Generalisierte Berechnungen
in
iterativen Arrays

Inauguraldissertation
zur
Erlangung des Doktorgrades

der Naturwissenschaftlichen Fachbereiche
der Justus–Liebig–Universität Gießen

vorgelegt von
Andreas Klein
geboren in Lich

Gießen, 1999

D 26

Dekan :	Prof. Dr. Albrecht Beutelspacher
I. Berichterstatter :	PD Dr. Martin Kutrib
II. Berichterstatter :	Prof. Dr. Henner Kröger

Inhaltsverzeichnis

Abbildungsverzeichnis

Danksagungen

Ich möchte mich bei all denjenigen bedanken, die zur Ermöglichung dieser Arbeit beigetragen haben.

Mein besonderer Dank gilt Herrn PD Dr. Martin Kutrib, der mir Anregungen gab und jederzeit für fachliche Gespräche zur Verfügung stand. Durch sein Verständnis war es mir möglich, die Arbeit in relativ kurzer Zeit zu erstellen.

Für die Mühe, vorläufige Versionen dieser Arbeit zu lesen, bedanke ich mich ganz herzlich bei Thomas Buchholz.

Kapitel 1

Einleitung

Bereits 1936 hat A.M. Turing die berühmte Arbeit „On computable numbers, with an application to the Entscheidungsproblem“ [47] veröffentlicht. In dieser Arbeit stellt er ein formales Modell für Computer vor, das heute als Turing–Maschine bekannt ist. Turing–Maschinen wurden bald zur Grundlage vieler Untersuchungen in der theoretischen Informatik. Ein besonders interessanter Zweig ist die Komplexitätstheorie, in der Funktionen hinsichtlich des Arbeitsaufwands, der zu ihrer Berechnung erforderlich ist, untersucht werden. (Eine Einführung findet sich in [38].)

Es ist klar, daß die Zeit, die ein Algorithmus benötigt, von der verwendeten Maschine abhängt. Turing–Maschinen sind auf der einen Seite einfach genug, um als Grundlage für theoretische Untersuchungen zu dienen, auf der anderen Seite modellieren sie moderne Computer (mit einem Prozessor) hinreichend genau, so daß die gewonnenen Erkenntnisse für die Praxis relevant sind.

Die Leistung von sequentiellen Maschinen stößt heute bereits an physikalische Grenzen. Daher hat für den Bau von Supercomputern das Interesse an parallel verarbeitenden Maschinen zugenommen. Verfügte einer der ersten Parallelrechner — der ILLIAC IV — noch über nur 64 Prozessoren, so wurde mit dem Durchbruch der VLSI–Technologie der Bau von Rechneranlagen mit einigen zehntausend Prozessoren möglich (W.D. Hills [22]).

Vor diesem Hintergrund gewinnen abstrakte Modelle für Parallelrechner an Bedeutung. Das wichtigste dieser Modelle sind die Zellularräume, die von J. von Neumann erfunden wurden [35]. Daß solche formalen Modelle für die Anwendung interessant sind, zeigen die Arbeiten von T. Toffoli und N. Margolus [46] sowie K. Preston Jr. und M.J.B. Duff [37], in denen Projekte beschrieben werden, bei denen Ergebnisse aus der Theorie unmittelbare Anwendungen finden. Eine weitere Anwendung von Zellularräumen ergibt sich bei der Diskretisierung von Differentialgleichungen [49].

Das Modell des Zellularraums besteht aus Zellen, die in einem d–dimensionalen Gitter angeordnet sind, so daß jede Zelle nur mit ihren unmittelbaren Nachbarn verbunden ist. Jede Zelle kann nur endlich viele Zustände annehmen. Während der Berechnung wechseln die Zellen synchron ihren Zustand. Dabei hängt der Nachfolgezustand einer Zelle nur von den Zuständen ihrer Nachbarn ab. Die Eingabe wird vor dem Start der Berechnung in die Zellen geschrieben, findet also parallel statt.

Aus praktischen Gründen scheint es angebracht, Automaten mit sequentiellem Eingabemodus zu untersuchen. (Man denke an einen massiv parallelen Computer, bei dem ein „normaler PC“ als Schnittstelle zwischen Mensch und Maschine benutzt wird.) Ein einfaches Modell besteht aus einem Zellularraum, bei dem eine Zelle zusätzlich mit einem Eingabeband verbunden ist. Solche Automaten werden iterative Arrays (IA) genannt.

Im Zusammenhang mit formalen Sprachen wurden iterative Arrays in [12] eingeführt. Dort wird gezeigt, daß die Familie $\mathscr{L}_{rt}$(IA) von Sprachen, die in Realzeit von einem iterativen Array erkannt werden können, eine weder bezüglich Konkatenation noch Spiegelung abgeschlossene Boolesche Algebra ist.

Im Vergleich mit der Chomsky–Hierarchie ergeben sich gravierende Unterschiede. Es gibt sogar kontextfreie Sprachen, die von keinem d–dimensionalen iterativen Array in Realzeit erkannt werden können. Andererseits wurde in [11] gezeigt, daß für jede kontextfreie Sprache ein 2–dimensionaler Linearzeit–IA–Parser existiert. Auch im Vergleich mit Turing–Maschinen ergeben sich Unterschiede in der Berechnungsmächtigkeit. Zum Beispiel lassen sich Palindrome nicht in weniger als n^2 Zeitschritten von einer (einbändigen) Turing–Maschine erkennen. Aber iterative Arrays können Palindrome in Realzeit erkennen.

Iterative Arrays finden nicht nur bei der Spracherkennung eine Anwendung. In [2] steht Musterverarbeitung im Vordergrund. In [15] werden iterative Arrays zum Berechnen von Primzahlen eingesetzt und in [30] wird ein iteratives Array, das zwei Zahlen in binärer Darstellung in Realzeit multipliziert, beschrieben.

Charakterisierungen von iterativen Arrays durch eingeschränkte Turing–Maschinen finden sich in [23, 25, 26], wo auch einige Beschleunigungssätze bewiesen werden.

Iterative Arrays, bei denen die einzelnen Zellen nicht entlang einer Geraden, sondern in Form eines (Binär-)Baumes miteinander verbunden werden, heißen iterative Baumautomaten (ITA). Solche Automaten werden in [13] betrachtet.

Nichtdeterministische iterative Arrays werden z.B. in [43] betrachtet. Doch bisher wurde noch nie untersucht, inwieweit der Grad des Nichtdeterminis-

mus die Mächtigkeit des Modells beeinflußt. Da bei einem iterativen Array eine Zelle als Ursprung ausgezeichnet werden muß, liegt es nahe, diese auch noch auf andere Art von den restlichen Zellen zu unterscheiden. So werden zum Beispiel in [42] iterative Arrays untersucht, bei denen jede Zelle zusätzlich mit der Ursprungszelle verbunden ist.

In dieser Arbeit werden iterative Arrays untersucht, bei denen die Ursprungszelle nichtdeterministisch ist, während alle anderen Zellen deterministisch sind. Außerdem wird die Anzahl der nichtdeterministischen Übergänge für die Ursprungszelle eingeschränkt. Der Schwerpunkt der Untersuchungen liegt dabei auf Spracherkennung. Allerdings lassen sich Methoden zur Spracherkennung oft zu numerischen Methoden umwandeln. Zum Beispiel kann ein System, das $\{ww \mid w \in \Sigma^*\}$ erkennt, dazu benutzt werden, das Skalarprodukt zweier Vektoren zu berechnen. Man muß lediglich die elementaren Operationen $\cdot$ und $+$ als Vergleich und „und“ interpretieren.

Ein Schwerpunkt liegt auf der Untersuchung von Modellen mit eingeschränktem Nichtdeterminismus.

Eingeschränkter Nichtdeterminismus wurde bei Turing–Maschinen erstmals in [16] betrachtet. Weitere interessante Resultate finden sich in [8, 17]. Eingeschränkter Nichtdeterminismus für endliche Automaten wurde in [28] untersucht. In [39, 40] wird eingeschränkter Nichtdeterminismus bei Kellerautomaten studiert. In dieser Arbeit untersuche ich nun erstmals eingeschränkten Nichtdeterminismus bei iterativen Arrays.

Ebenfalls interessant sind die Guess–and–Check Modelle [9], bei denen eingeschränkter Nichtdeterminismus für beliebige Spachklassen definiert wird. Diese Modelle erkennen ein Wort w, indem sie zunächst ein Wort u raten (Guess). Die Länge von u hängt von der Länge von w ab. Danach muß ein deterministischer Automat das Wort $u \bullet w$ erkennen (Check). Das Problem bei diesen Modellen ist, daß die Länge des geratenen Wortes u extern vorgegeben wird. Im Gegensatz dazu wird bei dem in dieser Arbeit untersuchten Modell der iterativen Arrays mit eingeschränktem Nichtdeterminismus keine externe Beschränkung des Nichtdeterminismus benutzt. Die daraus resultierenden Unterschiede werden in Anhang B ausführlich betrachtet.

Die Arbeit gliedert sich wie folgt:

- In Kapitel 2 werden Standardbezeichnungen und das zu untersuchende Modell eingeführt.

- Kapitel 3 behandelt die grundlegenden Resultate für die Untersuchung von iterativen Arrays mit eingeschränktem Nichtdeterminismus ($f(n)$ – NIAs). Insbesondere werden Möglichkeiten betrachtet, wie $f(n)$ – NIAs auf gewisse Normalformen beschränkt werden können. In

Abschnitt 3.4 wird ein für die Untersuchung von $f(n)-$NIAs zentrales Lemma erarbeitet.

- In Kapitel 4 werden die Abschlußeigenschaften der Sprachfamilien untersucht. Das Kapitel ist in zwei Abschnitte unterteilt. Im ersten Abschnitt untersuchen wir die Familie der Sprachen, die von nichtdeterministischen iterativen Arrays in Realzeit erkannt werden ($\mathscr{L}_{rt}$(NIA)). Dabei spielen Ergebnisse aus der Theorie der abstrakten Sprachfamilien eine wichtige Rolle. Im zweiten Teil untersuchen wir die Sprachfamilien mit eingeschränktem Nichtdeterminismus $\mathscr{L}_{rt}(f(n)-\text{NIA})$. Die Ergebnisse aus diesem Abschnitt werden in den folgenden Kapiteln benötigt, um Hierarchiesätze zu beweisen.

- Kapitel 5 untersucht das Verhältnis von Nichtdeterminismus zu anderen Ressourcen (Zeit, Speicherplatz). Insbesondere beweisen wir mehrere Hierarchiesätze.

- In Kapitel 6 werden die nichtdeterministischen iterativen Arrays zu alternierenden iterativen Arrays (AIA) verallgemeinert. Insbesondere beweisen wir, daß AIAs in Realzeit echt mächtiger sind als NIAs.

- Anhang A stellt Resultate über berechenbare Funktionen zusammen, die in den vorangegangenen Kapitel benutzt wurden.

- In Anhang B vergleichen wir die Definition des beschränkten Nichtdeterminismus mit Guess–and–Check Modellen [9].

Kapitel 2

Einführende Definitionen und Begriffe

In diesem Kaptitel wiederholen wir kurz die wichtigsten Definitionen für parallele Automaten und führen dann das Modell der iterativen Arrays mit eingeschränktem Nichtdeterminismus formal ein.

2.1 Allgemeine Bezeichnungen

2.1.1 Mengen und Funktionen

Mit $\mathbb{N} = \{1, 2, 3, \dots\}$ bezeichnen wir die Menge der natürlichen Zahlen und mit $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ die Menge der natürlichen Zahlen einschließlich der Null. $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ ist die Menge der ganzen Zahlen. Die Potenzmenge einer Menge S bezeichnen wir mit 2^S.

In der gesamten Arbeit bezeichnet log den Logarithmus zur Basis 2. Allerdings folgt aus den Reduktionssätzen in Abschnitt 3.3, daß im allgemeinen die Basis keinen Einfluß auf die Resultate hat.

Häufig benutzen wir $\log(n)$, $\sqrt{n}$ usw. als Funktionen von $\mathbb{N}$ nach $\mathbb{N}$. In diesem Fall steht $\log(n)$ kurz für $\lceil \log(n) \rceil$ usw.

Die i–fache Verkettung einer Funktion f bezeichnen wir mit $f^{[i]}$, d.h. $f^{[1]} = f$ und $f^{[i+1]} = f \circ f^{[i]}$ für $i \in \mathbb{N}$.

2.1.2 Formale Sprachen

Ein Alphabet Σ ist eine endliche, nichtleere Menge von Zeichen. Ein Wort w über Σ ist eine beliebige Aneinanderreihung von Zeichen. Die Anzahl n der Zeichen in einem Wort w bezeichnen wir als die Länge von w und schreiben $|w| = n$. (Formal ist w ein n-Tupel von Elementen aus Σ.)

Das Wort, das aus 0 Zeichen besteht, nennen wir das leere Wort und bezeichnen es mit ε.

Mit Σ^* bezeichnen wir die Menge aller Wörter über Σ, und mit Σ^+ bezeichnen wir die Menge aller nichtleeren Wörter über Σ.

Eine Teilmenge L von Σ^* heißt formale Sprache über Σ.

Für ein Wort $w = a_1 \cdots a_n$ bezeichen wir die Spiegelung $a_n \cdots a_1$ von w mit w^R.

Für zwei Mengen L_1 und L_2 von Wörtern definieren wir die Konkatenation L_1L_2 von L_1 und L_2 durch

$$L_1L_2 = \{w_1w_2 \mid w_1 \in L_1 \text{ und } w_2 \in L_2\}.$$

2.1.3 Vereinbarungen für Variablennamen

In der Arbeit werden wir für bestimmte Variablen immer dieselben Bezeichnungen verwenden. Dies sind:

- Σ für das Eingabealphabet.
- a_i für einen Buchstaben. Die Zeichen \$, # und • werden für spezielle Buchstaben, die als Trennsymbole benutzt werden, verwendet.
- w und u sind Wörter über einem Alphabet.
- n bezeichnet üblicherweise die Länge des Eingabewortes.
- S, F_a, F_r sind Mengen von Zuständen.
- δ und Δ bezeichnen die lokale und globale Überführungsfunktion.
- $t : \mathbb{N}_0 \to \mathbb{N}$ ist stets eine Funktion zur Zeitbeschränkung.
- $f : \mathbb{N}_0 \to \mathbb{N}_0$ bezeichnet eine Funktion, die die Anzahl der nichtdeterministischen Schritte beschränkt.
- x, y und z sind Vektoren im $\mathbb{Z}^d$ und bezeichnen meistens die Koordinaten einer Zelle.

2.2 Zellularräume

Zellularräume wurden von J. von Neumann [35] als Modell für selbstreproduzierende Systeme vorgeschlagen. Seitdem wurden Zellularräume und ihre „endliche“ Variante, die Zellularautomaten, in vielerlei Hinsicht untersucht.

Definition 2.1

1. *Für eine Raumdimension $d \in \mathbb{N}$ und ein beliebiges $k \in \mathbb{N}$ sei N ein k-Tupel von Elementen aus $\mathbb{Z}^d$. N heißt Nachbarschaftsindex vom Grad k, wenn N aus k paarweise verschiedenen Elementen aus $\mathbb{Z}^d$ besteht und eine Komponente von N gleich $0 \in \mathbb{Z}^d$ ist.*

2. *Seien $x, y \in \mathbb{Z}^d$. y heißt benachbart zu x (bezüglich N), wenn es ein $z \in N$ mit $y = x + z$ gibt. Die Menge aller zu x benachbarten Elemente aus $\mathbb{Z}^d$ bezeichnen wir mit $x + N$.*

Wegen $0 \in N$ ist jedes Element zu sich selbst benachbart. Oft wird auf diese Zusatzforderung verzichtet.

Der wichtigste Nachbarschaftsindex ist das *von–Neumann–Raster*. N besteht bei diesem Raster aus 0 und den Vektoren $\pm e_k$, $k = 1, \ldots, d$. (Dabei bezeichnet e_k den k-ten Einheitsvektor.) Im Fall $d = 1$ ist das von–Neumann–Raster also gleich $(-1, 0, 1)$.

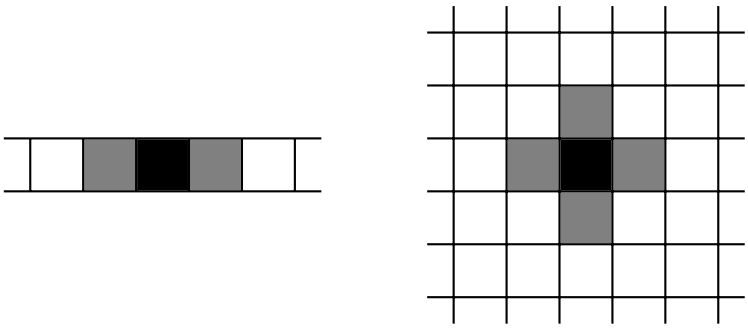


Abbildung 2.1: Eine Zelle (schwarz) und ihre Nachbarn (grau) im von–Neumann–Raster. Links für $d = 1$, rechts für $d = 2$. (Die Zelle ist auch zu sich selbst benachbart.)

Die Bedeutung des von–Neumann–Rasters liegt in der Tatsache, daß durch geschickte Wahl der Zustände des Zellularautomaten andere Raster auf dieses einfache Raster zurückgeführt werden können. (Siehe dazu [12] für iterative Arrays, [44] für Zellularräume.) Im folgenden werden wir Zellularräume zur Spracherkennung benutzen. Ein Wort wird von einem Zellularraum erkannt, wenn die Ursprungszelle einen ausgezeichneten Zustand annimmt. Man kann auch Varianten betrachten, in denen mit der gesamten Konfiguration eines Zellularraums und nicht mit der Ursprungszelle akzeptiert wird. Z.B. kann man Zellularautomaten betrachten, die ein Wort w

nur dann erkennen, wenn alle Zellen des Zellularraums in einen akzeptierenden Zustand wechseln. Probleme, die bei diesen Varianten entstehen, werden in [5] betrachtet.

Ein Zellularraum besteht aus endlichen Automaten (Zellen), die in einem d–dimensionalen Gitter angeordnet sind. Jeder Automat kann die ihm benachbarten Automaten „sehen“ und seinen Folgezustand von den Zuständen der Nachbarautomaten abhängig machen. Formal:

Definition 2.2
Ein 5*-Tupel* (S, d, N, δ, q_0) *heißt Zellularraum, falls gilt:*

1. S *ist eine endliche, nichtleere Menge von* Zuständen.
2. $d \in \mathbb{N}$ *ist die Raumdimension.*
3. N *ist ein* d*–dimensionaler Nachbarschaftsindex.*
4. $\delta : S^{|N|} \to S$ *ist die* lokale Überführungsfunktion.
5. $q_0 \in S$ *ist der* Ruhezustand, *für den* $\delta(q_0, \ldots, q_0) = q_0$ *gilt.*

Häufig läßt sich die Zustandsmenge S als $S_1 \times \cdots \times S_k$ schreiben. Wir sagen dann, daß eine Zelle des Zellularraums aus k Registern besteht, deren Einträge aus S_1 bis S_k stammen. Die Gesamtheit der i-ten Register aller Zellen des Zellularraums bezeichnen wir als die i-te Spur des Zellularraums.

Jeder einzelne Automat kann Zustände aus S annehmen. Nun wird der Gesamtzustand des Zellularraums definiert.

Definition 2.3
Eine Abbildung $c_t : \mathbb{Z}^d \to S$ *heißt* Konfiguration *oder* globaler Zustand *des Zellularraums zum Zeitpunkt* $t \in \mathbb{N}_0$. *Die Menge aller Konfigurationen bezeichnen wir mit* C. *Für* $t = 0$ *sprechen wir auch von der* Startkonfiguration.

Eine Konfiguration beschreibt also den Gesamtzustand des Raumes zum Zeitpunkt t. Die nächste Definition legt fest, wie ein Zellularraum Berechnungen ausführt, d.h. wie der Zellularraum aus einer Konfiguration c_t die *Nachfolgekonfiguration* c_{t+1} zum Zeitpunkt $t + 1$ ermittelt.

Definition 2.4
Sei (S, d, N, δ, q_0) *ein Zellularraum mit* $N = (z_1, \ldots, z_k)$. *Wir definieren die* globale Überführungsfunktion $\Delta : C \to C$ *durch* $\Delta(c) = c'$ *genau dann, wenn* $c'(i) = \delta(c(i) + z_1, \ldots, c(i) + z_k)$ *gilt.*

Da wir im folgenden nur Sprachverarbeitung betrachten, ist der Spezialfall $d = 1$ besonders wichtig (Wörter sind „eindimensionale Objekte“). Außerdem lassen sich viele Ergebnisse unmittelbar von $d = 1$ auf $d \in \mathbb{N}$ verallgemeinern.

Wie bereits erwähnt, lassen sich allgemeine Raster auf das von–Neumann–Raster zurückführen.

Wir definieren daher:

Definition 2.5
Ist N das von–Neumann–Raster, so schreiben wir kurz (S, d, δ, q_0) für den Zellularraum (S, d, N, δ, q_0). Gilt zusätzlich $d = 1$ so schreiben wir (S, δ, q_0) statt (S, d, N, δ, q_0).

Ein Zellularautomat ist ein Zellularraum, der nur einen endlichen Teil des Raumes für Berechnungen zur Verfügung hat. Zur Einfachheit führen wir nur eindimensionale Zellularautomaten mit von–Neumann–Raster ein.

Definition 2.6
*Ein Zellularautomat (*CA*) ist ein 4-Tupel $A = (S, \delta, F, \#)$, für das gilt:*

1. *$(S \cup \{\#\}, \delta, \#)$ ist ein Zellularraum.*
2. *Der* Grenzzustand *$\# \notin S$ erfüllt zusätzlich die Bedingung*

$$\delta(x, y, z) = \# \iff y = \# \quad .$$

   (D.h. eine Zelle kann den Grenzzustand weder verlassen noch aus einem anderen Zustand in den Grenzzustand wechseln.)

3. *$F \subseteq S$ ist eine Menge von* Endzuständen *oder* akzeptierenden Zuständen.

Für ein Wort $w = a_1 \cdots a_n \in S^+$ definieren wir die zu w gehörige Startkonfiguration c_w durch

$$c_w(i) = \begin{cases} a_i & \text{für } i = 1, \ldots, n \\ \# & \text{sonst} \end{cases} .$$

Wir sagen, der Zellularautomat A akzeptiert w in Realzeit, falls $\Delta^{[n]}(c_w) = c'$ und $c'(1) \in F$ ist. A akzeptiert w in Linearzeit, falls $\Delta^{[2n]}(c_w) = c'$ und $c'(1) \in F$ gilt. Außerdem kann frei festgelegt werden, ob A das leere Wort akzeptiert oder nicht. Für ein Alphabet $\Sigma \subseteq S$ nennen wir die Menge

$$\{w \in \Sigma^* \mid w \text{ wird von } A \text{ akzeptiert}\}$$

die von A akzeptierte Sprache.

Die Menge aller in Realzeit bzw. Linearzeit von Zellularautomaten erkennbaren Sprachen wird mit $\mathscr{L}_{rt}(\mathrm{CA})$ bzw. $\mathscr{L}_{lt}(\mathrm{CA})$ bezeichnet.

Neben Sprachverarbeitung gibt es noch weitere Aspekte im Zusammenhang mit Zellularautomaten. Aufgrund seiner vielfältigen Anwendung in Algorithmen ist das *Firing Squad Synchronization Problem (FSSP)* eines der wichtigsten Probleme. Die Aufgabe lautet: Für ein festes $n \in \mathbb{N}$ wird die Anfangskonfiguration c_0 beschrieben durch

$$c_0(i) = \begin{cases} G & \text{für } i = 1 \\ q_0 & \text{für } i = 2, \ldots, n \\ \# & \text{sonst} \end{cases},$$

wobei $G, q_0, \# \in S$ ausgezeichnete Zustände sind. Dabei erfüllt der Ruhezustand q_0 die Bedingungen

$$\delta(q_0, q_0, q_0) = q_0 \quad \text{und} \quad \delta(q_0, q_0, \#) = q_0,$$

d.h. die Zelle am rechten Rand (Zelle an der Position n) kann die Information, daß sie eine Randzelle ist, nicht benutzen. Gesucht ist ein Zellularautomat A, der unabhängig von der Wahl von n die Zellen $1, \ldots, n$ dazu veranlaßt, gleichzeitig in einen ausgezeichneten Zustand $f \in S$ überzugehen. Dabei darf keine Zelle vor den anderen Zellen den Zustand f annehmen.

(Die „anschauliche“ Bedeutung des Problems ist: Es stehen n Soldaten (Zellen) in einer Reihe, der General (G) am Anfang der Reihe muß den Soldaten befehlen, gleichzeitig Salut zu schießen (f). Es wird vorausgesetzt, daß die Soldaten sich jeweils nur mit ihren Nachbarn verständigen können.)

Es gibt Algorithmen, die das FSSP-Problem in $2n-2$ Zeitschritten lösen, und $2n-2$ ist die minimale Anzahl von Schritten, die zur Lösung des FSSP-Problems benötigt werden. Mazoyer beschreibt in [32] einen Automaten mit 6 Zuständen, der das FSSP-Problem zeitoptimal löst. Im folgenden ist wichtig, daß es Lösungen des FSSP-Problems gibt, die weniger als $2n$ Zeitschritte benötigen.

Das FSSP-Problem wird zum Beispiel gebraucht, um zu zeigen, daß iterative Arrays Zellularautomaten und Zellularräume simulieren können (siehe nächster Abschnitt).

2.3 Iterative Arrays

Bei Zellularautomaten wird angenommen, daß die Eingabe bereits zum Zeitpunkt $t = 0$ vorliegt. Anschaulich bedeutet dies, daß jede Zelle über eine Verbindung zum Eingabemedium verfügt und die Eingabe parallel erfolgt.

Aus praktischer Sicht erscheint eine sequentielle Eingabe jedoch leichter realisierbar.

Es ist nun naheliegend, das Modell der Zellularräume entsprechend abzuändern. S. N. Cole hat in diesem Zusammenhang 1966 (siehe [12]) das Modell der iterativen Arrays vorgeschlagen. Das Modell geht von einem Zellularraum aus, bei dem sich am Anfang alle Zellen im Ruhezustand befinden und eine Zelle während der Berechnung sequentiell die Eingabe von einem externen Speicherband einliest. Dadurch wird natürlich diese Zelle vor allen anderen ausgezeichnet.

Definition 2.7
Ein iteratives Array ist ein 10*-Tupel* $(S, \Sigma, F_a, F_r, d, N, \delta, \delta_0, q_0, \sqcup)$*, für das gilt:*

1. S *ist eine endliche, nichtleere Menge von Zuständen.*
2. Σ *ist eine endliche, nichtleere Menge von Eingabesymbolen.*
3. $F_a \subseteq S$ *ist die Menge der akzeptierenden Endzustände und* $F_r \subseteq S$ *ist die Menge der nichtakzeptierenden Endzustände. Es gilt* $F_a \cap F_r = \emptyset$*. Die Menge* $F = F_a \cup F_r$ *ist die Menge aller Endzustände.*
4. $d \in \mathbb{N}$ *ist die Raumdimension.*
5. N *ist ein* d*-dimensionaler Nachbarschaftsindex.*
6. $\delta_0 : S^{|N|} \times (\Sigma \cup \{\sqcup\}) \to S$ *ist die lokale Überführungsfunktion für die Ursprungszelle* 0*.*
7. $\delta : S^{|N|} \to S$ *ist die lokale Überführungsfunktion für die Zellen* $x \neq 0$*.*
8. q_0 *ist der Ruhezustand, für den* $\delta(q_0, \ldots, q_0) = q_0$ *gilt.*
9. $\sqcup \notin \Sigma$ *ist das Eingabeendesymbol (leeres Feld).*

Im Vergleich zu Zellularräumen haben wir also eine spezielle lokale Überführungsfunktion δ_0, da das Verhalten der Ursprungszelle zusätzlich von einer externen Eingabe abhängt. Das Symbol $\sqcup$ (Leerzeichen) wird als Füllsymbol benutzt, das nach Ende der eigentlichen Eingabe als externe Dauereingabe anliegt.

Im Gegensatz zu Zellularautomaten ist bei iterativen Arrays (wie bei Zellularräumen) der Raum unbeschränkt.

Auch bei iterativen Arrays ist der Fall des von–Neumann–Rasters besonders wichtig. Wir definieren daher:

Definition 2.8
Ist N das von–Neumann–Raster, so schreiben wir $(S, \Sigma, F_a, F_r, d, \delta, \delta_0, q_0, \sqcup)$ für das iterative Array $(S, \Sigma, F_a, F_r, d, N, \delta, \delta_0, q_0, \sqcup)$.

Gilt zusätzlich $d = 1$, so schreiben wir $(S, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup)$ statt $(S, \Sigma, F_a, F_r, d, N, \delta, \delta_0, q_0, \sqcup)$.

Definition 2.9
Ein Paar (w, c_t) heißt Konfiguration des iterativen Arrays, *falls $w \in \Sigma^*$ das noch zu lesende Eingabewort und c_t eine Konfiguration im Sinne von Zellularräumen ist.*

Die Nachfolgekonfiguration *(w', c_{t+1}) wird definiert durch:*

- *$w' = a_2 \cdots a_n$, falls $w = a_1 a_2 \cdots a_n$ $(n \in \mathbb{N})$ und $w' = \varepsilon$, falls $w = \varepsilon$ gilt.*
- *$c_{t+1}(x) = \delta(c_t(x + z_1), \ldots, c_t(x + z_k))$. (Dabei ist $N = (z_1, \ldots, z_k)$).*
- *$c_{t+1}(0) = \delta_0(c_t(z_1), \ldots, c_t(z_k), a_1)$, falls $w = a_1 \cdots a_n$ und*
  $c_{t+1}(0) = \delta_0(c_t(z_1), \ldots, c_t(z_k), \sqcup)$, falls $w = \varepsilon$.

Die globale Überführungsfunktion Δ ist die Abbildung, die jeder Konfiguration ihre Nachfolgekonfiguration zuordnet.

Zum Zeitpunkt $t = 0$ befinden sich alle Automaten im Ruhezustand q_0. Diese Konfiguration bezeichnen wir mit c_0.

Die Abbildung 2.2 zeigt schematisch die Anfangskonfiguration eines eindimensionalen iterativen Arrays bei Eingabe von $w = a_1 \cdots a_n$.

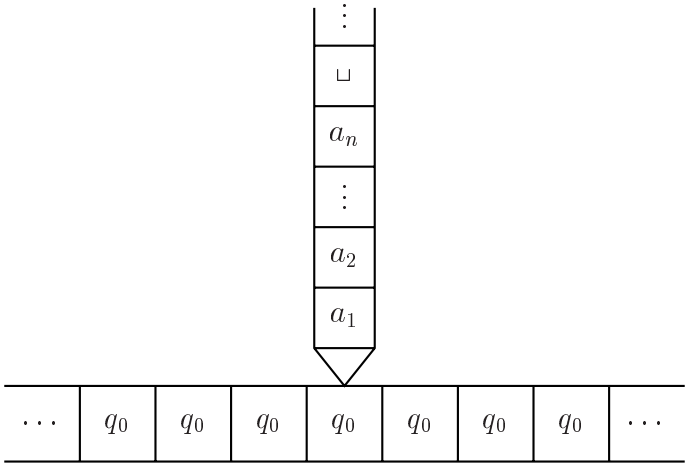


Abbildung 2.2: Die Anfangskonfiguration eines iterativen Arrays.

Die nächste Definiton gibt an, wann ein iteratives Array seine Berechnung beendet:

Definition 2.10
Die Berechnung eines iterativen Arrays $A = (S, \Sigma, F_a, F_r, d, N, \delta, \delta_0, q_0, \sqcup)$ *bei Eingabe eines Wortes* $w \in \Sigma^*$ *wird beschrieben durch die Folge der Konfigurationen* $a = (w, c_0)$, $\Delta(a)$, $\Delta^{[2]}(a)$ *usw.*

Ist t *die kleinste natürliche Zahl, für die* $\Delta^{[t]}(a) = (w', c')$ *mit* $c'(0) \in F$ *gilt, so nennen wir* (w', c') *das Ergebnis der Berechnung von* A*. Wir sagen, die Berechnung von* A *auf* w *dauert* t *Zeitschritte* ($time_A(w) = t$)*. (Das iterative Array hält nach* t *Zeitschritten an.)*

Definition 2.11

1. *Ein iteratives Array* A *akzeptiert ein Wort* w*, wenn* (w', c') *das Ergebnis der Berechnung von* A *auf* w *ist und* $c'(0) \in F_a$ *gilt. (Es ist möglich, daß das iterative Array* A *bei Eingabe von* w *nie eine Konfiguration* (w', c') *mit* $c'(0) \in F$ *erreicht. In diesem Fall sagen wir: Das iterative Array* A *rechnet bei Eingabe* w *unendlich lange. Solche Wörter* w *werden von* A *nicht akzeptiert.)*

2. *Die Menge aller Wörter, die von* A *akzeptiert werden, nennen wir die von* A *akzeptierte Sprache* $L(A)$*. Die Familie aller Sprachen, die von einem* d*–dimensionalen iterativen Array mit von–Neumann–Raster akzeptiert werden können, bezeichnen wir mit* $\mathscr{L}(\mathrm{IA}_d)$*.*

3. *Für eine Funktion* $t : \mathbb{N}_0 \to \mathbb{N}$ *sagen wir,* A *ist* $t(n)$*–zeitbeschränkt, wenn für jedes Wort* $w \in \Sigma^*$ *gilt:* $time_A(w) \leq t(|w|)$*.*

   Die Familie aller Sprachen, die von $t(n)$*–zeitbeschränkten,* d*-dimensionalen iterativen Arrays mit von–Neumann-Raster erkannt werden, bezeichnen wir mit* $\mathscr{L}_{t(n)}(\mathrm{IA}_d)$*.*

4. *Im Fall* $d = 1$ *lassen wir die Angabe von* d *weg und sprechen nur von* $\mathscr{L}(\mathrm{IA})$ *und* $\mathscr{L}_{t(n)}(\mathrm{IA})$*.*

Die Funktionen $t(n) = n + 1$ und $t(n) = 2n$ kürzen wir mit rt (Realzeit) und lt (Linearzeit) ab. (Realzeit ist die minimale Zeit, die ein iteratives Array benötigt, um zu überprüfen, daß es die gesamte Eingabe gelesen hat.)

Da iterative Arrays im Gegensatz zu Zellularautomaten die Eingabe erst während der Berechnung erhalten, ist es naheliegend, daß $\mathscr{L}_{t(n)}(\mathrm{CA})$ mächtiger ist als $\mathscr{L}_{t(n)}(\mathrm{IA})$. Der folgende Satz faßt die wichtigsten Beziehungen zwischen diesen beiden Sprachfamilien zusammen.

Satz 2.12
Für jede Zeitbeschränkung $t : \mathbb{N}_0 \to \mathbb{N}$, die $t(n) \leq k \cdot n$ für eine Konstante $k \geq 1$ erfüllt, gilt $\mathscr{L}_{t(n)}(\mathrm{IA}) \subseteq \mathscr{L}_{t(n)}(\mathrm{CA})$.

Es gilt weiter:

$$\mathscr{L}_{rt}(\mathrm{IA}) \subsetneqq \mathscr{L}_{rt}(\mathrm{CA})$$
$$\mathscr{L}_{lt}(\mathrm{IA}) = \mathscr{L}_{lt}(\mathrm{CA})$$

.

Beweis
Die Beziehungen $\mathscr{L}_{t(n)}(\mathrm{IA}) \subseteq \mathscr{L}_{t(n)}(\mathrm{CA})$ und $\mathscr{L}_{rt}(\mathrm{IA}) \subsetneqq \mathscr{L}_{rt}(\mathrm{CA})$ werden in [12] bewiesen. Den Beweis von $\mathscr{L}_{lt}(\mathrm{CA}) \subseteq \mathscr{L}_{lt}(\mathrm{IA})$ führen wir an dieser Stelle vor, da er die Bedeutung des FSSP-Problems aufzeigt und allgemeine Techniken, die wir später brauchen, verdeutlicht.

Sei L eine Sprache, die von einem Zellularautomaten A in Linearzeit erkannt werden kann. Ein iteratives Array A', das L in Linearzeit erkennt, arbeitet wie folgt:

Solange eine Eingabe vorliegt, wird sie gelesen und auf der ersten Spur gespeichert. D.h. bei Eingabe $w = a_1 \dots a_n$ wird a_i in der Zelle an Position $i-1$ gespeichert. Sobald das Zeichen $\sqcup$ gelesen wird, startet die Zelle an Position 0 einen FSSP-Algorithmus auf der zweiten Spur. Nach $2n-2$ Zeitschritten werden die Zellen $0, \dots, n-1$ im zweiten Register ein f enthalten. Sobald eine Zelle im zweiten Register ein f enthält, simuliert sie in ihrem ersten Register den Automaten A. Der FSSP–Algorithmus sorgt also dafür, daß alle Zellen des iterativen Arrays *gleichzeitig* mit der Simulation von A beginnen. Das iterative Array A' braucht für diese Berechnung insgesamt $(n+1)+(2n-2)+2n$ Zeitschritte. In Kapitel 3 wird gezeigt, daß sich diese Berechnung auf $2n$ Zeitschritte beschleunigen läßt.

Die Abbildung 2.3 zeigt beispielhaft den Zustand des iterativen Arrays während der Berechnung. Die Eingabe ist $w = a_1a_2a_3a_4$. □

Die Technik, einen Zellularautomaten durch ein iteratives Array zu simulieren, wird häufig benötigt. Wir werden in späteren Beweisen dies kurz als „Das iterative Array liest die Eingabe ein, synchronisiert sich mittels FSSP und simuliert den Zellularautomaten.“
beschreiben.

2.4 Nichtdeterministische iterative Arrays

Wir modifizieren nun das Modell der iterativen Arrays, indem wir eine nichtdeterministische Überführungsfunktion für die Ursprungszelle zulassen. In

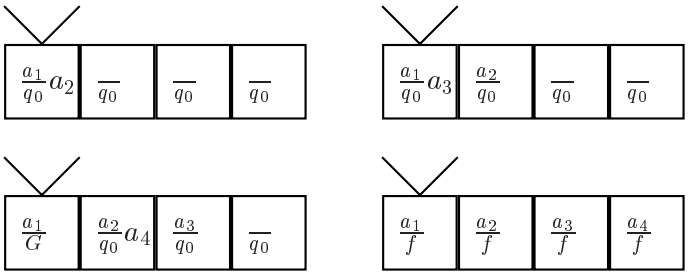


Abbildung 2.3: Die Initialisierungsphase der Simulation. Die obere Zeile zeigt das iterative Array beim Lesen der Eingabe an zwei aufeinanderfolgenden Zeitpunkten. Die Eingabe wird in dem oberen Register gespeichert. Die Abbildung unten links zeigt das iterative Array nach dem Lesen des ersten $\sqcup$ Zeichens. Die Ursprungszelle startet im unteren Register einen FSSP–Algorithmus. Die letzte Abbildung zeigt das Ende der Initialisierungsphase. Alle Zellen beginnen nun gleichzeitig, den Zellularautomaten zu simulieren.

den folgenden Definitionen beschränken wir uns der Einfachheit halber auf eindimensionale iterative Arrays mit von–Neumann–Nachbarschaft.

Definition 2.13
Ein nichtdeterministisches iteratives Array (NIA) *ist ein* 8*–Tupel*

$$(S, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup),$$

für das gilt:

1. S *ist eine endliche, nichtleere Menge von Zuständen.*
2. Σ *ist ein endliches, nichtleeres Eingabealphabet.*
3. $F_a \subseteq S$ *ist die Menge der akzeptierenden Endzustände und* $F_r \subseteq S$ *ist die Menge der nichtakzeptierenden Endzustände. Es gilt* $F_a \cap F_r = \emptyset$. *Die Menge* $F = F_a \cup F_r$ *ist die Menge der Endzustände.*
4. $\delta_0 : S^3 \times (\Sigma \cup \{\sqcup\}) \to 2^S \setminus \{\emptyset\}$ *ist die nichtdeterministische Überführungsfunktion für die Ursprungszelle* 0.
5. $\delta : S^3 \to S$ *ist die lokale Überführungsfunktion für alle Zellen* $x \neq 0$.
6. q_0 *ist der Ruhezustand, für den* $\delta(q_0, q_0, q_0) = q_0$ *gilt.*
7. $\sqcup \notin \Sigma$ *ist das Eingabeendesymbol.*

Die Überführungsfunktion δ_0 ist jetzt nichtdeterministisch.

Die nächste Definition legt fest, wie ein NIA Berechnungen ausführt. Wie bei deterministischen iterativen Arrays bezeichnen wir die Startkonfiguration, bei der sich alle Zellen im Ruhezustand befinden, mit c_0.

Die globale Überführungsfunktion Δ ist nun eine Abbildung, die für jede Konfiguration des iterativen Arrays eine Menge von Nachfolgekonfigurationen liefert.

Definition 2.14

1. *Für ein* NIA A *sagen wir,* (w', c') *ist ein Ergebnis der Berechnung von* A *bei Eingabe von* w*, wenn es für ein* $t \in \mathbb{N}$ *eine Folge von Konfigurationen* (w_k, c_k) *mit* $k = 0, 1, \ldots, t$ *gibt, die die folgenden Bedingungen erfüllt:*

   - $(w, c_0) = (w_0, c_0)$,
   - $(w', c') = (w_t, c_t)$,
   - $(w_k, c_k) \in \Delta((w_{k-1}, c_{k-1}))$ *für* $k = 1, \ldots, t$,
   - $c_k(0) \notin F$ *für* $k < t$ *und* $c_t(0) = c'(0) \in F$.

   Wir nennen (w', c') *eine Endkonfiguration.* A *erreicht diese Konfiguration in* t *Zeitschritten. (Wir sagen auch* A *hält an, wenn* A *eine Endkonfiguration erreicht.) Wir sagen, daß* A *höchstens* t *Zeitschritte bei Eingabe* w *benötigt (*$time_A(w) \leq t$*), wenn beginnend bei* (w, c_0) *jede Wahl von Nachfolgekonfigurationen nach spätestens* t *Zeitschritten eine Endkonfiguration ergibt.*

2. *Ein* NIA A *erkennt ein Wort* w*, wenn es ausgehend von* (w, c_0) *eine Endkonfiguration* (w', c') *erreichen kann, bei der* $c'(0) \in F_a$ *gilt.*

3. *Die Menge aller Wörter, die von* A *erkannt werden, bezeichnen wir mit* $L(A)$*. Die Familie aller Sprachen, die von nichtdeterministischen iterativen Arrays erkannt werden können, bezeichnen wir mit* $\mathscr{L}$(NIA).

4. *Für eine Funktion* $t : \mathbb{N}_0 \to \mathbb{N}$ *sagen wir,* A *ist* $t(n)$*–zeitbeschränkt, wenn* $time_A(w) \leq t(|w|)$ *für jedes Wort* $w \in \Sigma^*$ *gilt.*

   Die Familie aller Sprachen, die von $t(n)$*–zeitbeschränkten* NIA*s akzeptiert werden, bezeichnen wir mit* $\mathscr{L}_{t(n)}$(NIA).

Satz 2.15
Die Lexikonsprache

$$L = \{w_1\$ \ldots \$w_k\#y \mid w_i, y \in \{0,1\}^+ \text{ und } w_i = y \text{ für ein } i \in \{1, \ldots, k\}\}$$

liegt in $\mathscr{L}_{rt}$(NIA).

Beweis
Nach jedem \$ Zeichen entscheidet der Automat, ob das folgende Wort w_i in einer FiFo–Schlange gespeichert werden soll. Die folgende Abbildung zeigt, wie ein iteratives Array eine FiFo–Schlange simuliert.

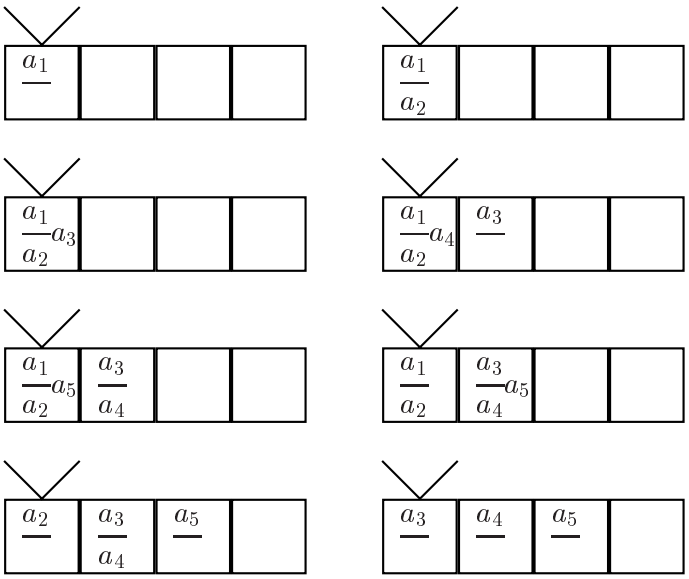


Abbildung 2.4: Eine FiFo–Schlange. In den ersten 5 Schritten wird jeweils ein Zeichen in der Schlange gespeichert. Danach folgt ein Schritt, in dem weder aus der Schlange gelesen noch in die Schlange gespeichert wird. In den letzten zwei Schritten wird jeweils ein Zeichen aus der Schlange gelesen.

Jede Zelle benötigt drei Register, um eine FiFo-Schlange zu simulieren. In den beiden ersten Registern werden die Einträge der Schlange gespeichert. Das dritte Register dient als Überlaufregister. Es gelten folgende Regeln:

- Ist eines der beiden ersten Register leer und enthält die rechte Nachbarzelle Einträge, so wird der erste dieser Einträge von rechts nach links bewegt.

- In jedem Schritt wird das Zeichen im Überlaufregister um eine Zelle nach rechts kopiert. Ist die letzte Zelle der Schlange erreicht, so wird der Eintrag aus dem Überlaufregister in eines der beiden ersten Register kopiert.

Da jede Zelle zwei Einträge der FiFo–Schlange speichert, kommt es beim Auslesen der Schlange zu keiner Verzögerung.

Nach dem # Zeichen vergleicht der Automat das Wort y mit dem gespeicherten w_i. Es wird nur erkannt, wenn diese beiden Wörter gleich sind. □

Im nächsten Abschnitt führen wir ein Maß für die Anzahl der nichtdeterministischen Schritte ein.

2.5 Eingeschränkter Nichtdeterminismus

Eine Konfiguration (w, c) heißt *nichtdeterministisch*, wenn $|\Delta((w, c))| \geq 2$ gilt. (D.h. eine nichtdeterministische Konfiguration hat mindestens zwei Nachfolgekonfigurationen.) Eine Konfiguration (w, c) heißt *deterministisch*, wenn $|\Delta((w, c))| = 1$ gilt.

Definition 2.16
Wir sagen, die Anzahl der nichtdeterministischen Schritte von A ist durch die Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ beschränkt, wenn für alle $w \in \Sigma^*$ beginnend mit der Konfiguration (w, c_0) jede Wahl von Nachfolgekonfigurationen höchstens $f(|w|)$ nichtdeterministische Konfigurationen liefert, bevor A eine Endkonfiguration erreicht.

Wir nennen A dann ein $f(n)-$NIA.

Die Familie aller Sprachen, die von $f(n)-$NIAs akzeptiert werden, bezeichnen wir mit $\mathscr{L}(f(n)-\text{NIA})$ und die Familie aller Sprachen, die von $t(n)$–zeitbeschränkten $f(n)-$NIAs akzeptiert werden, heißt $\mathscr{L}_{t(n)}(f(n)-\text{NIA})$.

Ein wichtiger Spezialfall ist $f(n) = 0$ für alle n. In diesem Fall gilt immer $|\Delta((w, c))| = 1$, d.h. das iterative Array ist deterministisch.

Bemerkung 2.17
Die Definition eines $f(n)-$NIA darf nicht mit den β–Hierarchien bei Turing–Maschinen verwechselt werden.

Man sagt, eine Sprache liegt in β_f, wenn es für jedes Wort w ein Wort u, das höchstens die Länge $f(|w|)$ hat, gibt, so daß $u \bullet w$ von einer deterministischen Turing–Maschine in Polynomialzeit erkannt werden kann (siehe [14]). ($\bullet$ ist ein Trennsymbol, das weder in u noch in w vorkommt.) Die Interpretation dieser Definition ist, daß die Turing–Maschine erst $f(n)$ Zeichen rät und dann mittels der geratenen Zeichen das Wort w überprüft (siehe auch das guess–and–check Modell aus [9]).

Dies führt zu keinen Problemen, wenn man nur „vernünftige“ Funktionen f betrachtet, z.B. $f_k(n) = \log^k(n)$ (siehe [19]). Betrachtet man jedoch beliebige Funktionen f, so können gewaltige Probleme auftreten. Z.B. sei

M eine (nichtberechenbare) Teilmenge der natürlichen Zahlen, dann liegt $\{a^n | n \in M\}$ in β_f mit

$$f(n) = \begin{cases} 0 & \text{für } n \in M \\ 1 & \text{für } n \notin M \end{cases}.$$

Die Turing–Maschine erkennt einfach alle Paare $u \bullet w$, bei denen u nicht das leere Wort ist.

Bei unserer Definition eines $f(n) - \text{NIA}$ wird direkt gefordert, daß die Schranke f durch das NIA *„konstruiert“ wird. Dies garantiert, daß alle Sprachfamilien $\mathscr{L}_{t(n)}(f(n) - \text{NIA})$ innerhalb der rekursiv aufzählbaren Sprachen liegen und daß*

$$\mathscr{L}_{t(n)}(f(n) - \text{NIA}) \subseteq \mathscr{L}_{t(n)}(f'(n) - \text{NIA})$$

aus $f(n) \leq f'(n)$ folgt.

In Anhang B wird dieser Unterschied noch genauer erläutert.

Wir betrachten nun ein typisches Beispiel für die Berechnung eines $f(n) -$ NIA. Dazu definieren wir zunächst eine eingeschränkte Version der Lexikonsprache aus Satz 2.15.

Definition 2.18
Sei $h : \mathbb{N} \to \mathbb{N}$ eine Abbildung. Die „beschränkte Lexikonsprache“ $L_{h(n)}$ ist definiert durch

$$L_{h(n)} = \{w_1\$ \ldots \$w_{h(n)}\#y\# \mid w_i, y \in \{0,1\}^n, y = w_i \text{ für ein } i \in \{1, \ldots, h(n)\}\}.$$

(Im Vergleich zur „allgemeinen Lexikonsprache“ haben alle Wörter im Lexikon die gleiche Länge und die Anzahl der Wörter hängt von deren Länge ab.)

Die Bedeutung der eingeschränkten Lexikonsprache ergibt sich aus dem folgenden Satz:

Satz 2.19
Gilt für zwei Abbildungen $h : \mathbb{N} \to \mathbb{N}$ und $f : \mathbb{N}_0 \to \mathbb{N}_0$ $\{(a^n b)^{h(n)}\} \in \mathscr{L}_{rt}(\text{IA})$ und $f((h(n)+1)(n+1)) \geq \log(h(n))$, so liegt $L_{h(n)}$ in $\mathscr{L}_{rt}(f(n) - \text{NIA})$.

Beweis
Wir konstruieren ein $f(n) - \text{NIA}$ A, das $L_{h(n)}$ erkennt. A hat drei Register.

Im ersten Register wird ein deterministisches iteratives Array simuliert, das $(a^n b)^{h(n)}$ erkennt. Dabei werden 0 und 1 als a sowie \$ und # als b behandelt. Auf diese Weise kann A überprüfen, ob die Eingabe von der richtigen

Form ist (d.h. ob alle w_i gleichlang sind und die Anzahl der w_i gleich $h(n)$ ist). Viele Funktionen erfüllen diese Bedingung (siehe Bemerkung 2.20).

Das zweite Register enthält einen Binärzähler. Durch die nichtdeterministischen Schritte wird der Zähler auf einen Wert zwischen 1 und $h(n)$ gesetzt. Nach jedem \$ wird der Zähler um eins vermindert.

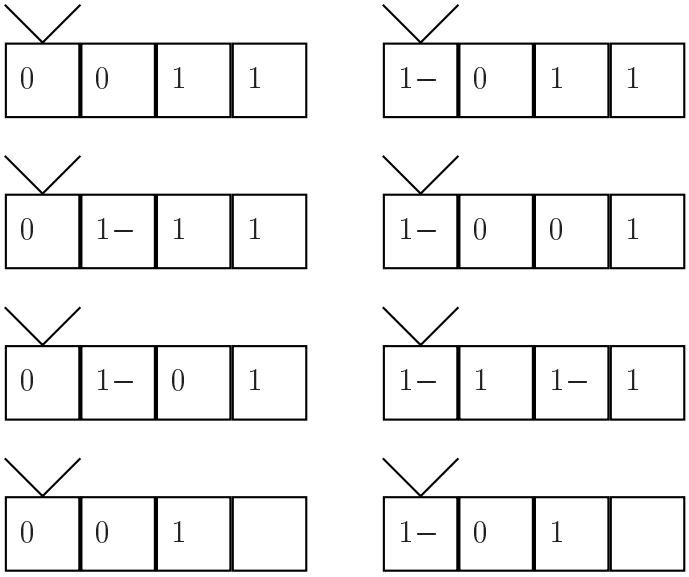


Abbildung 2.5: Ein Binärzähler. In jedem Schritt wird der Zähler um 1 vermindert.

Ein Binärzähler wird ähnlich einer FiFo–Schlange initialisiert. Um den Zähler vermindern zu können, braucht jede Zelle zwei Register. (In unserem Fall ist daher das zweite Register in zwei Unterregister unterteilt.) Im ersten Unterregister wird jeweils eine Ziffer gespeichert. Das zweite Unterregister dient als Überlaufregister. Das niedrigstwertige Bit des Zählers wird in der Ursprungszelle gespeichert. (Der Zähler im Beispiel steht daher am Anfang auf 12.) Es gelten folgende Regeln:

- Steht im Überlaufregister der linken Nachbarzelle ein $-$ Zeichen, so muß der Eintrag in der Zelle um eins vermindert werden und das $-$ Zeichen wird gelöscht.

- Wird der Eintrag einer Zelle von 1 auf 0 vermindert, so ist keine weitere Aktion notwendig. Wird der Eintrag einer Zelle von 0 auf 1 vermindert, so muß zusätzlich ein $-$ Zeichen in das Überlaufregister geschrieben werden.

- Wird das höchstwertige Bit vermindert, so wird der Eintrag in dieser Zelle ganz gelöscht. (Keine Zahl benötigt eine 0 als erste Ziffer.)

Ist der Zähler 0, so beginnt A das Wort w_i hinter dem betreffenden \$ Zeichen abzuspeichern. Dies geschieht im dritten Register mit Hilfe einer FiFo–Schlange.

Nach dem # Zeichen vergleicht der Automat das Wort y mit dem gespeicherten w_i. Es wird nur erkannt, wenn diese beiden Wörter gleich sind und die Berechnung im ersten Register erfolgreich ist. Auf diese Weise können nur Wörter erkannt werden, bei denen $y = w_i$ für ein $i \geq 2$ gilt. Um auch Wörter der Form mit $y = w_1$ erkennen zu können, rät das iterative Array im ersten Zeitschritt, ob es w_1 speichern soll. Wird w_1 gespeichert, so muß die Berechnung mit dem Binärzähler nicht durchgeführt werden.

Es ist klar, daß nur Wörter aus L erkannt werden können. Andererseits gibt es auch für jedes Wort aus L eine Folge von Wahlmöglichkeiten, bei der dieses Wort erkannt wird.

Wir müssen noch dafür sorgen, daß der Automat höchstens $\log(h(n))$ nichtdeterministische Schritte macht, um den Binärzähler zu raten. Der Trick ist, eine Stelle des Binärzählers nur dann zu raten, wenn sie gebraucht wird, z.B. wissen wir, daß $\ldots 10 - 1 = \ldots 01$ gilt, unabhängig von den vorherigen Stellen. Nur wenn alle hinteren Stellen 0 sind, müssen wir eine weitere Stelle des Binärzählers raten. Dies läuft dann nach folgendem Muster ab.

- Der Binärzähler steht auf $\sqcup 00000$. Die Ursprungszelle schickt das Signal, daß der Zähler um 1 vermindert werden muß.

- Nach (im Beispiel) 5 Schritten wurden alle hinteren Stellen auf 1 gesetzt. Die Zelle, die $\sqcup$ enthält, schickt ein Signal zur Ursprungszelle, daß eine neue Stelle des Binärzählers geraten werden muß.

- Nach weiteren 5 Schritten erhält die Ursprungszelle das Signal. Sie rät nun entweder eine 0, eine 1 oder ein E. (Das E bedeutet, der Zähler ist 100000 gewesen und wird nun 11111 sein. Es gibt *keine* weiteren Stellen.) Das geratene Symbol wird jetzt an die Zelle mit den $\sqcup$ geschickt.

- Nach weiteren 5 Schritten kann der Zähler auf den neuesten Stand gebracht werden. Er lautet nun $\sqcup 011111$, $\sqcup 111111$ oder 11111.

Benutzt man diese Form des Zählers, so hat das iterative Array nach dem Lesen des k-ten \$ Zeichens erst $\log(k)$ nichtdeterministische Schritte gemacht. Das iterative Array kann feststellen, daß zwischen zwei \$ Zeichen jeweils n andere Zeichen stehen. Die Berechnung wird sofort durch Wechseln in einen nichtakzeptierenden Endzustand abgebrochen, falls dies nicht

der Fall ist. Da $\{(a^n b)^{h(n)} | n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{IA})$ gilt, kann das iterative Array feststellen, wann das $h(n)$–te \$ Zeichen kommt. Die Berechnung wird sofort beendet, wenn das $h(n)$-te \$ Zeichen auftritt. (Wörter aus $L_{h(n)}$ enthalten das Zeichen \$ nur $(h(n) - 1)$ mal.) Auf diese Weise kann verhindert werden, daß mehr als $\log(h(n))$ nichtdeterministische Schritte durchgeführt werden. □

Bemerkung 2.20
Viele Funktionen h erfüllen die Voraussetzung $\{(a^n b)^{h(n)} | n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{IA})$. Z.B. $h(n) = n$, $h(n) = \sqrt{n}$ oder $h(n) = \log(n)$. Allgemein kann man zeigen: Jede Funktion h, für die $\{a^n b^{h(n)} | n \in \mathbb{N}\}$ in $\mathscr{L}_{lt}(\text{CA})$ enthalten ist, erfüllt diese Voraussetzung (siehe Anhang A).

Die Lexikonsprache $L_{h(n)}$ wird in späteren Abschnitten häufig als Beispiel einer Sprache aus $\mathscr{L}_{rt}(f(n) - \text{NIA})$ benutzt.

Kapitel 3

Grundlagen

In diesem Kapitel werden einige grundlegende Eigenschaften von iterativen Arrays mit eingeschränktem Nichtdeterminismus erarbeitet.

Insbesondere werden in Abschnitt 3.3 Reduktionssätze für Beschränkungen von Zeit und Nichtdeterminismus bewiesen. In Abschnitt 3.4 werden Äquivalenzklassen auf Wörtern definiert, und es wird ein Lemma erarbeitet, mit dem wir zeigen können, daß bestimmte Sprachen nicht in Realzeit von $f(n)$ – NIAs erkannt werden können.

3.1 Bezeichnungen

3.1.1 Asymptotik

Häufig ist nicht die exakte Gestalt einer Funktion von Interesse sondern nur ihr Wachstumsverhalten.

Bei asymptotischen Aussagen benutzen wir folgende Bezeichnungen.

Definition 3.1
Für Funktionen $f, g : \mathbb{N} \to \mathbb{N}$ schreiben wir:

1. $f(n) = O(g(n))$, *wenn*

$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

gilt.

2. $f(n) = o(g(n))$, *wenn*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

gilt.

3. $f(n) = \Omega(g(n))$, *wenn*

$$\liminf_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

gilt.

4. $f(n) = \omega(g(n))$, *wenn*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

gilt.

5. $f(n) = \Theta(g(n))$, *wenn sowohl* $f(n) = O(g(n))$ *als auch* $f(n) = \Omega(g(n))$ *gilt.*

Zum Beispiel bedeutet $f(n) = o(1)$, daß $\lim_{n\to\infty} f(n) = 0$ gilt. $f(n) = \omega(1)$ bedeutet: $\lim_{n\to\infty} f(n) = \infty$. $f(n) = O(1)$ bedeutet: f ist (nach oben) beschränkt.

Es gilt $\sum_{i=1}^{n} i = O(n^2)$ oder $\sum_{i=1}^{n} i = \frac{1}{2}n^2 + O(n)$.

(Die zweite Aussage bedeutet: Es gibt eine Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(n) = O(n)$ und $\sum_{i=1}^{n} i = \frac{1}{2}n^2 + f(n)$.)

Ein weiteres Beispiel ist $\log(n) = n^{o(1)}$. Dies bedeutet: $\log(n)$ wächst langsamer als jede Potenz von n.

Aussagen mit „O-Termen“ sind so zu lesen, daß für jede Funktion, die die linke Seite erfüllt, auch die rechte Seite erfüllt ist. Die Umkehrung muß nicht gelten. (Zum Beispiel gilt $n^{o(1)} = o(n)$, aber nicht $o(n) = n^{o(1)}$!)

Ein wichtiges Beispiel ist:

Lemma 3.2
Für $h(n) = O(n)$ *gilt:*

$$\binom{2^n}{h(n)} = 2^{\Theta(nh(n))} \tag{3.1}$$

Beweis
Es gilt

$$\begin{aligned}\binom{2^n}{h(n)} &= \frac{2^n \cdot (2^n-1)\ldots(2^n-h(n)+1)}{h(n)\cdot\ldots\cdot 1}\\ &\geq \frac{(2^n-h(n)+1)^{h(n)}}{h(n)^{h(n)}}\\ &= \left(\frac{2^n-h(n)+1}{h(n)}\right)^{h(n)}\\ &\geq \left(\frac{2^{n-1}}{2^{\log(h(n))}}\right)^{h(n)}\\ &= \left(2^{n-O(\log(n))}\right)^{h(n)}\\ &= 2^{\Omega(nh(n))}\end{aligned}$$

und

$$\begin{aligned}\binom{2^n}{h(n)} &= \frac{2^n \cdot (2^n-1)\ldots(2^n-h(n)+1)}{h(n)\cdot\ldots\cdot 1}\\ &\leq (2^n)^{h(n)}\\ &= 2^{O(nh(n))}.\end{aligned}$$

□

Dieses Lemma wird bei späteren Beweisen noch häufiger benutzt.

3.1.2 Abschlußeigenschaften

Die Untersuchung von Abschlußeigenschaften ist ein wichtiger Teil der Theorie formaler Sprachen. Grundlegende Eigenschaften von Sprachfamilien spiegeln sich in ihren Abschlußeigenschaften wider. Oft werden auch Abschlußeigenschaften als Hilfsmittel zum Beweis weiterer Sätze benötigt.

Wichtige Operationen sind Homomorphismen und Substitutionen.

Definition 3.3
Seien Σ, Σ' zwei Alphabete.

Eine Abbildung $s : \Sigma^* \to 2^{\Sigma'^*}$, heißt Substitution falls die folgenden Bedingungen erfüllt sind.

1. *$s(w) \neq \emptyset$ für alle $w \in \Sigma^*$.*

2. $s(\varepsilon) = \{\varepsilon\}$.

3. $s(w_1w_2) = s(w_1)s(w_2) = \{xy | x \in s(w_1), y \in s(w_2)\}$ *für alle* $w_1, w_2 \in \Sigma^*$.

Für eine Sprache L definieren wir:

$$s(L) = \bigcup_{w \in L} s(w)$$

Die Substitution s heißt ε–frei, wenn es kein Wort $w \neq \varepsilon$ in Σ^* gibt, für daß $\varepsilon \in s(w)$ gilt.

Eine Substitution h heißt Homomorphismus, falls für jedes $w \in \Sigma^*$ die Sprache $h(w)$ nur ein Wort enthält. Wir schreiben kurz $h(w) = w'$ statt $h(w) = \{w'\}$. Ein Homomorphismus heißt ε–frei, wenn er eine ε–freie Substitution ist.

Für einen Homomorphismus h definieren wir:

$$h^{-1}(L) = \{w \mid h(w) \in L\} \tag{3.2}$$

Wir nennen die Abbildung h^{-1} einen inversen Homomorphismus.

Bemerkung 3.4
Eine Substitution ist gemäß dieser Definition eine Abbildung von $\Sigma^* \to 2^{\Sigma'^*}$. Diese Abbildung wird ausgedehnt zu einer Abbildung von Sprachen $2^{\Sigma^*} \to 2^{\Sigma'^*}$. Wir werden im folgenden oft zwischen den beiden Interpretationen einer Substitution wechseln, ohne dies explizit anzugeben.

Bei einem Homomorphismus identifizieren wir einelementige Teilmengen von Σ'^* mit dem entsprechenden Wort und betrachten daher Homomorphismen auch als Abbildungen von $\Sigma^* \to \Sigma'^*$.

Faßt man einen Homomorphismus h als Abbildung von $\Sigma^* \to \Sigma'^*$ auf und betrachtet die Mengen Σ^*, Σ'^* zusammen mit der Konkatenation von Wörtern als die von Σ bzw. Σ' erzeugten freien Monoide, so ist h ein Homomorphismus zwischen Monoiden. (Dies erklärt die Wahl des Wortes „Homomorphismus“.)

Zunächst werden die im folgenden betrachteten Operationen auf Sprachen und die damit verbundenen Abschlußeigenschaften formalisiert.

Definition 3.5
Für eine Sprachfamilie $\mathscr{L}$ sagen wir:

1. *$\mathscr{L}$ ist abgeschlossen unter Vereinigung bzw. Durchschnitt, wenn aus $L_1, L_2 \in \mathscr{L}$ auch $L_1 \cup L_2 \in \mathscr{L}$ bzw. $L_1 \cap L_2 \in \mathscr{L}$ folgt. $\mathscr{L}$ ist abgeschlossen unter Komplementbildung, wenn für jede Sprache $L \in \mathscr{L}$,*

die über einem Alphabet Σ definiert ist, $\bar{L} = \Sigma^* \backslash L \in \mathscr{L}$ folgt. Diese drei Abschlußeigenschaften fassen wir unter dem Begriff „Boolesche Operationen“ zusammen.

2. *$\mathscr{L}$ ist abgeschlossen unter Konkatenation, wenn mit $L_1, L_2 \in \mathscr{L}$ auch die Sprache $L_1 L_2 \in \mathscr{L}$ ist. $\mathscr{L}$ ist abgeschlossen unter Iteration (oder Kleenescher Hüllenbildung [29]), wenn $L^* = \bigcup_{i \in \mathbb{N}_0} L^i \in \mathscr{L}$ aus $L \in \mathscr{L}$ folgt. Dabei ist $L^0 = \{\varepsilon\}$ und $L^{i+1} = LL^i$. $\mathscr{L}$ heißt abgeschlossen unter ε–freier Iteration, wenn mit $L \in \mathscr{L}$ auch $L^+ = \bigcup_{i \in \mathbb{N}} L^i \in \mathscr{L}$ gilt.*

3. *$\mathscr{L}$ heißt abgeschlossen unter Schnitt mit regulären Mengen, wenn für $L \in \mathscr{L}$ und eine reguläre Menge R, die Sprache $L \cap R$ in $\mathscr{L}$ liegt.*

4. *$\mathscr{L}$ heißt abgeschlossen unter* markierter Konkatenation, *wenn mit L_1 und L_2 auch $L_1 \bullet L_2$ in $\mathscr{L}$ liegt. Dabei ist $\bullet$ ein beliebiges Zeichen, das nicht in den Alphabeten von L_1 und L_2 vorkommt. $\mathscr{L}$ heißt abgeschlossen unter* markierter Iteration, *wenn mit $L \in \mathscr{L}$ auch die Sprache $(L\bullet)^*$ in $\mathscr{L}$ ist. Dabei ist $\bullet$ ein beliebiges Zeichen, das nicht im Alphabet von L vorkommt.*

5. *$\mathscr{L}$ heißt abgeschlossen unter (ε–freien) Homomorphismen, wenn für jeden (ε–freien) Homomorphismus h aus $L \in \mathscr{L}$ die Beziehung $h(L) \in \mathscr{L}$ folgt.*

6. *$\mathscr{L}$ ist abgeschlossen unter inversen Homomorphismen, wenn für jeden Homomorphismus h und jede Sprache $L \in \mathscr{L}$ gilt: $h^{-1}(L) \in \mathscr{L}$.*

7. *$\mathscr{L}$ heißt abgeschlossen unter Substitution, wenn für jede Sprache $L \subset \Sigma^*$ aus $\mathscr{L}$ und jede Substitution $s : \Sigma^* \to 2^{\Sigma'^*}$ mit $s(a) \in \mathscr{L}$ für alle Zeichen $a \in \Sigma$ die Sprache $s(L)$ in $\mathscr{L}$ liegt. $\mathscr{L}$ heißt abgeschlossen unter ε–freier Substitution, wenn für jede ε–freie Substitution s, die die oben genannte Bedingung erfüllt, $s(L) \in \mathscr{L}$ ist.*

8. *$\mathscr{L}$ heißt abgeschlossen unter Spiegelung, wenn mit $L \in \mathscr{L}$ auch*

$$L^R = \left\{ w^R = w_n \dots w_1 \mid w = w_1 \dots w_n \in L \right\}$$

in $\mathscr{L}$ liegt.

Im allgemeinen betrachtet man nur Sprachfamilien $\mathscr{L}$, für die mit L auch $L \cup \{\varepsilon\}$ und $L \backslash \{\varepsilon\}$ in $\mathscr{L}$ liegen. Zum Beispiel lassen sich durch monotone Grammatiken keine Sprachen definieren, die das leere Wort enthalten. Trotzdem definiert man die Familie aller kontextsensitiven Sprachen, so daß das

leere Wort in einer kontextsensitiven Sprache enthalten sein kann, und erreicht damit, daß die Familie der kontextsensitiven Sprachen die obige Bedingung erfüllt. Dies ist wichtig, damit alle kontextfreien Sprachen und nicht nur die ε–freien kontextfreien Sprachen in der Familie der kontextsensitiven Sprachen enthalten sind.

Auch die von uns betrachteten Sprachfamilien $\mathscr{L}_{rt}(\mathrm{CA})$, $\mathscr{L}_{t(n)}(\mathrm{NIA})$, $\mathscr{L}_{t(n)}(f(n)-\mathrm{NIA})$ usw. erfüllen diese Voraussetzung.

3.2 Zeitberechenbare Funktionen

Häufig muß man für eine Funktion $t : \mathbb{N}_0 \to \mathbb{N}$ zusätzliche Eigenschaften annehmen, um Resultate über die Sprachfamilien der Form $\mathscr{L}_{t(n)}(\mathrm{IA})$ zu erhalten. Eine der wichtigsten Eigenschaften in diesem Zusammenhang ist die *Zeitberechenbarkeit*.

Zum Beispiel gilt aufgrund der Definition 2.14 $\mathscr{L}_{t(n)}(\mathrm{IA}) \subseteq \mathscr{L}_{t'(n)}(\mathrm{IA})$ für $t(n) \leq t'(n)$. Später zeigen wir die Echtheit der Inklusion unter der Voraussetzung, daß t' zeitberechenbar ist und $t(n) = o(t'(n))$ gilt.

Wir führen nun den Begriff der Zeitberechenbarkeit ein.

Definition 3.6
Wir nennen eine Funktion $t : \mathbb{N}_0 \to \mathbb{N}$ (mit $t(n) \geq n+1$) zeitberechenbar, *wenn die Sprache $\{a^n b^{t(n)-1-n} \mid n \in \mathbb{N}\}$ in Realzeit von einem iterativen Array erkannt werden kann. Die Menge aller zeitberechenbaren Funktionen bezeichnen wir mit $\mathscr{C}$.*

Dies ist eine sinnvolle Einschränkung auf im Modell berechenbare Funktionen. Wir werden in diesem Abschnitt sehen, daß viele Funktionen zeitberechenbar sind. Außerdem folgt aus Resultaten der allgemeinen Komplexitätstheorie, daß ohne weitere Zusatzvoraussetzungen die Echtheit der Inklusion $\mathscr{L}_{t(n)}(\mathrm{IA}) \subseteq \mathscr{L}_{t'(n)}(\mathrm{IA})$ nicht bewiesen werden kann (siehe [3]).

Der folgende Satz gibt eine alternative Charakterisierung der zeitberechenbaren Funktionen.

Satz 3.7
Die beiden folgende Aussagen sind äquivalent:

1. *$t : \mathbb{N}_0 \to \mathbb{N}$ ist zeitberechenbar.*

2. *Es gibt ein iteratives Array, dessen Ursprungszelle bei jeder Eingabe der Länge n zum Zeitpunkt $t(n)$ in einen ausgezeichneten Zustand wechselt.*

Beweis
Es sei t zeitberechenbar. Ein iteratives Array A, das bei jeder Eingabe der Länge n zum Zeitpunkt $t(n)$ einen ausgezeichneten Zustand annimmt, simuliert einfach das iterative Array A', das $\{a^n b^{t(n)-1-n} \mid n \in \mathbb{N}\}$ erkennt. (O.B.d.A. benötigt das iterative A' genau $n+1$ Zeitschritte für seine Realzeitberechnung.) Dabei wird jedes Zeichen der Eingabe als a und jedes Eingabeendesymbol $\sqcup$ als b interpretiert. Zusätzlich überprüft A beim Lesen des k–ten Zeichen $\sqcup$, ob A' das Wort $a^n b^{k-1}$ erkennt. A wechselt in seinen ausgezeichneten Zustand, wenn dies der Fall ist. Dann gilt

$$n + k = n + (t(n) - n - 1) + 1 = t(n),$$

d.h. A wechselt zum Zeitpunkt $t(n)$ in den ausgezeichneten Zustand.

Für die umgekehrte Richtung sei A ein iteratives Array, das bei jeder Eingabe der Länge n nach $t(n)$ Schritten in einen ausgezeichneten Zustand übergeht. Ein iteratives Array A' kann $\{a^n b^{t(n)-1-n} \mid n \in \mathbb{N}\}$ erkennen, indem es die as als ein beliebiges Eingabezeichen und die bs als das Eingabeendesymbol $\sqcup$ interpretiert und dabei A simuliert. Das Wort $a^n b^{t(n)-n-1}$ wird erkannt, wenn die Simulation von A ihren ausgezeichneten Zustand annimmt, sobald das erste tatsächliche $\sqcup$ Zeichen gelesen wird. □

Bemerkung 3.8
Häufig wird auch Aussage 2 von Satz 3.7 als Definition von Zeitberechenbarkeit verwendet.

Man kann zeitberechenbare Funktionen auch für andere Maschinenmodelle definieren. Man schreibt dann $\mathscr{C}$(Automatentyp) für die Klasse der durch Automatentyp berechenbaren Funktionen. Da wir in dieser Arbeit nur iterative Arrays betrachten, schreiben wir kurz $\mathscr{C}$ statt $\mathscr{C}$(IA).

Ein Überblick, inwieweit die Klasse der zeitberechenbaren Funktionen vom verwendeten Automatentyp abhängt, gibt [7].

Weiter definieren wir *zeitkonstruierbare* Funktionen.

Definition 3.9
Eine streng monotone Funktion $t : \mathbb{N}_0 \to \mathbb{N}$ heißt zeitkonstruierbar, *wenn es ein iteratives Array gibt, das ohne Eingabe zu den Zeitpunkten $t(1), t(2), t(3), \ldots$ einen ausgezeichneten Zustand annimmt. Wir bezeichnen die Klasse aller zeitkonstruierbaren Funktionen mit $\mathscr{F}$.*

Lemma 3.10
Ist $t : \mathbb{N}_0 \to \mathbb{N}$ zeitkonstruierbar, dann ist t auch zeitberechenbar.

Beweis
Ein iteratives Array A kann zu Beginn seiner Berechnung einen Zeitkonstruierer für t starten. Zusätzlich speichert A seine n Eingabesymbole in einem Keller ab. Ein Keller (oder LiFo–Speicher) ist eine Variante der FiFo–Schlange, bei der das zuletzt gespeicherte Symbol als erstes gelesen wird.

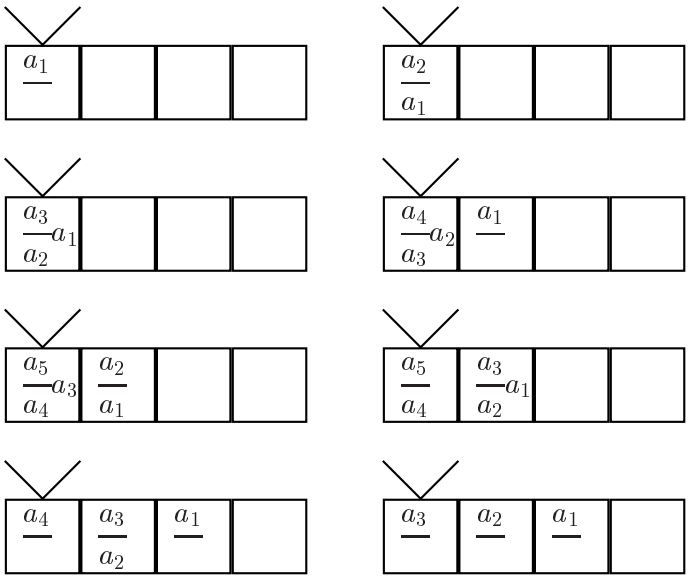


Abbildung 3.1: Ein Keller. In den ersten 5 Schritten wird jeweils ein Zeichen in dem Keller gespeichert. Danach folgt ein Schritt, in dem weder aus dem Keller gelesen noch in den Keller gespeichert wird. In den letzten zwei Schritten wird jeweils ein Zeichen aus dem Keller gelesen.

Wie bei der FiFo–Schlange benötigt jede Zelle drei Register, um einen Keller zu simulieren. In den beiden ersten Registern werden die Einträge des Kellers gespeichert. Das dritte Register dient als Überlaufregister. Es gelten folgende Regeln:

- Ist eines der beiden ersten Register leer und enthält die rechte Nachbarzelle Einträge, so wird der erste dieser Einträge von rechts nach links bewegt.
- In jedem Schritt wird das Zeichen im Überlaufregister um eine Zelle nach rechts in das erste Register kopiert. Der Eintrag des ersten Registers wandert in das zweite Register und der Eintrag des zweiten Registers wird in dem zugehörigen Überlaufregister gespeichert.

Da jede Zelle zwei Einträge des Kellers speichert, kommt es beim Auslesen zu keiner Zeitverzögerung.

Zu den Zeitpunkten $t(1), t(2), \ldots, t(n)$ entfernt A ein Zeichen vom Keller. Auf diese Weise kann A bis n zählen und somit den Zeitpunkt $t(n)$ markieren. Gemäß Satz 3.7 ist daher t eine zeitberechenbare Funktion. □

In [33] wird die Klasse der zeitkonstruierbaren Funktionen genauer untersucht. Zum Beispiel sind n, $2n$, $n + \log(n)$, n^2, 2^n, $n!$ zeitkonstruierbar und somit auch zeitberechenbar. Außerdem ist die Klasse der zeitkonstruierbaren Funktionen abgeschlossen unter Addition, Multiplikation und Kompostion.

3.3 Reduktionssätze

In die Definition der Sprachklasse $\mathscr{L}_{t(n)}(f(n) - \mathrm{NIA})$ gehen die Funktionen t und f für die Beschränkung der Zeit bzw. des Nichtdeterminismus ein. In diesem Abschnitt untersuchen wir, wie diese Funktionen auf gewisse „Normalformen“ reduziert werden können.

Zuerst betrachten wir das Beschleunigungslemma für die Zeit.

Lemma 3.11 (Beschleunigungslemma)
Für jede Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ für die Anzahl der nichtdeterministischen Schritte gelten die Beziehungen:

1. *Für jede Funktion $t : \mathbb{N}_0 \to \mathbb{N}$ mit $t(n) \geq n + 1$ und jede Konstante $k \in \mathbb{N}$ gilt:*

$$\mathscr{L}_{t(n)+k}(f(n) - \mathrm{NIA}) = \mathscr{L}_{t(n)}(f(n) - \mathrm{NIA}) \tag{3.3}$$

   (Additive Konstanten in der Zeitbeschränkung können weggelassen werden.)

2. *Für jede Funktion $t : \mathbb{N}_0 \to \mathbb{N}$ und jede Konstante $k \in \mathbb{N}$ gilt:*

$$\mathscr{L}_{n+kt(n)}(f(n) - \mathrm{NIA}) = \mathscr{L}_{n+t(n)}(f(n) - \mathrm{NIA}) \tag{3.4}$$

   (Oberhalb von Realzeit können multiplikative Konstanten in der Zeitbeschränkung weggelassen werden.)

Beweis
Teil 2 des Lemmas wird analog zu dem entsprechenden Lemma für deterministische iterative Arrays in [25] (Theorem 3.1.) bewiesen. Der Trick ist, daß jede Zelle des schnellen Automaten in k Register aufgeteilt wird. In jedem dieser Register wird eine Zelle des langsamen Automaten simuliert. In den ersten n Schritten kann kein Geschwindigkeitsvorteil erzielt werden, da

die Eingabe nur in n Schritten gelesen werden kann. Danach ist eine Beschleunigung um Faktor k möglich, da jedes Register mit allen Registern, die höchstens k Einheiten weit entfernt sind, verbunden ist.

Bei Teil 1 des Lemmas simuliert nur die Ursprungszelle des schnellen Automaten k Zellen des langsamen Automaten. □

Der Teil 2 des Lemmas rechtfertigt insbesondere die Bezeichnung Linearzeit für die Funktion $t(n) = 2n$, denn alle Zeitbeschränkungen der Form $k \cdot n$ mit einer Konstanten $k > 1$ ergeben die gleiche Komplexitätsklasse.

Für die Beschränkung der nichtdeterministischen Schritte gelten ganz ähnliche Aussagen.

Lemma 3.12 (Reduktion der nichtdeterministischen Schritte)
Für jede Funktion $t : \mathbb{N}_0 \to \mathbb{N}$ mit $t(n) \geq n + 1$ für die Zeitbeschränkung gelten die Beziehungen:

1. *Für jede Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ und jede Konstante $k \in \mathbb{N}$ gilt:*

$$\mathscr{L}_{t(n)}((f(n) + k) - \text{NIA}) = \mathscr{L}_{t(n)}(f(n) - \text{NIA}) \tag{3.5}$$

   (Endlich viele nichtdeterministische Schritte können deterministisch simuliert werden.)

2. *Für jede Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ und jede Konstante $k \in \mathbb{N}$ gilt:*

$$\mathscr{L}_{t(n)}(kf(n) - \text{NIA}) = \mathscr{L}_{t(n)}(f(n) - \text{NIA}) \tag{3.6}$$

   (Jeder nichtdeterministische Schritt kann endlich viele nichtdeterministische Schritte simulieren.)

Beweis
Aufgrund der Definition von $\mathscr{L}(f(n) - \text{NIA})$ folgt aus $f(n) \leq f'(n)$ für fast alle $n \in \mathbb{N}$, daß $\mathscr{L}_{t(n)}(f(n) - \text{NIA}) \subseteq \mathscr{L}_{t(n)}(f'(n) - \text{NIA})$ gilt. Wir müssen daher jeweils nur eine Inklusion zeigen.

Offensichtlich reicht es für den Beweis von Teil 1, den Fall $k = 1$ zu betrachten. Wir zeigen $\mathscr{L}_{t(n)}((f(n) + 1) - \text{NIA}) \subseteq \mathscr{L}_{t(n)}(f(n) - \text{NIA})$.

Sei A ein $(f(n) + 1) - \text{NIA}$. In jedem nichtdeterministischen Schritt verzweige sich A höchstens in r verschiedene Berechnungspfade. Ein $f(n) - \text{NIA}$ A', das A simuliert, verfügt über r verschiedene Register. Den ersten nichtdeterministischen Schritt von A simuliert A' mit einem deterministischen Schritt, indem es jede Wahlmöglichkeit von A in einem Register simuliert.

Die nächsten nichtdeterministischen Schritte von A simuliert A' durch nichtdeterministische Schritte. A' erkennt, wenn in mindestens einem Register die Simulation von A zu einem erkennenden Endzustand führt. Wir

müssen noch sicherstellen, daß A' nicht mehr als $f(n)$ nichtdeterministische Schritte macht, d.h. daß alle Register von A' ihre nichtdeterministischen Schritte gleichzeitig durchführen.

Dazu legt A' für jedes der r Register einen Keller an.

In diesen Kellern kann A' jeweils Zahlen zwischen 1 und r (einschließlich) speichern.

Falls A' im Register i einen nichtdeterministischen Schritt von A simulieren muß, überprüft A', ob der Keller für das Register i leer ist. Ist dies der Fall, so rät A' in einem nichtdeterministischen Schritt für jedes der r Register eine Zahl zwischen 1 und r. (A' kann sich also in einem nichtdeterministischen Schritt in r^r Berechnungspfade verzweigen.) Nun ist der Keller nicht leer und A' kann die im nächsten Absatz beschriebene Rechnung ausführen.

Ist der Keller nicht leer, so liest A' eine Zahl k vom Keller und wählt danach die k-te Verzweigung von A. Somit kann in diesem Fall A' den nichtdeterministischen Schritt von A deterministisch simulieren. (Hat A in dem entsprechenden Schritt weniger als k Verzweigungsmöglichkeiten, so simuliert A' die erste Möglichkeit von A).

Teil 2 kann auf ähnliche Art gezeigt werden. Wir müssen

$$\mathscr{L}_{t(n)}(kf(n) - \text{NIA}) \subseteq \mathscr{L}_{t(n)}(f(n) - \text{NIA})$$

beweisen.

Sei A ein $kf(n) - \text{NIA}$ und A' ein $f(n) - \text{NIA}$. Wieder benutzt A' einen Keller, in dem es Anweisungen speichert, wie es einen nichtdeterministischen Schritt von A zu simulieren hat.

Jedesmal wenn A' einen nichtdeterministischen Schritt von A simulieren muß, benutzt A' seinen Keller. Ist der Keller nicht leer, so befolgt A' die Anweisungen, die im Keller gespeichert wurden, und kann auf diese Art den nichtdeterministischen Schritt von A deterministisch simulieren. Ist der Keller leer, so benutzt A' einen nichtdeterministischen Schritt, um den nichtdeterministischen Schritt von A zu simulieren. Zusätzlich rät A' noch Verzweigungsanweisungen für die nächsten $k-1$ nichtdeterministischen Schritte von A und speichert sie in dem Keller.

Auf diese Weise kann A' jeweils k nichtdeterministische Schritte von A durch einen simulieren. □

Insbesondere folgt aus Teil 1 des Lemmas, daß endlich viele zusätzliche nichtdeterministische Schritte die Berechnungsfähigkeit des Modells nicht steigern. Im folgenden werden wir daher nur noch $f(n) - \text{NIAs}$ mit $f = \omega(1)$ betrachten. Aus Teil 2 des Lemmas folgt

$$\mathscr{L}_{t(n)}(\Theta(f(n)) - \text{NIA}) = \mathscr{L}_{t(n)}(f(n) - \text{NIA}) \quad .$$

Für entsprechend definierte nichtdeterministische Zellularautomaten, erhält man nur für $f(n) > 0$ ein zu Teil 1 analoges Lemma (siehe [4]).

3.4 Äquivalenzklassen

Daß eine Sprache L von einem $f(n)-$NIA erkannt werden kann, läßt sich durch Angabe eines Algorithmus beweisen. In diesem Abschnitt wird ein Hilfsmittel erarbeitet, mit dem wir zeigen können, daß bestimmte Sprachen *nicht* von einem $f(n)-$NIA erkannt werden können.

Zunächst definieren wir für eine feste Sprache L eine Äquivalenzrelation auf der Menge aller Wörter:

Definition 3.13
Sei $L \subseteq \Sigma^*$ eine Sprache über dem Alphabet Σ und sei $l \in \mathbb{N}$ eine Konstante.

1. *Zwei Wörter w und w' über Σ heißen l–äquivalent bezüglich L, wenn*

$$ww_l \in L \iff w'w_l \in L \text{ für alle } w_l \in \Sigma^l \tag{3.7}$$

   gilt.

2. *Mit $N(n,l,L)$ bezeichnen wir die Anzahl der l-Äquivalenzklassen bezüglich L unter den Wörtern der Länge $n-l$ (d.h. $|ww_l| = n$).*

Für Sprachen L, die von $f(n)-$NIAs in Realzeit erkannt werden können, läßt sich $N(n,l,L)$ abschätzen.

Lemma 3.14
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine Abbildung mit $f(n) \leq n+1$. Gilt $L \in \mathscr{L}_{rt}(f(n)-\text{NIA})$, so gibt es Konstanten s und r, so daß

$$N(n,l,L) \leq (s^l)^{r^{f(n)}} \tag{3.8}$$

für alle $l, n \in \mathbb{N}$ gilt.

Beweis
Sei $A = (S, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup)$ ein $f(n)-$NIA, das L in Realzeit akzeptiert. Wir definieren:

$$r = \max\{|\delta_0(s_0, s_1, s_2, a)| \mid s_0, s_1, s_2 \in S \text{ und } a \in \Sigma \cup \{\sqcup\}\}$$

Um eine obere Schranke für die Anzahl der l-Äquivalenzklassen zu bestimmen, betrachten wir die Menge aller möglichen Konfigurationen von A

zum Zeitpunkt $n-l$, d.h. A muß noch l Eingabesymbole lesen. Der Zustand der Ursprungszelle nach dem Lesen der gesamten Eingabe hängt jetzt nur noch von den Zellen an den Positionen $-l-1,\ldots,0,\ldots,l+1$ ab. Da dies $2l+3$ Zellen sind, ergeben sich $|S|^{2l+3}$ verschiedene Möglichkeiten. Für $s=|S|^3$ ergibt sich: $|S|^{2l+3} \leq |S|^{3l} = s^l$ für $l \geq 2$.

Durch die $f(n)$ nichtdeterministischen Schritte können höchstens $r^{f(n)}$ Berechnungspfade entstehen. Das iterative Array A hat also für die Entscheidung, ob es akzeptiert oder nicht, höchstens s^l verschiedene Möglichkeiten in $r^{f(n)}$ Berechnungspfaden, d.h. die Anzahl $N(n,l,L)$ ist durch $(s^l)^{(r^{f(n)})}$ beschränkt. □

Man kann genau wie im determistischen Fall auch d–dimensionale nichtdeterministische iterative Arrays definieren. Die entsprechenden Sprachklassen bezeichen wir mit $\mathscr{L}_{t(n)}(\mathrm{NIA}_d)$ bzw. $\mathscr{L}_{t(n)}(f(n)-\mathrm{NIA}_d)$.

Der Beweis von Lemma 3.14 läßt sich leicht auf mehrdimensionale iterative Arrays verallgemeinern. Im d–dimensionalen Fall wird die Ursprungszelle in l Schritten nur von Zellen beeinflußt, die in einem Würfel mit der Kantenlänge $2l+3$ um den Ursprung liegen. Die Argumentation von Lemma 3.14 führt dann zu folgender Abschätzung.

Lemma 3.15
Sei $f:\mathbb{N}_0\to\mathbb{N}_0$ eine Abbildung mit $f(n)\leq n+1$. Gilt $L\in\mathscr{L}_{rt}(f(n)-\mathrm{NIA}_d)$, so gibt es Konstanten s und r, so daß

$$N(n,l,L) \leq (s^{l^d})^{r^{f(n)}} \tag{3.9}$$

für alle $l,n\in\mathbb{N}$ gilt.

Man kann das Lemma 3.14 auch auf Zeitbeschränkungen oberhalb von Realzeit verallgemeinern.

Lemma 3.16
Sei $f:\mathbb{N}_0\to\mathbb{N}_0$ eine Beschränkung der nichtdeterministischen Schritte und $t:\mathbb{N}_0\to\mathbb{N}$ eine Zeitbeschränkung. Gilt $L\in\mathscr{L}_{t(n)}(f(n)-\mathrm{NIA})$, so gibt es Konstanten s und r, so daß

$$N(n,l,L) \leq (s^{l+t(n)-n})^{r^{f(n)}} \tag{3.10}$$

für alle $l,n\in\mathbb{N}$ gilt.

Beweis
Interpretiert man das Zeichen ␣ als ein „normales“ Zeichen des Alphabets, so kann man jedes $t(n)$–zeitbeschränkte iterative Array als ein Realzeit beschränktes iteratives Array auffassen, das statt des Wortes w das Wort $w␣^{t(n)-n}$

als Eingabe erhält. Setzen wir also $L' = \{w\sqcup^{t(n)-n} \mid w \in L\}$, so können wir Lemma 3.14 auf L' anwenden und erhalten

$$N(n,l,L) = N(t(n), l + (t(n) - n), L') \leq (s^{l+t(n)-n})^{rf(n)}.$$

□

Kapitel 4

Abschlußeigenschaften

Die Familie $\mathcal{REG}$ der regulären Sprachen ist als die Familie aller Sprachen, die durch endliche Automaten akzeptiert werden, definiert. Der Satz von Kleene und Myhill [34] liefert folgende alternative Charakterisierung der regulären Sprachen.

Satz 4.1 (Kleene und Myhill)
Die Familie der regulären Sprachen ist die kleinste Menge, die alle endlichen Sprachen enthält und unter Vereinigung, Konkatenation und Iteration abgeschlossen ist.

Eine ähnliche Charakterisierung für von nichtdeterministischen iterativen Arrays erzeugte Sprachen ist sehr nützlich für die weitere Untersuchung.

Um eine derartige Charakterisierung formulieren zu können, muß man zunächst eine Menge von Operationen ermitteln, unter denen die entsprechende Sprachfamilie abgeschlossen ist. Im folgenden untersuchen wir die Abschlußeigenschaften der Sprachfamilien $\mathscr{L}_{rt}(f(n) - \text{NIA})$ und $\mathscr{L}_{rt}(\text{NIA})$.

4.1 Nichtdeterministische iterative Arrays

Zunächst untersuchen wir nichtdeterministische iterative Arrays ohne Beschränkung des Nichtdeterminismus. Hauptergebnis dieses Abschnitts ist eine Charakterisierung von $\mathscr{L}_{rt}(\text{NIA})$ mittels ε–freier Homomorphismen. Aus diesem Ergebnis können wir dann viele Eigenschaften der Familie $\mathscr{L}_{rt}(\text{NIA})$ ableiten.

4.1.1 Charakterisierung durch Homomorphismen

Grundlage aller Resultate dieses Kapitels ist der folgende Satz, der die Sprachfamilie $\mathscr{L}_{rt}$(NIA) mit der Sprachfamilie $\mathscr{L}_{rt}$(IA) in Verbindung bringt:

Satz 4.2

1. *Für jede Sprache $L \in \mathscr{L}_{rt}$(NIA) gibt es eine Sprache $L' \in \mathscr{L}_{rt}$(IA) und einen ε–freien Homomorphismus h mit $h(L') = L$.*

2. *Für jede Sprache $L \in \mathscr{L}_{rt}$(IA) und jeden ε–freien Homomorphismus h liegt $h(L)$ in $\mathscr{L}_{rt}$(NIA).*

Beweis
Zum Beweis von 1 betrachten wir eine Sprache $L \in \mathscr{L}_{rt}$(NIA). Sei

$$A = (S, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup)$$

ein NIA, das L in Realzeit erkennt.

Ähnlich zum Beweis von Teil 1 von Lemma 3.12 können wir A so konstruieren, daß A im letzten Zeitschritt keinen nichtdeterministischen Schritt ausführt, d.h. daß $|\delta_0(a, b, c, \sqcup)| = 1$ für alle $a, b, c \in S$ gilt.

Wir definieren die Sprache L' über dem Alphabet $\Sigma \times S$ durch

$$(a_1, s_1) \cdots (a_n, s_n) \in L',$$

falls $w = a_1 \cdots a_n \in L$ gilt und A bei einer erkennenden Berechnung nach Eingabe von w die Konfigurationen $c_0, \ldots, c_{n+1}$ durchläuft mit $c_i(0) = s_i$. (Das heißt, die Ursprungszelle von A durchläuft die Zustände $s_0, \ldots, s_{n+1}$. Es gilt $s_0 = q_0$ und $s_{n+1} \in F_a$.)

Als Homomorphismus h wählen wir die Projektion auf die erste Komponente. Offensichtlich ist $h(L') = L$. Es bleibt zu zeigen, daß $L' \in \mathscr{L}_{rt}$(IA) gilt.

Ein deterministisches iteratives Array A' kann L' erkennen, indem es deterministisch verifiziert, ob die Ursprungszelle von A die Zustände $s_1 \ldots s_n$ annehmen kann. Der restliche Teil der Rechnung von A ist deterministisch und kann direkt verifiziert werden. Formal gilt

$$A' = (S \cup \{err\}, \Sigma \times S, F_a, F_r \cup \{err\}, \delta, \delta_0', q_0, \sqcup)$$

mit

1. $\delta_0'(a, b, c, (s, d)) = d$, falls $d \in \delta_0(a, b, c, s)$ gilt.

2. $\delta_0'(a, b, c, (s, d)) = err$, falls $d \notin \delta_0(a, b, c, s)$ gilt.

3. $\delta'_0(a,b,c,\sqcup) = \delta_0(a,b,c,\sqcup)$. (Beachte, daß $|\delta_0(a,b,c,\sqcup)| = 1$ gilt.)

Es ist offensichtlich, daß A' ein Wort genau dann akzeptiert, wenn es in L' liegt.

Zum Beweis von 2 betrachten wir eine Sprache $L \in \mathscr{L}_{rt}(\text{IA})$ und einen ε–freien Homomorphismus $h : \Sigma^* \to \Sigma'^*$. L sei über dem Alphabet $\Sigma = \{a_1, \ldots, a_n\}$ definiert. Es gelte $h(a_i) = a_i^{(1)} a_i^{(2)} \cdots a_i^{(k_i)}$.

Wir definieren das Alphabet $\Sigma'' = \{\hat{a}_i^{(j)} \mid 1 \leq i \leq n \text{ und } 1 \leq j \leq k_i\}$. Dabei seien im Gegensatz zu Σ' alle $\hat{a}_i^{(j)}$ verschieden.

Wir betrachten die Homomorphismen $h' : \Sigma''^* \to \Sigma'^*$ und $h'' : \Sigma''^* \to \Sigma^*$ mit

$$h'(\hat{a}_i^{(j)}) = a_i^{(j)}$$

und

$$h''(\hat{a}_i^{(j)}) = \begin{cases} a_i & \text{für } j = 1 \\ \varepsilon & \text{für } j \neq 1 \end{cases}.$$

Es sei

$$L' = h''^{-1}(L) \cap \{\hat{a}_1^{(1)} \ldots \hat{a}_1^{(k_1)}, \ldots, \hat{a}_n^{(1)} \ldots \hat{a}_n^{(k_n)}\}^*.$$

Da $\mathscr{L}_{rt}(\text{IA})$ abgeschlossen ist bezüglich inversen Homomorphismen und Schnitt mit regulären Mengen (siehe [41] und [12]), folgt $L' \in \mathscr{L}_{rt}(\text{IA})$. Nach Konstruktion gilt $h'(L') = h(L)$.

Sei A ein deterministisches iteratives Array, das L' in Realzeit erkennt. Ein NIA, das $h'(L')$ in Realzeit erkennt, rät bei Eingabe von $a_i^{(j)}$ das Urbild $\hat{a}_i^{(j)}$ unter h' und simuliert danach A. Offensichtlich kann das NIA auf diese Weise nur Wörter aus $h'(L')$ erkennen und zu jedem Wort aus $h'(L')$ gibt es eine erkennende Berechnung des NIA. Also gilt $h(L) = h'(L') \in \mathscr{L}_{rt}(\text{NIA})$. □

Korollar 4.3
Sei $t : \mathbb{N}_0 \to \mathbb{N}$ eine monoton wachsende Funktion, so daß für jede Konstante k die Gleichung $t(kn) = O(t(n))$ erfüllt ist.

Dann gilt

1. *Für jede Sprache $L \in \mathscr{L}_{n+t(n)}(n - \text{NIA})$ gibt es eine Sprache $L' \in \mathscr{L}_{n+t(n)}(\text{IA})$ und einen ε–freien Homomorphismus h mit $h(L') = L$.*

2. *Für jede Sprache $L \in \mathscr{L}_{n+t(n)}(\text{IA})$ und jeden ε–freien Homomorphismus h liegt $h(L)$ in $\mathscr{L}_{n+t(n)}(n - \text{NIA})$.*

Beweis
Zum Beweis von Teil 1 benötigten wir lediglich, daß das nichtdeterministische iterative Array für jedes Zeichen der Eingabe höchstens einen nichtdeterministischen Schritt ausgeführt hat.

Zum Beweis von Teil 2 benötigen wir nur noch den Abschluß von $\mathscr{L}_{n+t(n)}(\mathrm{IA})$ unter inversen Homomorphismen und Schnitt mit regulären Mengen. Daß $\mathscr{L}_{n+t(n)}(\mathrm{IA})$ diese Abschlußeigenschaften besitzt, wird durch die Forderung an t gewährleistet. □

Bemerkung 4.4
Auf ähnliche Weise kann man auch die kontextfreien Sprachen als Abschluß der deterministischen kontextfreien Sprachen unter ε–freien Homomorphismen charakterisieren. (Da die kontextfreien Sprachen unter beliebigen Homomorphismen abgeschlossen sind, bilden sie sogar den Abschluß der deterministischen kontextfreien Sprachen unter Homomorphismen (siehe [1]).)

In [4] wird die Sprachfamilie $\mathscr{L}_{rt}(1G-\mathrm{OCA})$ eingeführt und gezeigt, daß sie den Abschluß der Sprachfamilie $\mathscr{L}_{rt}(\mathrm{OCA})$ unter ε–freien Homomorphismen bildet.

Nach Satz 4.2 ist die Sprachfamilie $\mathscr{L}_{rt}(\mathrm{NIA})$ der Abschluß von $\mathscr{L}_{rt}(\mathrm{IA})$ unter ε–freien Homomorphismen. Wir definieren allgemein:

Definition 4.5
Für eine Sprachfamilie $\mathscr{L}$ setzen wir:

$$H(\mathscr{L}) = \{h(L) \mid L \in \mathscr{L} \text{ und } h \text{ ist ein } \varepsilon\text{-freier Homomorphismus}\} \quad (4.1)$$

Wir nennen $H(\mathscr{L})$ den Abschluß von $\mathscr{L}$ unter ε–freien Homomorphismen.

Man beachte, daß H keinen speziellen Homomorphismus bezeichnet.

4.1.2 AFLs und prä–AFLs

Alle Eigenschaften der Sprachfamilie $\mathscr{L}_{rt}(\mathrm{NIA})$ lassen sich aus Satz 4.2 ableiten. Grundlage dieser Untersuchungen ist der Begriff der *abstrakten Sprachfamilie*. Eine Sprachfamilie ist eine Menge von Sprachen, die mindestens eine nichtleere Sprache enthält.

Definition 4.6
Eine Sprachfamilie $\mathscr{L}$ heißt abstrakte Sprachfamilie (engl.: abstract family of languages, AFL), wenn sie unter den folgenden Operationen abgeschlossen ist: inversen Homomorphismen, ε–freien Homomorphismen, Schnitt mit regulären Mengen, Vereinigung, Konkatenation und ε–freier Iteration.

Ist $\mathscr{L}$ außerdem abgeschlossen unter beliebigen Homomorphismen, so heißt $\mathscr{L}$ eine volle AFL.

Die in der Definition 4.6 aufgezählten Abschlußeigenschaften sind nicht unabhängig voneinander, denn es gilt folgender Satz:

Satz 4.7 (Greibach, Hopcroft [20])
Eine Sprachfamilie $\mathscr{L}$, für die mit L auch $L\cup\{\varepsilon\}$ und $L\backslash\{\varepsilon\}$ in $\mathscr{L}$ liegen und die unter Konkatenation, Iteration, inversen Homomorphismen und ε–freien Homomorphismen abgeschlossen ist, ist eine AFL.

Im folgenden wollen wir Bedingungen für eine Sprachfamilie $\mathscr{L}$ angeben, unter denen $H(\mathscr{L})$ eine AFL ist. Dazu führen wir den Begriff der prä–AFL ein.

Definition 4.8
Eine Sprachfamilie $\mathscr{L}$ heißt prä–AFL, wenn mit L auch $L\cup\{\varepsilon\}$ und $L\backslash\{\varepsilon\}$ in $\mathscr{L}$ liegen und $\mathscr{L}$ abgeschlossen ist unter markierter Konkatenation, markierter Iteration, inversen Homomorphismen und Schnitt mit regulären Mengen.

Die für eine prä–AFL geforderten Abschlußeigenschaften sind deutlich schwächer als die Eigenschaften einer AFL. Zum einen wird der Abschluß unter ε–freien Homomorphismen und Vereinigung nicht gefordert, zum anderen wurden die Konkatenation und Iteration auf die markierten Varianten abgeschwächt.

Wie bereits in Abschnitt 3.1.2 erwähnt, werden praktisch alle Sprachfamilien $\mathscr{L}$ so definiert, daß mit L auch $L\cup\{\varepsilon\}$ und $L\backslash\{\varepsilon\}$ in $\mathscr{L}$ liegen. Also stellt diese zusätzliche Forderung keine große Einschränkung dar.

Die Bedeutung einer prä–AFL ergibt sich aus dem folgenden Satz.

Satz 4.9 (Ginsburg, Greibach, Hopcroft [18])
Ist $\mathscr{L}$ ein prä-AFL, so ist $H(\mathscr{L})$ eine AFL.

Dieser Satz läßt sich auf die verschiedensten Sprachfamilien anwenden, z.B. bilden die deterministischen kontextfreien Sprachen ein prä–AFL. Die kontextfreien Sprachen sind Abschluß der deterministisch kontextfreien Sprachen unter Homomorphismen, also bilden die kontextfreien Sprachen eine AFL.

Uns interessiert die Anwendung auf iterative Arrays.

Korollar 4.10
$\mathscr{L}_{rt}(\mathrm{IA})$ ist eine prä–AFL, also ist $H(\mathscr{L}_{rt}(\mathrm{IA})) = \mathscr{L}_{rt}(\mathrm{NIA})$ eine AFL.

Beweis
Daß $\mathscr{L}_{rt}$(IA) unter inversen Homomorphismen und Schnitt mit regulären Mengen abgeschlossen ist, wurde in [41] und [12] gezeigt. Wir zeigen hier nur, daß iterative Arrays auch unter den etwas ungewöhnlicheren Operationen markierte Konkatenation und markierte Iteration abgeschlossen sind.

Sei $A = (S, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup)$ ein iteratives Array, das die Sprache L in Realzeit erkennt. Wir nehmen o.B.d.A. an, daß A zu Beginn ein Signal mit Geschwindigkeit 1 erzeugt, das alle durchlaufenen Zellen in den Zustand q_0 versetzt, d.h. A kann seine Berechnung auch dann durchführen, wenn nicht alle Zellen zu Beginn im Ruhezustand q_0 sind. Außerdem nimmt die Ursprungszelle von A den Ruhezustand nur am Anfang an.

Das iterative Array $A' = (S \cup \{err, acc\}, \Sigma \cup \{\bullet\}, \{acc\}, \{err\}, \delta', \delta_0', q_0, \sqcup)$ ist definiert durch:

Für $a, b, c \in S$ gilt

$$\delta(a,b,c)' = \delta(a,b,c)$$

und

$$\delta_0'(a,b,c,s) = \begin{cases} \delta_0(a,b,c,s) & \text{falls } s \in \Sigma \\ q_0 & \text{falls } s = \bullet \text{ und } b \in F_a \\ err & \text{falls } s = \bullet \text{ und } b \notin F_a \\ acc & \text{falls } s = \sqcup \text{ und } b = q_0. \end{cases}$$

(Man beachte, daß der zugrundeliegende Nachbarschaftsindex gleich $(-1, 0, 1)$ ist!) Das iterative Array A' hält an, sobald die Ursprungszelle den Zustand err oder acc annimmt, d.h. wir brauchen die lokalen Überführungsfunktionen für diese Zustände nicht zu definieren. (Die von dem iterativen Array akzeptierte Sprache hängt nicht von der Definition der Überführungsfunktionen in diesen Fällen ab.)

Die Konstruktion stellt sicher, daß A' ein Wort nur erkennen kann, wenn es mit $\bullet$ endet. Steht zwischen zwei $\bullet$ Zeichen kein Wort aus L, so geht A' in den Zustand err über, den es nie wieder verlassen kann. Daher erkennt A' genau die Wörter der Form $(L\bullet)^*$.

Ein iteratives Array, das die markierte Konkatenation zweier Sprachen erkennt, läßt sich auf analoge Weise konstruieren. □

Als nächstes untersuchen wir, wie sich der Abschluß bezüglich Durchschnittsbildung in die Theorie der AFLs einfügt.

Satz 4.11 (Ginsburg, Greibach, Hopcroft [18])
Ist $\mathscr{L}$ eine prä-AFL, die unter Durchschnittsbildung abgeschlossen ist, so ist $H(\mathscr{L})$ eine AFL, die unter Durchschnittsbildung und ε–freier Substitution abgeschlossen ist.

Korollar 4.12
$\mathscr{L}_{rt}$(NIA) ist eine AFL, die unter Durchschnittsbildung und ε–freier Substitution abgeschlossen ist.

Beweis
$\mathscr{L}_{rt}$(IA) ist unter Durchschnittsbildung abgeschlossen [12]. Zusammen mit Korollar 4.10 folgt daher, daß $\mathscr{L}_{rt}$(NIA) eine bezüglich Durchschnittsbildung und ε–freier Substitution abgeschlossene AFL ist. □

4.1.3 Weitere Abschlußeigenschaften

In diesem Abschnitt betrachten wir noch einige andere Abschlußeigenschaften von $\mathscr{L}_{rt}$(NIA), die nicht mit der allgemeinen Theorie der abstrakten Sprachfamilien behandelt werden können.

Satz 4.13
$\mathscr{L}_{rt}$(NIA) ist unter Spiegelung abgeschlossen.

Beweis
Sei $L \in \mathscr{L}_{rt}$(NIA) eine Sprache über Σ. Nach Satz 4.2 gibt es eine Sprache $L' \in \mathscr{L}_{rt}$(IA) über Σ' und einen Homomorphismus $h : \Sigma'^* \to \Sigma^*$ mit $h(L') = L$. Wir definieren eine Sprache L'' über $\Sigma' \times \Sigma'$ durch

$$L'' = \{(a_1, a_n)(a_2, a_{n-1}) \cdots (a_n, a_1) \mid a_1 \cdots a_n \in L'\}.$$

Es gilt $L'' \in \mathscr{L}_{rt}(IA)$. Ein iteratives Array, das L'' erkennt, überprüft, ob das Wort in der ersten Komponente aus L' ist und ob die zweite Komponente das Spiegelbild der ersten ist. Der zweite Teil der Berechnung ist möglich, da Palindrome von iterativen Arrays in Realzeit erkannt werden (siehe [12]). (Um festzustellen, ob ein Wort ein Palindrom ist, muß jeweils das i–te Zeichen mit dem $(n-i)$–ten verglichen werden. Zum Erkennen, daß ein Wort aus L'' ist, müssen wir überprüfen, ob das i–te und das $(n-i)$–te Paar durch Vertauschen ihrer beiden Komponenten ineinander übergehen. Man kann daher jeden Algothimus zum Erkennen von Palindromen direkt in einen Algorthmus zum Erkennen von L'' abwandeln.)

Sei $p_2 : (\Sigma' \times \Sigma')^* \to \Sigma'^*$ die Projektion auf die zweite Komponente und $h^R : \Sigma'^* \to \Sigma^*$ der durch $h^R(w) = h(w^R)^R$ definierte Homomorphismus.

Es gilt $h^R \circ p_2(L'') = h(L')^R = L^R$. Nach Satz 4.2 liegt die Spiegelung L^R von L in $\mathscr{L}_{rt}$(NIA). □

Wie die meisten zeit- und raumbeschränkten Varianten berechnungsuniverseller Modelle ist auch die Sprachfamilie $\mathscr{L}_{rt}$(NIA) nicht abgeschlossen unter Homomorphismen und Substitution. Es gilt nämlich der folgende Satz.

Satz 4.14
Die Familie der rekursiv aufzählbaren Sprachen ist der Abschluß von $\mathscr{L}_{rt}(\mathrm{IA})$ *unter Homomorphismen.*

Insbesondere sind die Sprachfamilien $\mathscr{L}_{t(n)}(\mathrm{IA})$ *und* $\mathscr{L}_{t(n)}(\mathrm{NIA})$ *nicht abgeschlossen unter Homomorphismen und Substitution für alle Zeitbeschränkungen* $t : \mathbb{N}_0 \to \mathbb{N}$.

Beweis
Eine rekursiv aufzählbare Sprache L über Σ kann von einen deterministischen iterativen Array A ohne Zeitbeschränkung akzeptiert werden.

Es sei $b \notin \Sigma$. Wir definieren

$$L' = \{wb^{t-n-1} \mid w \in L \text{ und } time_A(w) \leq t\}.$$

Ein iteratives Array kann L' in Realzeit erkennen, indem es A simuliert und dabei die Zeichen b als das Eingabeendesymbol $\sqcup$ behandelt.

Sei $h : (\Sigma \cup \{b\})^* \to \Sigma^*$ ein Homomorphismus mit $h(a) = a$ für $a \in \Sigma$ und $h(b) = \varepsilon$. Offensichtlich gilt $h(L') = L$, d.h. die rekursiv aufzählbare Sprache L ist homomorphes Bild einer Sprache L' aus $\mathscr{L}_{rt}(\mathrm{IA})$. □

Der Abschluß unter Komplementbildung ist für nichtdeterministische Automaten nur schwer zu zeigen oder zu wiederlegen. So konnte mittlerweile gezeigt werden, daß die von bandbeschränkten nichtdeterministischen Turing–Maschinen erkannten Sprachfamilien unter Komplementbildung abgeschlossen sind ([45, 27]). Das entsprechende Problem für zeitbeschränkte nichtdeterministische Turing–Maschinen ist immer noch ungelöst.

Für allgemeine nichtdeterministische iterative Arrays ist der Abschluß bezüglich Komplementbildung noch nicht geklärt. Im nächsten Abschnitt, werden wir jedoch zeigen, daß iterative Arrays mit beschränktem Nichtdeterminismus nicht unter Komplementbildung abgeschlossen sind (Satz 4.22).

4.2 Iterative Arrays mit beschränktem Nichtdeterminismus

Nun betrachten wir nichtdeterministische iterative Arrays mit Beschränkung des Nichtdeterminismus. Für dieses allgemeinere Modell werden wir keine Charakterisierung durch Abschlußeigenschaften ähnlich zu Satz 4.2 oder dem Satz von Kleene und Myhill erhalten.

Trotzdem werden sich die in diesem Abschnitt bewiesenen Abschlußeigenschaften als wichtig für die spätere Untersuchung herausstellen. Zum Beispiel

werden wir in Kapitel 6 den Satz 4.18 zum Beweis, daß alternierende iterative Arrays mächtiger sind als nichtdeterministische iterative Arrays, benutzen.

Außerdem interessiert uns die Frage, wieviel Symmetrie im Vergleich zu den Grenzfällen $\mathscr{L}_{rt}(\text{IA})$ und $\mathscr{L}_{rt}(\text{NIA})$ verlorengeht.

4.2.1 Konkatenation und Iteration

Die Sprachfamilie $\mathscr{L}_{rt}(\text{NIA})$ ist unter Konkatenation und Iteration abgeschlossen. Im Gegensatz dazu ist $\mathscr{L}_{rt}(\text{IA})$ nur unter markierter Konkatenation und markierter Iteration abgeschlossen. (Siehe dazu auch Abschnitt 4.1.)

Im Fall der allgemeinen Sprachfamilien $\mathscr{L}_{rt}(f(n)-\text{NIA})$ sind die Verhältnisse etwas komplizierter. Wir betrachten zunächst den Fall der markierten Konkatenation.

Satz 4.15
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine monoton wachsende Funktion. Dann ist $\mathscr{L}_{rt}(f(n)-\text{NIA})$ abgeschlossen unter markierter Konkatenation.

Beweis
Für zwei Sprachen L_1 und L_2, die von den $f(n) - \text{NIAs}$ A_1 und A_2 in Realzeit erkannt werden, ist ein $f(n) - \text{NIA}$ A zu konstruieren, das die Sprache $\{w_1 \bullet w_2 \mid w_i \in L_i, i = 1, 2\}$ in Realzeit erkennt.

Sei $n_1 = |w_1|$, $n_2 = |w_2|$ und $|w_1 \bullet w_2| = n = n_1 + n_2 + 1$.

A simuliert zunächst den Automaten A_1. Nach dem Symbol $\bullet$ weiß A, daß das Wort w_1 gelesen wurde.

Falls der Automat A_1 sich nun in einem nichtakzeptierenden Endzustand befinden würde, so ist die Rechnung von A zu Ende und A akzeptiert nicht.

Falls sich A_1 in einem akzeptierenden Endzustand befinden würde, generiert A ein Signal, das alle Zellen in den Ruhezustand zurückversetzt, und beginnt danach mit der Simulation von A_2. (In diesem Fall gilt bereits $w_1 \in L_1$.) Nach dem ersten $\sqcup$ Zeichen kann A entscheiden, ob auch $w_2 \in L_2$ gilt.

Es gilt also $L(A) = L_1 \bullet L_2$ und A arbeitet in Realzeit. Dabei benötigt A höchstens $f(n_1) + f(n_2)$ nichtdeterministische Schritte. ($f(n_1)$ für die Simulation von A_1 und $f(n_2)$ für die Simulation von A_2.) Da f monoton ist, gilt $f(n_1) + f(n_2) \leq 2f(n)$. Nach Lemma 3.12 können wir die Anzahl der nichtdeterministischen Schritte auf $f(n)$ reduzieren. □

Aus Satz 4.15 erhalten wir sofort:

Korollar 4.16
Unter den Voraussetzungen von Satz 4.15 gilt für Funktionen $f(n) = \Omega(\log(n))$:
Die Familie $\mathscr{L}_{rt}(f(n) - \text{NIA})$ ist abgeschlossen unter Konkatenation.

Beweis
Da $f(n) = \Omega(\log(n))$ ist, kann der Automat A, der L_1L_2 erkennt, bei Abarbeitung von w_1w_2 mit $w_i \in L_i$ die Zahl $|w_1|$ in binärer Form raten. In jedem Zeitschritt wird der entsprechende Zähler um 1 vermindert. A kann feststellen, wann der Zähler gleich 0 ist, d.h. A kann den Übergang von w_1 nach w_2 bestimmen. In allen anderen Punkten arbeitet A genau wie der entsprechende Automat für die markierte Konkatenation.

Wir müssen noch sicherstellen, daß A beim Raten von $|w_1|$ nicht mehr als $\log(n)$ nichtdeterministische Schritte ausführt, d.h., daß die von A geratene Binärzahl nicht zu groß ist. Dazu bedienen wir uns desselben Verfahrens wie im Beweis von Satz 2.19. In jedem Schritt vermindern wir den Binärzähler um 1. Die i–te Stelle des Binärzählers wird erst nach dem Lesen des 2^{i-1}-ten Zeichens geraten. □

Im allgemeinen ist jedoch $\mathscr{L}_{rt}(f(n) - \text{NIA})$ nicht abgeschlossen bezüglich Konkatenation, denn es gilt:

Satz 4.17
Ist $f(n) = o(\log(\log(n)))$, so ist $\mathscr{L}_{rt}(f(n) - \text{NIA})$ nicht abgeschlossen unter Konkatenation.

Beweis
Sei $\Sigma = \{0, 1, a, b\}$. Wir setzen

$$L_1 = \Sigma^* \quad \text{und} \quad L_2 = \{v \mid v \text{ ist ein Palindrom über } \Sigma\}.$$

Sowohl L_1 als auch L_2 können von deterministischen iterativen Arrays in Realzeit erkannt werden.

Wir zeigen, daß $L = L_1L_2$ nicht in $\mathscr{L}_{rt}(f(n) - \text{NIA})$ liegt. Es sei

$$W_n = \{0w1 \mid w \in \{a, b\}^n\}.$$

W_n hat 2^n Elemente. Für eine n–elementige Teilmenge $U = \{w_1, \ldots, w_n\}$ von W_n definieren wir ein Wort u durch

$$u = \begin{cases} \varepsilon & \text{falls } U = \emptyset \\ u_n & \text{sonst} \end{cases}.$$

Dabei werden die u_i rekursiv definiert durch

$$u_0 = \varepsilon\ , \qquad u_{i+1} = w_{i+1}^R u_i^R u_i.$$

Das Wort u hat daher die Länge $(n + 2)(2^n - 1)$.

Die Wörter u wurden so konstruiert, daß für $w \in W_n$ gilt:

$$w \in U \iff uw \in L$$

(siehe [12]).

Wir betrachten die Anzahl $N((n+2)2^n, n+2, L)$ aller $(n+2)$–Äquivalenzklassen unter den Wörtern der Länge $(n+2)(2^n-1)$. Wie oben gezeigt, gibt es für jede n–elementige Teilmenge U von W_n eine solche Äquivalenzklasse. Aus Lemma 3.2 folgt daher

$$N((n+2)2^n, n+2, L) \geq \binom{2^n}{n} = 2^{\Theta(n^2)}.$$

Andererseits gibt es nach Lemma 3.14 $r, s \in \mathbb{N}$ mit

$$\begin{aligned} N((n+2)2^n, n+2, L) &\leq (s^n)^{r^{f((n+2)2^n)}} \\ &= (s^n)^{r^{o(\log\log(((n+2)2^n)))}} \\ &= (s^n)^{o(\log(((n+2)2^n)))} \\ &= 2^{o(n^2)} \end{aligned} .$$

Dies ist ein Widerspruch, d.h. $L \notin \mathscr{L}_{rt}(f(n) - \text{NIA})$. □

Nun wenden wir uns dem Fall der markierten Iteration zu. Dabei haben wir das interessante Phänomen, daß die beiden Grenzfälle $\mathscr{L}_{rt}(\text{IA})$ und $\mathscr{L}_{rt}(\text{NIA})$ abgeschlossen sind bezüglich markierter Iteration. Dies gilt jedoch nicht für die Zwischenstufen. Ähnliches hat man auch bei der Chomsky-Hierarchie. Die regulären und kontextsensitiven Sprachen sind abgeschlossen bezüglich Komplement und Schnittbildung, die kontextfreien Sprachen jedoch nicht.

Satz 4.18
Für eine monoton wachsende, unbeschränkte Funktion $h : \mathbb{N}_0 \to \mathbb{N}_0$ sei $\{(a^n b)^{h(n)}\}$ eine Sprache aus $\mathscr{L}_{rt}(\text{IA})$. Weiter sei $h(n) = n^{o(1)}$. Ist $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine Funktion mit $f((h(n)+1)(n+1)) = \Theta(\log(h(n)))$, dann ist $\mathscr{L}_{rt}(f(n) - \text{NIA})$ nicht abgeschlossen bezüglich markierter Iteration.

Beweis
Wir betrachten die eingeschränkte Lexikonsprache Sprache

$$L_{h(n)} = \{w_1\$\ldots\$w_{h(n)}\#y\# \mid w_i, y \in \{0,1\}^n, y = w_i \text{ für ein } i \in \{1,\ldots,h(n)\}\}.$$

In Satz 2.19 haben wir gezeigt, daß $L_{h(n)} \in \mathscr{L}_{rt}(f(n) - \text{NIA})$ gilt. Dazu benötigten wir nur $\{(a^n b)^{h(n)}\} \in \mathscr{L}_{rt}(\text{IA})$ und $f((h(n)+1)(n+1)) = \Omega(\log(h(n)))$.

Da h monoton wachsend ist, können wir o.B.d.A. annehmen, daß auch f monoton wachsend ist. (Nach Lemma 3.12 kann f durch eine monoton wachsende Funktion $f' : \mathbb{N}_0 \to \mathbb{N}_0$ mit $f'((h(n)+1)(n+1)) = \Theta(\log(h(n)))$ ersetzt werden.)

Außerdem gibt es unendlich viele $n \in \mathbb{N}$ mit $h(n^2) \leq h(n)^2$. (Angenommen dies wäre nicht der Fall, dann gäbe es ein $n \in \mathbb{N}$ mit $h(n^{2^k}) > h(n)^{2^k}$ für alle $k \in \mathbb{N}$. Ein Widerspruch zu $h(n) = n^{o(1)}$.)

Wir betrachten nun die markierte Iteration L' von $L_{h(n)}$ und zeigen, daß L' nicht von einem $f(n) - \text{NIA}$ erkannt werden kann. Angenommen es gäbe ein $f(n) - \text{NIA}$ A, das L' in Realzeit erkennt. Die maximale Anzahl von Möglichkeiten in einem nichtdeterministischen Schritt von A sei r. Wir betrachten das Wort $x_1 \bullet \cdots \bullet x_k \bullet$ mit $x_i \in L$. Mit n_i bezeichnen wir den Parameter, der in der Definition von x_i vorkommt (d.h. n_i ist die Länge der Teilworte in x_i). Für ein festes n_k wählen wir $n_i = |x_{i+1} \bullet \cdots \bullet x_k \bullet|$ für $i = 1, \ldots, k-1$. Außerdem wählen wir n_k, so daß $h(n_k^2) \leq h(n_k)^2$ gilt.

Die Länge l_i des Worts $x_i \bullet \cdots \bullet x_k \bullet$ ist dann durch

$$\begin{aligned} l_k &= (h(n_k)+1)(n_k+1)+1 \\ l_i &= (h(l_{i+1})+1)(l_{i+1}+1)+1+l_{i+1} \\ &= (h(l_{i+1})+2)(l_{i+1}+1) \end{aligned}$$

bestimmt. Wegen $h(n) = n^{o(1)}$ ist für genügend großes n_k, $l_1 < n_k^2$. (Denn ist $l_{i+1} < n_k^2$, so folgt $l_i = (h(l_{i+1})+2)(l_{i+1}+1) = O(h(n_k)^2 l_{i+1})$. Also folgt $l_i = O(h(n_k)^{2(k-i+1)} n_k) \leq n_k^2$, da $h(n) = n^{o(1)}$ gilt.)

Dem iterativen Array A stehen daher nur

$$\begin{aligned} f(l_1) &\leq f(n_k^2) \\ &\leq f(h(n_k^2+1)(n_k+1)) \\ &= \Theta(\log(h(n_k^2))) \\ &= \Theta(\log(h(n_k))) \end{aligned}$$

nichtdeterministische Schritte zur Verfügung, um $x_1 \bullet \cdots \bullet x_k \bullet$ zu erkennen. (Die letzte Gleichung gilt, da wir n_k so gewählt hatten, daß $h(n_k^2) \leq h(n_k)^2$ gilt.)

Insbesondere ist die Anzahl der nichtdeterministischen Schritte von A für große n_k durch $c\log(h(n_k))$ beschränkt. Dabei ist c eine von k *unabhängige* Konstante. (D.h. es gibt ein $n_0 \in \mathbb{N}$, so daß für $n_k \geq n_0$ A weniger als $c\log(h(n_k))$ nichtdeterministische Schritte zur Verfügung hat. Dabei hängt c nicht von k ab. Die Schranke n_0 darf jedoch von k abhängen.)

Wir betrachten nun die Äquivalenzklassen von L', die entstehen, wenn wir im Teilwort x_i hinter dem ersten # Zeichen abschneiden. Jede mögliche Teilmenge von $h(n_i)$ Wörtern aus $\{0,1\}^{n_i}$ definiert ein Äquivalenzklasse. (Denn

ist $V = \{w_1, \ldots, w_{h(n_i)}\}$ eine solche Teilmenge und $v = w_1\$\ldots\$w_{h(n_i)}\#$, dann ist $y \in V$ äquivalent zu $vy\# \in L_{h(n)}$.)

Die Anzahl der Äquivalenzklassen ist daher mindestens $\binom{2^{n_i}}{h(n_i)} = 2^{\Theta(n_i h(n_i))}$ (siehe Lemma 3.2). Nach Lemma 3.14 muß sich daher A in mindestens $r^{c' \log(h(n_i))}$ Berechnungen verzweigen, um x_i erkennen zu können. Dabei hängen die Konstanten r und c' nur von dem Automaten A aber nicht von i oder k ab.

Da die Teilwörter $x_1, \ldots, x_k$ voneinander unabhängig sind, muß sich A, um den $x_1 \bullet x_2 \bullet$ Teil zu erkennen, in $r^{c' \log(h(n_1))} \cdot r^{c' \log(h(n_2))}$ Pfade verzweigen.

Begründung:

Sei m die Anzahl aller Berechnungspfade. Wir teilen die Menge aller Berechnungspfade in Klassen $K_1, \ldots, K_a$ ein. Für jedes Wort $x_1 \in L$ gibt es eine Klasse aller Berechnungspfade, die Wörter der Form $x_1 \bullet \ldots$ erkennt. Unter diesen Klassen gibt es mindestens $r^{c' \log(h(n_1))}$ Klassen, die paarweise keinen Berechnungspfad gemeinsam haben. (Das ist genau die Aussage des Äquivalenzklassenlemmas.) Sei x_1 ein Wort, das zu einer dieser $r^{c' \log(h(n_1))}$ Klassen gehört. Um Wörter der Form $x_1 \bullet x_2$ zu erkennen, muß sich der Automat in mindestens $r^{c' \log(h(n_2))}$ verschiedene Berechnungspfade verzweigen. Diese $r^{c' \log(h(n_2))}$ Pfade müssen alle in der zu x_1 gehörenden Klasse liegen. (Denn nur Pfade aus der zu x_1 gehörenden Klasse erkennen den x_1 Anteil.) Also umfaßt jede der Klassen $K_1, \ldots, K_a$ mindestens $r^{c' \log(h(n_2))}$ Berechnungspfade. Da es unter den Klassen $K_1, \ldots, K_a$ mindestens $r^{c' \log(h(n_1))}$ paarweise disjunkte Klassen gibt, folgt $m \geq r^{c' \log(h(n_1))} \cdot r^{c' \log(h(n_2))}$.

Setzt man diese Argumentation fort (Induktion), dann folgt, daß sich A in mindestens $(r^{c' \log(h(n_1))}) \ldots (r^{c' \log(h(n_k))}) \geq (r^{c' \log(h(n^2))})^k \geq (r^{c' \log(h(n_k))})^k$ verschiedene Berechnungen verzweigen muß, um $x_1 \bullet \cdots \bullet x_k \bullet$ zu erkennen.

Wählt man nun k so, daß $kc' > c$ gilt (c' und c sind von k unabhängig), so erhält man einen Widerspruch. (Da h' unbeschränkt ist gilt $c' > 0$.) A muß mindestens $kc' \log(h(n_k))$–mal raten, hat aber nur $c \log(h(n_k))$ nichtdeterministische Schritte zur Verfügung.

Also ist $L' \notin \mathscr{L}_{rt}(f(n) - \text{NIA})$. □

Beispiel 4.19

Für die Funktion $f(n) = \log\log(n)$ und $h(n) = \log(n)$ sind alle Voraussetzungen des Satzes 4.18 erfüllt. Die folgende Argumentation zeigt anschaulich, warum $\mathscr{L}_{rt}(f(n) - \text{NIA})$ bezüglich markierter Iteration nicht abgeschlossen ist:

Das Wort $x_1 \bullet \cdots \bullet x_k \bullet$ aus dem Beweis von Satz 4.18 hat eine Länge in

der Größenordnung von $\log(n)^k n$. *Für große* n *ist also*

$$|x_1 \bullet \cdots \bullet x_k \bullet | \leq n^2$$

und daher ist $f(|x_1 \bullet \cdots \bullet x_k \bullet |) = O(f(n))$. *Ein Automat, der die markierte Iteration von* L *erkennen soll, hat daher weniger als* $2 \log\log(n)$ *nichtdeterministische Schritte zur Verfügung.*

Um ein Wort x_i *zu erkennen, muß ein Automat den Index* j *raten, für den* $w_{i,j} = y_i$ *gilt.* ($x_i = w_{i,1}\$ \ldots \$w_{i,h(|y|)}\#y_i\#$) *Dazu braucht er mindestens* $c \log(h(|y_i|)) \geq c \log\log(n)$ *nichtdeterministische Schritte, wobei* c *nur von der gewählten Basis abhängt. Für jedes der* k *Teilwörter* x_i *muß der Automat einen Index raten, d.h. er braucht* $kc \log\log(n)$ *nichtdeterministische Schritte. Man kann nun* k *so wählen, daß* $kc > 2$ *gilt. Dies liefert einen Widerspruch, d.h. es gibt keinen Automaten, der die markierte Iteration erkennt.*

Korollar 4.20
Unter den Voraussetzungen von Satz 4.18 ist $\mathscr{L}_{rt}(f(n) - \mathrm{NIA})$ *nicht abgeschlossen bezüglich Iteration.*

Der Unterschied zwischen der markierten Konkatenation und der markierten Iteration ist, daß ein iteratives Array zum Erkennen der markierten Konkatenation $w_1 \bullet w_2$ nur doppelt so viele nichtdeterministische Schritte ausführen muß wie zum Erkennen der Teilwörter w_1 und w_2. Dieser Mehraufwand kann nach dem Reduktionslemma 3.12 ausgeglichen werden. Um die markierte Iteration $w_1 \bullet \ldots w_k \bullet$ zu erkennen, benötigt ein iteratives Array jedoch k–mal so viele nichtdeterministische Schritte. (Dabei liegen $|w_i|$ und $|w_1 \bullet \ldots w_n \bullet |$ in der gleichen Größenordung, d.h. das iterative Array erhält keine zusätzlichen nichtdeterministischen Schritte.) Da $k \in \mathbb{N}$ beliebig ist, kann dieser zusätzliche Aufwand nicht reduziert werden.

4.2.2 Boolesche Operationen

Alle Sprachfamilien, die von $f(n) - \mathrm{NIAs}$ erkannt werden, sind abgeschlossen unter Durchschnitt und Vereinigung. Wir erhalten dieses Resultat über die „Zwei-Register-Technik“, die man in dieser Form auch bei endlichen Automaten oder deterministischen iterativen Arrays anwenden kann.

Satz 4.21
Für jede Zeitbeschränkung $t : \mathbb{N}_0 \to \mathbb{N}$ *und jede Beschränkung der nichtdeterministischen Schritte* $f : \mathbb{N}_0 \to \mathbb{N}_0$ *ist die Sprachfamilie* $\mathscr{L}_{t(n)}(f(n) - \mathrm{NIA})$ *abgeschlossen bezüglich Durchschnitt und Vereinigung.*

Beweis
Für zwei Sprachen L_1 und L_2 aus $\mathscr{L}_{t(n)}(f(n) - \text{NIA})$ ist

$$L_1 \cap L_2 \in \mathscr{L}_{t(n)}(f(n) - \text{NIA}) \quad \text{und} \quad L_1 \cup L_2 \in \mathscr{L}_{t(n)}(f(n) - \text{NIA})$$

zu zeigen. Den $f(n) - \text{NIA}$, der die Sprache L_i in $t(n)$ Zeitschritten erkennt, bezeichnen wir mit A_i $(i = 1, 2)$.

Formal ist:

$$A_i = (S^{(i)}, \Sigma, F_a^{(i)}, F_r^{(i)}, \delta^{(i)}, \delta_0^{(i)}, q_0^{(i)}, \sqcup).$$

Den Automaten A, der $L_1 \cap L_2$ erkennt, definieren wir durch

$$A = (S, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup)$$

mit $S = S^{(1)} \times S^{(2)}$, $F_a = F_a^{(1)} \times F_a^{(2)}$, $F_r = (F_r^{(1)} \times S^{(2)}) \cup (S^{(1)} \times F_r^{(2)})$ und $q_0 = (q_0^{(1)}, q_0^{(2)})$.

Die lokalen Überführungsfunktionen werden durch

$$\delta((s_1^{(1)}, s_1^{(2)}), (s_2^{(1)}, s_2^{(2)}), (s_3^{(1)}, s_3^{(2)})) = (\delta^{(1)}(s_1^{(1)}, s_2^{(1)}, s_3^{(1)}), \delta^{(2)}(s_1^{(2)}, s_2^{(2)}, s_3^{(2)}))$$

für alle $s_i^{(j)} \in S^{(j)}$ mit $i = 1, 2, 3$ und $j = 1, 2$ definiert. Analog wird auch δ_0 komponentenweise definiert.

Da die lokalen Überführungsfunktionen komponentenweise definiert sind, folgt, daß die Zelle j zur Zeit t im Automaten A den Zustand (s_1, s_2) annimmt, falls zur Zeit t die Zelle j des Automaten A_i den Zustand s_i angenommen hat $(i = 1, 2)$.

Da $(s_1, s_2) \in F_a$ äquivalent zu $s_1 \in F_a^{(1)}$ und $s_2 \in F_a^{(2)}$ ist, akzeptiert der Automat A nur, wenn A_1 und A_2 akzeptieren, d.h. $L(A) = L_1 \cap L_2$. Es ist klar, daß A spätestens anhält, wenn sowohl A_1 und A_2 angehalten haben. Also ist A durch $t(n)$–zeitbeschränkt.

A führt immer dann einen nichtdeterministischen Schritt aus, wenn A_1 oder A_2 einen nichtdeterministischen Schritt macht, die Anzahl der nichtdeterministischen Schritte von A ist daher durch $f(n) + f(n)$ beschränkt. Nach Lemma 3.12 liegt daher $L_1 \cap L_2$ in $\mathscr{L}_{t(n)}(f(n) - \text{NIA})$.

Um $L_1 \cup L_2$ zu erkennen, müssen in dem Automaten A die Endzustandsmengen durch $F_a = (F_a^{(1)} \times S^{(2)}) \cup (S^{(1)} \times F_a^{(2)})$ und $F_r = F_r^{(1)} \times F_r^{(2)}$ definiert werden. □

Die Untersuchung, ob $\mathscr{L}_{rt}(f(n) - \text{NIA})$ bezüglich Komplementbildung abgeschlossen ist, ist deutlich schwieriger.

Satz 4.22
Eine Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ erfülle die Voraussetzungen von Satz 4.18. Dann ist $\mathscr{L}_{rt}(f(n) - \text{NIA})$ nicht abgeschlossen bezüglich Komplementbildung.

Beweis
Sei L eine Sprache aus $\mathscr{L}_{rt}(f(n)-\text{NIA})$, für die die markierte Konkatenation L' von L nicht in $\mathscr{L}_{rt}(f(n)-\text{NIA})$ liegt; z.B. die Sprache aus dem Beweis von Satz 4.18. Wir unterscheiden zwei Fälle. (Erinnerung: Das Komplement einer Sprache L wird mit $\bar{L}$ bezeichnet.)

Fall 1: $\bar{L} \notin \mathscr{L}_{rt}(f(n)-\text{NIA})$.
In diesem Fall ist L eine Sprache, die zeigt, daß $\mathscr{L}_{rt}(f(n)-\text{NIA})$ nicht bezüglich Komplementbildung abgeschlossen ist.

Fall 2: $\bar{L} \in \mathscr{L}_{rt}(f(n)-\text{NIA})$.
Wir zeigen, daß $\bar{L}'$ in $\mathscr{L}_{rt}(f(n)-\text{NIA})$ liegt. Ein $f(n)-\text{NIA}$ A, das $\bar{L}'$ erkennt, überprüft, ob eine Eingabe der Form $x_1 \bullet \cdots \bullet x_k \bullet$ vorliegt. Alle Eingaben, die nicht dieser Form entsprechen, werden akzeptiert. (Dies kann sogar ein deterministisches iteratives Array.)

Ist die Eingabe von der Form $x_1 \bullet \cdots \bullet x_k \bullet$, so simuliert A für jedes Wort x_i den Automaten, der $\bar{L}$ erkennt. Dabei benutzt A für jedes Wort x_i dieselben nichtdeterministischen Wahlmöglichkeiten, d.h. A speichert seine Wahl in einem Keller ab und benutzt bei einem neuen x_i die im Keller gespeicherten Angaben, um einen nichtdeterministischen Schritt deterministisch zu simulieren. (Verzweigt sich also der Automat, der $\bar{L}$ erkennt, in einem nichtdeterministischen Schritt in r Berechnungspfade, so speichert A auf dem Keller Zahlen zwischen 1 und r, die angeben, welche Verzweigung A wählen muß. Da A nur für ein $i \in \{1,\ldots,k\}$ die Beziehung $x_i \in \bar{L}$ nachweisen muß, kann A für alle x_i immer dieselben Verzweigungen wählen.)

Auf diese Weise benötigt A höchstens $\max\{f(|x_i|) \mid i = 1,\ldots,k\}$ nichtdeterministische Schritte. Da f monoton ist, folgt

$$\max\{f(|x_i|)\} \leq f(|x_1 \bullet \ldots x_k \bullet|).$$

Also gilt $\bar{L}' \in \mathscr{L}_{rt}(f(n)-\text{NIA})$ aber $L' \notin \mathscr{L}_{rt}(f(n)-\text{NIA})$.

Damit ist gezeigt, daß $\mathscr{L}_{rt}(f(n)-\text{NIA})$ nicht bezüglich Komplementbildung abgeschlossen ist. □

Bemerkung 4.23
Der Beweis von Satz 4.22 ist nicht konstruktiv, da keine Sprache aus $\mathscr{L}_{rt}(f(n)-\text{NIA})$ angegeben wird, deren Komplement nicht in $\mathscr{L}_{rt}(f(n)-\text{NIA})$ liegt.

4.2.3 Homomorphismen

Wir wissen bereits, daß $\mathscr{L}_{rt}(\text{NIA})$ unter ε–freien Homomorphismen abgeschlossen ist. Der folgende Satz zeigt, daß „alle“ anderen Sprachfamilien $\mathscr{L}_{rt}(f(n) - \text{NIA})$ nicht unter ε–freien Homomorphismen abgeschlossen sind.

Satz 4.24
Für $f : \mathbb{N}_0 \to \mathbb{N}_0$ ist die Sprachfamilie $\mathscr{L}_{rt}(f(n) - \text{NIA})$ genau dann unter ε–freien Homomorphismen abgeschlossen, wenn $\mathscr{L}_{rt}(f(n) - \text{NIA}) = \mathscr{L}_{rt}(\text{NIA})$ gilt.

Beweis
Da $\mathscr{L}_{rt}(\text{NIA})$ nach Korollar 4.10 unter ε–freien Homomorphismen abgeschlossen ist, folgt aus $\mathscr{L}_{rt}(f(n) - \text{NIA}) = \mathscr{L}_{rt}(\text{NIA})$, daß $\mathscr{L}_{rt}(f(n) - \text{NIA})$ unter ε–freien Homomorphismen abgeschlossen ist.

Zum Beweis der umgekehrten Richtung nehmen wir nun an, daß die Familie $\mathscr{L}_{rt}(f(n) - \text{NIA})$ unter ε–freien Homomorphismen abgeschlossen ist. Da $\mathscr{L}_{rt}(\text{IA}) \subseteq \mathscr{L}_{rt}(f(n) - \text{NIA})$ gilt und nach Satz 4.2 die Familie $\mathscr{L}_{rt}(\text{NIA})$ der Abschluß von $\mathscr{L}_{rt}(IA)$ unter ε–freien Homomorphismen ist, folgt:

$$\mathscr{L}_{rt}(\text{NIA}) \subseteq \mathscr{L}_{rt}(f(n) - \text{NIA})$$

Andererseits gilt trivialerweise

$$\mathscr{L}_{rt}(f(n) - \text{NIA}) \subseteq \mathscr{L}_{rt}(\text{NIA}).$$

□

Bemerkung 4.25
Im nächsten Kapitel zeigen wir, daß für $f(n) = o(\log(n))$

$$\mathscr{L}_{rt}(f(n) - \text{NIA}) \subsetneq \mathscr{L}_{rt}(\text{NIA})$$

gilt (Korollar 5.4).

Also ist für $f(n) = o(\log(n))$ die Familie $\mathscr{L}_{rt}(f(n) - \text{NIA})$ unter ε–freien Homomorphismen nicht abgeschlossen.

Die Sprachfamilien $\mathscr{L}_{rt}(f(n) - \text{NIA})$ sind also im allgemeinen nicht abgeschlossen bezüglich Homomorphismen und Substitution. Als nächstes betrachten wir inverse Homomorphismen.

Satz 4.26
Es sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine monotone Funktion. f erfülle weiterhin die Bedingung: Für jedes $k \in \mathbb{N}$ gibt es eine Konstante c, so daß $f(kn) \leq cf(n)$ für fast alle $n \in \mathbb{N}$ gilt.

Dann ist $\mathscr{L}_{rt}(f(n) - \text{NIA})$ abgeschlossen unter inversen Homomorphismen.

Beweis
Es sei $L \in \mathscr{L}_{rt}(f(n) - \text{NIA})$ und A ein $f(n) - \text{NIA}$, das L erkennt. L sei über dem Alphabet Σ definiert. Für einen Homomorphismus $h : \Sigma'^* \to \Sigma^*$ konstruieren wir ein $f(n) - \text{NIA}$ A', das $h^{-1}(L)$ erkennt. Es gelte $|h(a)| \leq k$ für alle $a \in \Sigma'$.

Jede Zelle von A' simuliert k Zellen von A. Dabei rechnet jede Zelle von A' immer soweit, wie es die verfügbare Information zuläßt. (D.h. hat zum Beispiel die Nachbarzelle nur einen Schritt von A simuliert, so kann auch nur ein Schritt von A simuliert werden. Liest A' gerade die Eingabe a, so kann A' mit seiner Ursprungszelle genau $h(a)$ Schritte von A simulieren.) Das Verfahren ist genau dasselbe wie bei deterministischen iterativen Arrays.

Die Abbildung 4.1 zeigt die Berechnung von A' im Fall $k = 2$.

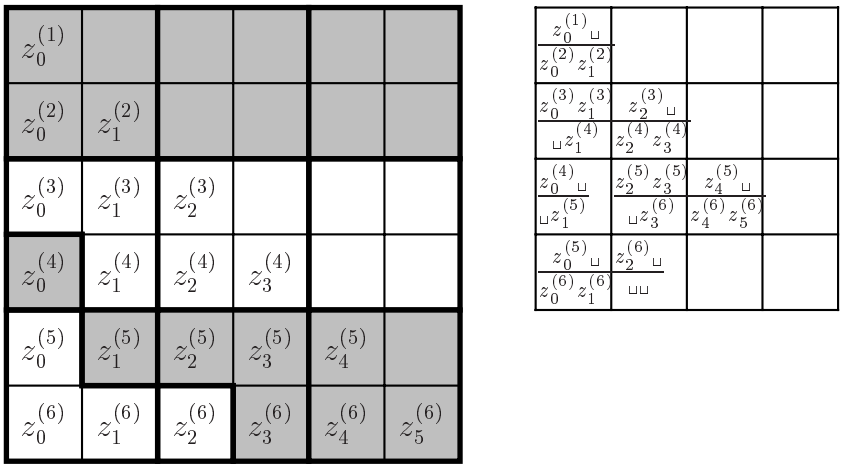


Abbildung 4.1: Dargestellt ist links die Berechnung von A. Die oberste Zeile zeigt die Zustände der einzelnen Zellen von A zum Zeitpunkt 1, die unterste die Zustände zum Zeitpunkt 6. Die Eingabe von A war *acbbac*. Es gilt $h(\hat{a}) = ac$ und $h(\hat{b}) = b$, d.h $k = 2$ und die Eingabe von A' ist $\hat{a}\hat{b}\hat{b}\hat{a}$. Die Schritte von A, die A' in einem Schritt simuliert, sind fett umrandet. Z.B. simuliert die Ursprungszelle (links) von A' im 2–ten Schritt den 3–ten Schritt der Zelle 0 und den 3–ten und 4–ten Schritt der Zelle 1 von A. (Die Zeitschritte 1 und 3 von A' sind grau unterlegt.) Das rechte Bild zeigt die Berechnung von A'.

Man sieht, daß A' die Berechnung von A genau simuliert, d.h A' akzeptiert ein Wort w genau dann, wenn $h(w)$ von A akzeptiert wird.

A' benötigt höchstens so viele nichtdeterministische Schritte wie A. Wegen der Voraussetzung an f gibt es eine Konstante c, so daß $f(h(w)) \leq cf(|w|)$ für

alle Wörter w gilt. Nach Lemma 3.12 gilt daher $h^{-1}(L) \in \mathscr{L}_{rt}(f(n)-\text{NIA})$. □

4.2.4 Spiegelung

Als letztes betrachten wir noch den Abschluß bezüglich Spiegelung. Da ein iteratives Array in Realzeit viel Zeit hat, die ersten Zeichen seiner Eingabe zu bearbeiten, aber direkt nach den Lesen des letzten Zeichen eine Entscheidung fällen muß, ist es naheliegend, daß $\mathscr{L}_{rt}(f(n)-\text{NIA})$ nicht bezüglich Spiegelung abgeschlossen ist.

Satz 4.27
Für $f(n) = o(\log(n))$ ist $\mathscr{L}_{rt}(f(n) - \text{NIA})$ nicht abgeschlossen bezüglich Spiegelung.

Beweis
Die Lexikonsprache

$$L_n = \{w_1\$ \dots \$w_n\#y\# \mid w_i, y \in \{0,1\}^n, w_i = y \text{ für ein } i \in \{1,\dots,n\}\}$$

liegt nicht in $\mathscr{L}_{rt}(f(n) - \text{NIA})$.

Wir betrachten die Anzahl $N((n+1)^2, n+1, L_n)$ der Äquivalenzklassen, die entstehen, wenn wir nach dem ersten # Zeichen abschneiden. Wie im Beweis von Satz 4.18 sehen wir, daß diese Anzahl größer als $\binom{2^n}{n} = 2^{\Theta(n^2)}$ ist. Allerdings kann nach Lemma 3.14 ein $f(n) - \text{NIA}$ nur

$$\begin{aligned} s^{(n+1)r^{f((n+1)^2)}} &= s^{(n+1)r^{o(\log((n+1)^2))}} \\ &= 2^{o(n^2)} \end{aligned}$$

solcher Äquivalenzklassen unterscheiden. Es folgt $L_n \notin \mathscr{L}_{rt}(f(n) - \text{NIA})$.

Die Spiegelung L' von L_n jedoch kann leicht von einem deterministischen iterativen Array erkannt werden. Das iterative Array muß lediglich y in einer FiFo-Schlange abspeichern. Nach jedem \$ Zeichen beginnt das iterative Array, y aus der Schlange zu lesen und mit den entsprechenden w_i zu vergleichen. Gleichzeitig wird y erneut in der FiFo–Schlange gespeichert. Ein Wort wird genau dann erkannt, wenn $y = w_i$ für ein $i \in \{1,\dots,n\}$ gilt. □

Die folgende Tabelle gibt eine Übersicht über die Abschlußeigenschaften.

	$\mathscr{L}_{rt}$(IA)	$\mathscr{L}_{rt}(f(n)-$NIA)	$\mathscr{L}_{rt}$(NIA)
$\bullet$	–	–	$\surd$
$+$	–	–	$\surd$
markierte $\bullet$	$\surd$	$\surd$	$\surd$
markierte $+$	$\surd$	–	$\surd$
$\cup$	$\surd$	$\surd$	$\surd$
$\cap$	$\surd$	$\surd$	$\surd$
Komplement	$\surd$	–	?
hom^{-1}	$\surd$	$\surd$	$\surd$
ε-freie hom	–	–	$\surd$
ε-freie sub	–	–	$\surd$
Spiegelung	–	–	$\surd$

Abbildung 4.2: Ein $\surd$ markiert, daß die Sprachfamilie abgeschlossen bezüglich der angegeben Operation ist. Bis auf den Abschluß von $\mathscr{L}_{rt}$(NIA) bezüglich Komplementbildung haben wir in allen anderen Fällen gezeigt, daß die entsprechende Sprachfamilie nicht abgeschlossen unter der betreffenden Operation ist.
Die Ergebnisse aus der mittleren Spalte gelten für unbeschränkte Funktionen $f : \mathbb{N}_0 \to \mathbb{N}_0$, die nicht zu schnell wachsen ($f(n) = o(\log(n))$ oder $f(n) = o(\log\log(n))$). Außerdem müssen die Funktionen f noch einige Berechenbarkeitsforderungen erfüllen.

Kapitel 5

Hierarchiesätze

In diesem Kapitel untersuchen wir das Verhältnis von Nichtdeterminismus zu anderen Ressourcen. Insbesondere beweisen wir mehrere Hierarchiesätze und verallgemeinern bereits bekannte Resultate über deterministische iterative Arrays.

Wir beweisen eine dichte Hierarchie über die Anzahl der nichtdeterministischen Schritte (Korollar 5.4). In [12] wird eine Hierarchie über die Dimension iterativer Arrays gezeigt. Wir werden dieses Resultat auf nichtdeterministische iterative Arrays verallgemeinern (Korollar 5.14). Ebenfalls in [12] wird $\mathscr{L}_{rt}(\mathrm{IA}) \subsetneqq \mathscr{L}_{lt}(\mathrm{IA})$ gezeigt. Wir werden dieses Ergebnis zu einer dichten Hierarchie verallgemeinern (5.9). Für nichtdeterministische iterative Arrays existiert jedoch eine Komplexitätslücke bei kleinen Zeitschranken (Satz 5.7).

5.1 Nichtdeterministische Hierarchien

Für $g(n) \leq f(n)$ gilt offensichtlich $\mathscr{L}_{t(n)}(g(n) - \mathrm{NIA}) \subseteq \mathscr{L}_{t(n)}(f(n) - \mathrm{NIA})$. Aus Satz 3.12 wissen wir, daß für $f(n) \in \Theta(g(n))$ die beiden Sprachfamilien zusammenfallen. In diesem Abschnitt untersuchen wir, unter welchen Bedingungen $\mathscr{L}_{t(n)}(g(n) - \mathrm{NIA}) \subsetneqq \mathscr{L}_{t(n)}(f(n) - \mathrm{NIA})$ gilt.

Satz 5.1
Seien $f, g : \mathbb{N}_0 \to \mathbb{N}_0$ Funktionen, für die $g(n) \leq f(n)$ gilt. Sei ferner $t : \mathbb{N}_0 \to \mathbb{N}$ eine beliebige Funktion.

Weiter gebe es eine Funktion $h : \mathbb{N} \to \mathbb{N}$ mit $h(n) \leq 2^{n-2}$. h erfülle die Voraussetzungen:

1. $\{(a^n b)^{h(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{IA})$.
2. $f((n+1)(h(n)+1)) = \Omega(\log(h(n)))$ *und*
   $g((n+1)(h(n)+1)) = o(\log(h(n)))$.

3. $t((n+1)(h(n)+1)) = O(n)$.

Dann gibt es eine Sprache L, die in $\mathscr{L}_{rt}(f(n) - \text{NIA})$ aber nicht in $\mathscr{L}_{n+t(n)}(g(n) - \text{NIA})$ liegt.

Beweis
Die Lexikonsprache $L_{h(n)}$ liegt in $\mathscr{L}_{rt}(f(n) - \text{NIA})$. (Folgt aus Satz 2.19 mit 1 und 2.)

Wir zeigen nun, daß $L_{h(n)}$ nicht in $\mathscr{L}_{n+t(n)}(g(n) - \text{NIA})$ liegt. Dazu benutzen wir das Lemma 3.16.

Wir schätzen die Anzahl $N((h(n)+1)(n+1), n+1, L_{h(n)})$ der $(n+1)$–Äquivalenzklassen bezüglich $L_{h(n)}$ unter den Wörtern der Länge $h(n)(n+1)$ ab. (Das heißt, wir betrachten den Teil bis zum ersten # Zeichen.) Für jede $h(n)$–elementige Teilmenge $V = \{w_1, \ldots, w_{h(n)}\}$ von $\{0,1\}^n$ sei v das Wort $w_1\$ \ldots \$w_{h(n)}\#$. Für $y \in \{0,1\}^n$ gilt:

$$y \in V \iff vy\# \in L_{h(n)}.$$

Also ist $N((h(n)+1)(n+1), n+1, L_{h(n)}) \geq \binom{2^n}{h(n)} = 2^{\Theta(nh(n))}$.

Angenommen, $L_{h(n)} \in \mathscr{L}_{n+t(n)}(g(n) - \text{NIA})$. Nach Lemma 3.14 gibt es in diesem Fall Konstanten s und r, für die

$$\begin{aligned} N((h(n)+1)(n+1), n+1, L_{h(n)}) &\leq (s^{n+1+t((h(n)+1)(n+1))})^{r^{g((h(n)+1)(n+1))}} \\ &= (s^{O(n)})^{r^{o(\log(h(n)))}} \quad \text{nach 2) und 3)} \\ &= 2^{o(nh(n))} \end{aligned}$$

gilt.

Dies ist ein Widerspruch, d.h. $L_{h(n)} \notin \mathscr{L}_{rt}(g(n) - NIA)$. □

Bemerkung 5.2

1. *Voraussetzung 1 wurde von Satz 2.19 übernommen. Sie ist notwendig, um einen $f(n) - \text{NIA}$ zu konstruieren, der $L_{h(n)}$ in Realzeit erkennt. In Anhang A zeigen wir, daß viele Funktionen diese Voraussetzung erfüllen.*

2. *Für jede Funktion $g : \mathbb{N}_0 \to \mathbb{N}_0$ mit $g(n) = o(\log(n))$ und jede monoton wachsende Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ mit $f(n) = \omega(g(n))$ gibt es eine Funktion h, so daß die Voraussetzung 2 erfüllt ist.*

   Denn:

O.B.d.A sei $f(n) \leq \log(n)$. *Wir definieren*

$$h(n) = \max\{m \in \mathbb{N} : f((n+1)(m+1)) \geq 3\log(m)\}.$$

Das Maximum existiert, da die angegebene Menge durch n *beschränkt ist. Aus der Definition von* h *folgt sofort*

$$f((n+1)(h(n)+1)) = \Theta(\log(h(n)))$$

und damit auch $g((n+1)(h(n)+1)) = o(\log(h(n)))$.

3. *Ist* $g : \mathbb{N}_0 \to \mathbb{N}_0$ *monoton wachsend, so folgt aus Voraussetzung* 2, *daß* $g(n) = o(\log(n))$ *gilt. Dies ist trivial für* $h(n) = O(n)$. *Für* $h(n) = \omega(n)$ *folgt* $g(O(h(n))) = o(\log(h(n)))$ *und somit* $g(n) = o(\log(n))$.

4. *Voraussetzung* 3 *drückt aus, daß die Anzahl der Schritte nach dem Lesen der Eingabe im Vergleich zur Eingabelänge nicht zu groß werden darf. Ist* $t(n) = O(\sqrt{n})$, *so ist Voraussetzung* 3 *für alle Funktionen* h *mit* $h(n) = O(n)$ *erfüllt. In diesem Fall erhalten wir eine* dichte *Hierarchie bis zu* $f(n) = \log(n)$.

   Dies gilt insbesondere für $t(n) = 1$ *für alle* $n \in \mathbb{N}_0$, *d.h. für Realzeit.*

5. *Die Voraussetzungen besagen daher im wesentlichen* $g(n) = o(f(n))$. *Da wir bereits bewiesen haben, daß für* $g(n) = \Theta(f(n))$ *die Familien* $\mathscr{L}_{rt}(g(n) - \mathrm{NIA})$ *und* $\mathscr{L}_{rt}(f(n) - \mathrm{NIA})$ *gleich sind, ist* $g(n) = o(f(n))$ *notwendig für die Echtheit der Inklusion.*

   Da unter schwachen Zusatzvoraussetzungen $g(n) = o(f(n))$ *auch hinreichend für die Echtheit der Inklusion ist, folgt, daß die Abschätzungen aus Lemma 3.14 scharf sind.*

Beispiel 5.3
Ein typisches Beispiel für Funktionen, die die Voraussetzungen 1 *bis* 3 *erfüllen, ist:*

$h(n) = \log^{[i]}(n)$ *und* $f(n) = \log^{[i+1]}(n)$ *für* $i \in \mathbb{N}_0$. *Eine beliebige Funktion* $g : \mathbb{N}_0 \to \mathbb{N}_0$ *mit* $g(n) = o(f(n))$ *und eine beliebige Funktion* $t : \mathbb{N}_0 \to \mathbb{N}$ *mit* $t(n) = O(\sqrt{n})$.

(Erinnerung: $\log^{[i]}$ *bezeichnet die* i*–fache Verkettung der Logarithmus Funktion und* $\log^{[0]}(n) = n$ *für alle* $n \in \mathbb{N}_0$.*)*

Korollar 5.4 (Nichtdeterministische Hierarchien)
Unter den Voraussetzung von Satz 5.1 gilt:

$$\mathscr{L}_{n+t(n)}(g(n) - \mathrm{NIA}) \subsetneq \mathscr{L}_{n+t(n)}(f(n) - \mathrm{NIA}).$$

5.2 Nichtdeterminismus und Zeit

Satz 5.1 zeigt insbesondere, daß unterhalb von $n + \sqrt{n}$ Zeitschritten weniger Nichtdeterminismus nicht durch mehr Zeit kompensiert werden kann. Umgekehrt ist es jedoch möglich, Zeit durch Nichtdeterminismus zu ersetzen, d.h. unter geeigneten Voraussetzungen gilt $\mathscr{L}_{n+t(n)}(\text{IA}) \subseteq \mathscr{L}_{rt}(t(n) - \text{NIA})$.

Definition 5.5
Eine Funktion $g : \mathbb{N}_0 \to \mathbb{N}_0$ heißt durch ein iteratives Array in Linearzeit raumberechenbar, wenn es ein iteratives Array A und ein Konstante c gibt, so daß A nach jeder Eingabe der Länge n nach höchstens cn Schritten anhält und dabei die Zellen $0, \ldots, g(n) - 1$ mit speziellem Zustand markiert hat.

Lemma 5.6
Alle zeitberechenbaren Funktionen $g : \mathbb{N}_0 \to \mathbb{N}_0$ mit $g(n) = O(n)$ sind in Linearzeit raumberechenbar.

Beweis
Da $g(n)$ zeitberechenbar ist, gibt es ein iteratives Array, das bei jeder Eingabe der Länge n nach $g(n)$ Zeitschritten in einen ausgezeichneten Zustand wechselt. In jedem dieser Zeitschritte kann das iterative Array eine Zelle markieren. (Es schreibt einfach ein Symbol in einen kellerartigen Speicher, der pro Zelle genau ein Zeichen speichert.) □

Wir können nun den folgenden Satz zeigen:

Satz 5.7 (Komplexitätslücke bei (sub)linearen Zeitschranken)
Sei $t : \mathbb{N}_0 \to \mathbb{N}$ eine monotone Funktion mit $t(n) = O(n)$. Es gebe eine in Linearzeit raumberechenbare, monotone Funktion $g : \mathbb{N}_0 \to \mathbb{N}_0$ mit $g(n/16) = \Omega(t(n))$ und $g(n) = O(t(n))$.

Dann gilt

$$\mathscr{L}_{n+t(n)}(t(n) - \text{NIA}) = \mathscr{L}_{rt}(t(n) - \text{NIA}).$$

Beweis
Offensichtlich gilt

$$\mathscr{L}_{rt}(t(n) - \text{NIA}) \subseteq \mathscr{L}_{n+t(n)}(t(n) - \text{NIA}).$$

Für die umgekehrte Richtung betrachten wir ein $t(n) - \text{NIA}$ A, das eine Sprache L in $n + t(n)$ Zeitschritten erkennt. Wir konstruieren ein $t(n) - \text{NIA}$ A', das L in Realzeit erkennt.

Die Idee ist, daß A' seine nichtdeterministischen Schritte nutzt, um die Zustände der Zellen $-2t(n), \ldots, 2t(n)$ zum Zeitpunkt $n - t(n)$ zu raten. Danach kann A' ermitteln, ob A bis zum Zeitpunkt $n+t(n)$ in einen erkennenden

Zustand übergeht. Außerdem muß A' überprüfen, ob es die Zustände der Zellen $-2t(n), \ldots, 2t(n)$ korrekt geraten hat.

Das Hauptproblem beim Umsetzten dieser Idee ist, daß wir verhindern müssen, daß A' zu viele nichtdeterministische Schritte ausführt. Dafür benötigen wir die Berechenbarkeit der Funktion g.

O.B.d.A. können wir $t(n) \leq g(n/16)$ und $g(n/8) \leq n/14$ annehmen, denn nach Voraussetzung gibt es Konstanten k_1, k_2, für die $g(n/8) \leq k_1 \frac{n}{8}$ und $t(n) \leq k_2 g(n/16)$ gilt. Gemäß den Lemmata 3.11 und 3.12 können wir die Funktionen g und t so abändern, daß $k_1 = \frac{8}{14}$ und $k_2 = 1$ gilt.

Die Ursprungszelle von A' simuliert die Ursprungszelle von A. Alle anderen Zellen von A' simulieren jeweils 2 Zellen von A. A' arbeitet nun nach folgendem Schema.

1. A' simuliert die ersten n Schritte von A direkt. Dafür benötigt A' n Zeitschritte. Diese Berechnung findet auf der ersten Spur von A' statt. In weiteren Spuren führt A' gleichzeitig noch die zusätzlichen Berechnungen aus, die in den nächsten Punkten beschrieben werden.

2. Zusätzlich startet A' einen Zeitkonstruierer für 2^n. Nach jedem Zeitpunkt 2^k ($k \in \mathbb{N}$) „rät“ A' deterministisch, daß nun bereits mehr als $\frac{1}{16}$ der Eingabe gelesen wurde. A' berechnet also den Fall, daß $2^k > \frac{1}{16}n$ und den Fall $2^k \leq \frac{1}{16}n$. Dafür benötigt A' 4 Register, in denen die zu 2^k, 2^{k+1}, 2^{k+2} und 2^{k+3} gehörenden Fälle behandelt werden. Nach dem Zeitpunkt 2^{k+4} weiß A', daß $2^k \leq \frac{1}{16}n$ gilt und kann die Berechnung für den Fall $2^k > \frac{1}{16}n$ löschen.

   (Das Behandeln von mehreren Fällen in verschiedenen Registern nennt man deterministisches Raten. Eine weitere Anwendung von deterministischem Raten ist zum Beispiel: „Das iterative Array rät deterministisch, daß nur noch 8 *b*s in der Eingabe vorkommen.“ Siehe auch Lemma A.1.)

3. Ab jetzt betrachten wir die Berechnung von A', nachdem es $2^k \geq \frac{1}{16}n$ „geraten“ hat. Beginnend mit dem Zeitpunkt 2^k berechnet A' die Zahl $x = g(2^k)$. Dafür benötigt A' höchstens $3 \cdot 2^k$ Zeitschritte, das heißt A' hat $g(2^k)$ berechnet, bevor mehr als die Hälfte der Zeit verbraucht wurde. (2^k Zeitschritte, um eine Eingabe der Länge 2^k zu lesen. Die Eingabe wird von A' selbst erzeugt. Weitere 2^k Schritte für die Berechnung von x. Die in Linearzeit stattfindende Berechnung kann ähnlich wie im Beweis von Lemma 3.11 entsprechend beschleunigt werden. Abschließend benötigt A' maximal $x \leq 2^k$ Schritte zum Markieren der x Zellen.)

4. Bisher hat A' $4 \cdot 2^k$ Schritte ausgeführt. Auf einer zusätzlichen Spur von A' rät die Ursprungszelle von A' die Zustände, die die Zellen $-2x, \ldots, 2x$ von A nach dem $(n-x)$–ten Zeitschritt haben. Für das Raten braucht A' x Zeitschritte.

   (Dabei benutzt A die Zustände der Zellen $-2g(2^{k-1}), \ldots, 2g(2^{k-1})$, die es zuvor für den Fall $2^{k-1} > \frac{1}{16}n$ geraten hat, erneut. Insgesamt muß A daher nur $g(2^k)$ nichtdeterministische Schritte ausführen.)

5. Danach führt A' eine FSSP–Synchronisation auf diesen Zellen aus, um die gleichzeitige Simulation von A zu veranlassen ($4x$ Zeitschritte).

6. A' simuliert die Berechnung von A in den Zeitschritten von $n-x$ bis $n+x$ ($2x$ Zeitschritte). Da A' den Zustand der Zellen $-2x, \ldots, 2x$ kennt, kann A' ermitteln, ob die Ursprungszelle von A bis zum Zeitpunkt $n+x$ in einen erkennenden Zustand wechselt.

   Für den Fall, daß A in den Zeitschritten $n-x$ bis $n+x$ nichtdeterministische Schritte ausführt, generiert A' beim Raten der Zellen $-2x, \ldots, 2x$ Anweisungen, welche Wahlmöglichkeit von A simuliert werden soll, d.h. A' speichert auf einem Keller Zahlen, die angeben welche Verzweigung A wählen soll. Damit kann A' diesen Teil der Berechnung von A deterministisch simulieren.

7. A' akzeptiert nur, wenn die Ursprungszelle von A bis zum Zeitpunkt $n+x$ in einen akzeptierenden Zustand wechselt.

8. A' muß noch überprüfen, ob es die Zustände der Zellen $-2x, \ldots, 2x$ von A zum Zeitpunkt $n-x$ korrekt geraten hat.

   Dazu speichert A' die (durch Raten entstandenen) Zustände der Zellen $\pm i$ zum Zeitpunkt $n - \lceil \frac{i}{2} \rceil$ ab $(i = 0, \ldots, x)$. D.h. A' merkt sich den (geratenen) Zustand der Ursprungszelle zum Zeitpunkt n, die Zustände der Zellen $-2, -1, 1, 2$ zum Zeitpunkt $n-1$, usw.

   Die Zellen $-x$ und x von A' generieren zu jedem Zeitpunkt ein Signal, das mit voller Geschwindigkeit zur Ursprungszelle läuft. (Nur an dieser Stelle nutzt A' aus, daß jeweils zwei Zellen von A durch eine Zelle von A' simuliert werden.) Diese Signale überprüfen auf ihrem Weg, ob der Zustand, der durch Simulation von A entstanden ist, mit dem Zustand, der durch das Raten entstanden ist, übereinstimmt. Die Signale werden unterbrochen, wenn dies in einer der Zellen $-2x, \ldots, 2x$ von A' nicht der Fall ist.

Die Ursprungszelle akzeptiert zum Zeitpunkt $n+1$ nur dann, wenn der durch Simulation entstandene Zustand mit dem gespeicherten Zustand übereinstimmt, d.h. wenn zu diesem Zeitpunkt die beiden Signale ankommen.

Dieses Vorgehen garantiert, daß die von A' geratenen Zustände von A zum Zeitpunkt $n-x$ korrekt waren.

A' kann mit diesem Schritt beginnen, sobald $\frac{n}{2}+x$ Zeitschritte vergangen sind (Schritt 2 bis 4). Da $\frac{n}{2}+x < n-x$ gilt, hat A' genügend Zeit für diesen Schritt.

Nach Voraussetzung an g folgt, daß $x \geq t(n)$ ist, d.h. A' die Berechnung von A über genügend viele Zeitschritte simuliert. Außerdem ist x klein genug, so daß die oben beschriebene Simulation in Realzeit ablaufen kann. (Sie dauert $n/2+x+4x+2x \leq n$ Zeitschritte, $n/2$ Zeitschritte bis $g(2^k)$ berechnet ist (Schritt 2 und 3) und $7x$ Zeitschritte zur Simulation von A' (Schritt 4 bis 7).)

A' braucht zur Simulation von A höchstens $t(n)+x$ nichtdeterministische Schritte. ($t(n)$ zur Simulation von A und x zum Raten der Zellen $-2x, \ldots, 2x$.) Nach Voraussetzung an g folgt:

$$t(n)+x \leq t(n)+g(n) = O(t(n)).$$

Satz 3.12 zeigt daher,

$$L = L(A) = L(A') \in \mathscr{L}_{rt}(t(n)-\mathrm{NIA}).$$

□

Damit ist gezeigt, daß man bei iterativen Arrays zusätzliche Zeitschritte durch nichtdeterministische Schritte ersetzen kann, denn unter den Voraussetzungen von Satz 5.7 gilt

$$\mathscr{L}_{n+t(n)}(\mathrm{IA}) \subsetneq \mathscr{L}_{rt}(t(n)-\mathrm{NIA}).$$

Die Echtheit der Inklusion ergibt sich aus dem Hierarchiesatz 5.4.

Ein wichtiger Spezialfall, von Satz 5.7 ist

$$\mathscr{L}_{rt}(\mathrm{NIA}) = \mathscr{L}_{lt}(\mathrm{NIA}).$$

Dadurch ergibt sich die Verschärfung des Beschleunigungslemmas 3.11.

Korollar 5.8
Für $t : \mathbb{N}_0 \to \mathbb{N}$ mit $t(n) \geq n+1$ für alle $n \in \mathbb{N}_0$ gilt

$$\mathscr{L}_{\Theta(t(n))}(\mathrm{NIA}) = \mathscr{L}_{t(n)}(\mathrm{NIA}).$$

5.3 Deterministische Zeithierarchien

In den vorangegangenen Abschnitten haben wir gesehen, daß Zeit eine schwächere Ressource ist als Nichtdeterminismus. Bei nichtdeterministischen iterativen Arrays gibt es eine Komplexitätslücke bei kleinen Zeitbeschränkungen.

Trotzdem können wir eine dichte Zeithierarchie für deterministische iterative Arrays beweisen.

Wir unterscheiden dabei zwischen Zeitschranken unterhalb von Linearzeit (Satz 5.9) und oberhalb von Linearzeit (Satz 5.10). Während Satz 5.10 allgemeine Methoden verwendet, wie sie auch für ähnliche Sätze bei anderen Maschinenmodellen eingesetzt werden, ist Satz 5.9 speziell auf deterministische iterative Arrays zugeschnitten. Es sind auch keine allgemeinen Methoden bekannt, um Zeitbeschränkungen zwischen Real- und Linearzeit zu behandeln.

Bei nichtdeterministischen Automaten fallen diese beiden Komplexitätsklassen häufig zusammen, z.B.

$$\begin{aligned} \mathscr{L}_{rt}(\text{NIA}) &= \mathscr{L}_{lt}(\text{NIA}) \quad \text{Korollar 5.8} \\ \mathscr{L}_{rt}(1G - \text{OCA}) &= \mathscr{L}_{lt}(1G - \text{OCA}) \quad \text{siehe [4]} \\ NTIME(n) &= NTIME(kn) \quad \text{siehe [38].} \end{aligned}$$

Für deterministische Automaten ist nur wenig bekannt, z.B. erhält man für einbändige Turing–Maschinen mit Hilfe von Crossing–Sequenzen

$$DTIME_1(o(n \log(n))) = DTIME_1(n) = \mathcal{REG}$$

(siehe [21] und [38]).

Bei unidirektionalen Zellularautomaten weiß man, daß

$$\mathscr{L}_{rt}(\text{OCA}) \subsetneqq \mathscr{L}_{n+\log(n)}(\text{OCA})$$

gilt und für Sprachen über einem einelementigen Alphabet eine Komplexitätslücke zwischen Realzeit und $n + \log(n)$ besteht (siehe [6]).

Das Problem, ob

$$\mathscr{L}_{rt}(\text{CA}) = \mathscr{L}_{lt}(\text{CA})$$

gilt, ist eines der bekanntesten offenen Probleme der Theorie der parallelen Automaten (siehe z.B. [24] zur Bedeutung dieses Problems).

Satz 5.9
Seien $t : \mathbb{N}_0 \to \mathbb{N}$ und $t' : \mathbb{N}_0 \to \mathbb{N}$ zwei Funktionen, die den folgenden Bedingungen genügen:

1. *$t'(n) \leq t(n)$ für alle $n \in \mathbb{N}_0$.*

2. *Es gibt eine Funktion* $h : \mathbb{N}_0 \to \mathbb{N}$ *mit* $h(n) \geq (n+1)^2$, *so daß* $t(h(n)) = \Omega(n^2)$ *und* $t'(h(n)) = o(n^2)$ *gilt.*

3. *Außerdem gelte* $\{a^{h(n)-n}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{NIA})$.

Dann gilt

$$\mathscr{L}_{n+t'(n)}(\text{IA}) \subsetneqq \mathscr{L}_{n+t(n)}(\text{IA}).$$

Beweis
Da $n + t'(n) \leq n + t(n)$ nach Voraussetzung 1 gilt, folgt

$$\mathscr{L}_{n+t'(n)}(\text{IA}) \subseteq \mathscr{L}_{n+t(n)}(\text{IA}).$$

Wir zeigen nun die Echtheit der Inklusion.

Da $\{a^{h(n)-n}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{NIA})$ ist, gibt es nach Satz 4.2 eine Sprache $L' \in \mathscr{L}_{rt}(\text{IA})$ und einen ε–freien, längenerhaltenden Homomorphismus h' mit $h'(L') = \{a^{h(n)-n}b^n \mid n \in \mathbb{N}\}$. L' sei über dem Alphabet Σ' definiert.

Wir definieren die Sprache L über dem Alphabet $\{\sqcup, \#, \$, 0, 1\} \times \Sigma'$ wie folgt:

Ein Wort w ist genau dann in L, wenn die folgenden Bedingungen erfüllt sind:

1. Die Projektion auf die zweite Komponente ist ein Wort aus L'.

2. Gilt $|w| = h(n)$ für ein $n \in \mathbb{N}$, so hat die erste Komponente von w die Form
$$\sqcup^{h(n)-(n+1)^2+1} w_1\$w_2\$\ldots\$w_n\#y,$$
wobei $w_i, y \in \{0,1\}^n$ und $y = w_i$ für ein $i \in \{1, \ldots, n\}$ gilt.

(Damit ist auch L eine Art Lexikonsprache.)

Zunächst zeigen wir, daß $L \notin \mathscr{L}_{n+t'(n)}(\text{IA})$ gilt. Dazu betrachten wir die Anzahl $N(h(n), n, L)$ der n–Äquivalenzklassen von L unter den Wörtern der Länge $h(n)$. Wenn wir die zweite Komponente festhalten und nur die erste Komponente betrachten, sehen wir, daß $N(h(n), n, L) \geq \binom{2^n}{n} = 2^{\Theta(n^2)}$ gilt. Andererseits kann ein $(n+t'(n))$–zeitbeschränktes iteratives Array nach Lemma 3.16 nur $s^{n+t'(h(n))}$ von n-Äquivalenzklassen unter den Wörtern der Länge $h(n)$ unterscheiden. Nach Voraussetzung gilt $t'(h(n)) = o(n^2)$, d.h. das $n + t'(n)$ zeitbeschränkte iterative Array kann höchstens $2^{o(n^2)}$ Äquivalenzklassen unterscheiden. Es folgt $L \notin \mathscr{L}_{n+t'(n)}(\text{IA})$.

Um $L \in \mathscr{L}_{n+t(n)}(\text{IA})$ zu zeigen, geben wir ein iteratives Array A an, das L in $n + t(n)$ Zeitschritten erkennt. A arbeitet nach folgendem Schema:

1. A kann erkennen, ob die zweite Komponente ein Wort aus L' ist. (L' ist sogar in Realzeit erkennbar.)

2. A überprüft, ob $|y| = n$ gilt. Dies ist genau dann der Fall, wenn h' angewandt auf die zweite Komponente für alle Zeichen bis einschließlich dem # ein a ergibt und ein b für alle Zeichen nach dem #.

3. A kann auch überprüfen, ob die erste Komponente die Form $\sqcup^{h(n)-(n+1)^2+1} w_1\$ \ldots \$w_n\#y$ mit $w_i, y \in \{0,1\}^n$ hat. Denn es gilt $\{(a^n b)^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{IA})$, d.h. A kann überprüfen, ob es genau n Teilwörter w_i gibt und ob alle diese Teilwörter w_i gleich lang sind. Zusätzlich kann A noch überprüfen, ob w_n und y gleich lang sind.

4. Die obigen Rechnungen konnte A in Realzeit ausführen. Für die Überprüfung von $y = w_i$ für ein i braucht A die zusätzlichen $t(n)$ Zeitschritte.

   Dazu speichert A alle w_i in einer FiFo–Schlange ab und vergleicht nach dem # Zeichen die w_i mit dem y. A vergleicht jedes Zeichen von y mit dem entsprechenden Zeichen von w_i. Zusätzlich speichert A das Teilwort y in einer zweiten FiFo-Schlange und kann deshalb für jedes Wort w_i das Wort y erneut lesen.

   A benötigt $\Theta(n^2)$ Schritte für den Vergleich. Wegen $t(h(n)) = \Omega(n^2)$ liegt daher $L = L(A)$ in $\mathscr{L}_{t(n)}(\mathrm{IA})$.

□

Damit haben wir eine *dichte* Zeithierarchie für Zeitbeschränkungen zwischen Real- und Linearzeit bewiesen. Die Voraussetzung 2 besagt lediglich, daß $t'(n) = o(t(n))$ und $t'(n) = o(n^2)$ gilt. Die zusätzliche Voraussetzung 3 an die Funktion h läßt sich leicht erfüllen, da die Klasse, der durch nichtdeterministische iterative Arrays in Realzeit erkennbaren Sprachen, sehr groß ist. Zum Beispiel erfüllen k^n, n^k $(k \in \mathbb{N})$ die Voraussetzung 3. Die Klasse aller Funktionen, die die Voraussetzung 3 erfüllen, ist außerdem abgeschlossen unter Addtion, Multiplikation und Verkettung von Funktionen.

Für Zeitbeschränkungen oberhalb von Linearzeit erhalten wir einen Hierarchiesatz ähnlich zu dem bekannten Resultat über Turing–Maschinen.

Satz 5.10
Seien $t', t : \mathbb{N}_0 \to \mathbb{N}$ zwei Funktionen mit $t'(n) \geq n+1$ und $t(n) \geq t'(n)$. Ist $t'(n) = o(t(n))$ und ist $t(n)$ zeitberechenbar, so gilt

$$\mathscr{L}_{t'(n)}(\mathrm{IA}) \subsetneq \mathscr{L}_{t(n)}(\mathrm{IA}).$$

Beweis
Es gibt einen universellen Zellularautomaten (siehe zum Beispiel [31]). Ein

universeller Zellularautomat benötigt jeweils k Schritte, um einen Schritt eines vorgegebenen Automaten A zu simulieren. k hängt dabei nur von A, aber nicht von der Länge der Eingabe ab.

Die Menge $\mathscr{L}_{t(n)}(\mathrm{IA})$ ist abgeschlossen bezüglich Komplementbildung. Wir können daher den allgemeinen Hierarchiesatz (Theorem 3.2.2 in [38]) anwenden. In [38] wird der allgemeine Hierarchiesatz nur für Turing–Maschinen definiert, der Beweis kann jedoch wörtlich auf Zellularräume und iterative Arrays übertragen werden. □

5.4 Nichtdeterministische Zeithierarchien

Die Ergebnisse des vorangegangen Abschnitts lassen sich nur bedingt auf nichtdeterministische iterative Arrays übertragen. Wir können jedoch folgende Zeithierarchie beweisen.

Satz 5.11
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine Beschränkung der nichtdeterministischen Schritte.

Seien $t : \mathbb{N}_0 \to \mathbb{N}$ und $t' : \mathbb{N}_0 \to \mathbb{N}$ zwei Funktionen, die den folgenden Bedingungen genügen:

1. *$t'(n) \leq t(n)$ für alle $n \in \mathbb{N}_0$.*
2. *Es gibt eine Funktion $h : \mathbb{N}_0 \to \mathbb{N}$ mit $h(n) \geq (n+1)(n2^{f(h(n))}+1)$, so daß $t(h(n)) = \Omega(n^2)$ und $t'(h(n)) = o(n^2)$ gilt.*
3. *$f(h(n)) = o(\log(n))$ und $\{(a^n b)^{n2^{f(h(n))}}\} \in \mathscr{L}_{rt}(\mathrm{IA})$.*
4. *$\{a^{h(n)-n}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{NIA})$.*

Dann gilt

$$\mathscr{L}_{n+t'(n)}(f(n) - \mathrm{NIA}) \subsetneqq \mathscr{L}_{n+t(n)}(f(n) - \mathrm{NIA}).$$

Beweis
Da $n + t'(n) \leq n + t(n)$ nach Voraussetzung 1 gilt, folgt

$$\mathscr{L}_{n+t'(n)}(f(n) - \mathrm{NIA}) \subseteq \mathscr{L}_{n+t(n)}(f(n) - \mathrm{NIA}).$$

Wir zeigen nun die Echtheit der Inklusion.

Da $\{a^{h(n)-n}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{NIA})$ ist, gibt es nach Satz 4.2 eine Sprache $L' \in \mathscr{L}_{rt}(\mathrm{IA})$ und einen ε–freien, längenerhaltenden Homomorphismus h' mit $h'(L') = \{a^{h(n)-n}b^n \mid n \in \mathbb{N}\}$. L' sei über dem Alphabet Σ' definiert.

Wir definieren die Sprache L über dem Alphabet $\{\sqcup, \#, \$, 0, 1\} \times \Sigma'$ wie folgt:

Ein Wort w ist genau dann in L, wenn die folgenden Bedingungen erfüllt sind:

1. Die Projektion auf die zweite Komponente ist ein Wort aus L'.

2. Gilt $|w| = h(n)$ für ein $n \in \mathbb{N}$, so hat die erste Komponente von w die Form
$$\sqcup^{h(n)-(n+1)(n2^{f(h(n))}+1)+1} w_1\$w_2\$\ldots\$w_{n2^{f(h(n))}}\#y,$$
wobei $w_i, y \in \{0,1\}^n$ und $y = w_i$ für ein $i \in \{1, \ldots, n2^{f(h(n))}\}$ gilt.

(Damit ist auch L eine Art Lexikonsprache.)

Im Vergleich mit Satz 5.9 haben wir die Anzahl der Wörter im Lexikon erhöht.

Durch Anwendung des Äquivalenzklassenlemmas 3.16 erhalten wir analog zum Beweis von Satz 5.9 $L \notin \mathscr{L}_{n+t'(n)}(f(n) - \text{NIA})$. Wir müssen daher nur noch ein $f(n) - \text{NIA}$ konstruieren, das L in $n + t(n)$ Zeitschritten erkennt. Das entsprechende $f(n) - \text{NIA}$ A arbeitet nach folgendem Schema:

1. A kann erkennen, ob die zweite Komponente aus L' ist. Dies ist sogar in Realzeit möglich.

2. A überprüft, ob $|y| = n$ gilt. Dies ist genau dann der Fall, wenn h' angewandt auf die zweite Komponente für alle Zeichen bis einschließlich dem # ein a ergibt und ein b für alle Zeichen nach dem #.

3. A kann auch überprüfen, ob die erste Komponente die Form $\sqcup^{h(n)-(n+1)(n2^{f(h(n))}+1)+1} w_1\$\ldots\$w_{n2^{f(h(n))}}\#y$ mit $w_i, y \in \{0,1\}^n$ hat. Denn nach Voraussetzung gilt $\{(a^n b)^{n2^{f(h(n))}} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{IA})$, d.h. A kann überprüfen, ob es genau $n2^{f(h(n))}$ Teilwörter w_i gibt und ob alle diese Teilwörter w_i gleich lang sind. Zusätzlich kann A noch überprüfen, ob w_n und y gleich lang sind.

4. Die obigen Rechnungen konnte A in Realzeit ausführen. Für die Überprüfung von $y = w_i$ für ein i braucht A die zusätzlichen $t(n)$ Zeitschritte.

   A unterteilt die erste Komponente in $2^{f(h(n))}$ Blöcke mit je n Teilwörtern w_i, d.h. der erste Block enthält $w_1, \ldots, w_n$ und der letzte Block enthält $w_{n2^n-n+1}, \ldots, w_{n2^n}$. Da $|w_i| = n$ für alle $i \in \mathbb{N}$ gilt, muß A nur überprüfen, daß jeder Block genauso viele Wörter enthält, wie Zeichen in einem Wort w_i enthalten sind.

Seine $f(h(n))$ nichtdeterministischen Schritte benutzt A, um in binärer Darstellung die Nummer i eines Blocks, in dem das Wort y vorkommt, zu raten. Wie im Beweis von Satz 2.19 kann A garantieren, daß höchstens $f(h(n))$ nichtdeterministische Schritte ausgeführt werden.

Nun speichert A alle Wörter $w_{n(i-1)+1}, \ldots, w_{ni}$ in einer FiFo–Schlange ab und vergleicht nach dem # Zeichen diese Wörter nacheinander mit y.

A benötigt $\Theta(n^2)$ Schritte für den Vergleich. Wegen $t(h(n)) = \Omega(n^2)$ liegt $L = L(A)$ daher in $\mathscr{L}_{n+t(n)}(f(n) - \text{NIA})$.

□

Bemerkung 5.12
Wir wählen eine feste Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ für die Beschränkung der nichtdeterministischen Schritte. In Satz 5.7 haben wir gezeigt, daß für $f(n) - \text{NIA}$ eine Komplexitätslücke zwischen den Zeitschranken $t_1(n) = n + 1$ und $t_2(n) = n + f(n)$ besteht. Der obige Satz zeigt, daß es für $f(n) - \text{NIA}$ eine dichte Zeithierarchie zwischen $t_3(n) = n + 2^{f(n)}$ und $t_4(n) = 2n$ existiert. In diesem Bereich vordern die Vorrausetzungen des obigen Satzes nur, daß $t'(n) = o(t(n))$ gilt und $t(n)$ gewissen leicht erfüllbaren Konstruierbarkeits Bedingungen genügt. (Vergeiche auch die Bemerkungen zu Satz 5.9.)

Zwischen t_2 und t_3 können wir das Klassenlemma nicht verwenden, um eine dichte Hierarchie zu zeigen, denn in diesem Bereich fallen alle oberen Schranken zusammen. (Es gilt $(s^{l+t(n)-n})^{rf(n)} = \Theta((s^l)^{rf(n)})$ für alle Zeitschranken $t(n)$ mit $t(n) \leq t_3(n)$.)

Das Problem für Hierarchiesätze oberhalb von t_4 ist, daß die Familie $\mathscr{L}_{t(n)}(f(n) - \text{NIA})$ im allgemeinen nicht bezüglich Komplementbildung abgeschlossen ist. Daher kann der auf Diagonalisierung beruhende allgemeine Hierarchiesatz nicht angewandt werden.

5.5 Speicherhierarchien

Bisher haben wir Hierarchien für Nichtdeterminismus und Zeit bewiesen. Es existiert auch eine Hierarchie über die Dimension der iterativen Arrays. Für deterministische iterative Arrays wurde diese Hierarchie bereits in [12] gezeigt. Eine ähnliche Hierarchie für Linerarzeitberechnungen durch iterative Arrays, bei denen alle Zellen nichtdeterministisch sind, wird in [43] bewiesen.

Es ist offensichtlich, daß $\mathscr{L}_{rt}(f(n) - \text{NIA}_d) \subseteq \mathscr{L}_{rt}(f(n) - \text{NIA}_{d+1})$ gilt. Die Echtheit der Inklusion folgt aus dem folgenden Satz:

Satz 5.13
Es sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ mit $f(n) = o(\log(n))$. Für jede Dimension $d \in \mathbb{N}$ gibt es eine Sprache L, die in $\mathscr{L}_{rt}(\mathrm{IA}_{d+1})$ liegt, aber nicht in $\mathscr{L}_{rt}(f(n) - \mathrm{NIA}_d)$.

Beweis
Es sei L die Lexikonsprache

$$L = \{w_1\$w_2\$\ldots\$w_{n^d}\#y\#^{n(2d+1)} \mid w_i, y \in \{0,1\}^n \text{ und } y = w_i \text{ für ein } i \in \{1,\ldots,n^d\}\}.$$

(L ist ein Variante der Lexikonsprache L_{n^d}, die dem Automaten noch zusätzlich $n(2d+1)$ Schritte für die Überprüfung zur Verfügung stellt.)

Der Beweis $L \notin \mathscr{L}_{rt}(f(n) - \mathrm{NIA}_d)$ verläuft analog zu dem entsprechenden Beweis aus Satz 5.1: Die Anzahl $N((n+1)(n^d) + n(2d+2), n(2d+2), L)$ der Äquivalenzklassen, die durch Teilworte der Form $w_1\$\ldots\$w_{n^d}\#$ bestimmt werden, ist gleich $\binom{2^n}{n^d} = 2^{\Theta(n^{d+1})}$. Aber ein $f(n) - \mathrm{NIA}_d$ kann nach Lemma 3.15 in Realzeit nur

$$(s^{(n(2d+1)^d)})^{(r^{f((n+1)(n^d)+n(2d+1))})} = (2^{O(n^d)})^{o(n)} = 2^{o(n^{d+1})}$$

Äquivalenzklassen unterscheiden.

Wir zeigen nun $L \in \mathscr{L}_{rt}(\mathrm{IA}_{d+1})$. Ein $(d+1)$–dimensionales iteratives Array, das L erkennt, benutzt nur Zellen aus einem $(d+1)$–dimensionalen Hyperwürfel mit der Kantenlänge n. In diesem Würfel kann das iterative Array alle Wörter w_i speichern. Das iterative Array speichert in einem d–dimensionalen Hyperwürfel jeweils die ersten Buchstaben der n^d Wörter. Die einzelnen Wörter werden in n aufeinanderfolgenden Zellen in Richtung der $(d+1)$-ten Koordinate geschrieben. Abbildung 5.1 zeigt am Beispiel eines zweidimensionalen iterativen Arrays, wie die Wörter w_i gespeichert werden können. Dabei gilt $w_i = w_i^{(1)} w_i^{(2)} \ldots w_i^{(n)}$.

Im höherdimensionalen Fall muß das iterative Array außerdem noch ein Gebiet der Größe n^d markieren, damit es die Wörter w_i an der richtigen Stelle abspeichern kann. Diese Markierung kann leicht in $n(d+1)$ Schritten durchgeführt werden. (Man benutzt dazu ein Signal, das die Diagonale des n^d Hyperwürfels durchläuft. Mit Hilfe des Wortes w_1 kann das iterative Array bis n zählen und so den richtigen Zeitpunkt zum Stoppen des Signal ermitteln.) Also kann das iterative Array die Grenzen des n^d Hyperwürfels ermitteln, bevor es das $(n+1)$–te w_i speichern muß, d.h. das iterative Array kann dafür sorgen, daß alle w_i in dem Würfel gespeichert werden.

Nach dem ersten # Zeichen weiß das iterative Array, daß das Wort y anliegt. y wird mit allen gespeicherten w_i verglichen, und falls $w_i = y$ für ein i gilt, wird ein Erfolgssignal erzeugt, das zur Ursprungszelle läuft. Dies

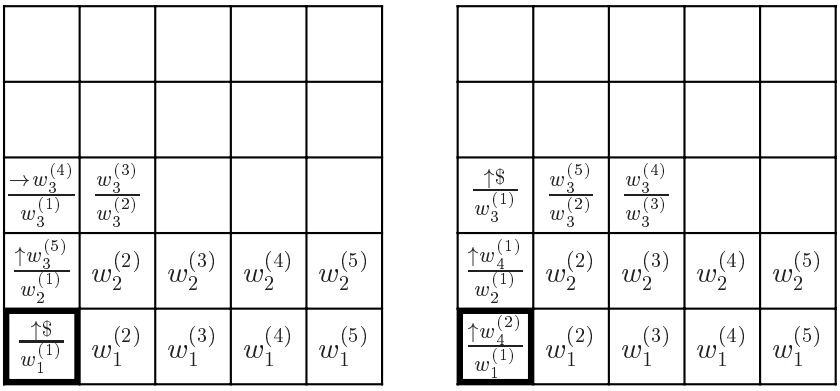


Abbildung 5.1: Beispiel für $d = 1$ und $n = 5$. Die Ursprungszelle ist durch einen Rahmen gekennzeichnet. Das rechte Diagramm zeigt das iterative Array zwei Zeitschritte nach dem linken Diagramm. Die Pfeile markieren in welche Richtung die Zeichen $w_i^{(j)}$ bewegt werden sollen. Ein \$ schaltet ein $\rightarrow$ in einen $\uparrow$ um.

geschieht, indem y an denselben Positionen gespeichert wird, wie die w_i. Beim Speichern kann das Wort y dann Zeichen für Zeichen mit dem bereits vorhanden Wort w_i verglichen werden.

Die $n(2d + 1)$ Zeichen # nach dem y zusammen mit den n Zeichen des y geben dem iterativen Array genug Zeit, den Vergleich durchzuführen ($n(d + 1)$ Zeitschritte) und das Erfolgssignal zum Ursprung zu schicken ($n(d + 1)$ Zeitschritte). □

Inbesondere haben wir damit die Hierarchie aus [12] auf nichtdeterministische iterative Arrays verallgemeinert.

Korollar 5.14
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$. *Für* $f(n) = o(\log(n))$ *gilt*

$$\mathscr{L}_{rt}(f(n) - \mathrm{NIA}_d) \subsetneqq \mathscr{L}_{rt}(f(n) - \mathrm{NIA}_{d+1})$$

für alle $d \in \mathbb{N}$.

5.6 Nichtdeterminismus und Speicher

Im letzten Abschnitt haben wir gesehen, daß ein iteratives Array mindestens $\log(n)$ nichtdeterministische Schritte machen muß, um die Dimension des Speichers um 1 zu senken. In diesem Abschnitt werden wir das Verhältnis von Nichtdeterminismus und Speicher noch etwas genauer untersuchen.

Nichtdeterminismus gibt einem Automaten in gewisser Hinsicht die Möglichkeit, mehrere Berechnungen gleichzeitig auszuführen. Andererseits kann man ein 2-dimensionales iteratives Array auch als mehrere parallel arbeitende 1-dimensionale iterative Arrays auffassen. In diesem Abschnitt untersuchen wir Möglichkeiten, wie man Nichtdeterminismus durch höhere Speicherdimensionen ersetzen bzw. durch nichtdeterministische Schritte Speicherplatz einsparen kann.

Zunächst zeigen wir, daß bei Realzeit ein Einsparen von Nichtdeterminismus nicht möglich ist.

Satz 5.15
Es gibt eine Sprache $L \in \mathscr{L}_{rt}(\log(n) - \text{NIA})$, die von keinem d-dimensionalen deterministischen iterativen Array in Realzeit erkannt werden kann.

Beweis
Wir betrachten wieder die Lexikonsprache

$$L = \bigcup_{k \in \mathbb{N}} L_k$$

Erinnerung :

$$L_k = \{w_1\$ \ldots \$w_k\#y\# \mid w_i, y \in \{0,1\}^n \text{ und } y = w_i \text{ für ein } i \in \{1, \ldots, k\}\}.$$

Ein $\log(n) - \text{NIA}$ kann den Index i, für den $w_i = y$ gilt, raten und überprüfen, ob es richtig geraten hat. Dazu rät das $\log(n) - \text{NIA}$ wie im Beweis von Satz 2.19 beschrieben einen Binärzähler, der nach jedem \$ Zeichen um 1 vermindert wird. Ist der Zähler gleich 0, wird das betreffende w_i in einer FiFo–Schlange abgespeichert und nach dem ersten # Zeichen mit dem Wort y verglichen. Also gilt $L \in \mathscr{L}_{rt}(\log(n) - \text{NIA})$.

Wir zeigen nun, daß L von keinem d-dimensionalen deterministischen iterativen Array in Realzeit erkannt werden kann. Dazu betrachten wir die Anzahl $N((n+1)(n^d+1), n+1, L)$ der $(n+1)$–Äquivalenzklassen bezüglich L unter den Wörtern der Länge $(n+1)n^d$.

Für jede Menge $\{w_1, \ldots, w_{n^d}\}$ gibt es eine Äquivalenzklasse, d.h. die Anzahl aller Äquivalenzklassen ist $\binom{2^n}{n^d} = 2^{\Theta(n^{d+1})}$. Nach dem Äquivalenzklassenlemma kann ein d-dimensionales deterministisches iteratives Array nur $s^{n^d} = 2^{O(n^d)} = 2^{o(n^{d+1})}$ Äquivalenzklassen unterscheiden. Dies ist ein Widerspruch, d.h. $L \notin \mathscr{L}_{rt}(\text{IA}_d)$. □

Wir können dieses Lemma auf beliebige unbeschränkte Funktionen f verallgemeinern.

Satz 5.16
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ *eine unbeschränkte Funktion. Es gebe eine Funktion* $h : \mathbb{N}_0 \to \mathbb{N}_0$ *mit* $f(h(n)) = \Omega(\log(m))$ *und* $\{a^{h(n)-n}b^n | n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{NIA})$.

Dann gibt es eine Sprache aus $\mathscr{L}_{rt}(f(n) - \text{NIA})$, *die nicht in* $\mathscr{L}_{rt}(\text{IA}_2)$ *liegt.*

Beweis
Wir benutzen eine ähnliche Technik wie beim Beweis von Satz 5.9.

Da $\{a^{h(n)-n}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\text{NIA})$ ist, gibt es nach Satz 4.2 eine Sprache $L' \in \mathscr{L}_{rt}(\text{IA})$ und einen ε–freien, längenerhaltenden Homomorphismus h' mit $h'(L) = \{a^{h(n)-n}b^n \mid n \in \mathbb{N}\}$. L' sei über dem Alphabet Σ' definiert.

Wir definieren eine Sprache L über dem Alphabet $\{\sqcup, \#, \$, 0, 1\} \times \Sigma'$ wie folgt:

Ein Wort w ist genau dann in L enthalten, wenn die folgenden Bedingungen erfüllt sind.

1. Die zweite Komponente ist ein Wort aus L'.

2. Gilt $|w| = h(m)$ für ein $m \in \mathbb{N}$, so hat die erste Komponente von w die Form
$$\sqcup^{h(m)-m} w_1\$w_2\$\ldots\$w_k\#y\#,$$
wobei $k \in \mathbb{N}$, $w_i, y \in \{0,1\}^n$ für ein $n \in \mathbb{N}$ und $w_i = y$ für ein $i \in \{1, \ldots, k\}$.
(Es gilt $|w_1\$w_2\$\ldots\$w_k\#y\#| = m$.)

Wenn wir die zweite Komponente festhalten und nur die erste Komponente betrachten, erhalten wir wie im Beweis des vorherigen Lemmas, daß $L \notin \mathscr{L}_{rt}(\text{IA}_d)$ gilt. (Wir betrachten die Anzahl $N(h(m), n+1, L)$ mit $m = (n+1)(n^d+1)$.)

Ein $f(n) - \text{NIA}$, das L in Realzeit erkennt, arbeitet wie folgt.

Ein iteratives Array kann deterministisch überprüfen, ob die zweite Komponente in L' liegt. Außerdem kann es überprüfen, ob die erste Komponente mit $h(m)-m$ Zeichen $\sqcup$ beginnt. (In der ersten Komponente muß genau dann ein $\sqcup$ stehen, wenn die zweite Komponente durch h' auf a abgebildet wird.)

Da $f(h(m)) = \Omega(\log(m))$ gilt, stehen einem $f(n) - \text{NIA}$ $\log(m)$ nichtdeterministische Schritte zur Verfügung, um den $w_1\$w_2\$\ldots\$w_k\#y\#$ Teil zu erkennen. Auf diesem Teil kann also das $\log(n) - \text{NIA}$ aus dem vorherigen Lemma simuliert werden.

Es folgt $L \in \mathscr{L}_{rt}(f(n) - \text{NIA})$. □

Wir haben somit gezeigt, daß nichtdeterministische iterative Arrays in Realzeit Sprachen erkennen können, die von keinem deterministischen iterativen Array in Realzeit erkannt werden können. Für die Simulation nichtdeterministischer iterativer Arrays durch deterministische iterative Arrays müssen wir daher einen Zeitverlust in Kauf nehmen. (Die Erhöhung der Dimension alleine reicht nicht aus, um Nichtdeterminismus einzusparen.)

Der folgende Satz zeigt, daß die Erhöhung der Dimension um 1 und linearer Zeitverlust ausreichend sind, um $o(\log(n))$ nichtdeterministische Schritte einzusparen.

Satz 5.17
Ist $f : \mathbb{N}_0 \to \mathbb{N}_0$ *eine Funktion mit* $f(n) = o(\log(n))$, *so gilt*

$$\mathscr{L}_{lt}(f(n) - \text{NIA}) \subseteq \mathscr{L}_{lt}(\text{IA}_2).$$

Beweis
Sei A ein $f(n) - \text{NIA}$. In jedem nichtdeterministischen Schritt verzweigt sich A in maximal r Möglichkeiten. Die Eingabe sei $w = a_1 \dots a_n$.

Die Simulation von A durch ein 2–dimensionales deterministisches iteratives Array A' läuft in zwei Schritten ab.

1. **Schritt : Vorbereitung der Simulation**
   A' generiert im Zahlsystem der Basis r Zahlen von 0 bis $r^{f(n)} - 1$ und speichert sie in seinen Zellen ab (siehe Abbildung 5.2 unteres Register). In jeder Zeile steht einer Zahl.

   A bestimmt nicht die Zahl $f(n)$. Stattdessen generiert A einfach alle Zahlen zur Basis r. In der folgenden Rechnung werden jedoch nur die Zahlen von 0 bis $r^{f(n)} - 1$ benutzt.

   Außerdem liest A' die Eingabe ein und speichert sie in jeder Zeile ab. ($a_1 \cdots a_8$ in Abbildung 5.2.)

   Anschließend startet A' in jeder Zeile ein FSSP, um die Zellen für die anschließende Simulation zu synchronisieren. Die Phase 1 endet in Zeile i nach genau $i+3n$ Schritten. (i Schritte bis die Eingabe Zeile i erreicht, n weitere Schritte bis auch das n–te Zeichen von w die i–te Zeile erreicht und $2n$ Schritte für das FSSP.)

   Daher geht jede der Zeilen 0 bis $r^{f(n)} - 1$ nach spätestens $r^{f(n)} + 3n$ Zeitschritten in Phase 2 über.

2. **Schritt : Simulation**
   Nun simuliert A' die $r^{f(n)}$ verschiedenen Berechnungspfade des Automaten A. Immer wenn A eine nichtdeterministische Wahl trifft, benutzt

$\frac{a_1}{r-1}$	$\frac{a_2}{r-1}$	$\frac{a_3}{r-1}$	a_4	a_5	a_6	a_7	a_8
$\frac{a_1}{r-2}$	$\frac{a_2}{r-1}$	$\frac{a_3}{r-1}$	a_4	a_5	a_6	a_7	a_8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{a_1}{1}$	$\frac{a_2}{0}$	$\frac{a_3}{0}$	a_4	a_5	a_6	a_7	a_8
$\frac{a_1}{0}$	$\frac{a_2}{0}$	$\frac{a_3}{0}$	a_4	a_5	a_6	a_7	a_8

Abbildung 5.2: Ein Beispiel für $n = 8$. Das Wort $w = a_1 \cdots a_8$ wurde in allen Zeilen gespeichert. In den zweiten Registern wurde ein Binärzähler generiert. Das niedigstwertige Bit des Zählers steht links, z.B. hat der Zähler in der zweiten Zeile von unten den Wert 1. Die Ursprungszelle ist durch den Rahmen gekennzeichnet.

A' die nichtdeterministische Möglichkeit von A, die durch die Zahl im zweiten Register bestimmt wird. (In der untersten Zeile wählt A' immer die erste Möglichkeit, denn der Zähler ist gleich 0. In der zweiten Zeile wählt A' im ersten Schritt die zweite Möglichkeit und danach immer die erste Möglichkeit, da die erste Stelle des Zählers gleich 1 und alle anderen Stellen gleich 0 sind. In der Zeile $r^{f(n)} - 1$ wählt A' immer die r-te Möglichkeit von A.)

Auf diese Weise werden alle möglichen Berechnungen von A simuliert. Da die Anzahl der nichtdeterministischen Schritte von A durch $f(n)$ beschränkt ist, reichen die Zeilen 0 bis $r^{f(n)} - 1$ von A' zur Simulation von A aus.

Falls eine dieser Berechnungen zum Erfolg führt, schickt A' ein Erfolgssignal zur Ursprungszelle und akzeptiert. Anderenfalls wird A' nicht akzeptieren.

Insgesamt dauert diese Phase der Simulation höchstens $2n+r^{f(n)}$ Schritte. ($2n$ Schritte für die eigentliche Simulation und maximal weitere $r^{f(n)}$ Schritte bis das Signal bei der Ursprungszelle eintrifft.)

Falls A' akzeptiert, so steht dies nach höchstens

$$(r^{f(n)} + 3n) + (2n + r^{f(n)}) \leq c \cdot n$$

Schritten fest (c konstant). A' kann den Zeitpunkt $c \cdot n$ berechnen. Falls A' bis zu diesem Zeitpunkt noch nicht akzeptiert hat, wechselt er in einen nichtakzeptieren Endzustand. Dadurch wird erreicht, daß A' nach spätestens $c \cdot n$ Zeitschritten anhält. Gemäß Lemma 3.11 läßt sich die Berechnung von A' auf $2n$ Zeitschritte beschleunigen, d.h. die Berechnung von A' findet in Linearzeit statt. □

Um nichtdeterministische iterative Arrays *ohne* Beschränkung des Nichtdeterminismus mit linearem Zeitverlust deterministisch simulieren zu können, benötigt man eine mächtigere Speicherstruktur als beim 2–dimensionalen iterativen Array.

Bei einem iterativen Array sind die einzelnen Zellen in Form eines d–dimensionalen Gitters miteinander verbunden. Wir betrachten nun einen Automaten, bei dem die einzelnen Zellen in Form eines Binärbaums miteinander verbunden werden. Die Wurzel des Baums wird zusätzlich mit einem externen Eingabeband verbunden, dient also als Ursprungszelle. Bei den anderen Zellen wird noch zwischen „linken“ und „rechten“ Nachfolgerknoten unterschieden.

Definition 5.18
*Ein iterativer Baumautomat (engl.: iterative tree automaton (*ITA*)) ist ein* 9*–Tupel* $(S, \Sigma, F_a, F_r, \delta_l, \delta_r, \delta_0, q_0, \sqcup)$*, für das gilt:*

1. S, Σ, F_a, F_r, q_0 *und* $\sqcup$ *sind eine endliche Zustandsmenge, ein endliches Eingabealphabet, die Menge der akzeptierenden und nicht akzeptierenden Endzustände, der Ruhezustand und das leere Bandsymbol. (Die Bedeutung ist dieselbe wie bei iterativen Arrays.)*

2. $\delta_l : S^4 \to S$ *und* $\delta_r : S^4 \to S$ *sind die Überführungsfunktionen für die linken bzw. rechten Nachfolgerzellen.*

3. $\delta_0 : S^3 \times \Sigma \to S$ *ist die Überführungsfunktion der Ursprungszelle.*

4. *Es gilt* $\delta_l(q_0, q_0, q_0, q_0) = q_0$ *und* $\delta_r(q_0, q_0, q_0, q_0) = q_0$.

Bemerkung 5.19
Der neue Zustand einer Zelle hängt außer von dem aktuellen Zustand der Zelle, ihres Vorgängerknotens und ihrer beiden Nachfolgerknoten zusätzlich davon ab, ob die Zelle linker oder rechter Nachfolger ist. Diese Unterscheidung ist wichtig, da ITAs *sonst nicht mächtiger wären als eindimensionale iterative Arrays.*

In [13] werden iterative Baumautomaten untersucht. Insbesondere wird dort gezeigt, daß eine Variante der allgemeinen Lexikonsprache aus Satz 2.15

in $\mathscr{L}_{rt}$(ITA) liegt, aber von keinen d–dimensionalen iterativen Array in Realzeit erkannt werden kann. Ein weiteres interessantes Resultat ist, daß iterative Baumautomaten das $\mathcal{NP}$–vollständige Erfüllbarkeitsproblem in Linearzeit lösen können.

Wir zeigen nun, daß iterative Baumautomaten ohne großen Zeitverlust nichtdeterministische iterative Arrays simulieren können.

Satz 5.20
Seien $t : \mathbb{N}_0 \to \mathbb{N}$ und $f : \mathbb{N}_0 \to \mathbb{N}_0$ zwei Funktionen.

Wird eine Sprache L von einem $f(n)-$NIA in $t(n)$ Zeitschritten erkannt, so gibt es eine Konstante c, so daß L in $t(n)+cf(n)$ Zeitschritten von einem deterministischen ITA *erkannt wird.*

Insbesondere gilt $\mathscr{L}_{lt}$(NIA) $\subseteq \mathscr{L}_{lt}$(ITA).

Beweis
Sei A ein $f(n)-$NIA, das L in $t(n)$ Zeitschritten erkennt. Wir konstruieren einen iterativen Baumautomaten A', der A simuliert.

Wir betrachten zunächst den Fall, daß A deterministisch ist. In diesem Fall simuliert A' in der Ebene i die Zelle i von A. D.h. die Wurzel von A' entspricht der Ursprungszelle von A. Die beiden Nachfolger sind immer im gleichen Zustand und simulieren die Zelle 1, usw. Es ist klar, daß in diesem Fall A' das iterative Array A ohne Zeitverlust simulieren kann.

Nun betrachten wir den Fall, daß sich A in einem nichtdeterministischen Schritt in r verschiedene Berechnungen verzweigt. Jeweils in einem solchen Fall verlagert A' seine Berechnung um $x = \log(r)$ Schichten im Baum nach unten. In dieser Schicht besitzt A' $2^x \geq r$ Knoten. Damit kann A' die r verschiedenen Berechnungen von A simulieren. Im Vergleich zu A benötigt A' zusätzlich x Schritte. (Siehe Abbildung 5.3 zur Veranschaulichung.)

Im Beispiel hat sich A in 3 Möglichkeiten verzweigt. A' muß daher seine Simulation von A um zwei Ebenen verlagern. Dadurch erhält A' genügend Platz, um vier verschiedene Berechnungen von A zu simulieren. Die drei ersten Plätze benutzt A', um die drei Möglichkeiten von A zu simulieren. Der letzte Platz bleibt ungenutzt (dargestellt durch die schwarzen Knoten). Außerdem sieht man, daß A' die Zelle 1 von A jeweils doppelt simuliert, usw. Die Zahlen 0 und 1 in dem oberen Knoten sind für die Mißerfolgssignale und werden später erläutert.

Jedesmal wenn A einen nichtdeterministischen Schritt ausführt, verlagert A' seine Berechnung um weitere Ebenen nach unten. Auf diese Weise hat A' immer eine ausreichende Zahl an parallelen Rechnungen, um alle Wahlmöglichkeiten von A zu simulieren. Das Verlagern der Simulation, um x Ebenen nach unten, kostet x Zeitschritte. Während dieser x Zeitschritte ist die Simulation von A unterbrochen. Da x nur von dem Automaten

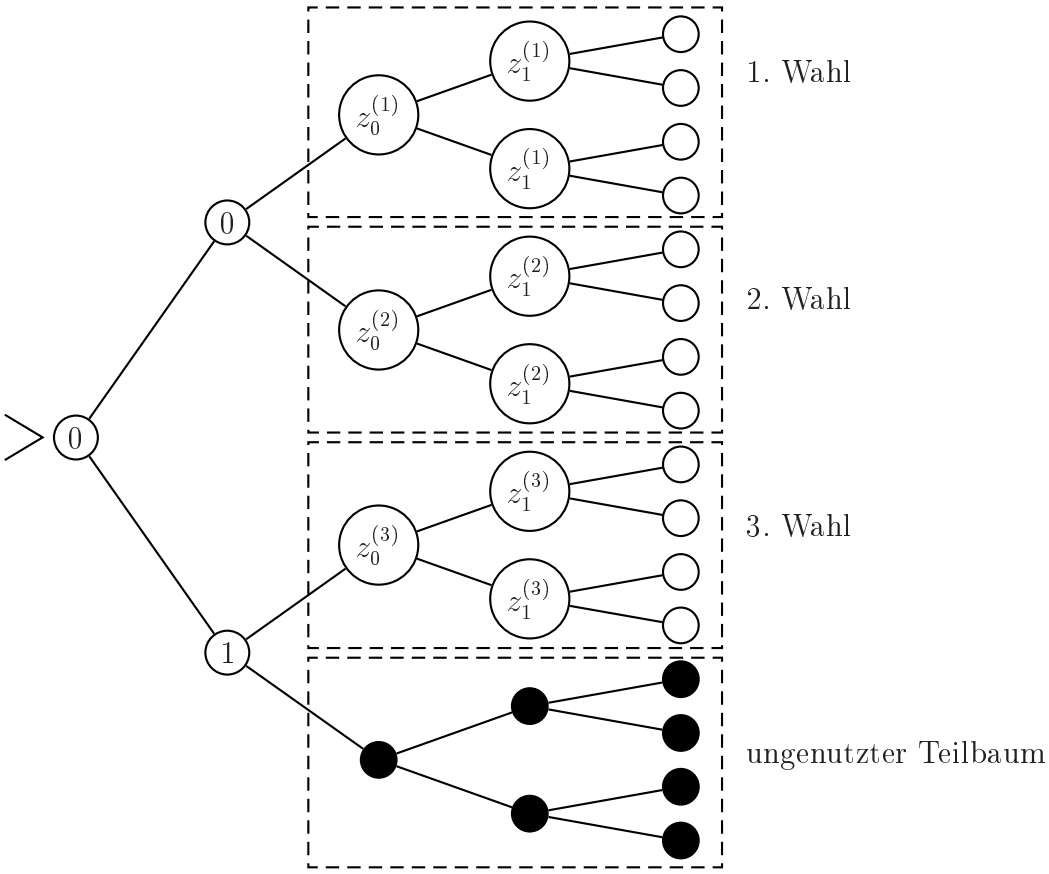


Abbildung 5.3: Der iterative Baumautomat A', nachdem sich A in drei Berechnungspfade verzweigt hat.

A abhängt, können die Knoten von A' alle zum Verlagern der Berechnung notwendigen Entscheidungen in ihren endlichen Speichern treffen.

Erkennt eine Berechnung von A, so erzeugt die entsprechende Berechnung von A' ein Erfolgssignal, das zur Ursprungszelle geschickt wird. Ist

$$x_{max} = \max\{log(r) | \text{A verzweigt in } r \text{ Berechnungen}\},$$

so kommt das Erfolgssignal nach spätestens $t(n) + 2x_{max}f(n)$ Zeitschritten bei der Ursprungszelle an. ($t(n)$ Zeitschritte, um die $t(n)$ Schritte von A zu simulieren, höchstens x_{max} Zeitschritte für jeden nichtdeterministischen Schritt von A und höchstens $x_{max}f(n)$ Zeitschritte bis das Signal die Ursprungszelle erreicht.)

Wir brauchen jetzt nur noch zu zeigen, daß A' auch im Fall, daß A nicht

erkennt, immer nach $t(n)+cf(n)$ Zeitschritten anhält. Dazu erzeugt A', falls eine simulierte Berechnung von A in einen nicht akzeptierenden Endzustand gelangt, ein Mißerfolgssignal. Außerdem erzeugt jeder unbenutzte Teilbaum von A' ein Mißerfolgssignal. Diese Signale werden in Richtung der Ursprungszelle geschickt. Allerdings leitet jeder Knoten nur jedes zweite Mißerfolgssignal weiter. (Im Beispiel haben die Ursprungszelle und der obere Knoten bisher noch kein Mißerfolgssignal erhalten (Zustand 0), während der untere Knoten bereits ein Mißerfolgssignal empfangen hat (Zustand 1).) Empfängt die Ursprungszelle das zweite Mißerfolgssignal, so weiß A', daß *alle* simulierten Berechnungen von A in einem nicht akzeptierenden Zustand enden. A' wechselt dann in einen Zustand aus F_r und hält an. □

Die folgende Abbildung faßt die von uns bewiesen Hierarchiesätze zusammen.

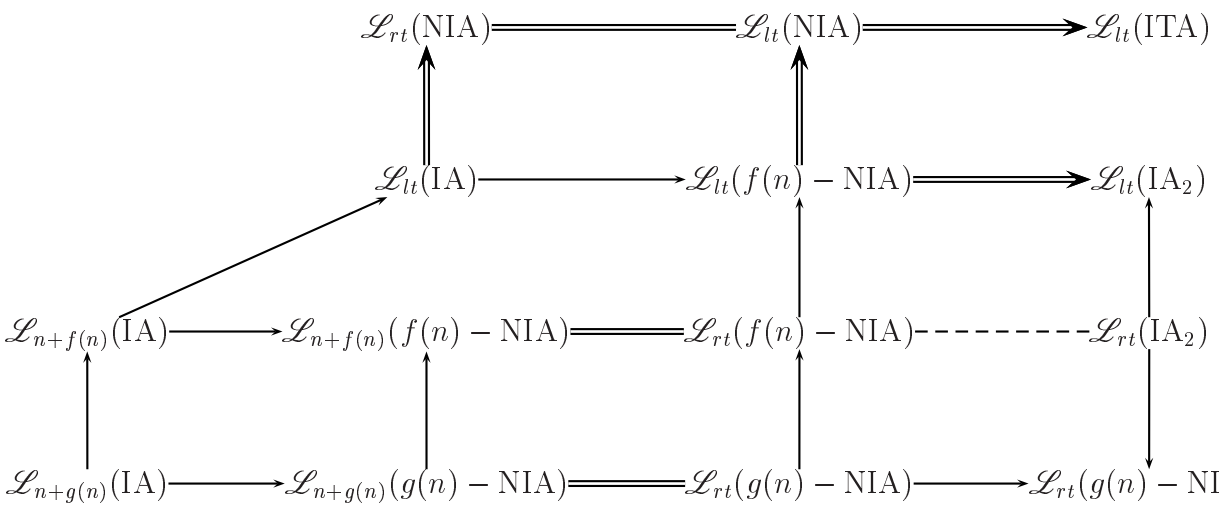
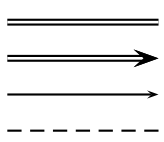


Abbildung 5.4: $f, g : \mathbb{N}_0 \to \mathbb{N}_0$ sind Funktionen mit $g(n) = o(f)$ und $f(n) = o(\log(n))$. Zusätzlich müssen f und g noch einige Brechenbarkeitsanforderungen erfüllen. Es bedeutet:



$=$
$\subseteq$
$\subset$
$\neq$
unvergleichbar

Kapitel 6

Alternierende iterative Arrays

6.1 Definition

Ein Automat akzeptiert nach einem nichtdeterministischen Schritt genau dann, wenn eine der Nachfolgekonfigurationen akzeptiert. Bei einer Verallgemeinerung des Nichtdeterminismus läßt man Konfigurationen zu, bei denen der Automat nur dann akzeptiert, wenn alle Nachfolgekonfigurationen akzeptieren. Man spricht dann von *alternierenden* Automaten. Alternierende Automaten wurden zuerst in [10] für Turing–Maschinen eingeführt.

In diesem Abschnitt definieren wir iterative Arrays mit eingeschränkten Alternierungen ($f(n) - \mathrm{AIA}$).

Definition 6.1
*Ein alternierendes iteratives Array (*AIA*) ist ein 9-Tupel*

$$(S_e, S_u, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup),$$

für das gilt:

1. *S_e ist die Menge der* existentiellen *Zustände und S_u ist die Menge der* universellen *Zustände. Es gilt $S_e \cap S_u = \emptyset$ und $S = S_e \cup S_u$ ist endlich. (S ist die Zustandsmenge.)*

2. *$(S, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup)$ erfüllt die Definition 2.13 eines* NIA.

Genau wie für NIAs können wir globale Konfigurationen und deren Nachfolgekonfigurationen definieren:

1. Eine Konfiguration (w, c) heißt Endkonfiguration, wenn $c(0) \in F$ mit $F = F_a \cup F_r$ gilt.

2. Ein AIA A benötigt höchstens t Zeitschritte bei Eingabe von w, wenn beginnend bei (w, c_0) jede Wahl von Nachfolgekonfigurationen nach höchstens t Schritten eine Endkonfiguration ergibt. Wir schreiben $time_A(w) \leq t$.

3. Sei $t : \mathbb{N}_0 \to \mathbb{N}$ mit $t(n) \geq n+1$. Das AIA A heißt $t(n)$–zeitbeschränkt, wenn $time_A(w) \leq t(|w|)$ für alle Wörter $w \in \Sigma^*$ gilt.

4. Eine Konfiguration ist nichtdeterministisch, wenn es mehr als eine Nachfolgekonfiguration gibt.

5. Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$. Das AIA A benötigt höchstens $f(n)$ nichtdeterministische Schritte, wenn für alle $w \in \Sigma^*$ beginnend bei (w, c_0) jede Wahl von Nachfolgekonfigurationen bis zum Erreichen einer Endkonfiguration höchstens $f(|w|)$ nichtdeterministische Konfigurationen liefert. Wir nennen A in diesem Fall ein $f(n)$ – AIA.

Um die von einem alternierenden iterativen Array akzeptierte Sprache zu definieren, benötigen wir noch einige Hilfsmittel.

Definition 6.2
Eine Konfiguration (w, c) *heißt* existentiell, *wenn* $c(0) \in F_e$ *gilt. Gilt* $c(0) \in F_u$, *so heißt die Konfiguration* universell.

Wir definieren nun eine (partielle) Funktion *accept* von der Menge aller Konfiguration nach $\mathbb{B} = \{wahr, falsch\}$.

Ist (w, c) eine Endkonfiguration, so gilt

$$accept((w,c)) = \begin{cases} wahr & \text{falls } c(0) \in F_a \\ falsch & \text{falls } c(0) \in F_r \end{cases}.$$

Für alle anderen existentiellen Konfigurationen (w, c) gilt

$$accept((w,c)) = \begin{cases} wahr & \text{falls } accept((w',c')) = wahr \\ & \text{für eine Nachfolgekonfiguration } (w',c') \text{ von } (w,c) \\ falsch & \text{falls } accept((w',c')) = falsch \\ & \text{für alle Nachfolgekonfigurationen } (w',c') \text{ von } (w,c) \end{cases}$$

Für alle anderen universellen Konfigurationen (w, c) gilt

$$accept((w,c)) = \begin{cases} wahr & \text{falls } accept((w',c')) = wahr \\ & \text{für alle Nachfolgekonfigurationen } (w',c') \text{ von } (w,c) \\ falsch & \text{falls } accept((w',c')) = falsch \\ & \text{für eine Nachfolgekonfiguration } (w',c') \text{ von } (w,c) \end{cases}$$

Nun können wir die von einem AIA akzeptierte Sprache definieren.

Definition 6.3
Ein AIA A *akzeptiert das Wort* w*, wenn* $accept((w, c_0)) = wahr$ *gilt.* A *lehnt* w *ab, falls* $accept((w, c_0)) = falsch$ *gilt. Ist* $accept((w, c_0))$ *nicht definiert, so läuft* A *bei Eingabe von* w *endlos, d.h. es gibt eine Folge von Nachfolgekonfigurationen die nicht in eine Endkonfiguration mündet.*

Die Menge $L(A) = \{w \in \Sigma^* \mid A \text{ akzeptiert } w\}$ *ist die von* A *akzeptierte Sprache. Die Familie aller Sprachen, die von* $t(n)$*–zeitbeschränkten* $f(n) -$ AIA*s akzeptiert werden, bezeichnen wir mit* $\mathscr{L}_{t(n)}(f(n) - \text{AIA})$.

Wie bei einem (nicht)deterministischen iterativen Array gibt es also auch bei einem AIA drei mögliche Fälle.

1. Der Automat erkennt das Wort nach einer endlichen Anzahl von Schritten. $(accept((w, c_0)) = wahr)$
2. Der Automat stellt nach endlich vielen Schritten fest, daß das Wort nicht zu der Sprache gehört. $(accept((w, c_0)) = falsch)$
3. $accept(w, c_0)$ ist nicht definiert. In diesem Fall sagen wir, der Automat hält nicht an. Das Wort liegt dann nicht in der von dem Automaten erkannten Sprache.

Natürlich sind in der Praxis nur zeitbeschränkte Automaten, bei denen der dritte Fall nicht auftreten kann, von Bedeutung.

6.2 Grundlegende Eigenschaften

Ein nichtdeterministisches iteratives Array ist ein Spezialfall des alternierenden iterativen Arrays mit $S_u = \emptyset$. Dies liefert:

Lemma 6.4
Für alle Zeitbeschränkungen $t : \mathbb{N}_0 \to \mathbb{N}$ *und alle Beschränkungen des Nichtdeterminismus* $f : \mathbb{N}_0 \to \mathbb{N}_0$ *gilt*

$$\mathscr{L}_{t(n)}(f(n) - \text{NIA}) \subseteq \mathscr{L}_{t(n)}(f(n) - \text{AIA}).$$

Die Sätze aus Kapitel 3 lassen sich auf alternierende iterative Arrays verallgemeinern. Zunächst ist es jedoch vorteilhaft, eine geeignete Normalform für alternierende iterative Arrays nachzuweisen.

Definition 6.5
Ein alternierendes iteratives Array heißt in Normalform, wenn für jedes Wort w und jede Wahl von Nachfolgekonfigurationen die Folge der nichtdeterministischen Konfigurationen $c_{t_0}, c_{t_1}, \ldots$ die folgende Bedingung erfüllt:

$$c_{t_i} \text{ ist } \begin{cases} \text{existentiell} & \text{falls } i \text{ gerade ist.} \\ \text{universell} & \text{falls } i \text{ ungerade ist.} \end{cases} \tag{6.1}$$

(D.h. existentielle und universelle nichtdeterministische Konfigurationen wechseln sich ab. Für deterministische Konfigurationen wird nicht festgelgt, ob sie existentiell oder universell sind.)

Lemma 6.6
Zu jedem alternierenden iterativen Array A gibt es ein alternierendes iteratives Array A' in Normalform, so daß $L(A') = L(A)$ gilt.

Ist außerdem A $t(n)$–zeitbeschränkt, so ist auch A' $t(n)$–zeitbeschränkt. Ist die Anzahl der nichtdeterministischen Schritte von A durch $f(n)$ beschränkt, so gilt dies auch für A'.

Beweis
A kann sich bei jedem nichtdeterministischen Schritt in maximal r Nachfolgekonfigurationen verzweigen. Die Ursprungszelle von A' hat ein Register, in dem sie eine Zahl von 1 bis r abspeichern kann. Dieses Register ist am Anfang leer. Außerdem enthält die Ursprungszelle von A' noch ein weiteres Register, in dem sie abspeichert, ob der nächste nichtdeterministische Schritt existentiell oder universell sein muß. Die Mengen S'_u und S'_e des Automaten A' werden anhand des dritten Registers definiert. (D.h. ist S die Zustandsmenge von A, so hat A' die Zustandsmenge $S \times \{\sqcup, 1, \ldots, r\} \times \{\exists, \forall\}$.)

A' hat den Ruhezustand $(q_0, \sqcup, \exists)$. In jedem nichtdeterministischen Schritt füllt A' das zweite Register mit einer nichtdeterministisch gewählten Zahl zwischen 1 und r und ändert das dritte Register von $\exists$ auf $\forall$ und umgekehrt.

Es ist klar, daß A' deterministische Schritte von A direkt simulieren kann. Wir betrachten nun einen nichtdeterministischen Schritt von A. Wir nehmen zunächst an, daß dieser nichtdeterministische Schritt von A existentiell ist. Für A' gibt es nun zwei Fälle:

Fall 1: A' befindet sich in einem existentiellen Zustand, d.h. das dritte Register von A' ist gleich $\exists$. In diesem Fall kann A' das Verhalten von A direkt simulieren.

Fall 2: A' befindet sich in einem universellen Zustand, d.h. das dritte Register von A' ist gleich $\forall$. In diesem Fall ist die Zahl im zweiten Register durch einen existentiellen Schritt von A' entstanden. A' benutzt die Wahl von A, die durch diese Zahl angegeben wird. (Hat A in diesem Schritt nur r' Möglichkeiten zur Verfügung und ist die Zahl im zweiten Register größer als r', so wählt A die Möglichkeit r'.)

Da die Zahl im zweiten Register durch einen universellen Schritt von A' erzeugt wurde, wird A' genau wie A nur dann akzeptieren, wenn alle Nachfolgekonfigurationen zum Akzeptieren führen. Somit hat A' den nichtdeterministischen Schritt von A deterministisch simuliert.

Außerdem rät A' universell eine neue Zahl für das zweite Register und ändert das dritte Register.

Entsprechend geht A' bei einem universellen Schritt von A vor.

Bei der obigen Betrachtung haben wir vorausgesetzt, daß das zweite Register von A' eine Zahl enthält. Wir müssen nun noch den Fall des ersten nichtdeterministischen Schrittes betrachten. In diesem Fall ist der Eintrag im zweiten Register gleich $\sqcup$. Es gibt zwei Möglichkeiten.

Fall 1: Der erste nichtdeterministische Schritt von A ist existentiell. In diesem Fall hat A' keine Probleme, diesen Schritt von A zu simulieren.

Fall 2: Der erste nichtdeterministische Schritt von A ist universell. In diesem Fall simuliert A' alle Wahlmöglichkeiten von A deterministisch auf r verschiedenen Spuren. A' wird nur akzeptieren, wenn alle Simulationen auf den r Spuren zu akzeptierenden Endzuständen von A führen. Außerdem rät A existentiell für jedes dieser Register eine Zahl zwischen 1 und r. (D.h. die Zustandsmenge von A' ist gleich $(S \times \{\sqcup, 1, \ldots, r\})^r \times \{\exists, \forall\}$.)

Damit ist gezeigt, daß sich die Zustände von A und A' nach jedem Zeitschritt entsprechen, d.h. $L(A') = L(A)$ und A und A' benötigen gleichviel Zeit für ihre Berechnung. Außerdem macht A' genau dann einen nichtdeterministischen Schritt, wenn A einen nichtdeterministischen Schritt ausführt. □

Wir können nun die Sätze aus Abschnitt 3.3 auf alternierende iterative Arrays übertragen.

Lemma 6.7 (Zeitreduktion)
Für jede Funktion $f : \mathbb{N}_0 \to \mathbb{N}_0$ für die Anzahl der nichtdeterministischen Schritte gelten die Beziehungen:

1. *Für jede Funktion* $t : \mathbb{N}_0 \to \mathbb{N}$ *mit* $t(n) \geq n+1$ *und jede Konstante* $c \in \mathbb{N}$ *gilt:*

$$\mathscr{L}_{t(n)+c}(f(n) - \text{AIA}) = \mathscr{L}_{t(n)}(f(n) - \text{AIA}) \tag{6.2}$$

(Additive Konstanten in der Zeitbeschränkung können weggelassen werden.)

2. *Für jede Funktion* $t : \mathbb{N}_0 \to \mathbb{N}$ *und jede Konstante* $c \in \mathbb{N}$ *gilt:*

$$\mathscr{L}_{n+ct(n)}(f(n) - \text{AIA}) = \mathscr{L}_{n+t(n)}(f(n) - \text{AIA}) \tag{6.3}$$

(Oberhalb von Realzeit können multiplikative Konstanten in der Zeitbeschränkung weggelassen werden.)

Beweis
Der Beweis verläuft analog zum Beweis von Lemma 3.11. Jede Zelle des „schnellen“ Automaten simuliert c Zellen des „langsamen Automaten. In den ersten n Schritten kann keine Beschleunigung stattfinden, da die Eingabe nur zeichenweise gelesen wird.

Ein zusätzliches Problem bereiten die Wechsel zwischen universellen und existentiellen Schritten. Eventuell muß der schnelle Automat in einem Schritt universelle und existenzielle Schritte des langsamen Automaten simulieren. Um dieses Problem zu lösen, nehmen wir an, daß der zu simulierende Automat in Normalform vorliegt. Auch der schnelle Automat wird in Normalform konstruiert.

Die Ursprungszelle des schnellen Automaten speichert genau wie im Beweis von Lemma 6.6 in einem zusätzlichen Register Zahlen, die nichtdeterministisch erzeugt wurden. Muß z.B. im Fall $c = 2$ mit einem existentiellen Schritt ein existentieller und ein universeller Schritt des langsamen Automaten simuliert werden, so kann der schnelle Automat den existenziellen Schritt direkt simulieren und die durch einen universellen Schritt entstandenen Zahlen im Zusatzregister benutzen, um den universellen Schritt zu simulieren. Wie im Beweis von Lemma 6.6 werden in jedem nichtdeterministischen Schritt neue Zahlen für das Zusatzregister erzeugt. □

Lemma 6.8 (Reduktion der nichtdeterministischen Schritte)
Für jede Funktion $t : \mathbb{N}_0 \to \mathbb{N}$ *mit* $t(n) \geq n+1$ *für die Zeitbeschränkung gelten die Beziehungen:*

1. *Für jede Funktion* $f : \mathbb{N}_0 \to \mathbb{N}_0$ *und jede Konstante* $c \in \mathbb{N}$ *gilt:*

$$\mathscr{L}_{t(n)}((f(n) + c) - \text{AIA}) = \mathscr{L}_{t(n)}(f(n) - \text{AIA}) \tag{6.4}$$

(Endlich viele nichtdeterministische Schritte können deterministisch simuliert werden.)

2. *Für jede Funktion* $f : \mathbb{N}_0 \to \mathbb{N}_0$ *und jede Konstante* $c \in \mathbb{N}$ *gilt:*

$$\mathscr{L}_{t(n)}(cf(n) - \text{AIA}) = \mathscr{L}_{t(n)}(f(n) - \text{AIA}) \tag{6.5}$$

(Jeder nichtdeterministische Schritt kann endlich viele nichtdeterministische Schritte simulieren.)

Beweis
Der Beweis verläuft analog zum Beweis von Lemma 3.12. Der Beweis von Teil 1 kann direkt übernommen werden.

Beim Beweis von Teil 2 des Lemmas 3.12 haben wir einen Keller, in dem die nichtdeterministischen Schritte des zu simulierenden $cf(n) - \text{NIA}$ gespeichert wurden, benutzt. Für alternierende iterative Arrays benötigen wir zwei Keller, einen für die existentiellen und einen für die universellen Schritte. Zusätzlich nehmen wir an, daß die alternierenden iterativen Arrays in Normalform sind. Dadurch wird garantiert, daß das $f(n) - \text{AIA}$ in seinen Kellern immer Anweisungen vorfindet, wie das $cf(n) - \text{AIA}$ zu simulieren ist. □

Mit Hilfe des Lemmas über die Normalform eines alternierenden iterativen Arrays können wir nun auch das Äquivalenzklassenlemma für AIAs beweisen.

Lemma 6.9
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ *eine Abbildung mit* $f(n) \leq n+1$. *Gilt* $L \in \mathscr{L}_{rt}(f(n) - \text{AIA})$, *so gibt es Konstanten* s *und* r, *so daß*

$$N(n, l, L) \leq (s^l)^{r^{f(n)}} \tag{6.6}$$

für alle $l, n \in \mathbb{N}$ *gilt.*

Beweis
Das alternierende iterative Array sei in Normalform. Bei jedem nichtdeterministischen Schritt verzweige es sich in *genau* r Berechnungspfade. (Dies kann immer ereicht werden, indem man einige Wahlmöglichkeiten doppelt benutzt.) Wie beim nichtdeterministischen iterativen Array hängt das Verhalten des Automaten nach $n - l$ Zeitschritten nur noch von den Zuständen der Zellen $-l - 1$ bis $l + 1$ in den $r^{f(n)}$ möglichen Berechnungspfaden ab. (Da das alternierende iterative Array normalisiert ist und in jedem nichtdeterministischen Schritt in genau r Berechnungspfade verzweigt, ist die Verteilung der universellen und existentiellen Schritte festgelegt, kann also das Verhalten unterschiedlicher Automaten nicht beeinflussen.)

Daher folgt dieselbe Schranke wie für nichtdeterministische iterative Arrays. □

Entsprechend können wir auch die anderen Varianten des Äquivalenzklassenlemmas übertragen.

Lemma 6.10
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine Abbildung mit $f(n) \leq n+1$. Gilt $L \in \mathscr{L}_{rt}(f(n)-\mathrm{AIA}_d)$, so gibt es Konstanten s und r, so daß

$$N(n,l,L) \leq (s^{l^d})^{rf(n)} \tag{6.7}$$

für alle $l, n \in \mathbb{N}$ gilt.

Lemma 6.11
Sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine Beschränkung der nichtdeterministischen Schritte und $t : \mathbb{N}_0 \to \mathbb{N}$ eine Zeitbeschränkung. Gilt $L \in \mathscr{L}_{t(n)}(f(n) - \mathrm{AIA})$, so gibt es Konstanten s und r, so daß

$$N(n,l,L) \leq (s^{l+t(n)-n})^{rf(n)} \tag{6.8}$$

für alle $l, n \in \mathbb{N}$ gilt.

6.3 Vergleich mit NIAs

Bis auf Lemma 4.18 und seine Folgerung 4.22 haben alle Beweise, daß eine Sprache L nicht in $\mathscr{L}_{rt}(f(n) - \mathrm{NIA})$ liegt, das Äquivalenzklassenlemma benutzt. Wir können sie daher auf alternierende iterative Arrays übertragen.

Zum Beispiel wird aus Satz 5.1 der Satz:

Satz 6.12
Seien $f, g : \mathbb{N}_0 \to \mathbb{N}_0$ Funktionen, für die $g(n) \leq f(n)$ gilt. Sei ferner $t : \mathbb{N}_0 \to \mathbb{N}$ eine beliebige Funktion.

Weiter gebe es eine Funktion $h : \mathbb{N} \to \mathbb{N}$ mit $h(n) \leq 2^{n-2}$. h erfülle die Voraussetzungen:

1. $\{(a^n b)^{h(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(\mathrm{IA})$.
2. $f((n+1)(h(n)+1)) = \Omega(\log(h(n)))$ *und* $g((n+1)(h(n)+1)) = o(\log(h(n)))$.
3. $t((n+1)(h(n)+1)) = O(n)$.

Dann gibt es eine Sprache L, die in $\mathscr{L}_{rt}(f(n) - \text{NIA})$, aber nicht in $\mathscr{L}_{n+t(n)}(g(n) - \text{AIA})$ liegt.

Dies bedeutet, daß man hier keine nichtdeterministischen Schritte einsparen kann, indem man einen alternierenden Automaten benutzt. Außerdem folgt die Hierarchie (Korollar 6.13):

Korollar 6.13 (Alternierende Hierarchien)
Unter den Voraussetzung von Satz 6.12 gilt:

$$\mathscr{L}_{n+t(n)}(g(n) - \text{AIA}) \subsetneqq \mathscr{L}_{n+t(n)}(f(n) - \text{AIA}).$$

Insbesondere gilt

$$\mathscr{L}_{rt}(g(n) - \text{AIA}) \subsetneqq \mathscr{L}_{rt}(f(n) - \text{AIA}).$$

Trotzdem sind alternierende iterative Arrays bei gleicher Beschränkung der nichtdetermistischen Schritte echt mächtiger als nichtdeterministische iterative Arrays. Um dies zu beweisen, betrachten wir die Abschlußeigenschaften von AIAs bezüglich boolescher Operationen.

Lemma 6.14
Für alle Zeitbeschränkungen $t : \mathbb{N}_0 \to \mathbb{N}$ und alle Beschränkungen des Nichtdeterminismus $f : \mathbb{N}_0 \to \mathbb{N}_0$ ist die Sprachfamilie $\mathscr{L}_{t(n)}(f(n) - \text{AIA})$ abgeschlossen unter booleschen Operationen.

Beweis
Seien A_1 und A_2 zwei $t(n)$–zeitbeschränkte Automaten, die $f(n)$ nichtdeterministische Schritte ausführen.

Ein Automat A', der die Vereinigung $L(A_1) \cup L(A_2)$ zweier Sprachen erkennt, entscheidet in einem ersten existentiellen Schritt, ob er A_1 oder A_2 simuliert. A' braucht also einen Zeitschritt und einen nichtdeterministischen Schritt mehr als A_1 und A_2. Nach den Lemmata 6.7 und 6.8 liegt daher $L(A_1) \cup L(A_2) = L(A')$ in $\mathscr{L}_{t(n)}(f(n) - \text{AIA})$.

Analog folgt $L(A_1) \cap L(A_2) \in \mathscr{L}_{t(n)}(f(n) - \text{AIA})$.

Ist $A = (S_e, S_u, \Sigma, F_a, F_r, \delta, \delta_0, q_0, \sqcup)$ ein alternierendes iteratives Array, so definieren wir $\bar{A} = (S_u, S_e, \Sigma, F_r, F_a, \delta, \delta_0, q_0, \sqcup)$.

Es gilt $\overline{L(A)} = L(\bar{A})$. Dies folgt direkt aus den de Morgan–Regeln

$$\neg(x \vee y) = \neg x \wedge \neg y \quad \text{und} \quad \neg(x \wedge y) = \neg x \vee \neg y.$$

□

Jetzt folgt leicht, daß alternierende iterative Arrays mächtiger sind als nichtdeterministische iterative Arrays.

Satz 6.15
Für eine monoton wachsende, unbeschränkte Funktion $h : \mathbb{N} \to \mathbb{N}$ *mit* $h(n) = n^{o(1)}$ *sei* $\{(a^n b)^{h(n)} | n \in \mathbb{N}\}$ *eine Sprache aus* $\mathscr{L}_{rt}(\mathrm{IA})$. *Eine Funktion* $f : \mathbb{N}_0 \to \mathbb{N}_0$ *erfülle die Voraussetzung*

$$f((h(n)+1)(n+1)) = \Theta(\log(h(n))).$$

Dann gilt

$$\mathscr{L}_{rt}(f(n) - \mathrm{NIA}) \subsetneqq \mathscr{L}_{rt}(f(n) - \mathrm{AIA}).$$

Beweis
Gemäß den Voraussetzungen an f und Satz 4.22 ist $\mathscr{L}_{rt}(f(n) - \mathrm{NIA})$ nicht abgeschlossen bezüglich Komplementbildung.

Aber nach Lemma 6.14 ist $\mathscr{L}_{rt}(f(n) - \mathrm{AIA})$ abgeschlossen bezüglich Komplementbildung. Es folgt

$$\mathscr{L}_{rt}(f(n) - \mathrm{NIA}) \neq \mathscr{L}_{rt}(f(n) - \mathrm{AIA}).$$

Nach Lemma 6.4 gilt jedoch

$$\mathscr{L}_{rt}(f(n) - \mathrm{NIA}) \subseteq \mathscr{L}_{rt}(f(n) - \mathrm{AIA}).$$

□

Anhang A

Wichtige Funktionsklassen

A.1 $\{(a^n b)^{f(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(IA)$

Eine wichtige Voraussetzung für Satz 2.19 und damit für eine große Anzahl weiterer Sätze war, daß eine Funktion f des $f(n)-$NIA die Bedingung $\{(a^n b)^{f(n)}\} \in \mathscr{L}_{rt}(IA)$ erfüllt. Im folgenden werden wir zeigen, daß diese Voraussetzung für eine große Klasse von Funktionen erfüllt ist.

Zunächst reduzieren wir das Problem auf ein bereits besser untersuchtes Problem.

Lemma A.1
Gilt für eine monoton wachsende Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(n) \leq n$, daß $\{a^{f(n)} b^n \mid n \in \mathbb{N}\}$ in $\mathscr{L}_{lt}(CA)$ liegt, so gilt auch $\{(a^n b)^{f(n)} \mid n \in \mathbb{N}\} \in \mathscr{L}_{rt}(IA)$.

Beweis
Wir nehmen zunächst an, daß $f(n) \geq 9$ gilt. Ein IA, das $(a^n b)^{f(n)}$ erkennt, arbeitet nach dem folgenden Schema:

1. Die ersten n Zeichen a werden in den Zellen 0 bis $n-1$ des iterativen Arrays abgespeichert. Dies ist analog zum Speichern in einer FiFo–Schlange.

2. Nun kann das iterative Array überprüfen, ob die Eingabe von der Form $(a^n b)^*$ ist.

3. Zusätzlich speichert das iterative Array jedes gelesene b ab. Die b werden in die Zellen n, $n+1$, usw. geschrieben.

4. Nach jedem b rät das iterative Array deterministisch, daß noch 8 weitere b kommen. Unter dieser Annahme beginnt es mit der Simulation eines

Zellularautomaten, der $a^n b^{f(n)}$ erkennt. (Siehe auch Beweis von Satz 5.7.)

5. Wegen $f(n) \leq n$ braucht das iterative Array höchstens $8n$ Zeitschritte für die Simulation des Zellularautomatens auf $a^n b^{f(n)}$ ($4n$ Zeitschritte für die Synchronisation und $4n$ Zeitschritte für die Simulation.) Außerdem weiß das iterative Array nach $8n$ Zeitschritten, ob die Annahme, daß noch genau 8 b folgen, korrekt war. Das iterative Array akzeptiert nur, wenn der Zellularautomat akzeptiert und die Annahme über die Anzahl der b korrekt war.

6. Da das iterative Array maximal 8 Zellularautomaten gleichzeitig simulieren muß, kann es dies auf 8 deterministischen Spuren tun.

Wir müssen nun noch den Fall $f(n) < 9$ behandeln.

Da f monoton ist, gibt es ganze Zahlen $0 = n_0, n_1, \ldots, n_8 \in \mathbb{N}_0 \cup \{\infty\}$ mit $f(n) = i$ für $n_{i-1} < n \leq n_i$ $(i = 1, \ldots, 8)$. Das iterative Array kann in seiner endlichen Kontrolle bis n_8 zählen und auf diese Art entscheiden, ob $f(n) < 9$ ist und welchen Wert $f(n)$ in diesem Fall hat. (Im Fall, daß $n_i = \infty$ und $n_{i-1} < \infty$ gilt, muß das iterative Array nur bis n_{i-1} zählen. Für alle $n > n_{i-1}$ gilt dann $f(n) = i$.) □

A.2 $\{a^{f(n)}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{lt}(CA)$

Im vorhergehenden Abschnitt haben wir gesehen, daß Funktionen, für die $\{a^{f(n)}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{lt}(CA)$ erfüllt ist, für die Untersuchung von iterativen Arrays wichtig sind.

Für eine monoton wachsende Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(n) \geq n$ definieren wir die Funktion $f^{-1} : \mathbb{N} \to \mathbb{N}$ durch $f^{-1}(n) = \max\{m \mid f(m) \leq n\}$. Es gilt:

Satz A.2
Ist $f \in \mathscr{F}$, so gilt $\{a^{f^{-1}(n)}b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{lt}(CA)$. ($\mathscr{F}$ bezeichnet die Klasse der zeitkonstruierbaren Funktionen (siehe Definition 3.9).)

Beweis
Um $\{a^{f^{-1}(n)}b^n \mid n \in \mathbb{N}\}$ zu erkennen, startet der CA mit der Zelle an der Grenze zwischen den a und den b einen Zeitkonstruierer für f. In jedem Zeitschritt verbraucht die Grenzzelle ein b und bei jeder Marke des Zeitkonstruierers wird ein a verbraucht.

Der Automat erkennt, wenn alle a und alle b verbraucht wurden. Dann wird ein Erfolgssignal an die Randzelle geschickt. Sind die b verbraucht, bevor die a verbraucht sind (d.h. die Anzahl der a ist kleiner als $f^{-1}(n)$) oder

sind die a verbraucht, bevor alle b verbraucht wurden (d.h. die Anzahl der a ist größer als $f^{-1}(n)$), so wird nicht akzeptiert. □

Die Klasse $\mathscr{F}$ der zeitkonstruierbaren Funktion wird in [33] ausführlich untersucht. Unter anderem wird dort bewiesen, daß k^n, k^2 ($k \in \mathbb{N}$) und $n!$ in $\mathscr{F}$ liegen und daß $\mathscr{F}$ abgeschlossen unter Addition, Multiplikation und Verkettung von Funktionen ist. Also erfüllen zum Beispiel $\log(n), \sqrt{n}$ und Verkettungen dieser Funktionen die Hierarchiesätze.

Anhang B

Guess–and–Check Modelle

Das Guess–and–Check Modell ist bei Turing–Maschinen die übliche Methode eingeschränkten Nichtdeterminismus zu definieren. Wie schon in Bemerkung 2.17 erwähnt, darf die Definition 2.16 von eingeschränktem Nichtdeterminismus nicht mit dem Guess–and–Check Modell verwechselt werden.

In diesem Anhang untersuchen wir das Verhältnis von Modellen mit eingeschränktem Nichtdeterminismus in Sinne von Definition 2.16 zu dem Guess–and–Check Modell aus [9].

Definition B.1 (Guess–and–Check Modell)
Es sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ eine Funktion und $\mathcal{C}$ eine Komplexitätsklasse. Eine Sprache L liegt in $GC(f(n), \mathcal{C})$, wenn es eine Sprache $L' \in \mathcal{C}$ und eine Konstante $c > 0$ gibt, so daß

$$x \in L \iff \exists y \in \{0,1\}^{cf(|x|)} \text{ mit } y \bullet x \in L'$$

gilt. Dabei ist $\bullet$ ein Trennzeichen, das weder in x noch in y vorkommt.

Man kann viele wichtige Sprachfamilien durch das Modell GC charakterisieren. Zum Beispiel gilt

$$\mathcal{NP} = GC(n^{O(1)}, \mathcal{P}).$$

Es gibt auch GC–Modelle, die nicht durch entsprechende nichtdeterministische Automaten beschrieben werden können.

Beispiel B.2
Betrachte eine Turing–Maschine mit einem random–access *Eingabeband. (Die Turing–Maschine hat ein zusätzliches Speicherband, auf dem sie eine Binärzahl speichern kann. Sie greift immer auf das Zeichen der Eingabe zu, das durch diese Binärzahl beschrieben wird.) Eine alternierende Turing–Maschine mit*

random–access Eingabeband kann daher die Eingabe in $O(\log(n))$ Zeitschritten verarbeiten.

Die Klasse aller alternierenden Turing–Maschinen, die ein random-access *Eingabeband haben, durch $O(\log(n))$ zeitbeschränkt sind, mit einem universellen Zustand beginnen und höchstens k Alternierungen (Wechsel zwischen universellen und existentiellen Zuständen) ausführen, wird in [9] mit Π_k^B bezeichnet.*

Die Bedeutung der $GC(f(n), \Pi_k^B)$ ergibt sich durch die enge Verbundenheit mit kombinatorischen Optimierungsproblemen. (Die Klassen $GC(\log^2, \Pi_2^B)$ und $GC(\log^2, \Pi_2^B)$ charakterisieren die Optimierungsklassen $LOGSNP$ und $LOGNP$, die in [36] eingeführt werden.)

Man beachte, daß die Anzahl der geratenen Bits größer als die Anzahl der von der Turing–Maschine ausgeführten Schritte ist!

Man muß jedoch noch zusätzliche Bedingungen an die Funktion f aus Definition B.1 stellen, um „vernünftige“ Komplexitätsklassen zu erhalten.

Beispiel B.3
Sei M eine beliebige Teilmenge von $\mathbb{N}$ und

$$f_M(n) = \begin{cases} 1 & n \in M \\ 0 & n \notin M \end{cases}.$$

Mit $\mathcal{REG}$ bezeichnen wir die Familie der regulären Sprachen.

Dann liegt die Sprache $L_M = \{0^n \mid n \in M\}$ in $GC(f_M(n), \mathcal{REG})$. (Es genügt, in Definition B.1 $c = 1$ und $L' = 0 \bullet 0^*$ zu wählen.)

Dies bedeutet, daß das gesamte Erkennen durch die Funktion f_M geschieht und „eigentlich“ nicht von dem zugrundeliegenden Automatenmodell geleistet wird.

Insbesondere können auch nicht rekursiv aufzählbare Mengen von einem GC–Modell erkannt werden, wenn die Funktion f nicht berechenbar ist. Im Gegensatz dazu sind alle Familien $\mathscr{L}_{t(n)}(f(n) - \text{NIA})$ in der Menge der rekursiv aufzählbaren Sprachen enthalten; unabhängig von der Wahl von t und f.

Ein weiterer Nachteil des GC-Modells ist, daß aus $f'(n) \leq f(n)$ nicht $GC(f'(n), \mathcal{C}) \subseteq GC(f(n), \mathcal{C})$ folgt. (Es ist nicht einmal sicher, daß f' von $GC(f(n), \mathcal{C})$ berechnet werden kann.) Man beachte, daß die entsprechende Inklusion für $f(n) - \text{NIA}$ trivial ist.

Wie das obige Beispiel zeigt, liegt eine Sprache aus $GC(f(n), \mathscr{L}_{rt}(\text{IA}))$ im allgemeinen nicht in $\mathscr{L}_{rt}(f(n) - \text{NIA})$. (Die Sprache kann sogar nichtberechenbar sein.) Die umgekehrte Inklusion ist jedoch richtig.

Satz B.4
Sei $t : \mathbb{N}_0 \to \mathbb{N}$ eine monoton wachsende Funktion. Für alle Funktionen $f : \mathbb{N}_0 \to \mathbb{N}_0$ gilt

$$\mathscr{L}_{n+t(n)}(f(n) - \text{NIA}) \subseteq GC(f(n), \mathscr{L}_{n+t(n)}(\text{IA})).$$

Beweis
Sei $L \in \mathscr{L}_{n+t(n)}(f(n) - \text{NIA})$ und A ein $f(n) - \text{NIA}$, das L in $n + t(n)$ Zeitschritten erkennt. In jedem nichtdeterministischen Schritt verzweigt sich A in höchstens r Berechnungspfade.

Wir setzen $c = \log(r)$. Nun definieren wir ein deterministisches iteratives Array A', das auf Wörtern der Form $y \bullet x$ arbeitet.

1. A' faßt jeweils $\log(r)$ Bits von y zu einer Zahl zwischen 1 und r zusammen. Diese Zahlen speichert A' in einem Keller ab.

2. Nach dem $\bullet$ Zeichen beginnt A' mit der Simulation von A. Jeder nichtdeterministische Schritt von A wird von A' deterministisch simuliert, indem A' eine Zahl vom Keller liest und die entsprechende Wahlmöglichkeit von A ausführt.

3. A' akzeptiert nur, wenn die Simulation von A zu einem akzeptierenden Endzustand führt.

Es ist klar, daß A' nur Wörter $y \bullet x$ mit $x \in L(A)$ akzeptieren kann. Umgekehrt gibt es jedoch für jedes Wort $x \in L(A)$ eine Folge von Wahlmöglichkeiten, nach der A akzeptiert. Für das entsprechende Wort y akzeptiert A' dann $y \bullet x$. Außerdem benötigt A' höchstens $|x| + 1 + |y| + t(|y|) \leq |y \bullet x| + t(|y \bullet x|)$ Zeitschritte.

Also erfüllen $c = \log(r)$ und $L' = L(A')$ die Definition B.1, d.h. $L \in GC(f(n), \mathscr{L}_{n+t(n)}(\text{IA}))$. □

Wie bereits oben gesagt, ist $GC(f(n), \mathscr{L}_{n+(n)}(\text{IA})$ im allgemeinen nicht in $\mathscr{L}_{n+t(n)}(f(n) - \text{NIA})$ enthalten, da das GC-Modell f vorgegeben bekommt, während der $f(n) - \text{NIA}$ f erst berechnen muß.

Die Gleichheit der beiden Modelle können wir daher nur für Linearzeit und berechenbare Funktionen f beweisen.

Satz B.5
Es sei $f : \mathbb{N}_0 \to \mathbb{N}_0$ in Linearzeit raumberechenbar (siehe Definition 5.5). Dann gilt

$$\mathscr{L}_{lt}(f(n) - \text{NIA}) = GC(f(n), \mathscr{L}_{lt}(\text{IA})).$$

Beweis
$\mathscr{L}_{lt}(f(n) - \text{NIA}) \subseteq GC(f(n), \mathscr{L}_{lt}(\text{IA}))$ haben wir bereits in Satz B.4 gezeigt.

Um die umgekehrte Inklusion zu beweisen, betrachten wir eine Sprache $L \in GC(f(n), \mathscr{L}_{lt}(\text{IA}))$. Es gibt eine Konstante c und ein Linearzeit deterministisches iteratives Array A, so daß

$$x \in L \iff \exists y \in \{0,1\}^{cf(|x|)} \text{ mit } y \bullet x \in L(A)$$

gilt.

Wir konstruieren ein $f(n) - \text{NIA}$ A', das L in Linearzeit erkennt. A' arbeitet nach dem folgenden Schema.

1. Als erstes liest A' seine Eingabe x ein und speichert sie in einer FiFo-Schlange ab.

2. A' berechnet $f(|x|)$.

3. Danach führt A' $f(n)$ nichtdeterministische Schritte aus. In jedem dieser Schritte rät A' genau c Bits. Diese werden in einem Keller abgespeichert. Den Inhalt des Kellers bezeichnen wir mit y.

4. Nun simuliert A' den deterministischen Automaten A auf $y \bullet x$.

Es ist klar, daß A' ein Wort x genau dann akzeptiert, wenn es ein $y \in \{0,1\}^{cf(|x|)}$ mit $y \bullet x \in L(A)$ gibt. Nach Konstruktion ist die Anzahl der nichtdeterministischen Schritte von A' durch $f(n)$ beschränkt. Wir müssen nur überprüfen, ob A' in Linearzeit arbeitet.

Da $f(n)$ in Linearzeit berechenbar ist, benötigt A' für die Schritte 1 und 2 $O(n)$ Zeitschritte. Außerdem gilt $f(n) = O(n)$. Für den Schritt 3 benötigt A' genau $f(n)$ Zeitschritte. Die Simulation von A benötigt $2|y \bullet x|$ Zeitschritte. Insgesamt läuft A' daher nur $O(n) + f(n) + 2(f(n) + 1 + n) = O(n)$ Zeitschritte. Nach Lemma 3.11 liegt $L(A')$ daher in $\mathscr{L}_{lt}(f(n) - \text{NIA})$. □

Literaturverzeichnis

[1] Berstel, J.: *Transductions and Context–Free Languages.* Teubner Studienbücher Informatik, Stuttgart, 1979.

[2] Beyer, W.T.: *Recognition of topological invariants by iterative arrays.* Technical Report, MIT, Cambridge, Proj. MAC, 1969.

[3] Borodin, A.: *Computational Complexity and the Existence of Complexity Gaps.* Journal of the ACM, 18:315–322, 1972.

[4] Buchholz, T., A. Klein und M. Kutrib: *One guess one-way cellular arrays.* In: L. Brim et al. (Herausgeber): *Mathematical Foundations of Computer Science*, Band 1450 der Reihe *LNCS*, 807–815, 1998.

[5] Buchholz, T., A. Klein und M. Kutrib: *On time reduction and simulation in cellular spaces.* International Journal of Computer Mathematics, 71:459–474, 1999.

[6] Buchholz, T., A. Klein und M. Kutrib: *On tally languages and generalized interacting automata.* In: Rosenburg, G. und W. Thomas (Herausgeber): *Developments in Language Theory*, World Sientific, Singapure, erscheint 2000.

[7] Buchholz, T. und M. Kutrib: *Some relations between massively parallel arrays.* Parallel Computing, 23:1643–1662, 1997.

[8] Buss, J. und J. Goldsmith: *Nondeterminism within P.* SIAM Journal on Computing, 22:560–572, 1993.

[9] Cai, L. und J. Chen: *On the amount of nondeterminism and the power of verifying.* SIAM Journal on Computing, 26:733–750, 1997.

[10] Chandra, A.K., D.C. Kozen und L.J. Stockmeyer: *Alternation.* Journal of the ACM, 28:114–133, 1981.

[11] Chang, J.H., O.H. Ibarra und M.A. Palis: *Parallel parsing on a one-way arrays of finite–state machines.* IEEE Transactions on Computers, C-36:64–75, 1987.

[12] Cole, S.N.: *Real-time computation by n-dimensional iterative arrays of finite-state machines.* In: *7th Annual Symposium on Switching and Automata Theory (IEEE)*, 53–77, 1966.

[13] Čulik II, K. und S. Yu: *Iterative tree automata.* Theoretical Computer Science, 32:227–247, 1984.

[14] Díaz, J. und J. Torán: *Classes of Bounded Nondeterminism.* Mathematical Systems Theory, 23:21–32, 1990.

[15] Fischer, P.C.: *Generation of primes by a one–dimensional realtime iterative array.* Journal of the ACM, 12:388–394, 1965.

[16] Fischer, P.C. und C.M. Kintala: *Real–time computations with restricted nondeterminism.* Mathematical Systems Theory, 12:219–231, 1979.

[17] Fischer, P.C. und C.M. Kintala: *Refining nondeterminism in relativized complexity classes.* SIAM Journal on Computing, 13:329–337, 1984.

[18] Ginsburg, S., S. Greibach und J. Hopcroft: *Pre-AFL.* Memoirs of the AMS, 87:41–51, 1969.

[19] Goldsmith, J., M.A. Levy und M. Mundhenk: *Limited Nondeterminism.* SIGACT Nems, 27:20–29, 1996.

[20] Greibach, S. und J. Hopcroft: *Independence of AFL operations.* Memoirs of the AMS, 87:33–40, 1969.

[21] Hennie, F.C.: *One–Tape, Off–Line Turing Machine Computations.* Information and Control, 8:553–578, 1965.

[22] Hillis, W.D.: *The Connection Machine.* MIT Press, Cambridge Massachusetts, 1985.

[23] Ibarra, O.H. und T. Jiang: *On one–way cellular arrays.* SIAM Journal on Computing, 16:1135–1154, 1987.

[24] Ibarra, O.H. und T. Jiang: *Relating the power of cellular arrays to their closure Properties.* Theoretical Computer Science, 57:225–238, 1988.

[25] Ibarra, O.H. und M.A. Palis: *Some results concerning linear iterative (systolic) arrays.* Journal of Parallel and Distributed Computing, 2:182–218, 1985.

[26] Ibarra, O.H. und M.A. Palis: *Two–dimensional iterative arrays: Characterisations and applications.* Theoretical Computer Science, 57:47–86, 1988.

[27] Immerman, N.: *Nondeterministic Space is Closed Under Complementation.* SIAM Journal on Computing, 17:935–938, 1988.

[28] Kintala, C.M. und D. Wotschke: *Amounts of nondeterminism in finite automata.* Acta Informatica, 13:199–204, 1984.

[29] Kleene, S.C.: *Representation of events in nerve nets and finite automata.* In: *Automata Studies*, Princeton Univ. Press, 3–42, Princeton, N.J., 1956.

[30] Knuth, D.E.: *The Art of Computer Programming*, Band 2: Seminumerical Algorithms. Addison–Wesley, 1981.

[31] Martin, B.: *Efficient Unidimensional Universal Cellular Automaton.* In: Havel, I.M. und V. Koubek (Herausgeber): *Mathematical Foundations of Computer Science*, Band 629 der Reihe *LNCS*, 374–382, 1992.

[32] Mazoyer, J.: *A six–state minimal time solution to the firing squad synchronisation problem.* Theoretical Computer Science, 50:183–238, 1987.

[33] Mazoyer, J. und V. Terrier: *Signals in one dimensional cellular automaton.* Theoretical Computer Science, 217:53–80, 1999.

[34] Myhill, J.: *Finite automata and the representation of events*, Band WADD TR 57-624, 112–137. Wright Patterson AFB, Ohio, 1957.

[35] Neumann, J. von: *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, 1966. Herausgegeben und vervollständigt von A.W. Burks.

[36] Papadimitriou, C.H. und M. Yannakakis: *On limited nondeterminism and complexity of the V-C dimension.* In: *8th Structure in Complexity Theory Conference (IEEE)*, 12–18, 1993.

[37] Preston Jr., K. und M.J.B. Duff: *Modern Cellular Automata, Theory and Applications.* Plenum Press, New York, 1984.

[38] Reischuk, K.R.: *Einführung in die Komplexitätstheorie*, Band 1: Grundlagen (Maschinenmodelle, Zeit- und Platzkomplexität, Nichtdeterminismus). Teubner, Stuttgart, 1999.

[39] Salomaa, K. und S. Yu: *Limited nondeterminism for pushdown automata.* Bulletin of the EATCS, 50:186–193, 1993.

[40] Salomaa, K. und S. Yu: *Measures of nondeterminism for pushdown automata.* Journal of Computer and System Sciences, 49:362–374, 1994.

[41] Seidel, S.R.: *Language Recognition and the Synchronization of Cellular Automata.* Doktorarbeit, University of Iowa, Iowa City, 1979.

[42] Seiferas, J.I.: *Iterative arrays with direct central control.* Acta Informatica, 8:177–192, 1977.

[43] Seiferas, J.I.: *Linear–time computation by nondeterministic multidimensional iterative arrays.* SIAM Journal on Computing, 6:487–504, 1977.

[44] Smith III, A.R.: *Cellular automata complexity trade–offs.* Information and Control, 18:466–482, 1971.

[45] Szelepcsényi, R.: *The Method of Forced Enumeration for Nondeterministic Automata.* Acta Informatica, 26:279–284, 1988.

[46] Toffoli, T. und N. Margolus: *Cellular Automata Machines.* MIT Press, Cambridge Massachusetts, 1988.

[47] Turing, A.M.: *On computational numbers, with an application to the Entscheidungsproblem.* Proceedings of the London Mathematical Society, Series 2, 42:231–265, 1936.

[48] Vollmar, R.: *Algorithmen in Zellularautomaten.* Teubner, Stuttgart, 1979.

[49] Weimar, J.R.: *Cellular automata for reaction-diffusion systems.* Parallel Computing, 23:1699–1715, 1997.