \Leftrightarrow (n+1)^{k-i}(n+1)^i \leq n^i n^{k+1-i} \Leftrightarrow (n+1)^{k-i} \leq \left(\frac{n}{n+1}\right)^i n^{k+1-i} \leq n^{k+1-i},$$

da aus $\frac{n}{n+1} \leq 1$ die Abschätzung $\left(\frac{n}{n+1}\right)^i \leq 1$ für alle $i \geq 1$ folgt. □

**Satz 6.47.** Für $n \geq 4$ und $k \leq n$ gibt es eine Sprache $L(n,k)$, die von einem $k$C-OCA mit $n$ Zuständen akzeptiert werden kann. Aber für $1 \leq j < k$ gilt: Jeder $j$C-OCA für $L(n,k)$ benötigt mehr als $n$ Zustände.

*Beweis.* Es sei $m = n-1$ und $L(n,k) = L_{n-1,k} = L_{m,k}$. Nach Lemma 6.9 gibt es einen $k$C-OCA für die Sprache $L(n,k)$ mit $n = m+1$ Zuständen. Da $j < k$ ist, gilt $j+1+i = k \Leftrightarrow j = k-i-1$ mit $0 \leq i \leq k-2$. Zum Widerspruchsbeweis nehmen wir nun an, daß die Sprache $L(n,k)$ von einem $l$C-OCA $A$ mit $m+1$ Zuständen akzeptiert wird. Nach Lemma 6.6 kann $A$ in einen DFA $M$ mit $m'$ Zuständen umgewandelt werden. $m'$ kann folgendermaßen abgeschätzt werden:

$$m' \leq (m+1)^j - (m+1)^{j-1} + 1 \leq (m+1)^j = (m+1)^{k-i-1} \leq m^{k+1-i-1} = m^{k-i} \leq m^k$$

Das steht im Widerspruch zu Lemma 6.8, in dem gezeigt wird, daß $m' \geq m^k + 1$ gilt. □

Wir haben in obigem Beweis ausgenutzt, daß es einen $k$C-OCA mit $m+1$ Zuständen für die Sprache $L_{m,k}$ gibt, und daß ein $k$C-OCA mit $n$ Zuständen in einen äquivalenten DFA mit höchstens $n^k - n^{k-1} + 1$ Zuständen umgewandelt werden kann. Gleiche Aussagen gelten aber nach den Lemmata 6.17, 6.16, 6.37 und 6.36 auch für die Modelle $k$C-CA, $k$C-OCA$_t$ und $k$C-OCA$(l)$ mit $1 \leq l \leq k$. Daher erhalten wir unmittelbar folgendes Korollar:

**Korollar 6.48.** Für $n \geq 4$ und $k \leq n$ gibt es eine Sprache $L(n,k)$, die von einem $k$C-CA ($k$C-OCA$_t$, $k$C-OCA$(l)$) mit $n$ Zuständen akzeptiert werden kann. Aber für $1 \leq j < k$ gilt: Jeder $j$C-CA ($j$C-OCA$_t$, $j$C-OCA$(l)$) für $L(n,k)$ benötigt mehr als $n$ Zustände.

## 6.5 Minimierung von Zellularautomaten mit beschränkter Zellenzahl

In diesem Abschnitt soll kurz das praktische Problem der Minimierung eines gegebenen Zellularautomaten mit beschränkter Zellenzahl diskutiert werden. Wir wissen aus Satz 4.25, daß

es für unbeschränkte Zellularautomaten keinen Minimierungsalgorithmus gibt. Die in dieser Arbeit betrachteten eingeschränkten Modelle akzeptieren gerade die regulären Sprachen, die oftmals durch deterministische oder nichtdeterministische endliche Automaten beschrieben werden. Für DFA gibt es einen effizienten Minimierungsalgorithmus von Hopcroft [Hop71], der in Zeit $O(n \log n)$ arbeitet, wobei $n$ die Anzahl der Zustände des gegebenen DFA ist. Für NFA gibt es ebenfalls einen Minimierungsalgorithmus, allerdings gibt es vermutlich keinen effizienten Minimierungsalgorithmus, da das Minimierungsproblem für NFA nach [JR93] PSPACE-vollständig ist.

Die Frage nach der Existenz eines (effizienten) Minimierungsalgorithmus für Zellularautomaten mit beschränkter Zellenzahl ist von großem praktischem Interesse, denn ein solcher Algorithmus würde ein Werkzeug liefern, eine gegebene reguläre Sprache optimal zu parallelisieren. Das heißt, wir können einen gegebenen (komplexen) DFA algorithmisch in eine feste Anzahl (einfacher) identischer, parallel arbeitender DFA zerlegen.

Wir erhalten folgende Ergebnisse: Zunächst können wir einen Minimierungsalgorithmus für Zellularautomaten mit beschränkter Zellenzahl angeben. Weiterhin zeigen wir, daß ein minimaler Zellularautomat nicht notwendigerweise eindeutig sein muß. Das deutet darauf hin, daß es vermutlich keinen effizienten Minimierungsalgorithmus gibt.

**Satz 6.49.** Es gibt einen Algorithmus, der einen gegebenen $k$C-OCA ($k$C-OCA$_t$, $k$C-CA, $k$C-OCA$(l)$) $A$ in einen äquivalenten $k$C-OCA ($k$C-OCA$_t$, $k$C-CA, $k$C-OCA$(l)$) $A'$ umwandelt, so daß $A'$ eine minimale Zustandszahl besitzt.

*Beweis.* Wir führen den Beweis nur für $k$C-OCA. Für $k$C-OCA$_t$, $k$C-CA und $k$C-OCA$(l)$ verläuft der Beweis analog. Wir beschreiben einen „brute force"-Algorithmus. Zunächst wandeln wir $A$ gemäß Lemma 6.6 in einen DFA $M$ um. Anschließend listen wir alle $k$C-OCA $A_1, \ldots, A_m$ mit $|A_i| < |A|$ auf. Dann wird für jedes $i \in \{1, \ldots, m\}$ der $k$C-OCA $A_i$ in einen DFA $M_i$ umgewandelt und die Äquivalenz $T(M) = T(M_i)$ der beiden DFA wird getestet. Gibt es kein $i \in \{1, \ldots, m\}$, so daß $T(M_i) = T(M)$ ist, dann muß $A$ bereits die minimale Größe besessen haben und wir geben $A$ zurück. Anderenfalls haben wir eine endliche Menge $\mathcal{M}$ äquivalenter $k$C-OCA $A_i$, die kleiner als $A$ sind, gefunden. Wir wählen einen Automaten $A' \in \mathcal{M}$ minimaler Größe und geben $A'$ zurück. □

**Satz 6.50.** Ein minimaler $k$C-OCA ist nicht notwendigerweise eindeutig.

*Beweis.* In Lemma 6.12 wird gezeigt, daß jeder $k$C-OCA für $L_p$ mindestens $p + 1$ Zustände benötigt. Wir geben nun zwei nichtisomorphe $k$C-OCA für $L_2$ mit drei Zuständen an. Die Verallgemeinerung der Konstruktion für Primzahlen $p \geq 3$ ist offensichtlich.

(1) Wir zählen in der Kommunikationszelle modulo 2. Sobald die Eingabe gelesen ist und der Rest 1 beträgt, senden wir einen akzeptierenden Zustand $g$ mit maximaler Geschwindigkeit nach links. Anderenfalls wird die Berechnung blockiert.

$A_1 = (\{0, 1, g\}, \{a\}, 0, \triangledown, k, \delta_r, \delta, \{g\})$, wobei

| $\delta$ | 0 | 1 | $g$ |
|---|---|---|---|
| 0 | 0 | 1 | $g$ |
| 1 | 0 | 1 | $g$ |
| $g$ | · | · | $g$ |

und

| $\delta_r$ | $a$ | $\triangledown$ |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | $g$ |
| $g$ | · | $g$ |

(2) Die Eingabe wird in die Kommunikationszelle geschoben, wobei $a$ dem Zustand 1 entspricht und $\triangledown$ dem Zustand 0 entspricht. Die vorletzte Zelle zählt nun modulo 2 und arbeitet wie die Kommunikationszelle in $A_1$.

$A_2 = (\{0, 1, g\}, \{a\}, 0, \triangledown, k, \delta_r, \delta, \{g\})$, wobei

| $\delta$ | 0 | 1 | $g$ |
|---|---|---|---|
| 0 | 0 | 1 | $g$ |
| 1 | $g$ | 0 | $g$ |
| $g$ | $\cdot$ | $\cdot$ | $g$ |

und

| $\delta_r$ | $a$ | $\triangledown$ |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 0 |
| $g$ | $\cdot$ | $\cdot$ |

□

**Satz 6.51.** Ein minimaler $k$C-OCA$_t$ ist nicht notwendigerweise eindeutig.

*Beweis.* Wir wissen nach Lemma 6.17, daß jeder $k$C-OCA$_t$ für die Sprache $L_{n,k}$ mindestens $n+1$ Zustände benötigt. Um die Behauptung zu beweisen, müssen wir daher nur zwei nichtisomorphe $k$C-OCA$_t$ für die Sprache $L_{n,k}$ mit $n+1$ Zuständen finden. Wir geben im folgenden für $n = 3$ zwei solche $k$C-OCA$_t$ an. Die Konstruktion kann analog für beliebiges $n \geq 4$ durchgeführt werden.

Im ersten $k$C-OCA$_t$ wird ein ternärer Zähler ähnlich wie in Beispiel 6.5 installiert. Wir beobachten, daß die Kommunikationszelle periodisch die Zustände $1, 2, +, 1, 2, +, \ldots$ annimmt. Außerdem ist die Berechnung in der Kommunikationszelle unabhängig vom linken Nachbarn. Die restlichen Zellen hängen von ihrem rechten Nachbarn insofern ab, daß ein + ein Erhöhen des gegenwärtigen Zustands bedingt. Alle anderen Zustände des rechten Nachbarn verändern den aktuellen Zustand nicht.

Wir konstruieren nun den zweiten $k$C-OCA$_t$ für $L_{n,k}$ mit $n+1$ Zuständen, indem wir die Reihenfolge der Zustände in der Kommunikationszelle vom linken Nachbarn abhängen lassen: Wir zählen in der Kommunikationszelle vorwärts $(1, 2, +)$, solange der linke Nachbar 0,2 oder + ist. Hat der linke Nachbar den Zustand 1, wird in der Kommunikationszelle rückwärts gezählt $(2, 1, +)$. Die Überführungsfunktion $\delta$ für die restlichen Zellen ist identisch zur Überführungsfunktion $\delta$ des ersten $k$C-OCA$_t$ und simuliert somit einen ternären Zähler. Da lediglich der Zustand + in der Kommunikationszelle Auswirkungen auf die restlichen Zellen hat, akzeptiert der zweite $k$C-OCA$_t$ ebenfalls die Sprache $L_{n,k}$. Damit haben wir zwei nichtisomorphe $k$C-OCA$_t$ für $L_{n,k}$ mit vier Zuständen. □

Indem wir in obigem Beweis $k$C-OCA$_t$ durch $k$C-CA oder $k$C-OCA$(l)$ mit $1 \leq l \leq k$ ersetzen, erhalten wir unmittelbar folgendes Korollar:

**Korollar 6.52.** Ein minimaler $k$C-CA ($k$C-OCA$(l)$) ist nicht notwendigerweise eindeutig.

## 6.6 Zusammenfassung

Wir haben in diesem Kapitel die Beschreibungskomplexität eingeschränkter Zellularautomaten mit einer festen Zellenzahl und unterschiedlichem Grad an zweiseitiger Kommunikation untersucht. Bezüglich der generativen Mächtigkeit beschreiben alle Modelle die regulären Sprachen. Bei einer festen Zellenzahl $k$ implizierte die Umwandlung in einen DFA einen

polynomiellen Größenzuwachs vom Grad $k$. Umgekehrt konnten DFA von eingeschränkten Zellularautomaten simuliert werden. Falls keine zweiseitige Kommunikation erlaubt war, waren keine Einsparungen möglich. In allen anderen Fällen waren quadratische Einsparungen möglich. Im Fall unärer Sprachen konnten darüber hinaus noch größere Einsparungen erzielt werden.

Weiterhin wurde untersucht, mit welchen Auswirkungen auf die Anzahl der Zustände der Grad der zweiseitigen Kommunikation reduziert werden kann. Als grobe Faustregel kann man zusammenfassen, daß bei einer Reduzierung vom Grad $m$ auf den Grad $l$ der Größenzuwachs durch ein Polynom vom Grad $m - l + 1$ beschränkt ist. In manchen Fällen konnten genauere obere Schranken bewiesen werden. Mit einer Ausnahme konnte auch gezeigt werden, daß diese oberen Schranken asymptotisch scharf sind.

Da sämtliche in diesem Kapitel betrachteten Zellularautomaten in polynomieller Zeit in DFA umgewandelt werden können, sind alle Entscheidbarkeitsfragen, die für DFA in polynomieller Zeit gelöst werden können, auch für eingeschränkte Zellularautomaten in polynomieller Zeit lösbar. Das ist eine wesentliche Verbesserung zum unbeschränkten Modell, da es dort fast keine entscheidbaren Fragen gab. Ebenfalls können eingeschränkte Zellularautomaten algorithmisch minimiert werden. Die Zeitkomplexität des Minimierungsproblems ist derzeit offen.

# Kapitel 7

# Diskussion und Ausblick

In dieser Arbeit haben wir die Beschreibungskomplexität unbeschränkter und beschränkter Zellularautomaten untersucht. Im unbeschränkten Fall haben wir parallele Modelle mit paralleler und sequentieller Verarbeitung der Eingabe betrachtet und haben nichtrekursive Tradeoffs zwischen allen zellularen Modellen erhalten, deren Sprachklassen entweder echt ineinander enthalten oder unvergleichbar sind. Zusammenfassend können wir hinsichtlich der Beschreibungskomplexität sagen:

- Parallele Modelle sind besser als sequentielle Modelle.
- Parallele Eingabe ist besser als sequentielle Eingabe.
- Linearzeit ist besser als Realzeit.
- Beidseitige Kommunikation ist besser als einseitige Kommunikation.
- Unvergleichbare Sprachklassen sind auch unvergleichbar bezüglich der Beschreibungskomplexität.

Die Existenz eines nichtrekursiven Tradeoffs bedeutet, daß zwischen den jeweiligen Automatenmodellen Einsparungen beliebiger Größe erzielt werden können. Das heißt beispielsweise für ein Problem, das durch einen Realzeit-OCA beschrieben werden kann, daß eine Beschreibung des Problems durch einen Linearzeit-OCA oder durch einen Realzeit-CA wesentlich kleiner sein kann. Stellt man also mehr Zeit als notwendig oder mehr Kommunikation als nötig zur Verfügung, kann das zu enormen Größeneinsparungen führen. Für praktische Anwendungen von Zellularautomaten kann also der Einsatz von mehr Ressourcen als notwendig zu sehr kurzen Beschreibungen führen. Die Kehrseite dieser nichtrekursiven Tradeoffs ist allerdings, daß diese sehr kurzen Beschreibungen nicht algorithmisch erzielt werden können. Zwar kann beispielsweise ein Realzeit-OCA in einen Realzeit-CA ohne Veränderung der Anzahl der Zustände umgewandelt werden; aber für Realzeit-CA gibt es keinen Minimierungsalgorithmus, der einen Realzeit-CA mit maximalen Einsparungen bezüglich der Zustandszahl konstruiert. Wir haben weiterhin gezeigt, daß für alle unbeschränkten Zellularautomatenmodelle fast alle Entscheidbarkeitsfragen unentscheidbar und nicht semientscheidbar sind. Das heißt, daß viele praktische Fragen wie das Testen, ob eine durch einen Zellularautomaten gegebene Spezifikation eines Problems die leere Menge, eine endliche oder eine reguläre Menge

ergibt, nicht algorithmisch lösbar sind. Auch ein Äquivalenztest für Zellularautomaten, das heißt die Frage, ob zwei gegebene Zellularautomaten das gleiche Problem beschreiben, kann nicht algorithmisch umgesetzt werden. Vielmehr können diese Entscheidbarkeitsfragen oft nur durch *ad hoc*-Ansätze gelöst werden, was mitunter recht aufwendig sein kann. Als weitere Konsequenz aus den Unentscheidbarkeitsergebnissen ergab sich, daß es auch Werkzeuge wie ein Pumpinglemma oder einen Minimierungsalgorithmus in allen untersuchten zellularen Modellen nicht geben kann.

Für zwei Zellularautomatenklassen, deren Sprachklassen zwar ineinander enthalten sind, die Echtheit der Inklusion aber derzeit unbekannt ist, können wir keine Aussagen hinsichtlich der Existenz nichtrekursiver Tradeoffs treffen. Hier wären zunächst die Sprachklassen bezüglich ihrer generativen Mächtigkeit zu trennen oder deren Äquivalenz zu zeigen, bevor man die Beschreibungskomplexität der Modelle untersucht. Beschreiben zwei Zellularautomatenklassen die gleiche Sprachklasse und kann ein Automat der einen Klasse algorithmisch in einen Automaten der anderen Klasse umgewandelt werden, dann ergibt sich, im Gegensatz zu den bisher behandelten Fällen, eine rekursive obere Schranke. Trotzdem ist es schwierig, untere Schranken zu beweisen, da es an Methoden fehlt, um zu zeigen, daß mindestens eine gewisse Anzahl an Zuständen für die Akzeptierung gewisser Sprachen notwendig ist. Mit ein Grund dafür könnten die in diesen Sprachklassen fehlenden Werkzeuge wie ein Pumpinglemma sein.

In [KK03] werden die Sprachklassen $\mathcal{L}_{rt}(\text{OCA})$ und $\mathcal{L}_{lt}(\text{OCA})$ genauer untersucht und als Ergebnis wird eine unendliche echte Hierarchie zwischen den Sprachklassen bewiesen, die von OCA mit Zeitkomplexitäten zwischen Realzeit und Linearzeit erzeugt werden. Im einzelnen gibt es das Ergebnis, daß $\mathcal{L}_{n+r_2(n)}(\text{OCA}) \subset \mathcal{L}_{n+r_1(n)}(\text{OCA})$ gilt für zwei steigende Funktionen $r_1, r_2 : \mathbb{N} \to \mathbb{N}$ mit $r_2 \log r_2 \in o(r_1)$ und mit einer Konstruierbarkeitsbedingung an $r_1^{-1}$. Hier wäre aus Sicht der Beschreibungskomplexität interessant, ob zwischen den Automatenklassen mit den entsprechenden Zeitkomplexitäten rekursive oder nichtrekursive Tradeoffs existieren. Eine ähnliche unendliche echte Hierarchie ist auch zwischen $\mathcal{L}_{rt}(\text{IA})$ und $\mathcal{L}_{lt}(\text{IA})$ bekannt [BKK00a]. Auch hier könnte untersucht werden, ob rekursive oder nichtrekursive Tradeoffs zwischen IA mit entsprechenden Zeitkomplexitäten existieren.

Nicht zuletzt aufgrund der Unhandlichkeit des unbeschränkten Modells haben wir im zweiten Teil dieser Arbeit Zellularautomaten mit einer festen Zellenzahl behandelt. Diese Beschränkung reduziert die generative Mächtigkeit des Modells auf die Klasse der regulären Sprachen. Wir haben beschränkte Modelle mit unterschiedlichen Graden an zweiseitiger Kommunikation betrachtet. Im Vergleich mit DFA können alle betrachteten Modelle polynomielle Einsparungen vom Grad $k$ liefern, wobei $k$ die Anzahl der Zellen ist. Umgekehrt können DFA mit einer Ausnahme immer von eingeschränkten Zellularautomaten mit quadratischen Einsparungen simuliert werden. Weiterhin wurde die Existenz inhärent sequentieller Sprachen gezeigt. Das heißt, es gibt Sprachen, deren Beschreibungsgröße durch den Einsatz zusätzlicher Zellen nicht verringert werden kann. Bei der Reduktion von zweiseitiger Kommunikation ergibt sich meistens ein polynomieller Zuwachs, wobei der Grad des Polynoms ungefähr der Anzahl der entfernten Zellen mit zweiseitiger Kommunikation entspricht. Alle in dieser Arbeit vorgestellten Umwandlungen zwischen verschiedenen eingeschränkten Modellen mit unterschiedlichem Grad an zweiseitiger Kommunikation sind algorithmisch durchzuführen und liefern polynomielle obere Schranken. Ist fast allen Fällen konnte auch gezeigt werden, daß diese Schranken asymptotisch scharf oder zumindest fast scharf sind.

Aufgrund der polynomiellen oberen Schranken und der konstruktiven Umwandlungsalgorithmen sind alle untersuchten zellularen Modelle in polynomieller Zeit in einen DFA umzuwandeln. Daher sind dann alle Fragen, die für DFA in polynomieller Zeit entscheidbar sind, auch für die eingeschränkten Zellularautomaten in polynomieller Zeit entscheidbar. Ein weiterer wichtiger Gesichtspunkt könnte die genaue Bestimmung der Zeitkomplexität der Entscheidbarkeitsfragen sein. Beispielsweise ist Leere, Universalität und Endlichkeit für DFA in linearer Zeit zu entscheiden [Weg93]. Nach Umwandlung eines $k$C-OCA in einen DFA sind diese Fragen dann natürlich in Zeit $O(n^k)$ zu lösen. Offen ist derzeit, ob es bessere Algorithmen für eingeschränkte Zellularautomaten gibt, die beispielsweise ebenfalls in linearer Zeit arbeiten.

Auch die algorithmische Zustandsminimierung von eingeschränkten Zellularautomaten ist von praktischem Interesse, da dadurch eine optimale Parallelisierung eines Problems algorithmisch erreicht werden kann. Für DFA sind effiziente Minimierungsalgorithmen bekannt. Für NFA ist das Problem PSPACE-vollständig. Weiterhin ist bekannt, daß bereits sehr wenig Nichtdeterminismus ausreicht, um das Minimierungsproblem zu erschweren. In [Mal03a] wird gezeigt, daß das Minimierungsproblem für NFA mit sehr wenig Nichtdeterminismus bereits NP-vollständig ist. Die hier betrachteten Zellularautomaten sind zwar ein deterministisches Modell, allerdings sind Minimalautomaten nicht notwendigerweise auch eindeutig, da Berechnungen nichtisomorph auf unterschiedliche Zellen verteilt werden können. Daher sind die in dieser Arbeit diskutierten beschränkten Zellularautomaten zwar algorithmisch zu minimieren, aber es gibt vermutlich keinen effizienten Minimierungsalgorithmus. Die NP-Vollständigkeit des Minimierungsproblems sollte mit einem ähnlichen Ansatz wie in [Mal03a] zu beweisen sein. Darüber hinaus könnte es sich lohnen zu untersuchen, unter welchen Voraussetzungen Minimieren effizient wird. In welcher Weise muß das Modell verändert oder abgeschwächt werden, um ein effizientes Minimieren zu ermöglichen?

Wir haben bislang sowohl im unbeschränkten als auch im beschränkten Fall deterministische zellulare Modelle betrachtet. Nun könnte es sehr interessant sein, insbesondere im beschränkten Fall, diese Modelle um die Ressource Nichtdeterminismus zu bereichern. Beispielsweise könnte man $k$C-CA mit NFA statt DFA in den einzelnen Zellen oder $k$C-OCA mit einer nichtdeterministischen Kommunikationszelle betrachten. Iterative Arrays mit nichtdeterministischer Kommunikationszelle führen zu einer größeren generativen Mächtigkeit [Kle99]. Wie verändert sich die Beschreibungskomplexität im Fall einer beschränkten Zellenzahl? Was verändert sich, wenn man eine nichtdeterministische, aber eindeutige Kommunikationszelle oder eine Kommunikationszelle mit beschränktem Nichtdeterminismus betrachtet? Generell stellt sich die Frage nach den Wechselwirkungen zwischen Parallelität und Nichtdeterminismus: Kann die eine Ressource die andere ersetzen? Wenn ja, wie verändert sich dabei die Beschreibungsgröße?

Eine interessante Erweiterung könnten auch Modelle sein, in denen man nicht eine feste oder eine lineare Zellenzahl, sondern eine sublineare Zellenzahl in Abhängigkeit von der Länge der Eingabe zuläßt. Welche Sprachklassen kann ein Zellularautomat mit einer logarithmischen Zellenzahl akzeptieren? Es stellt sich die Frage, ob man Ergebnisse und Methoden über logarithmisch platzbeschränkte Turingmaschinen auf entsprechende Zellularautomaten übertragen kann. Weiterhin ist bekannt, daß manche sublogarithmisch platzbeschränkte Turingmaschinen nur noch die regulären Sprachen akzeptieren [Sze94]. In diesem Zusammenhang könnte man untersuchen, ob Zellularautomaten mit sublogarithmischer Zellenzahl ebenfalls

nur reguläre Sprachen akzeptieren können. Falls ja, dann wäre wiederum die Beschreibungskomplexität im Vergleich mit DFA interessant.

Schließlich könnte man auch die Beschreibungskomplexität beschränkter und unbeschränkter Versionen ganz anderer zellularer Modelle betrachten. Beispielsweise könnte man zweidimensionale Zellularautomaten [Smi71, Ter99], systolische Baumautomaten [Gru90] oder zellulare neuronale Netze [Chu98] untersuchen.

# Literaturverzeichnis


[Bir93] Jean-Camille Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Math. Systems Theory*, 26(3):237–269, 1993.

[BKK00a] Thomas Buchholz, Andreas Klein, and Martin Kutrib. Iterative arrays with small time bounds. In *Mathematical foundations of computer science (MFCS 2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 243–252. Springer-Verlag, Berlin, 2000.

[BKK00b] Thomas Buchholz, Andreas Klein, and Martin Kutrib. On tally languages and generalized interacting automata. In Grzegorz Rozenberg and Wolfgang Thomas, editors, *Developments in Language Theory IV. Foundations, Applications, and Perspectives*, pages 316–325. World Scientific Publishing, Singapore, 2000.

[BKK02] Thomas Buchholz, Andreas Klein, and Martin Kutrib. On interacting automata with limited nondeterminism. *Fund. Inform.*, 52:15–38, 2002.

[BMS01] Stefania Bandini, Giancarlo Mauri, and Roberto Serra. Cellular automata: From a theoretical parallel computational model to its application to complex systems. *Parallel Comput.*, 27(5):539–553, 2001.

[Bun95] Gerhard Buntrock. Einige Bemerkungen zum Pumpen. In Martin Kutrib and Thomas Worsch, editors, *5. Theorietag „Automatentheorie und Formale Sprachen"*, pages 25–27. Universität Gießen, Gießen, 1995.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. Assoc. Comput. Mach.*, 30(2):323–342, 1983.

[CC84] Christian Choffrut and Karel Culik, II. On real-time cellular automata and trellis automata. *Acta Inform.*, 21(4):393–407, 1984.

[CGS86] Karel Culik, II, Jozef Gruska, and Arto Salomaa. Systolic trellis automata: stability, decidability and complexity. *Inform. and Control*, 71(3):218–230, 1986.

[Chr86] Marek Chrobak. Finite automata and unary languages. *Theoret. Comput. Sci.*, 47(2):149–158, 1986.

[Chu98] Leon O. Chua. *CNN: a paradigm for complexity*, volume 31 of *World Scientific Series on Nonlinear Science. Series A: Monographs and Treatises*. World Scientific Publishing, Singapore, 1998.

[Cod68] Edgar F. Codd. *Cellular automata*. Academic Press, New York, 1968.

[Col69] Stephen N. Cole. Real-time computation by $n$-dimensional iterative arrays of finite-state machines. *IEEE Trans. Computers*, C-18:349–365, 1969.

[CPPR87] Germinal Cocho, Rafael Pérez-Pascual, and José L. Rius. Discrete systems, cell-cell interactions and color pattern of animals. I. Conflicting dynamics and pattern formation. *J. Theoret. Biol.*, 125(4):419–435, 1987.

[CPPRS87] Germinal Cocho, Rafael. Pérez-Pascual, José L. Rius, and F. Soto. Discrete systems, cell-cell interactions and color pattern of animals. II. Clonal theory and cellular automata. *J. Theoret. Biol.*, 125(4):437–447, 1987.

[CVDKP94] Erzsébet Csuhaj-Varjú, Jürgen Dassow, Jozef Kelemen, and Gheorghe Păun. *Grammar systems*. Gordon and Breach Science Publishers, Yverdon, 1994.

[CVMVMV00] Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, Victor Mitrana, and György Vaszil. Parallel communicating pushdown automata systems. *Internat. J. Found. Comput. Sci.*, 11(4):633–650, 2000.

[DP91] Jürgen Dassow and Gheorghe Păun. On the succinctness of descriptions of context-free languages by cooperating/distributed grammar systems. *Comput. Artificial Intelligence*, 10(6):513–527, 1991.

[DPR97] Jürgen Dassow, Gheorghe Păun, and Grzegorz Rozenberg. Grammar systems. In *Handbook of formal languages, Vol. 2*, pages 155–213. Springer-Verlag, Berlin, 1997.

[DPS97] Jürgen Dassow, Gheorghe Păun, and Arto Salomaa. Grammars with controlled derivations. In *Handbook of formal languages, Vol. 2*, pages 101–154. Springer-Verlag, Berlin, 1997.

[EPR97] Andrzej Ehrenfeucht, Gheorghe Păun, and Grzegorz Rozenberg. Contextual grammars and formal languages. In *Handbook of formal languages, Vol. 2*, pages 237–293. Springer-Verlag, Berlin, 1997.

[Gar95] Max Garzon. *Models of massive parallelism*. Springer-Verlag, Berlin, 1995.

[GKK+02] Jonathan Goldstine, Martin Kappes, Chandra M. R. Kintala, Hing Leung, Andreas Malcher, and Detlef Wotschke. Descriptional complexity of machines with limited resources. *J.UCS*, 8(2):193–234, 2002.

[GKW90] Jonathan Goldstine, Chandra M. R. Kintala, and Detlef Wotschke. On measuring nondeterminism in regular languages. *Inform. and Comput.*, 86(2):179–194, 1990.

[GMNP97] Jozef Gruska, Angelo Monti, Margherita Napoli, and Domenico Parente. Succinctness of descriptions of SBTA-languages. *Theoret. Comput. Sci.*, 179(1-2):251–271, 1997.

[Gru90] Jozef Gruska. Synthesis, structure and power of systolic computations. *Theoret. Comput. Sci.*, 71(1):47–77, 1990.

[Har79] Juris Hartmanis. On the succinctness of different representations of languages. In *Automata, languages and programming (ICALP 1979)*, volume 71 of *Lecture Notes in Computer Science*, pages 282–288. Springer-Verlag, Berlin, 1979.

[Har80] Juris Hartmanis. On the succinctness of different representations of languages. *SIAM J. Comput.*, 9(1):114–120, 1980.

[Her97] Christian Herzog. Pushdown automata with bounded nondeterminism and bounded ambiguity. *Theoret. Comput. Sci.*, 181(1):141–157, 1997.

[Her99] Christian Herzog. *Die Rolle des Nichtdeterminismus in kontextfreien Sprachen.* PhD thesis, Johann Wolfgang Goethe-Universität, Frankfurt, 1999.

[HI68] Michael A. Harrison and Oscar H. Ibarra. Multi-tape and multi-head pushdown automata. *Inform. and Control*, 13:433–470, 1968.

[Hop71] John E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi, editor, *Theory of machines and computations*, pages 189–196. Academic Press, New York, 1971.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Co., Reading, Mass., 1979.

[JR93] Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

[KK03] Andreas Klein and Martin Kutrib. Fast one-way cellular automata. *Theoret. Comput. Sci.*, 295:233–250, 2003.

[Kle96] Reinhard Klemm. *Systems of communicating finite state machines as a distributed alternative to finite state machines.* PhD thesis, The Pennsylvania State University, 1996.

[Kle99] Andreas Klein. *Generalisierte Berechnungen in iterativen Arrays.* PhD thesis, Universität Gießen, 1999.

[KRS97] Lila Kari, Grzegorz Rozenberg, and Arto Salomaa. L systems. In *Handbook of formal languages, Vol. 1*, pages 253–328. Springer-Verlag, Berlin, 1997.

[Kut01a] Martin Kutrib. Automata arrays and context-free languages. In Carlos Martín-Vide and Victor Mitrana, editors, *Where Mathematics, Computer Science, Linguistics and Biology Meet*, pages 139–148. Kluwer Academic Publishers, Dordrecht, 2001.

[Kut01b] Martin Kutrib. Grundlagen der Zellularautomaten. Technical report, Institut für Informatik, Universität Gießen, 2001.

[Kut03] Martin Kutrib. On the descriptional power of heads, counters, and pebbles. In Erzsébet Csuhaj-Varjú, Chandra M.R. Kintala, Detlef Wotschke, and György Vaszil, editors, *Proceedings of the Fifth International Workshop on Descriptional Complexity of Formal Systems (DCFS 2003)*, pages 138–149. MTA SZTAKI, Budapest, Hungary, 2003.

[KW80] Chandra M. R. Kintala and Detlef Wotschke. Amounts of nondeterminism in finite automata. *Acta Inform.*, 13(2):199–204, 1980.

[Lan84] Christopher D. Langton. Self-reproduction in cellular automata. *Phys. D*, 10(1-2):135–144, 1984.

[Lei81] Ernst Leiss. Succinct representation of regular languages by Boolean automata. *Theoret. Comput. Sci.*, 13(3):323–330, 1981.

[LV93] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag, New York, 1993.

[Mal02] Andreas Malcher. Descriptional complexity of cellular automata and decidability questions. *J. Autom. Lang. Comb.*, 7(4):549–560, 2002.

[Mal03a] Andreas Malcher. Minimizing finite automata is computationally hard. In Zoltán Ésik and Zoltán Fülöp, editors, *Developments in Language Theory (DLT 2003)*, volume 2710 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, Berlin, 2003.

[Mal03b] Andreas Malcher. On one-way cellular automata with a fixed number of cells. *Fund. Inform.*, 58(3-4):355–368, 2003.

[Mal03c] Andreas Malcher. On two-way communication in cellular automata with a fixed number of cells. In Erzsébet Csuhaj-Varjú, Chandra M.R. Kintala, Detlef Wotschke, and György Vaszil, editors, *Proceedings of the Fifth International Workshop on Descriptional Complexity of Formal Systems (DCFS 2003)*, pages 162–173. MTA SZTAKI, Budapest, Hungary, 2003.

[Mal04a] Andreas Malcher. On the descriptional complexity of iterative arrays. *IEICE Transactions D*, E87-D(3):721–725, 2004.

[Mal04b] Andreas Malcher. On two-way communication in cellular automata with a fixed number of cells. *Theoret. Comput. Sci.*, 2004. to appear.

[MF71] Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In *IEEE Twelfth Annual Symposium on Switching and Automata Theory*, pages 188–191. IEEE, 1971.

[MT99] Jacques Mazoyer and Véronique Terrier. Signals in one-dimensional cellular automata. *Theoret. Comput. Sci.*, 217(1):53–80, 1999.

[MVMM02] Carlos Martín-Vide, Alexandru Mateescu, and Victor Mitrana. Parallel finite automata systems communicating by states. *Internat. J. Found. Comput. Sci.*, 13(5):733–749, 2002.

[Nak99] Katsuhiko Nakamura. Real-time language recognition by one-way and two-way cellular automata (extended abstract). In *Mathematical foundations of computer science (MFCS 1999)*, volume 1672 of *Lecture Notes in Computer Science*, pages 220–230. Springer-Verlag, Berlin, 1999.

[NM93] Martin A. Nowak and Robert M. May. The spatial dilemmas of evolution. *Internat. J. Bifur. Chaos Appl. Sci. Engrg.*, 3(1):35–78, 1993.

[Okh02] Alexander Okhotin. State complexity of linear conjunctive languages. In Jürgen Dassow, Maia Hoeberechts, Helmut Jürgensen, and Detlef Wotschke, editors, *Preproceedings of the Fourth International Workshop on Descriptional Complexity of Formal Systems (DCFS 2002)*, pages 256–270. University of Western Ontario, London, Ontario, Canada, 2002.

[Pău02] Gheorghe Păun. *Membrane computing*. Springer-Verlag, Berlin, 2002.

[RI89] Bala Ravikumar and Oscar H. Ibarra. Relating the type of ambiguity of finite automata to the succinctness of their representation. *SIAM J. Comput.*, 18(6):1263–1282, 1989.

[Ros66] Arnold L. Rosenberg. On multi-head finite automata. *IBM Journal of Research and Development*, 10:388–394, 1966.

[Sch78] Erik M. Schmidt. *Succinctness of descriptions of context-free, regular and finite languages*. PhD thesis, Cornell University, Ithaca, NY, 1978.

[Sei79] Steven R. Seidel. Language recognition and the synchronization of cellular automata. Technical Report 79-02, Department of Computer Science, University of Iowa, 1979.

[Smi70] Alvy R. Smith, III. Cellular automata and formal languages. In *IEEE Eleventh Annual Symposium on Switching and Automata Theory*, pages 216–224. IEEE, 1970.

[Smi71] Alvy R. Smith, III. Two-dimensional formal languages and pattern recognition by cellular automata. In *IEEE Twelfth Annual Symposium on Switching and Automata Theory*, pages 144–152. IEEE, 1971.

[SS77] Erik M. Schmidt and Thomas G. Szymanski. Succinctness of descriptions of unambiguous context-free languages. *SIAM J. Comput.*, 6(3):547–553, 1977.

[SS78] William J. Sakoda and Michael Sipser. Nondeterminism and the size of two-way finite automata. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 275–286. ACM, New York, 1978.

[Sze94] Andrzej Szepietowski. *Turing machines with sublogarithmic space*, volume 843 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.

[Ter95] Véronique Terrier. On real time one-way cellular array. *Theoret. Comput. Sci.*, 141(1-2):331–335, 1995.

[Ter99] Véronique Terrier. Two-dimensional cellular automata recognizer. *Theoret. Comput. Sci.*, 218(2):325–346, 1999.

[Val76] Leslie G. Valiant. A note on the succinctness of descriptions of deterministic languages. *Inform. and Control*, 32(2):139–145, 1976.

[vdLH92] Jan D. van der Laan and Paulien Hogeweg. Waves of crown-of-thorn starfish outbreaks—where do they come from? *Corel Reefs*, 11:207–213, 1992.

[Weg93] Ingo Wegener. *Theoretische Informatik*. Teubner, Stuttgart, 1993.

[Wei97] Jörg R. Weimar. *Simulation with cellular automata*. Logos-Verlag, Berlin, 1997.

# Zusammenfassung

Zellularautomaten sind ein massiv paralleles Berechnungsmodell, das aus sehr vielen identischen einfachen Prozessoren oder Zellen besteht, die homogen miteinander verbunden sind und parallel arbeiten. Es gibt Zellularautomaten in unterschiedlichen Ausprägungen. Beispielsweise unterscheidet man die Automaten nach der zur Verfügung stehenden Zeit, nach paralleler oder sequentieller Verarbeitung der Eingabe oder durch Beschränkungen der Kommunikation zwischen den einzelnen Zellen. Benutzt man Zellularautomaten zum Erkennen formaler Sprachen und betrachtet deren generative Mächtigkeit, dann kann bereits das einfachste zellulare Modell kontextsensitive Sprachen akzeptieren.

In dieser Arbeit wird die Beschreibungskomplexität von Zellularautomaten betrachtet. Es wird untersucht, wie sich die Beschreibungsgröße einer formalen Sprache verändern kann, wenn die Sprache mit unterschiedlichen Typen von Zellularautomaten oder sequentiellen Modellen beschrieben wird. Ein wesentliches Ergebnis im ersten Teil der Arbeit ist, daß zwischen zwei Automatenklassen, deren entsprechende Sprachklassen echt ineinander enthalten oder unvergleichbar sind, nichtrekursive Tradeoffs existieren. Das heißt, der Größenzuwachs beim Wechsel von einem Automatenmodell in das andere läßt sich durch keine rekursive Funktion beschränken.

Im zweiten Teil der Arbeit werden Zellularautomaten dahingehend beschränkt, daß nur eine feste Zellenzahl zugelassen ist. Zusätzlich werden Automaten mit unterschiedlichem Grad an bidirektionaler Kommunikation zwischen den einzelnen Zellen betrachtet, und es wird untersucht, welche Auswirkungen auf die Beschreibungsgröße unterschiedliche Grade an bidirektionaler Kommunikation haben können. Im Gegensatz zum unbeschränkten Modell können polynomielle und damit rekursive obere Schranken bei Umwandlungen zwischen den einzelnen Modellen bewiesen werden. Durch den Beweis unterer Schranken kann in fast allen Fällen auch die Optimalität der Konstruktionen belegt werden.