Die Berechnungsstärke von Forgetting-Automaten

Inauguraldissertation
zur Erlangung des akademischen Grades
„Doktor der Naturwissenschaften“

eingereicht beim
Fachbereich Mathematik und Informatik, Physik, Geographie
der Justus-Liebig-Universität Gießen

von
Jens Glöckler
geboren in Wetzlar

Gießen, 2008

D 26

Dekan:	Prof. Dr. Bernd Baumann (Justus-Liebig-Universität Gießen)
Gutachter:	Prof. Dr. Martin Kutrib (Justus-Liebig-Universität Gießen)
	Martin Plátek, CSc. (Karls-Universität Prag)

Datum der Disputation: 05. September 2008

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein Meilenstein in der Geschichte der theoretischen Informatik sind die Arbeiten des Linguisten Noam Chomsky (unter anderem [3, 4]) aus den Fünfziger Jahren des vorigen Jahrhunderts, in denen verschiedene Typen von *Grammatiken* definiert und ihre generativen Mächtigkeiten untersucht werden. Das langfristige Ziel seiner Arbeit ist es, natürliche Sprachen möglichst durch formale, mathematisch handhabbare Ersetzungssysteme darzustellen. Grammatiken sind solche Systeme, die mithilfe von *Regeln* oder *Produktionen* aus einem vorgegebenen *Startsymbol* Wörter über einer Menge von Zeichen, den *Terminalsymbolen*, erzeugen. Als Hilfsmittel werden dabei *Nichtterminalsymbole* verwendet, auf die in den entstehenden Zwischenstadien, den *Satzformen*, solange Regeln angewandt werden, bis schließlich ein Wort entsteht, das nur noch aus Terminalsymbolen besteht. In natürlich-sprachlichen Beispielen können grammatische Wortarten wie *Substantiv* oder *Verb* als derartige Nichtterminalsymbole fungieren. Hierbei ist zu beachten, dass ein Wort der natürlichen Sprache in der Grammatik als ein einzelnes Zeichen abgebildet werden kann. Ein aus den einzelnen Zeichen zusammengesetztes Wort der Grammatik entspricht somit einem vollständigen Satz der natürlichen Sprache. In der mathematischen Formalisierung erzeugt die Grammatik in einem Ableitungsprozess jeweils ein Wort. Die Menge aller ableitbaren Wörter ist die von der Grammatik erzeugte *Sprache*. Eine solche formale Sprache ist lediglich als eine Menge von Wörtern über dem zugrunde liegenden Alphabet definiert und ist im Folgenden nicht als eine natürliche Sprache aufzufassen.

Die Familien der Sprachen, die von den Chomsky-Grammatiken erzeugt werden, ergeben eine echte Hierarchie, die nach ihrem Begründer *Chomsky-Hierarchie* genannt wird:

1. REG (reguläre Sprachen),
2. CFL (kontextfreie Sprachen),

3. CSL (kontextsensitive Sprachen),
4. RE (rekursiv aufzählbare Sprachen).

Dabei ist jede Sprachfamilie jeweils eine echte Teilmenge der nachfolgenden und somit von allen weiteren Familien. Aus Sicht der Linguistik sind dabei insbesondere kontextfreie und kontextsensitive Sprachen von Interesse. Der Begriff *kontextfrei* gibt an, dass bei den Ersetzungsregeln der entsprechenden Grammatiken jeweils nur ein Nichtterminalsymbol durch ein neues Teilwort zu ersetzen ist. Es spielt also keine Rolle, in welchem Kontext das Symbol anzutreffen ist, das heißt, welche Teilwörter sich auf seiner linken beziehungsweise rechten Seite befinden. Bei kontextsensitiven Grammatiken hingegen kann dieser Kontext beachtet werden, und Nichtterminale können somit in unterschiedlichen Zusammenhängen unterschiedlich ersetzt werden.

Es hat sich im Laufe der Zeit herausgestellt, dass kontextfreie Grammatiken für die Beschreibung der in der Praxis auftretenden Phänomene nicht ausreichen (Zitat aus [7, Seite 24]):

„Indeed, the world seems to be non-context-free . . . “

Kontextsensitive Grammatiken sind aus Sicht der Linguistik hinreichend ausdrucksstark, sie sind jedoch leider aus praktischen Gesichtspunkten ungeeignet. Algorithmen, die für ein gegebenes Wort entscheiden, ob es in einer kontextsensitiven Sprache enthalten ist, haben im Allgemeinen eine (bezüglich der Länge des Wortes) exponentielle Laufzeit. Daher sind diese Algorithmen schon bei relativ geringer Wortlänge praktisch nicht mehr einsetzbar – die Laufzeit wächst schnell zu mehreren Jahren, Jahrhunderten und so weiter. Für die Frage, ob eine gegebene kontextsensitive Grammatik mindestens ein Wort erzeugen kann, also für das sogenannte *Leerheitsproblem*, existiert sogar – unabhängig von der Laufzeit – überhaupt kein Algorithmus.

Anstatt von kontextfreien zu kontextsensitiven Produktionsregeln überzugehen, wird daher auch versucht, die Anwendung der kontextfreien Regeln zu reglementieren. Obwohl dies auf den ersten Blick den Eindruck einer Schwächung der generativen Mächtigkeit erwecken könnte, werden damit tatsächlich ausdrucksstärkere Modelle geschaffen. Dies wird bei *Grammatiken mit kontrollierten Ableitungen* auf unterschiedliche Weise realisiert. So kann beispielsweise bei *regularly controlled grammars with / without appearance checking* festgelegt werden, dass die Regeln nur in einer bestimmten Reihenfolge angewandt werden dürfen, die durch eine reguläre Sprache über der Menge der Regeln festgelegt wird. Durch das optionale *appearance checking* kann zudem bestimmt werden, dass zunächst alle Regeln einer bestimmten Form angewandt werden müssen, bevor zu einer anderen Regel übergegangen werden darf. Dieses Konzept wird auch bei *matrix grammars*

with appearance checking verwendet. Hier werden Folgen von kontextfreien Produktionsregeln in den sogenannten *Matrizen* zusammengefasst und die Anwendungen der Regeln auf die so gegebene Reihenfolge beschränkt; die Auswahl der anzuwendenden Matrix ist jedoch frei. Ein anderer Ansatz ist es, nicht die Reihenfolge der anzuwendenden Regeln festzulegen, sondern Bedingungen an die Satzformen zu stellen, die während des Ableitungsprozesses entstehen. So wird bei *conditional grammars* jede Produktionsregel mit einer regulären Menge (über den Nichtterminalen und Terminalen der Grammatik) verbunden. Eine solche Regel kann nur angewandt werden, wenn die aktuelle Satzform in der angegebenen Menge enthalten ist. Bei *semi-conditional grammars* werden die Regeln stattdessen mit erlaubten beziehungsweise verbotenen Kontexten versehen, die in einer Satzform auftreten müssen beziehungsweise nicht auftreten dürfen. Diese sind ein Spezialfall der *conditional grammars*, wie auch die *random context grammars*, bei denen die Kontexte sogar nur aus Nichtterminalzeichen bestehen. Weitere Typen von Grammatiken mit kontrollierten Ableitungen führen einen gewissen Grad von Parallelität im Ableitungsprozess ein oder steuern die Ableitungen durch *Indizes* und definieren so ebenfalls Sprachfamilien, die oberhalb der kontextfreien Sprachen angesiedelt sind (siehe beispielsweise [1, 15, 16, 24, 37, 39, 54]).

Ein völlig anderer Ansatz wurde 1969 von Solomon Marcus in [38] mit dem Konzept der *contextual grammars* eingeführt, die einen grundsätzlich anderen Aufbau als die Chomsky-Grammatiken haben. Bei Letzteren werden getrennte Mengen von Terminal- und Nichtterminalsymbolen unterschieden. Aus einem ausgezeichneten Nichtterminalzeichen, dem *Startsymbol* oder *Axiom*, werden anhand der gegebenen Produktionsregeln solange Satzformen gebildet, bis ein Wort erreicht wird, das vollständig aus Terminalsymbolen besteht. Bei den contextual grammars hingegen wird eine Menge von Axiomen vorgegeben. Diese Axiome sind aus Zeichen des gleichen Alphabets wie auch die zu erzeugenden Wörter aufgebaut und sind daher selbst bereits Wörter aus der erzeugten Sprache. Mithilfe einer Menge von Kontexten und einer Auswahlfunktion werden dann in bestehenden Wörtern die Positionen ausgewählt, an denen ein Kontext eingefügt und somit ein neues, längeres Wort gebildet wird. In den vergangenen Jahrzehnten wurden zudem verschiedene Einschränkungen der allgemeinen Form dieser Grammatiken eingeführt (siehe beispielsweise [48]). Diese sollen die contextual grammars in geeigneter Weise einschränken, um für die von Seiten der Linguistik gestellten Fragen das bereits erwähnte Ziel eines einerseits ausreichend mächtigen und andererseits (unter anderem bezüglich der Laufzeit) praktisch durchführbaren Modells erreichen zu können.

Außer den bereits erwähnten (und vielen weiteren) Grammatikmodellen gibt es zudem das weite Gebiet der *Automatentheorie*, in dem abstrakte Maschinen als theoretische Modelle für Computer und für Berechnungen im

Allgemeinen betrachtet werden. Zu jeder Ebene der Chomsky-Hierarchie existieren hier jeweils Automatenmodelle, die exakt die gleichen Sprachfamilien beschreiben wie die Grammatiken, mithilfe derer die Hierarchie definiert wird. Die klassischen Modelle zur Charakterisierung dieser Ebenen sind (in der Reihenfolge wie zuvor):

1. endlicher Automat,
2. Kellerautomat,
3. linear beschränkter Automat,
4. Turingmaschine.

Die Automatentheorie beschäftigt sich mit umfangreichen Fragestellungen zu den oben genannten und vielen weiteren Automatenmodellen. Neben der fundamentalen Frage nach der Berechnungsstärke der Modelle werden algebraische Eigenschaften der durch sie akzeptierten Sprachfamilien, die möglichst platzsparende Beschreibung gegebener Sprachen, die effiziente Implementierung als Software sowie viele weitere Themen betrachtet. Die Automatentheorie und die Theorie der formalen Sprachen haben Anwendungen unter anderem in der Berechenbarkeitstheorie, in der Komplexitätstheorie, im Compilerbau, bei objektorientierter Modellierung, in Realzeit-Systemen, Kommunikationsprotokollen, theoretischer Biologie und – nicht zuletzt – in der Linguistik gefunden.

Wie im Bereich der Grammatiken, wurden auch in der Automatentheorie in den vergangenen Jahrzehnten diverse neue Modelle erforderlich, um Konzepte aus der Linguistik formalisieren zu können, beziehungsweise um zu versuchen, linguistische Fragestellungen zu lösen.

Der von Michal Chytil, Martin Plátek und Jörg Vogel betrachtete *Listenautomat* (englisch *list automaton*, siehe [6, 49]) ist ein solches Beispiel, das zudem auch aus der Sicht einer Implementierung interessant ist. Die Datenstruktur der doppelt verketteten Liste ist eine naheliegende und effiziente Form, die Eingabe eines Automaten zu übergeben. Listenautomaten arbeiten auf einer solchen Liste und verfügen über die Operationen *Einfügen*, *Löschen*, *Schreiben* und *Kopf nach links* beziehungsweise *Kopf nach rechts bewegen*; diese Operationen erscheinen aus praktischer Sicht sehr natürlich. Aus theoretischer Sicht ergibt sich durch die Listenautomaten zudem ein äußerst interessantes Ergebnis: Schränkt man die zur Verfügung stehenden Operationen in geeigneter Form ein, lassen sich die bereits erwähnten vier Ebenen der Chomsky-Hierarchie in einem gleichförmigen Automatenmodell charakterisieren. Erstaunlich ist diese Charakterisierung vor allem für die Ebene der kontextfreien Sprachen. Während die Darstellung von endlichen

Automaten, linear beschränkten Automaten und Turingmaschinen als Listenautomaten auf naheliegende Weise möglich ist, ist die Simulation eines Kellerautomaten keineswegs offensichtlich.

Eine mögliche Anwendung und die Motivation für das Konzept der Verkürzung durch Löschen liegen darin, zu erkennen, ob ein gegebener Satz syntaktisch korrekt aufgebaut ist. Durch eine schrittweise Verkürzung wird schließlich ein kurzer Satz erreicht, für den nun die Entscheidung „korrekt / nicht korrekt“ getroffen werden kann. Falls die Verkürzungen die Korrektheit oder Inkorrektheit nicht verändern, kann somit das Ergebnis für den ursprünglichen Satz durch diesen Prozess gewonnen werden. Als Beispiel betrachten wir die schrittweise Verkürzung des folgenden Satzes:

„Die schwarze Katze lief schnell davon.“
„Die Katze lief schnell davon.“
„Die Katze lief davon.“
„Die Katze lief.“

Eine mit dem Löschen vergleichbare Operation ist die des *Ausradierens*. Hier wird ein Eingabesymbol nicht komplett entfernt (zum Beispiel aus einer doppelt verketteten Liste), sondern lediglich mit einem *Leerzeichen* überschrieben. Der von dem Zeichen zuvor beanspruchte Platz bleibt jedoch – wie beim Ausradieren eines Wortes auf einem Blatt Papier – bestehen; die ausradierte Stelle ist also im Nachhinein noch als solche zu erkennen. Das Ausradieren wurde 1979 durch Burchard von Braunmühl und Rutger Verbeek als ein Spezialfall im Rahmen ihrer Arbeit [55] über *finite-change automata* eingeführt. Diese Automaten sind als Turingmaschinen mit zwei Bändern definiert, wobei das erste Band die Eingabe enthält. Über zwei Funktionen f bzw. g werden für eine gegebene Wortlänge obere Schranken für die Platzbeschränkung auf dem ersten bzw. zweiten Band fixiert. Eine feste Zahl k gibt an, wie oft ein einzelnes Feld des ersten Bandes mit einem neuem Symbol beschrieben werden darf. Durch die Einschränkung der Funktion f auf die identische Abbildung und der Zahl k auf den Wert 1 ergeben sich die *erasing automata* $E(g)$. Ein solcher Automat kann auf dem ersten Band nicht mehr als die mit der Eingabe beschriebenen Felder betreten (wie ein linear beschränkter Automat), er kann diese jedoch einmalig mit einem neuen Zeichen überschreiben. Es wird allerdings festgelegt, dass beim Überschreiben eines Feldes stets das gleiche Symbol verwendet werden muss. Zudem steht das g-platzbeschränkte zweite Band zur Verfügung.

Für das Überschreiben mit einem fest vorgegebenen Symbol wurde somit der Begriff des Ausradierens geprägt. In Analogie zum bereits vorgestellten Beispiel der schrittweisen Verkürzung eines Satzes durch Löschen kann das Ausradieren wie folgt zur Syntaxüberprüfung verwendet werden:

„Die schwarze Katze lief schnell davon.“
„Die _____ Katze lief schnell davon.“
„Die _____ Katze lief _____ davon.“
„Die _____ Katze lief _____ _____.“

In [55] wurde bewiesen, dass erasing automata $E(\log)$ mit einem logarithmisch beschränkten zweiten Band alle kontextfreien Sprachen akzeptieren können. Das Akzeptieren einer kontextfreien Sprache wird dabei über die rückwärtige Abarbeitung (das heißt vom gegebenen Wort zum Axiom) einer Grammatik in Chomsky-Normalform realisiert. Die in derselben Arbeit aufgestellte Vermutung, dass nicht alle kontextfreien Sprachen durch einen erasing automaton *ohne* weiteres Arbeitsband erkannt werden können, wurde allerdings 1991 von František Mráz und Martin Plátek in [42] widerlegt. Dabei wurde gezeigt, dass die Ableitungsschritte einer kontextfreien Grammatik in Chomsky-Normalform durch einen erasing automaton geraten werden können. Die bei der Verarbeitung verwendete Technik kann als Verallgemeinerung des parallelen Algorithmus zum Parsen von kontextfreien Sprachen aus [52] betrachtet werden. In [27] wurde 1992 zudem ein alternativer Beweis erbracht, bei dem die Rechtsableitung einer Grammatik in *TS(k)-Form* rückwärts abgebaut wird. Die TS(k)-Form ist dabei eine Normalform für kontextfreie Grammatiken, bei der die rechten Seiten der Produktionsregeln – bis auf Ausnahmen für das direkte Ableiten aus dem Startsymbol – mit mindestens k Terminalzeichen enden. Durch diese Normalform wird sichergestellt, dass die während der Abarbeitung der Grammatik auftretenden Nichtterminalzeichen durch das Ausradieren von geeigneten Eingabezeichen auf dem Band codiert hinterlassen werden können. Durch die Länge der rechten Regelseiten ist dabei ausreichend Platz für die Codierung der Nichtterminale vorhanden.

Ebenfalls im Jahre 1992 wurde von Petr Jančar, František Mráz und Martin Plátek in [28] erstmals die Kombination der Operationen Löschen und Ausradieren in einem neuen Automatenmodell betrachtet. Dieser neue Automatentyp wird als *forgetting automaton* bezeichnet. Er stellt den Mittelpunkt dieser Arbeit dar, in der wir ihn – nur zum Teil eingedeutscht – als *Forgetting-Automaten* bezeichnen. Auf eine vollständige Übersetzung, etwa als *vergessenden Automaten*, wird verzichtet. Die verschiedenen in [28] definierten Modelle der Forgetting-Automaten ergeben sich aus den Teilmengen, die aus der Menge aller möglichen Operationen gebildet werden können. Diese Operationen sind das Löschen DL, das Ausradieren ER und das schlichte Bewegen MV des Kopfes auf ein Nachbarfeld. Ferner wird jeweils eine mit Index L bzw. R versehene Version betrachtet, bei der der Index die Bewegungsrichtung angibt, die eingeschlagen werden kann. So bedeutet MV_L das Bewegen des Kopfes um ein Feld nach links, während ER_R das Ausradieren eines Feldes mit anschließender Rechtsbewegung bezeichnet. Bei den

Löschoperationen wird entsprechend das aktuelle Feld gelöscht und der Kopf anschließend auf dem zuvor links bzw. rechts liegenden Feld platziert.

Ein Forgetting-Automat ist somit ein Automat, der über mindestens eine der Operationen MV_L, MV_R, ER_L, ER_R, DL_L und DL_R verfügt. Da sich erstens 63 solcher Mengen von Operationen als Grundlage für einen Automaten betrachten lassen und zweitens zudem der nichtdeterministische und der deterministische Fall für jeden Typ eines Automaten unterschieden werden kann, ergibt sich durch diese Definition ein breites Feld von Modellen, das es zu untersuchen gilt. Ergebnisse zu bestimmten Typen der list automata und der erasing automata, die nun als Spezialfälle der Forgetting-Automaten aufgefasst werden können, ergänzen dabei die Betrachtungen aus der Arbeit [28], in der die Familien der von Forgetting-Automaten erkannten Sprachen in die Chomsky-Hierarchie eingeordnet wurden. In den weiteren Veröffentlichungen [26, 29, 30, 31, 43, 44] folgten neue Ergebnisse, unter anderem der durch Diagonalisierung geführte Beweis, dass auch das berechnungsmächtigste nichtdeterministische Forgetting-Automaten-Modell schwächer als ein deterministischer linear beschränkter Automat ist. Allerdings blieben auch einige Probleme ungelöst.

Ausgehend von den bisher betrachteten Forgetting-Automaten ist das Modell zudem in einer Reihe von Arbeiten erweitert beziehungsweise abgeändert worden. Durch die Einführung von *Restart-Automaten* in [32] wurde 1995 von Petr Jančar, František Mráz, Martin Plátek und Jörg Vogel die schrittweise Verkürzung einer Eingabe zur Syntaxanalyse („analysis by reduction“) mithilfe von mehreren Durchgängen der Berechnung modelliert. In jedem Durchgang muss dabei das Eingabeband verkürzt werden. Durch den *Restart* wird der Automat anschließend am Bandanfang neu gestartet. Diverse Varianten der Restart-Automaten wurden unter anderem in [33, 35, 36, 45, 46] betrachtet und führten zu interessanten Charakterisierungen bekannter Sprachfamilien. So wurden beispielsweise die *Church-Rosser-Sprachen* von Gundula Niemann und Friedrich Otto in [47] durch deterministische Restart-Automaten charakterisiert, die zusätzliche Sonderzeichen schreiben können. In [34, 50] betrachten Petr Jiřička und Jaroslav Král beziehungsweise Daniel Průša zweidimensionale Erweiterungen der Forgetting-Automaten. In diesem Fall liegen die Eingaben eines Automaten nicht mehr als eindimensionale Zeichenkette vor, sondern als zweidimensionale, rechteckige Matrix aus Symbolen. Eine solche Matrix kann als *zweidimensionales Wort* oder *Bild* bezeichnet werden. Eine Menge solcher Wörter wird daher *zweidimensionale Sprache* oder *Bildsprache* genannt. Da sich hier im Gegensatz zum eindimensionalen Fall das Entfernen eines einzelnen Feldes nicht sinnvoll realisieren lässt, wird im zweidimensionalen Fall nur das Ausradieren betrachtet. Die so definierten Forgetting-Automaten werden in den genannten Arbeiten schließlich mit anderen Modellen zur Beschreibung zweidimensionaler Sprachen (siehe zum Beispiel [10]) verglichen.

In dieser Arbeit wird die Untersuchung der Forgetting-Automaten fortgeführt und es wird versucht, weitergehende Einblicke in die Möglichkeiten und Grenzen dieses Modells zu gewinnen. Die weiteren Kapitel dieser Arbeit sind wie folgt aufgebaut: In Kapitel 2 legen wir einige grundlegende Schreibweisen und Bezeichnungen fest. Ferner werden die in den darauf folgenden Kapiteln verwendeten Automaten- und Grammatikmodelle definiert.

In Kapitel 3 wird das Modell des Forgetting-Automaten ausführlich beschrieben und formal definiert. Zu einigen typischen Beispielsprachen werden entsprechende Automaten konstruiert. Nach der Zusammenstellung der aus der Literatur bekannten Ergebnisse betrachten wir eine Reihe von Normierungen für bestimmte Modelle, die in den folgenden Kapiteln benötigt werden.

Kapitel 4 befasst sich mit der Berechnungsmächtigkeit für den allgemeinen Fall eines Eingabealphabets, während in Kapitel 5 nur unäre (das heißt aus *einem* Element bestehende) Alphabete betrachtet werden. Dabei werden die Modelle der Forgetting-Automaten sowohl untereinander als auch mit aus der Literatur bekannten, klassischen Modellen wie linearen und metalinearen Grammatiken sowie verschiedenen Arten von Zähler-Automaten verglichen. Der überwiegende Teil der Ergebnisse aus den Kapiteln 4 und 5 wurde bzw. wird auf der *Eleventh International Conference on Implementation and Application of Automata (CIAA 2006)* in Taipeh und der *Twelfth International Conference on Developments in Language Theory (DLT 2008)* in Kyoto präsentiert und in den Konferenzbänden der beiden Tagungen ([11, 14]) sowie der Zeitschrift *International Journal of Foundations of Computer Science* ([13]) veröffentlicht.

Im abschließenden Kapitel 6 werden die Familien der von Forgetting-Automaten erkannten Sprachen auf ihre Abschlusseigenschaften hin untersucht. Die Ergebnisse aus diesem Kapitel wurden größtenteils auf der Tagung *Mathematical and Engineering Methods in Computer Science (MEMICS 2007)* in Znojmo vorgestellt und im zugehörigen Konferenzband veröffentlicht ([12]).

Kapitel 2

Grundlagen

In diesem Kapitel werden wir zunächst einige grundlegende Begriffe und Notationen betrachten, die für dieses und die weiteren Kapitel von Bedeutung sind. Des Weiteren werden die benötigten grundlegenden Typen von Grammatiken und Automaten formal definiert.

2.1 Begriffe und Notationen

Es sei $\mathbb{N} := \{1, 2, 3, \dots\}$ die Menge der *natürlichen Zahlen*. Die Vereinigung $\mathbb{N} \cup \{0\}$ wird mit $\mathbb{N}_0$ bezeichnet.

Für eine Menge M bezeichnet $|M|$ deren *Mächtigkeit*. Für das *kartesische Produkt* der Mengen $M_1, M_2, \dots, M_n$ schreiben wir $M_1 \times M_2 \times \cdots \times M_n$; das k-fache kartesische Produkt einer Menge M wird auch mit M^k bezeichnet. Für die *Potenzmenge* von M schreiben wir 2^M.

Für Mengen M und N unterscheiden wir zwei verschiedene Notationen der Inklusion:

$M \subseteq N$	M ist eine *Teilmenge* von N,
$M \subset N$	M ist eine *echte Teilmenge* von N ($M \neq N$).

Wir nennen zwei Mengen M und N *unvergleichbar*, falls weder M in N noch N in M enthalten ist, das heißt es gilt:

$$M \not\subseteq N \qquad \text{und} \qquad N \not\subseteq M.$$

Ist f eine Abbildung von der Menge X in die Menge Y, so schreiben wir $f : X \to Y$. Für ein Element $x \in X$ bezeichnet $f(x)$ das Bild von x unter f.

Alphabete sind endliche, nichtleere Mengen. Die Elemente eines Alphabets werden als *Zeichen* oder *Symbole* bezeichnet. Ein *Wort* ist eine endliche Folge von Elementen des Alphabets, das durch Hintereinanderschreiben (*Konkatenieren*) seiner Zeichen entsteht. Das *leere Wort* wird mit λ bezeichnet.

Ist A ein Alphabet, so schreiben wir A^* für die Menge aller Wörter über A und A^+ für $A^* \setminus \{\lambda\}$. Mit A^n bezeichnen wir die Menge aller Wörter der Länge n über A, mit $A^{\leq n}$ die Menge aller Wörter der Länge kleiner oder gleich n. Für ein Wort $w \in A^*$ bezeichnet w^n die n-fache Konkatenation von w, wobei $w^0 = \lambda$. $|w|$ ist die Länge des Wortes w (mit $|\lambda| = 0$). Mit $|w|_a$ bezeichnen wir die Häufigkeit des Vorkommens von $a \in A$ im Wort $w \in A^*$. Teilmengen von A^* heißen *(formale) Sprachen* über A. Für eine Sprache $L \subseteq A^*$ bezeichnet $\overline{L} = (A^* \setminus L)$ das Komplement von L.

Für ein Wort $w = a_1 \cdots a_n \in A^*$ bezeichnen wir mit w^R das (zu w) *gespiegelte Wort*, also $w^R = a_n \cdots a_1$. Die *gespiegelte Sprache* L^R einer Sprache L ist definiert als die Menge aller gespiegelten Wörter aus L: $L^R = \{w^R \mid w \in L\}$.

Betrachten wir zwei Automaten, so bezeichnen wir diese als *äquivalent*, falls die von ihnen erkannten Sprachen gleich sind. Die von einem Automaten $\mathcal{A}$ erkannte Sprache bezeichnen wir dabei mit $L(\mathcal{A})$, die Familie der von Automaten eines Typs X erkannten Sprachen mit $\mathscr{L}(X)$.

Die Familie der von deterministischen Automaten des Typs X erkannten Sprachen bezeichnen wir mit $\mathscr{L}_{\text{det}}(X)$. Beschränken wir die Betrachtung auf Sprachen über einem *unären* Alphabet (das heißt einem Alphabet, welches nur aus einem Symbol besteht), dann bezeichnen wir die entsprechenden Sprachfamilien mit $\mathscr{L}^u(X)$ bzw. $\mathscr{L}^u_{\text{det}}(X)$.

Erweitern wir die Menge S der Zustände eines Automaten um weitere Zustände S', so gehen wir stets davon aus, dass die Vereinigung $S \cup S'$ disjunkt ist, das heißt es gilt $S \cap S' = \emptyset$. Durch Umbenennen der Zustände lässt sich dies gegebenenfalls immer sicherstellen.

2.2 Endliche Automaten

Ausgangspunkt unserer Betrachtungen von Automaten ist das Modell des *endlichen Automaten*. Charakteristisch hierfür sind einerseits die endliche Menge an *Zuständen*, in denen sich ein Automat befinden kann und die gewissermaßen das Gedächtnis darstellen, und andererseits die Tatsache, dass das Eingabeband nur gelesen – nicht beschrieben – werden kann. Zunächst betrachten wir *unidirektionale* Automaten, die sich nur in einer Richtung über das Eingabeband bewegen können und nach dem Lesen eines Zeichens jeweils entsprechend ihrer Überführungsfunktion einen neuen Zustand (den *Folgezustand*) annehmen. Befindet sich ein Automat nach dem Lesen des

letzten Zeichens der Eingabe in einem ausgezeichneten *Endzustand*, so akzeptiert er die Eingabe, anderenfalls akzeptiert er sie nicht. Auf diese Weise wird die von $\mathcal{A}$ *akzeptierte Sprache* beschrieben.

Zunächst definieren wir den *deterministischen* endlichen Automaten, bei dem der Folgezustand nach dem Lesen eines Zeichens stets eindeutig festgelegt ist.

2.2.1 Definition

(i) Ein *deterministischer endlicher Automat* (DFA, „deterministic finite automaton“) ist ein System $\mathcal{A} = \langle S, A, \delta, s_0, F\rangle$, wobei gilt:

(a) S ist die endliche Menge von *Zuständen*,

(b) A ist das *Eingabealphabet*,

(c) s_0 ist der *Startzustand*,

(d) $F \subseteq S$ ist die Menge der *Endzustände*,

(e) $\delta : S \times A \to S$ ist die totale *Überführungsfunktion*.

(ii) Die *erweiterte Überführungsfunktion* $\Delta : S \times A^* \to S$ von $\mathcal{A}$ wird für $a \in A, w \in A^*$ und $s \in S$ wie folgt definiert:

$$\begin{aligned} \Delta(s, \lambda) &= s \\ \Delta(s, wa) &= \delta(\Delta(s, w), a) \end{aligned}$$

(iii) Die von $\mathcal{A}$ *akzeptierte Sprache* ist

$$L(\mathcal{A}) = \{w \in A^* \mid \Delta(s_0, w) \in F\}.$$

Nun betrachten wir die *nichtdeterministischen endlichen Automaten*, die nach dem Lesen eines Zeichens in mehrere Folgezustände verzweigen können:

2.2.2 Definition

(i) Ein *nichtdeterministischer endlicher Automat* (NFA, „nondeterministic finite automaton“) ist ein System $\mathcal{A} = \langle S, A, \delta, s_0, F\rangle$, wobei gilt:

(a) – (d) wie in Definition 2.2.1,

(e) $\delta : S \times A \to 2^S$ ist die Überführungsfunktion.

(ii) Die erweiterte Überführungsfunktion $\Delta : S \times A^* \to 2^S$ von $\mathcal{A}$ wird für $a \in A, w \in A^*$ und $s \in S$ wie folgt definiert:

$$\begin{aligned} \Delta(s, \lambda) &= \{s\} \\ \Delta(s, wa) &= \bigcup_{s' \in \Delta(s,w)} \delta(s', a) \end{aligned}$$

(iii) Die von $\mathcal{A}$ akzeptierte Sprache ist

$$L(\mathcal{A}) = \{w \in A^* \mid \Delta(s_0, w) \cap F \neq \emptyset\}.$$

Die Familie der von endlichen Automaten erkannten Sprachen ist die Familie REG der regulären Sprachen, sowohl im deterministischen als auch im nichtdeterministischen Fall (siehe [23]). Auch die folgende Erweiterung auf eine bidirektionale Bewegung des Lesekopfes führt noch nicht aus der Familie der regulären Sprachen heraus.

Gegenüber den unidirektionalen Modellen gibt es folgende Unterschiede:

- Die Überführungsfunktion liefert nicht nur einen Folgezustand, sondern bestimmt zusätzlich auch eine Kopfbewegung ($-1 \mathrel{\widehat{=}}$ nach links, $1 \mathrel{\widehat{=}}$ nach rechts).
- Um die Arbeitsweise der Automaten beschreiben und das Akzeptieren definieren zu können, werden *Konfigurationen* und deren Übergang zu *Nachfolgekonfigurationen* betrachtet. Eine Konfiguration beschreibt die Eingabe, den Zustand und die Kopfposition, die den Gesamtzustand eines Automaten zu einem bestimmten Zeitpunkt ausmachen.
- Die Eingabe wird durch ein linkes und ein rechts Bandbegrenzungssymbol ($\triangleright$ und $\triangleleft$) eingefasst. Beim Erreichen eines Begrenzungssymbols kann der Automat sich wieder zurückbewegen und die Berechnung fortführen. In der Literatur wird das bidirektionale Modell üblicherweise ohne diese Symbole definiert (siehe zum Beispiel [23]), und der Automat *fällt* somit beim Erreichen des Eingabeendes vom Band. Die Berechnungsstärke ist jedoch in beiden Fällen gleich (siehe [53]). Wir betrachten hier die Variante mit Begrenzungen, da dieses Modell für die weiteren Kapitel von Bedeutung sein wird.
- Die Überführungsfunktion der bidirektionalen endlichen Automaten ist auch für den deterministischen Fall im Allgemeinen nicht total, sondern partiell. Gerät der Automat in eine Situation, in der die Überführungsfunktion nicht definiert ist (und somit keine Nachfolgekonfiguration existiert), so sprechen wir davon, dass der Automat *hält*.

2.2.3 Definition

(i) Ein *deterministischer bidirektionaler endlicher Automat* (2DFA, „two-way deterministic finite automaton [with endmarkers]“) ist ein System $\mathcal{A} = \langle S, A, \delta, s_0, \triangleright, \triangleleft, F\rangle$, wobei gilt:

(a) – (d) wie in Definition 2.2.1,

(e) $\triangleright, \triangleleft \notin A$ sind das linke bzw. das rechte Bandbegrenzungssymbol,

(f) $\delta : S \times (A \cup \{\triangleright, \triangleleft\}) \to S \times \{-1, 1\}$ ist die Überführungsfunktion. Für alle $s \in S$ gilt $\delta(s, \triangleright) \subseteq S \times \{1\}$ und $\delta(s, \triangleleft) \subseteq S \times \{-1\}$.

(ii) Eine *Konfiguration* von $\mathcal{A}$ ist ein Wort aus $S \triangleright A^* \triangleleft$ oder $\triangleright A^* S A^* \triangleleft$. Im Falle einer Konfiguration $\triangleright vsw \triangleleft$ befindet sich $\mathcal{A}$ im Zustand $s \in S$, vw ist die Eingabe und der Kopf von $\mathcal{A}$ befindet sich auf dem ersten Zeichen rechts von s.

(iii) Die Relation $\vdash$ beschreibt den Übergang von einer Konfiguration zur Nachfolgekonfiguration:

$$a_1 \cdots a_{i-1} s a_i \cdots a_n \vdash a_1 \cdots a_i s' a_{i+1} \cdots a_n \quad \text{falls } \delta(s, a_i) = (s', 1)$$

$$a_1 \cdots a_{i-1} s a_i \cdots a_n \vdash a_1 \cdots a_{i-2} s' a_{i-1} \cdots a_n \quad \text{falls } \delta(s, a_i) = (s', -1)$$

(iv) Die von $\mathcal{A}$ akzeptierte Sprache ist

$$L(\mathcal{A}) = \{w \in A^* \mid s_0 \triangleright w \triangleleft \vdash^* xsy,\ s \in F,\ \delta \text{ ist für } xsy \text{ undefiniert}\},$$

wobei $\vdash^*$ die reflexive, transitive Hülle der Relation $\vdash$ bezeichnet.

Ein *nichtdeterministischer bidirektionaler endlicher Automat* (2NFA, „two-way nondeterministic finite automaton“) kann wiederum analog definiert werden:

2.2.4 Definition

(i) Ein *nichtdeterministischer bidirektionaler endlicher Automat* (2NFA) ist ein System $\mathcal{A} = \langle S, A, \delta, s_0, F \rangle$, wobei gilt:

(a) – (e) wie in Definition 2.2.3,

(f) $\delta : S \times (A \cup \{\triangleright, \triangleleft\}) \to 2^{S \times \{-1,1\}}$ ist die Überführungsfunktion.

(ii) – (iv) wie in Definition 2.2.3.

Es gilt, wie bereits erwähnt:

$$\mathscr{L}(\text{DFA}) = \mathscr{L}(\text{NFA}) = \mathscr{L}(\text{2DFA}) = \mathscr{L}(\text{2NFA}) = \text{REG}.$$

Für den Beweis der Gleichheit der Berechnungsstärke von unidirektionalen und bidirektionalen endlichen Automaten werden zumeist *Kreuzungsfolgen* (*Crossing Sequences*, siehe [23, 51]) verwendet. Für die weiteren Betrachtungen in dieser Arbeit ist jedoch die Beweismethode aus [22, 53] zweckmäßiger, die im Folgenden beschrieben wird.

Es seien ein 2DFA $\mathcal{A} = \langle S, A, \delta, s_0, \rhd, \lhd, F\rangle$ und eine Eingabe $w = a_1 \cdots a_n$ gegeben. Ohne Beschränkung der Allgemeinheit setzen wir voraus, dass $\mathcal{A}$ eine Eingabe genau dann akzeptiert, wenn er beim erstmaligen Erreichen des rechten Bandbegrenzungssymbols einen Endzustand annimmt. Jedem Präfix $w_i = a_1 \cdots a_i$ von w, $i \in \{1, \ldots, n\}$, wird eine Abbildung $\Delta_i : S \to S \cup \{s_R\}$ zugeordnet, die die Zustandsänderung von $\mathcal{A}$ widerspiegelt, falls sich der Automat nach links zurück auf das Teilwort w_i bewegt und es wieder verlässt (oder in eine Schleife läuft). Gilt $\Delta_i(s) = t$ für ein $t \in S$, bedeutet dies, dass sich $\mathcal{A}$ schließlich auf das $(i+1)$-te Feld bewegt, wenn er im Zustand s auf auf das i-te Feld angesetzt wird; bei der Bewegung auf das $(i+1)$-te Feld nimmt $\mathcal{A}$ dabei den Zustand t an. Gilt $\Delta_i(s) = s_R$, gerät $\mathcal{A}$ in eine Schleife, wenn er im Zustand s auf das i-te Feld angesetzt wird; somit bewegt sich $\mathcal{A}$ niemals vom $(i+1)$-ten Feld nach rechts und wird die Eingabe nicht akzeptieren.

Die Anzahl verschiedener Abbildungen $\Delta_i : S \to S \cup \{s_R\}$ ist $(|S|+1)^{|S|}$ und somit endlich. Wir definieren daher einen DFA $\mathcal{A}' = \langle S', A, \delta', s_0', F'\rangle$ wie folgt: Die Zustandsmenge S' besteht aus Paaren (s, Δ), mit $s \in S$ und einer Abbildung $\Delta : S \to S \cup \{s_R\}$, sowie einem Zustand s_T, für den $\delta'(s_T, a) = s_T$ für alle $a \in A$ gilt, d. h. der als *Senke* fungiert. $\mathcal{A}'$ nimmt nach dem Lesen einer Eingabe w genau dann den Zustand (s, Δ) an, wenn der 2DFA $\mathcal{A}$ den Zustand s annimmt, sobald er das erste Mal w nach rechts verlässt und Δ die w zugeordnete Abbildung ist.

Die Abbildung δ' wird definiert durch

$$\delta'\big((s_1, \Delta_1), a\big) = (s_2, \Delta_2)$$

genau dann, wenn sich s_2 und Δ_2 aus s_1 und Δ_1 wie folgt ergeben:

1. $\Delta_2(t_1) = t$, falls es eine Folge von Zuständen $t_2, t_3, \ldots, t_n, t$ gibt, sodass $\delta(t_i, a) = (t_i', -1)$ und $\Delta_1(t_i') = t_{i+1}$ für alle $i \in \{1, \ldots, n-1\}$ gelten. Zudem ist $\delta(t_n, a) = (t, 1)$.

   Falls eine derartige Folge $t_1, \ldots, t_n, t$ existiert, können wir o. B. d. A. annehmen, dass die Folge keinen Zustand doppelt enthält.

2. $\Delta_2(t_1) = s_R$, falls es keine endliche Folge von Zuständen gibt, die Punkt 1 genügt.

3. $s_2 = \Delta_2(s_1)$. Falls jedoch $\Delta_2(s_1) = s_R$ gilt, existiert kein solcher Zustand s_2 und daher wird $\delta'((s_1, \Delta_1), a) = s_T$ definiert.

Auf diese Weise enthält der Zustand des DFAs $\mathcal{A}'$ stets die dem bereits gelesenen Teilwort zugeordnete Abbildung Δ und passt diese beim Lesen jedes weiteren Zeichens entsprechend an.

Der Anfangszustand s_0' wird definiert als (s_0, Δ_0), wobei Δ_0 die dem leeren Wort λ (d. h. dem sofortigen Erreichen des linken Bandbegrenzungssymbols)

zugeordnete Funktion ist. Die Menge der Endzustände F' wird definiert als $\{(s, \Delta) \mid s \in F\}$.

Der Beweis der Aussage $L(\mathcal{A}) = L(\mathcal{A}')$ erfolgt schließlich per Induktion und wird hier nicht ausgeführt.

2.3 Kellerautomaten und kontextfreie Sprachen

Nachdem wir im letzten Abschnitt endliche Automaten und reguläre Sprachen, das heißt die unterste Ebene der Chomsky-Hierarchie, betrachtet haben, widmen wir uns in diesem Abschnitt der nächsthöheren Ebene. Diese wird durch kontextfreie Grammatiken beziehungsweise Kellerautomaten beschrieben.

2.3.1 Definition

(i) Eine *kontextfreie Grammatik* ist ein System $\mathcal{G} = \langle N, T, S, P \rangle$, wobei gilt:

  (a) N ist die endliche Menge der *Nichtterminalsymbole*,

  (b) T ist die endliche Menge der *Terminalsymbole* mit $N \cap T = \emptyset$,

  (c) S ist das *Startsymbol*,

  (d) $P \subset N \times (N \cup T)^*$ ist die endliche Menge der *Produktionen*.

(ii) Die Ableitungsrelation $\Rightarrow$ beschreibt das Erzeugen von Wörtern:

$$uAv \Rightarrow uwv \quad \text{falls } u, v \in (N \cup T)^*, (A, w) \in P$$

(iii) Die von $\mathcal{G}$ erzeugte Sprache ist

$$L(\mathcal{G}) = \{w \in T^* \mid S \Rightarrow^* w\},$$

wobei $\Rightarrow^*$ die reflexive, transitive Hülle der Relation $\Rightarrow$ bezeichnet.

Eine Sprache heißt *kontextfrei*, wenn sie von einer kontextfreien Grammatik erzeugt wird.

Nun betrachten wir das Modell des Kellerautomaten, bei dem ein endlicher Automat um einen Kellerspeicher erweitert wird.

2.3.2 Definition

(i) Ein *nichtdeterministischer Kellerautomat* (PDA, „pushdown automaton“) ist ein System $\mathcal{A} = \langle S, A, \Gamma, \delta, s_0, \bot, F \rangle$, wobei gilt:

  (a) S ist die endliche Menge von *Zuständen*,

  (b) A ist das *Eingabealphabet*,

(c) Γ ist die endliche Menge von *Kellersymbolen*,

(d) s_0 ist der *Startzustand*,

(e) $\perp \in \Gamma$ ist das *Kellerendesymbol*,

(f) $F \subseteq S$ ist die Menge der *Endzustände*,

(g) die Überführungsfunktion δ bildet von $S \times (A \cup \{\lambda\}) \times \Gamma$ in die Menge der endlichen Teilmengen von $S \times \Gamma^*$ ab.

(ii) Eine *Konfiguration* von $\mathcal{A}$ ist ein Element aus $S \times A^* \times \Gamma^*$. Im Falle einer Konfiguration (s, w, z) befindet sich $\mathcal{A}$ im Zustand $s \in S$, w ist die restliche Eingabe und z der Kellerinhalt (dessen erstes Zeichen das oberste Kellersymbol ist).

(iii) Die Relation $\vdash$ beschreibt den Übergang von einer Konfiguration zur Nachfolgekonfiguration:

$$(s, aw, gz) \vdash (s', w, z'z) \quad \text{falls } (s', z') \in \delta(s, a, g) \text{ mit}$$
$$s, s' \in S, a \in A \cup \{\lambda\}, w \in A^*, g \in \Gamma \text{ und } z, z' \in \Gamma^*$$

(iv) Die von $\mathcal{A}$ akzeptierte Sprache ist

$$L(\mathcal{A}) = \{w \in A^* \mid (s_0, w, \perp) \vdash^* (s, \lambda, z),\ s \in F,\ z \in \Gamma^*\}.$$

Die deterministische Variante des Kellerautomaten ist – im Gegensatz zu den endlichen Automaten – weniger mächtig als das nichtdeterministische Modell:

2.3.3 Definition

(i) Ein *deterministischer Kellerautomat* (DPDA) ist ein Kellerautomat $\mathcal{A} = \langle S, A, \Gamma, \delta, s_0, \perp, F\rangle$, für den gilt:

(a) Für alle $s \in S$ und $g \in \Gamma$ ist $\delta(s, a, g)$ für alle $a \in A$ undefiniert, falls $\delta(s, \lambda, g)$ definiert ist.

(b) Für alle $s \in S$, $g \in \Gamma$ und $a \in A \cup \{\lambda\}$ gilt $|\delta(s, a, g)| \leq 1$.

(ii) Eine Sprache heißt *deterministisch kontextfrei*, wenn sie von einem deterministischen Kellerautomaten erkannt wird.

Die Familien der kontextfreien bzw. deterministisch kontextfreien Sprachen werden mit CFL bzw. DCFL bezeichnet. Es gilt:

$$\mathscr{L}(\text{DPDA}) = \text{DCFL} \subset \text{CFL} = \mathscr{L}(\text{PDA}).$$

2.4 Lineare und metalineare Grammatiken

In diesem Abschnitt betrachten wir Teilklassen der Familie der kontextfreien Sprachen, die sich durch Einschränkungen an die kontextfreien Grammatiken ergeben.

2.4.1 Definition

(i) Eine Grammatik $\mathcal{G} = \langle N, T, S, P\rangle$ heißt *k-linear* (für ein $k \in \mathbb{N}$), falls alle Produktionen aus P von einer der folgenden Formen sind:

$$(S \to v), \quad (X \to z) \quad \text{oder} \quad (X \to zYz')$$

mit $X, Y \in N$, $z, z' \in T^*$ und $v \in (N \cup T)^*$, wobei v höchstens k Zeichen aus N enthält und S auf keiner rechten Seite einer Produktion vorkommt.

(ii) Eine Grammatik heißt *linear*, wenn sie 1-linear ist.

(iii) Eine Grammatik heißt *metalinear*, wenn sie k-linear (für ein $k \in N$) ist.

(iv) Eine Sprache heißt *metalinear* bzw. *k-linear*, wenn sie durch eine metalineare bzw. k-lineare Grammatik erzeugt werden kann.

(v) Die Familien der linearen bzw. metalinearen Sprachen werden mit LIN bzw. METALIN bezeichnet.

Es gilt:

$$\text{REG} \subset \text{LIN} \subset \text{METALIN} \subset \text{CFL}$$

2.5 Deterministisch lineare Sprachen

2.5.1 Definition

Eine *lineare LR(1)-Grammatik* ist eine lineare Grammatik $G = \langle N, T, S, P\rangle$, bei der für alle Satzformen v_1yaw_1 und v_1yaw_2 (wobei $v_1, v_2, w_1, w_2 \in T^*$, $y \in T^*NT^* \cup T^*$ und $a \in T$ sind) gilt: Falls Ableitungen

$$\begin{aligned} S &\Rightarrow^* v_1Aaw_1 \Rightarrow v_1yaw_1 \\ S &\Rightarrow^* v_2Bw \;\; \Rightarrow v_1yaw_2 \end{aligned}$$

existieren, folgt daraus, dass

$$v_1 = v_2, A = B, w = aw_1$$

gilt. Weiterhin enthält N keine überflüssigen Symbole und S kommt nicht auf den rechten Seiten der Produktionen vor.

Ein *deterministischer 1-Turn-Kellerautomat* ist ein deterministischer Kellerautomat $\mathcal{A} = \langle S, A, \Gamma, \delta, s_0, \bot, F\rangle$, für den jede (vollständige) Berechnung

$$(s_1, w_1, y_1) \vdash \cdots \vdash (s_k, w_k, y_k)$$

auf einem akzeptierten Wort die folgende Eigenschaft hat: Es existiert ein $i \in \{1, \ldots, k\}$ derart, dass für die Kellerinhalte gilt:

$$|y_1| \leq \cdots \leq |y_{i-1}| \leq |y_i| > |y_{i+1}| \geq \cdots \geq |y_k|.$$

Die Familie der von deterministischen 1-Turn-Kellerautomaten akzeptierten Sprachen stimmt mit der Familie der von linearen LR(1)-Grammatiken erzeugten Sprachen überein (siehe [21]). Daher wählen wir *eine* Bezeichnung für beide Familien, nämlich DetLIN (die *Familie der deterministisch linearen Sprachen*).

Der Begriff „deterministisch linear“ wird in der Literatur unterschiedlich gebraucht. Eine Übersicht über verschiedene Notationen wird in [20] angegeben.

2.6 Linear beschränkte Automaten

2.6.1 Definition

(i) Ein *nichtdeterministischer linear beschränkter Automat* (LBA, „linear bounded automaton“) ist ein System $\mathcal{A} = \langle S, A, T, \delta, s_0, \rhd, \lhd, F\rangle$, wobei gilt:

(a) S ist die endliche Menge von *Zuständen*,

(b) A ist das *Eingabealphabet*,

(c) $T \supset A$ ist die endliche Menge von *Bandsymbolen*,

(d) s_0 ist der *Startzustand*,

(e) $\rhd, \lhd \in T \setminus A$ sind das linke bzw. das rechte Bandbegrenzungssymbol,

(f) $F \subseteq S$ ist die Menge der *Endzustände*,

(g) $\delta : S \times T \to 2^{S \times T \times \{-1,0,1\}}$ ist die Überführungsfunktion. Für alle $s \in S$ gilt $\delta(s, \rhd) \subseteq S \times \{\rhd\} \times \{1\}$ und $\delta(s, \lhd) \subseteq S \times \{\lhd\} \times \{-1\}$.

(ii) Eine *Konfiguration* von $\mathcal{A}$ ist ein Wort aus $S \rhd T^* \lhd$ oder $\rhd T^* S T^* \lhd$. Im Falle einer Konfiguration $\rhd xsy \lhd$ befindet sich $\mathcal{A}$ im Zustand $s \in S$, $\rhd xy \lhd$ ist der momentane Bandinhalt, und der Kopf von $\mathcal{A}$ befindet sich auf dem ersten Zeichen rechts von s.

(iii) Die Relation $\vdash$ beschreibt den Übergang von einer Konfiguration zur Nachfolgekonfiguration:

$$a_1 \cdots a_{i-1} s a_i \cdots a_n \vdash a_1 \cdots a_{i-1} a_i' s' a_{i+1} \cdots a_n \quad \text{falls } \delta(s, a_i) = (s', a_i', 1)$$

$$a_1 \cdots a_{i-1} s a_i \cdots a_n \vdash a_1 \cdots a_{i-1} s' a_i' a_{i+1} \cdots a_n \quad \text{falls } \delta(s, a_i) = (s', a_i', 0)$$

$$a_1 \cdots a_{i-1} s a_i \cdots a_n \vdash a_1 \cdots a_{i-2} s' a_{i-1} a_i' a_{i+1} \cdots a_n \quad \text{falls } \delta(s, a_i) = (s', a_i', -1)$$

(iv) Die von $\mathcal{A}$ akzeptierte Sprache ist

$$L(\mathcal{A}) = \{w \in A^* \mid s_0 \rhd w \lhd \vdash^* xsy,\ s \in F,\ \delta \text{ ist für } xsy \text{ undefiniert}\}.$$

Ein *deterministischer linear beschränkter Automat* (DLBA) ist ein linear beschränkter Automat $\mathcal{A} = \langle S, A, T, \delta, s_0, \rhd, \lhd, F\rangle$, für den gilt:

$$|\delta(s, a)| \leq 1 \quad \forall s \in S, a \in T.$$

Die von nichtdeterministischen bzw. deterministischen linear beschränkten Automaten erkannten Sprachen werden als *kontextsensitiv* bzw. *deterministisch kontextsensitiv* bezeichnet. Die Familien der kontextsensitiven bzw. der deterministisch kontextsensitiven Sprachen werden mit CSL bzw. DCSL bezeichnet. Üblicherweise werden kontextsensitive Sprachen und die Familie CSL über kontextsensitive Grammatiken definiert und es wird anschließend CSL $= \mathscr{L}$(LBA) gezeigt. Da in dieser Arbeit kontextsensitive Grammatiken nicht verwendet werden, verzichten wir hier auf diesen Weg.

Kapitel 3

Das Modell des Forgetting-Automaten

In diesem Kapitel werden die Grundlagen für die weitere Betrachtung der Forgetting-Automaten geschaffen. In Abschnitt 3.1 definieren wir zunächst das Modell des Forgetting-Automaten formal und beschreiben die verschiedenen möglichen Operationen, die ein Forgetting-Automat ausführen kann; diese Operationen sind das unverkennbare Merkmal der Automaten. Durch die Definition der Konfiguration eines Forgetting-Automaten und die möglichen Übergänge von einer Konfiguration zu einer Folgekonfiguration kann definiert werden, welches die von einem Automaten erkannte Sprache ist. Anschließend werden die verschiedenen Einschränkungen der Menge der zur Verfügung stehenden Operationen diskutiert, die a priori zu 63 verschiedenen Automatentypen führen. Es wird sich dabei schnell zeigen, dass einige davon auf triviale Weise zu derselben Berechnungsmächtigkeit führen und daher nicht einzeln betrachtet werden müssen. Einige triviale Modelle werden zudem direkt von der weiteren Betrachtung ausgeschlossen.

Einige Beispiele für das Erkennen von häufig verwendeten Sprachen werden in Abschnitt 3.2 angegeben. In Abschnitt 3.3 listen wir die bereits bekannten Ergebnisse aus den Arbeiten [26, 27, 28, 29, 30, 31, 43, 44, 49] auf und ordnen die oben erwähnten 63 Modelle – sowohl im deterministischen als auch im nichtdeterministischen Fall – anhand dieser Ergebnisse in diejenigen Klassen ein, die in den Kapiteln 4 bis 6 weiter betrachtet werden. Ergeben dabei mehrere Typen von Automaten dieselbe Berechnungsmächtigkeit, wählen wir *einen* als Repräsentanten für die Klasse dieser Modelle aus und geben anschließend nur noch diesen an.

In Abschnitt 3.4 beschäftigen wir uns schließlich mit einigen Normierungen, die die Diskussion bestimmter Automatenklassen im Folgenden vereinfachen werden, beziehungsweise die als Grundlage für spätere Beweise dienen können.

3.1 Formale Definition der Forgetting-Automaten

In diesem Abschnitt wird das Modell des Forgetting-Automaten beschrieben, welches die Grundlage aller weiteren Betrachtungen dieser Arbeit darstellt. Ein Forgetting-Automat kann in gewisser Weise als eine eingeschränkte Form eines linear beschränkten Automaten (LBA, siehe Definition 2.6.1) aufgefasst werden. Wir betrachten hier zunächst den Typ des Forgetting-Automaten, der über alle möglichen Operationen verfügt; die bereits erwähnten 63 Modelle ergeben sich dann durch weitere Einschränkungen an diese Menge der anwendbaren Operationen.

Der Forgetting-Automat besteht aus einer endlichen Zustandskontrolle und kann sich bidirektional über die durch die beiden Bandbegrenzungssymbole $\triangleright$ und $\triangleleft$ eingefasste Eingabe bewegen. Während ein linear beschränkter Automat die Eingabezeichen (außer den Begrenzungssymbolen) durch Symbole aus einem endlichen Bandalphabet beliebig oft überschreiben kann, sind die Veränderungsmöglichkeiten des Bandinhalts beim Forgetting-Automaten limitiert. Außer der normalen Bewegung über das Band sind nur die folgenden Operationen möglich:

Ausradieren eines Feldes Durch das Überschreiben eines Symbols mit einem fest definierten *Leerzeichen* kann der Inhalt des Feldes *ausradiert* werden. Betritt der Automat dieses Feld später erneut, ist der ursprüngliche Inhalt nicht mehr erkennbar – lediglich die Information, dass sich an dieser Position zuvor ein Eingabesymbol befand, kann daraus abgeleitet werden. Obwohl der Gehalt dieser Information auf den ersten Blick recht gering scheinen mag, wird sich herausstellen, dass das Ausradieren zu einer überraschend hohen Berechnungsstärke des Modells führt.

Da keine anderen Symbole geschrieben werden können, ist das Ausradieren eines Feldes ein irreversibler Vorgang. Das Ausradieren eines bereits ausradierten Feldes ist möglich, führt aber zu keiner Änderung des Bandinhaltes.

Löschen eines Feldes Das komplette Entfernen eines Feldes des Eingabebandes wird als *Löschen* bezeichnet. Es ist mit dem Löschen eines Elementes in der Datenstruktur einer doppelt verketteten Liste vergleichbar und verkürzt somit das Eingabeband. In diesem Fall weicht das Modell des Forgetting-Automaten von dem des LBA ab. Ein linear beschränkter Automat kann diese Operation durch das Schreiben eines Bandsymbols, das als bereits entfernt zu interpretieren ist, simulieren. Das tatsächliche Verkürzen des Bandes ist hier jedoch nicht möglich.

Der Begriff des Eingabebandes wird bei Forgetting-Automaten somit anders verwendet als bei den klassischen Automatenmodellen.

Mögliche Bezeichnungen wie *variables Band* oder *Liste* verwenden wir in dieser Arbeit jedoch nicht.

Wir definieren das Modell des Forgetting-Automaten und die verwendeten Operationen (mitsamt verschiedener Einschränkungen) nun formal wie folgt:

3.1.1 Definition

(i) Ein *Forgetting-Automat* ist ein System $\mathcal{A} = \langle S, A, \triangleright, \triangleleft, \sqcup, O, \delta, s_0, F \rangle$, wobei gilt:

(a) S ist eine endliche Menge von Zuständen.

(b) A ist das Eingabealphabet.

(c) $\triangleright, \triangleleft \notin A$ sind das linke bzw. das rechte Bandbegrenzungssymbol, $\sqcup \notin A$ ist das für das Ausradieren verwendete Leerzeichen.
Die Menge $A \cup \{\triangleright, \triangleleft, \sqcup\}$ der Bandsymbole wird mit $\hat{A}$ bezeichnet.

(d) O ist eine nichtleere Menge von Operationen (siehe unten).

(e) $\delta : S \times \hat{A} \to 2^{S \times O}$ ist die Überführungsfunktion.

(f) $s_0 \in S$ ist der Startzustand.

(g) $F \subseteq S$ ist die Menge der Endzustände.
Für Endzustände ist die Überführungsfunktion stets undefiniert, das heißt es gilt $\delta(s, a) = \emptyset \; \forall s \in F, a \in \hat{A}$.

(ii) Die Menge O besteht aus einer oder mehreren der folgenden Operationen (eine formale Beschreibung mittels Konfigurationsübergängen folgt in Teil (v) dieser Definition):

$\mathsf{MV_L}$**:** den Kopf ein Feld nach links bewegen

$\mathsf{MV_R}$**:** den Kopf ein Feld nach rechts bewegen

$\mathsf{DL_L}$**:** das aktuelle Feld löschen (das heißt komplett aus dem Eingabeband entfernen) und den Kopf auf das ursprünglich links vom aktuell gelegenen Feld bewegen

$\mathsf{DL_R}$**:** das aktuelle Feld löschen und den Kopf auf das ursprünglich rechts vom aktuell gelegenen Feld bewegen

$\mathsf{ER_L}$**:** das aktuelle Feld ausradieren (das heißt mit dem Leerzeichen $\sqcup$ überschreiben) und den Kopf nach links bewegen

$\mathsf{ER_R}$**:** das aktuelle Feld ausradieren und den Kopf nach rechts bewegen

(iii) Liest $\mathcal{A}$ das Begrenzungssymbol $\triangleright$ (bzw. $\triangleleft$), kann der Automat nur halten oder das Symbol unverändert mit einer $\mathsf{MV_R}$-Operation (bzw. $\mathsf{MV_L}$-Operation) verlassen:

$$\delta(s, \triangleleft) \subseteq S \times \{\mathsf{MV_L}\} \quad \text{und} \quad \delta(s, \triangleright) \subseteq S \times \{\mathsf{MV_R}\} \quad \forall s \in S.$$

Dabei sind $\mathsf{MV_L}$- bzw. $\mathsf{MV_R}$-Operationen auf $\triangleleft$ bzw. $\triangleright$ auch dann möglich, wenn diese Operationen nicht in O enthalten sind.

(iv) Im Allgemeinen ist ein Forgetting-Automat nach Teil (i) *nichtdeterministisch*. Ein Forgetting-Automat heißt *deterministisch*, wenn für seine Überführungsfunktion gilt:

$$|\delta(s,x)| \leq 1 \quad \text{für alle } s \in S \text{ und } x \in \hat{A}.$$

(v) Eine Konfiguration eines Forgetting-Automaten $\mathcal{A}$ ist eine Zeichenkette $w_1 s w_2$, wobei $w_2 \neq \lambda$ gilt, $w_1 w_2 \in \rhd(A \cup \{\sqcup\})^* \lhd$ der Inhalt des Bandes und s der aktuelle Zustand sind; $\mathcal{A}$ liest das erste Zeichen von w_2.

Mit $\vdash$ bezeichnen wir die Relation, die den Konfigurationsübergang gemäß δ beschreibt; $\vdash^*$ ist die reflexive, transitive Hülle von $\vdash$.

Der Konfigurationsübergang ist für die Operationen aus O dabei für einen Bandinhalt

$$w_0 w_1 \cdots w_n w_{n+1} = \rhd w_1 \cdots w_n \lhd \quad \text{mit } w_1, \ldots, w_n \in (A \cup \{\sqcup\})^*$$

und $i \in \{0, \ldots, n+1\}$ wie folgt definiert:

$$w_0 \cdots w_{i-1} s w_i \cdots w_{n+1} \vdash w_0 \cdots w_{i-2} t w_{i-1} \cdots w_{n+1}$$
$$\text{falls } (t, \mathsf{MV_L}) \in \delta(s, w_i)$$

$$w_0 \cdots w_{i-1} s w_i \cdots w_{n+1} \vdash w_0 \cdots w_i t w_{i+1} \cdots w_{n+1}$$
$$\text{falls } (t, \mathsf{MV_R}) \in \delta(s, w_i)$$

$$w_0 \cdots w_{i-1} s w_i \cdots w_{n+1} \vdash w_0 \cdots w_{i-2} t w_{i-1} w_{i+1} \cdots w_{n+1}$$
$$\text{falls } (t, \mathsf{DL_L}) \in \delta(s, w_i)$$

$$w_0 \cdots w_{i-1} s w_i \cdots w_{n+1} \vdash w_0 \cdots w_{i-1} t w_{i+1} \cdots w_{n+1}$$
$$\text{falls } (t, \mathsf{DL_R}) \in \delta(s, w_i)$$

$$w_0 \cdots w_{i-1} s w_i \cdots w_{n+1} \vdash w_0 \cdots w_{i-2} t w_{i-1} \sqcup w_{i+1} \cdots w_{n+1}$$
$$\text{falls } (t, \mathsf{ER_L}) \in \delta(s, w_i)$$

$$w_0 \cdots w_{i-1} s w_i \cdots w_{n+1} \vdash w_0 \cdots w_{i-1} \sqcup t w_{i+1} \cdots w_{n+1}$$
$$\text{falls } (t, \mathsf{ER_R}) \in \delta(s, w_i)$$

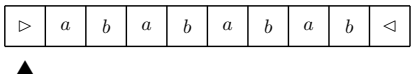
Nach Teil iii sind dabei auf den Bandbegrenzungssymbolen $w_0 = \rhd$ bzw. $w_{n+1} = \lhd$ nur $\mathsf{MV_R}$- bzw. $\mathsf{MV_L}$-Operationen erlaubt.

Ein Eingabewort wird genau dann von $\mathcal{A}$ akzeptiert, wenn es eine Berechnung gibt, die mit der Startkonfiguration $s_0 \rhd w \lhd$ beginnt und eine Konfiguration mit einem Endzustand erreicht.

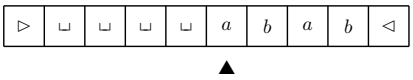
Die von $\mathcal{A}$ akzeptierte Sprache ist daher:

$$\mathscr{L}(\mathcal{A}) = \big\{ w \in A^* \;\big|\; s_0 \rhd w \lhd \vdash^* w_1 s w_2 \text{ mit } s \in F, w_2 \neq \lambda \text{ und } w_1 w_2 \in \rhd(A \cup \{\sqcup\})^* \lhd \big\}.$$

In Abbildung 3.1 werden zwei Beispiele für mögliche Bandinhalte dargestellt. Das schwarze Dreieck unterhalb des Bandes symbolisiert dabei die Kopfposition des Automaten.



(a) Zu Beginn der Berechnung steht der Lesekopf auf dem linken Bandbegrenzungssymbol.



(b) Der Lesekopf steht nach dem Ausradieren einiger Zeichen auf dem am weitesten links liegenden Eingabezeichen.

Abbildung 3.1: Beispiele für Bandinhalte.

Sind bei einem Forgetting-Automaten für eine Operation sowohl die Links- als auch die Rechtsbewegung möglich, das heißt, enthält die Menge O der Operationen des Automaten sowohl die mit L als auch die mit R indizierte Version, lassen wir den Index weg, um beide Operationen anzugeben:

$$\mathsf{MV} \mathrel{\widehat{=}} \mathsf{MV_L}, \mathsf{MV_R} \qquad \mathsf{ER} \mathrel{\widehat{=}} \mathsf{ER_L}, \mathsf{ER_R} \qquad \mathsf{DL} \mathrel{\widehat{=}} \mathsf{DL_L}, \mathsf{DL_R}$$

Um einen Forgetting-Automaten eines bestimmten Typs zu beschreiben, geben wir die möglichen Operationen des Automaten wie folgt an: Ist O eine Menge von Operationen, so bezeichnen wir die entsprechenden Automaten mit diesen Operationen als (O)-Automaten und die durch sie beschriebene Sprachfamilie als $\mathscr{L}(O)$, wobei auf die Mengenklammern der Menge O verzichtet wird. Ist die Menge der Operationen eines Automaten $\mathcal{A}$ beispielsweise $O = \{\mathsf{MV_R}, \mathsf{DL}\}$, bezeichnen wir $\mathcal{A}$ als ($\mathsf{MV_R}$, DL)-Automaten. Die Familie der von ($\mathsf{MV_R}$, DL)-Automaten erkannten Sprachen bezeichnen wir mit $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$.

Enthält die Menge O der Operationen eines Forgetting-Automaten $\mathcal{A} = \langle S, A, \rhd, \lhd, \sqcup, O, \delta, s_0, F\rangle$ keine ER-Operation, ist die Angabe des für das Ausradieren verwendeten Leerzeichens überflüssig. In diesem Fall geben wir den Automaten auch als System $\mathcal{A} = \langle S, A, \rhd, \lhd, O, \delta, s_0, F\rangle$ an.

Da die Menge der zur Verfügung stehenden Operationen eines Forgetting-Automaten eine nichtleere Teilmenge der sechselementigen Menge

$$\{\mathsf{MV_L}, \mathsf{MV_R}, \mathsf{DL_L}, \mathsf{DL_R}, \mathsf{ER_L}, \mathsf{ER_R}\}$$

ist, ergeben sich aus Definition 3.1.1 somit $2^6 - 1 = 63$ verschiedene Automatenmodelle, die betrachtet werden können. Zudem kann für jede dieser Mengen von Operationen jeweils der deterministische und der nichtdeterministische Fall unterschieden werden, sodass insgesamt 126 Modelle von Forgetting-Automaten angegeben werden können.

Im Folgenden werden wir jedoch sechs Modelle nicht weiter betrachten, deren Möglichkeiten sehr eingeschränkt sind. Dies sind die Modelle der deterministischen bzw. der nichtdeterministischen $(\mathsf{MV_L})$-, $(\mathsf{ER_L})$- und $(\mathsf{MV_L}, \mathsf{ER_L})$-Automaten. Ein Automat von einem der genannten Typen kann sich aufgrund der Festlegung, dass $\mathsf{MV_R}$ auf dem linken Bandbegrenzungssymbol stets erlaubt ist, nach rechts auf das erste Zeichen einer Eingabe bewegen – falls dieses überhaupt existiert, das heißt das Eingabewort nicht leer ist. Vom ersten Eingabezeichen ausgehend ist dann allerdings nur noch eine Linksbewegung möglich, die das erste Zeichen unverändert lässt bzw. ausradiert. Somit können keine weiteren Zeichen mehr gelesen werden.

Man beachte, dass ein $(\mathsf{DL_L})$-Automat im Gegensatz dazu durch abwechselnde $\mathsf{MV_R}$-Bewegungen auf dem linken Bandbegrenzungssymbol $\triangleright$ und $\mathsf{DL_L}$-Schritte auf dem jeweils nächsten verbliebenen Eingabezeichen bereits die gesamte Eingabe von links nach rechts lesen und somit einen endlichen Automaten simulieren kann.

Die Berechnungsstärke der zuvor genannten, stark eingeschränkten Modelle halten wir im nächsten Satz fest:

3.1.2 Satz

$$\begin{aligned}\mathscr{L}(\mathsf{MV_L}) &= \mathscr{L}_{\det}(\mathsf{MV_L}) = \mathscr{L}(\mathsf{ER_L}) = \mathscr{L}_{\det}(\mathsf{ER_L}) \\ &= \mathscr{L}(\mathsf{MV_L}, \mathsf{ER_L}) = \mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_L}) \\ &= \big\{BA^* \;\big|\; A \text{ ist ein Alphabet, } B \subseteq A\big\} \cup \\ &\quad \big\{BA^* \cup \{\lambda\} \;\big|\; A \text{ ist ein Alphabet, } B \subseteq A\big\}\end{aligned}$$

Beweis Es sei $\mathcal{A} = \langle S, A, \triangleright, \triangleleft, \sqcup, O, \delta, s_0, F\rangle$ ein Forgetting-Automat von einem der genannten Typen und $w \in A^*$ ein beliebiges Eingabewort. Der Automat $\mathcal{A}$ wird auf das linke Bandbegrenzungssymbol angesetzt und kann dieses (nach Definition 3.1.1, Teil iii) mit $\mathsf{MV_R}$ nach rechts verlassen. Es ist jedoch auch möglich, dass $\delta(s_0, \triangleright)$ gleich der leeren Menge ist, das heißt, die Überführungsfunktion ist hier undefiniert. In diesem Falle gilt:

$$L(\mathcal{A}) = \begin{cases} \emptyset = \emptyset A^* & \text{falls } s_0 \notin F, \\ A^* = AA^* \cup \{\lambda\} & \text{falls } s_0 \in F. \end{cases}$$

Falls $\delta(s_0, \triangleright) \neq \emptyset$ ist, kann sich $\mathcal{A}$ mit einer $\mathsf{MV_R}$-Operation nach rechts bewegen und erreicht dort entweder das rechte Bandbegrenzungssymbol (falls $w = \lambda$) oder das erste Eingabezeichen (falls $w \neq \lambda$). Für $w = \lambda$ besteht

die Möglichkeit, dass $\mathcal{A}$ die Eingabe akzeptiert (und somit $\lambda \in L$ gilt) oder nicht akzeptiert (und damit $\lambda \notin L$ gilt). Für $w \neq \lambda$ kann $\mathcal{A}$ keine Rechtsbewegungen mehr ausführen und folglich nicht mehr Zeichen als das linke Bandbegrenzungssymbol und das erste Eingabezeichen lesen. Für eine gewisse Teilmenge $B \subseteq A$ akzeptiert $\mathcal{A}$ die Eingabe $w = w_1 \cdots w_n$ daher für $w_1 \in B$ und verwirft sie für $w_1 \in A \setminus B$.

Die von $\mathcal{A}$ akzeptierte Sprache ist daher BA^* oder $BA^* \cup \{\lambda\}$. □

Die genannten Modelle werden im Folgenden komplett außer Acht gelassen. Ebenfalls von untergeordnetem Interesse sind die Modelle von Forgetting-Automaten, die die regulären Sprachen charakterisieren (eine vollständige Auflistung wird in den Tabellen 3.1 und 3.2 auf Seite 37 f. angegeben):

$$\mathscr{L}(\mathsf{MV}_\mathsf{R}) = \mathscr{L}(\mathsf{ER}_\mathsf{R}) = \mathscr{L}(\mathsf{DL}_\mathsf{R}) = \mathscr{L}(\mathsf{MV}) = \cdots = \mathrm{REG}.$$

Die entsprechenden Automatenmodelle stimmen mehr oder weniger direkt mit den wohlbekannten Modellen der endlichen Automaten (DFA / NFA) bzw. der bidirektionalen endlichen Automaten (2DFA / 2NFA) überein und werden daher hier nicht weiter untersucht. Ein deterministischer (MV_R)-Automat entspricht – bis auf die Bandbegrenzungssymbole – einem DFA, ein nichtdeterministischer (MV)-Automat einem 2NFA usw.

Sprechen wir von „allen Modellen“, „allen Forgetting-Automaten“ usw., so sind damit stets alle Modelle außer den genannten Modellen, die die regulären Sprachen charakterisieren, und denen in Satz 3.1.2 aufgezählten Modellen gemeint.

Zur Vereinfachung der Argumentation nehmen wir im Folgenden für einen Forgetting-Automaten $\mathcal{A} = \langle S, A, \rhd, \lhd, \sqcup, O, \delta, s_0, F\rangle$ ohne Beschränkung der Allgemeinheit an, dass $s_0 \notin F$ gilt. Aus $s_0 \in F$ folgt sonst $\mathscr{L}(\mathcal{A}) = A^*$; die Sprache A^* kann jedoch auch mithilfe zweier Zustände s_0 und s_1, wobei $s_0 \notin F$ und $s_1 \in F$, sowie der Festlegung $\delta(s_0, \rhd) = \{(s_1, \mathsf{MV}_\mathsf{R})\}$ erkannt werden.

3.2 Grundlegende Beispiele

Die Beschreibung eines Forgetting-Automaten wird in den späteren Kapiteln aufgrund der in der Regel sehr großen Anzahl von Zuständen zumeist nur noch in Worten erfolgen. Um eine Vorstellung von der Realisierung der Arbeitsweise eines Automaten zu bekommen, betrachten wir in diesem Abschnitt beispielhaft auch die formale Definition einiger Automaten zu bestimmten gegebenen Sprachen. Die dabei verwendeten Beispielsprachen wurden so ausgewählt, dass sie für die späteren Betrachtungen nützlich sind,

beziehungsweise in Beweisen direkt als Begründungen herangezogen werden können.

Zunächst betrachten wir eine Sprache, die oft als Beispiel für eine Sprache verwendet wird, welche nicht von einem endlichen Automaten erkannt werden und mithilfe des Pumping-Lemmas für reguläre Sprachen als nichtregulär nachgewiesen werden kann:

3.2.1 Beispiel $\{a^n b^n \mid n \in \mathbb{N}\} \in \mathscr{L}_{\text{det}}(\mathsf{ER_R}, \mathsf{DL})$

Ein deterministischer $(\mathsf{ER_R}, \mathsf{DL})$-Automat $\mathcal{A}$ kann die angegebene Sprache wie folgt erkennen:

Mithilfe von $\mathsf{ER_R}$-Operationen bewegt sich $\mathcal{A}$ über die a am Bandanfang, bis er das erste b erreicht. Nun beginnt $\mathcal{A}$, abwechselnd jeweils ein b mit einer $\mathsf{DL_L}$-Operation und ein Leerzeichen mit einer $\mathsf{DL_R}$-Operation zu löschen. Falls $\mathcal{A}$ erneut auf ein a trifft, hält er in einem nichtakzeptierenden Zustand, da in diesem Fall die Eingabe nicht von der Form a^*b^* war.

Durch das abwechselnde Löschen jeweils eines b und eines Leerzeichens (also eines ausradierten a) überprüft $\mathcal{A}$, ob beide Zeichen gleich oft auftreten und die Eingabe somit zu Beginn der Berechnung von der Form $a^n b^n$ (für ein $n \in \mathbb{N}$) war.

Formal definieren wir den Automaten als

$$\mathcal{A} = \langle \{s_0, s_1, s_2, s_3\}, \{a, b\}, \rhd, \lhd, \sqcup, \{\mathsf{ER_R}, \mathsf{DL}\}, \delta, s_0, \{s_3\} \rangle$$

mit der Überführungsfunktion δ wie folgt (da im deterministischen Fall $\delta(s, x)$ für jedes $s \in S$ und $x \in \hat{A}$ höchstens ein Element enthält, geben wir $\delta(s, x)$ hier nicht als Menge an):

Zustand	Eingabe	Folgezustand	Operation
s_0	a	s_0	$\mathsf{ER_R}$
s_0	b	s_1	$\mathsf{DL_L}$
s_0	$\sqcup$	undefiniert	
s_0	$\rhd$	s_0	$\mathsf{MV_R}$
s_0	$\lhd$	undefiniert	
s_1	a	undefiniert	
s_1	b	s_1	$\mathsf{DL_L}$
s_1	$\sqcup$	s_1	$\mathsf{DL_R}$
s_1	$\rhd$	undefiniert	
s_1	$\lhd$	s_2	$\mathsf{MV_L}$
s_2	a	undefiniert	
s_2	b	undefiniert	
s_2	$\sqcup$	undefiniert	
s_2	$\rhd$	s_3	$\mathsf{MV_R}$
s_2	$\lhd$	undefiniert	

(Fortsetzung der Tabelle)

Zustand	Eingabe	Folgezustand	Operation
s_3	a	undefiniert	
s_3	b	undefiniert	
s_3	$\sqcup$	undefiniert	
s_3	$\rhd$	undefiniert	
s_3	$\lhd$	undefiniert	

Ausgehend vom linken Bandbegrenzungssymbol bewegt sich $\mathcal{A}$ mit $\mathsf{ER_R}$-Schritten im Zustand s_0 bis zum ersten b und wechselt beim Löschen dieses Symbols in den Zustand s_1, indem er abwechselnd ein b und ein $\sqcup$ löscht. Wechselt $\mathcal{A}$ beim Erreichen des rechten Bandbegrenzungssymbols in den Zustand s_2, hat er gleich viele b und Leerzeichen (ursprünglich a) ausgelöscht. Da die Überführungsfunktion für s_2 nur auf dem linken Bandbegrenzungssymbol definiert ist, wird an dieser Stelle überprüft, ob die Eingabe vollständig gelöscht wurde, das heißt, dass sich keine Leerzeichen mehr auf dem Band befinden. $\mathcal{A}$ erreicht somit den akzeptierenden Zustand s_3 genau dann, wenn die Eingabe von der Form $a^n b^n$ (mit $n \in \mathbb{N}$) war.

Man beachte, dass die Sprache $\{a^n b^n \mid n \in \mathbb{N}\} = \{a^n b^n \mid n \geq 1\}$ das leere Wort nicht enthält. Der Automat $\mathcal{A}$ hält folgerichtig bei der Eingabe $w = \lambda$ im nichtakzeptierenden Zustand s_0 auf dem rechten Bandbegrenzungssymbol.

Die Sprache $\{a^n b^n \mid n \in \mathbb{N}_0\} = \{a^n b^n \mid n \geq 0\}$ kann jedoch selbstverständlich auch durch einen leicht modifizierten $(\mathsf{ER_R}, \mathsf{DL})$-Automaten $\mathcal{A}'$ akzeptiert werden, den wir hier der Vollständigkeit halber angeben:

$$\mathcal{A}' = \langle \{s_0, s_0', s_1, s_2, s_3\}, \{a, b\}, \rhd, \lhd, \sqcup, \{\mathsf{ER_R}, \mathsf{DL}\}, \delta', s_0, \{s_3\} \rangle.$$

Die Überführungsfunktion δ' ist dabei nur für s_0 und den neu hinzugefügten Zustand s_0' gegenüber δ verändert:

Zustand	Eingabe	Folgezustand	Operation
s_0	a	s_0'	$\mathsf{ER_R}$
s_0	b	undefiniert	
s_0	$\sqcup$	undefiniert	
s_0	$\rhd$	s_0	$\mathsf{MV_R}$
s_0	$\lhd$	s_3	$\mathsf{MV_L}$
s_0'	a	s_0'	$\mathsf{ER_R}$
s_0'	b	s_1	$\mathsf{DL_L}$
s_0'	$\sqcup$	undefiniert	
s_0'	$\rhd$	s_0'	$\mathsf{MV_R}$
s_0'	$\lhd$	undefiniert	

(Fortsetzung der Tabelle)

Zustand	Eingabe	Folgezustand	Operation
s_1	a	undefiniert	
s_1	b	s_1	$\mathsf{DL_L}$
s_1	$\sqcup$	s_1	$\mathsf{DL_R}$
s_1	$\rhd$	undefiniert	
s_1	$\lhd$	s_2	$\mathsf{MV_L}$
s_2	a	undefiniert	
s_2	b	undefiniert	
s_2	$\sqcup$	undefiniert	
s_2	$\rhd$	s_3	$\mathsf{MV_R}$
s_2	$\lhd$	undefiniert	
s_3	a	undefiniert	
s_3	b	undefiniert	
s_3	$\sqcup$	undefiniert	
s_3	$\rhd$	undefiniert	
s_3	$\lhd$	undefiniert	

□

Im obigen Beispiel wurde ein $(\mathsf{ER_R}, \mathsf{DL})$-Automat für das Erkennen der nichtregulären Sprache $\{a^n b^n \mid n \in \mathbb{N}\}$ gewählt, da dieser Automatentyp der schwächste bezüglich der Berechnungsstärke ist, wie sich im nächsten Abschnitt zeigen wird. An dieser Stelle sei jedoch bereits angemerkt, dass die gegebene Sprache beispielsweise auch durch einen $(\mathsf{MV_R}, \mathsf{DL})$-Automaten erkannt werden kann, indem sich dieser mit $\mathsf{MV_R}$-Schritten bis zum ersten b bewegt und anschließend jeweils ein a und ein b löscht.

Als nächstes Beispiel betrachten wir eine typische *Zählersprache*, das heißt eine Sprache, die von *Zähler-Automaten* – mit denen wir uns in den weiteren Kapiteln noch eingehender beschäftigen werden – erkannt werden kann:

3.2.2 Beispiel $\left\{w \in \{a,b\}^* \;\middle|\; |w|_a = |w|_b\right\} \in \mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$

Ein deterministischer $(\mathsf{ER_R}, \mathsf{DL})$-Automat $\mathcal{A}$ kann die angegebene Sprache wie folgt erkennen:

$\mathcal{A}$ löscht das erste Zeichen der Eingabe und bewegt sich mit $\mathsf{ER_R}$-Schritten nach rechts bis zum nächsten Zeichen, das sich von dem ersten unterscheidet; löscht $\mathcal{A}$ beispielsweise zunächst ein a, bewegt er sich anschließend bis zum ersten b. $\mathcal{A}$ löscht dieses Zeichen mit einer $\mathsf{DL_L}$-Operation. Falls $\mathcal{A}$ nun das linke Bandbegrenzungssymbol erreicht (in diesem Fall begann das Eingabewort mit ab oder ba), startet die beschriebene Prozedur von Neuem; anderenfalls hat $\mathcal{A}$ einige Zeichen, die gleich dem ersten gelöschten Zeichen sind, mit $\mathsf{ER_R}$ ausradiert und löscht nun das letzte Leerzeichen, um rechts wiederum nach dem nächsten Zeichen zu suchen, das sich vom zuerst gelöschten

unterscheidet. Dabei kann $\mathcal{A}$ schrittweise jeweils ein a und ein b löschen, während durch die Leerzeichen links vom Kopf die Differenz von bis dahin aufgetretenen a und b mitgezählt wird. $\mathcal{A}$ akzeptiert schließlich die Eingabe genau dann, wenn alle Symbole auf die beschriebene Weise gelöscht werden können.

Formal definieren wir den Automaten als

$$\mathcal{A} = \langle \{s_0, s_1, s_2, s_3\}, \{a, b\}, \rhd, \lhd, \sqcup, \{\mathsf{ER_R}, \mathsf{DL}\}, \delta, s_0, \{s_3\} \rangle$$

mit der Überführungsfunktion δ wie folgt:

Zustand	Eingabe	Folgezustand	Operation
s_0	a	s_1	$\mathsf{DL_R}$
s_0	b	s_2	$\mathsf{DL_R}$
s_0	$\sqcup$	undefiniert	
s_0	$\rhd$	s_0	$\mathsf{MV_R}$
s_0	$\lhd$	s_3	$\mathsf{MV_L}$
s_1	a	s_1	$\mathsf{ER_R}$
s_1	b	s_1	$\mathsf{DL_L}$
s_1	$\sqcup$	s_1	$\mathsf{DL_R}$
s_1	$\rhd$	s_0	$\mathsf{MV_R}$
s_1	$\lhd$	undefiniert	
s_2	a	s_2	$\mathsf{DL_L}$
s_2	b	s_2	$\mathsf{ER_R}$
s_2	$\sqcup$	s_2	$\mathsf{DL_R}$
s_2	$\rhd$	s_0	$\mathsf{MV_R}$
s_2	$\lhd$	undefiniert	
s_3	a	undefiniert	
s_3	b	undefiniert	
s_3	$\sqcup$	undefiniert	
s_3	$\rhd$	undefiniert	
s_3	$\lhd$	undefiniert	

Dabei sorgt die Festlegung $\delta(s_0, \lhd) = (s_3, \mathsf{MV_L})$ dafür, dass die Eingabe (durch Annehmen des Endzustands s_3) akzeptiert wird, wenn $\mathcal{A}$ vom linken Bandbegrenzungssymbol mit einem Rechtsschritt direkt zum rechten Bandbegrenzungssymbol gelangt. Dies ist genau dann der Fall, wenn die Eingabe leer oder nach dem Löschen von gleich vielen a und b komplett gelöscht ist.

□

Auch für das vorangegangene Beispiel lässt sich anmerken, dass die Sprache $\left\{w \in \{a,b\}^* \;\middle|\; |w|_a = |w|_b\right\}$ auf analoge Weise durch einen $(\mathsf{MV_R}, \mathsf{DL})$-Automaten erkannt werden kann.

Nun betrachten wir ein typisches Beispiel für eine deterministisch kontextfreie Sprache, also eine Sprache, die von einem deterministischen Kellerautomaten erkannt werden kann:

3.2.3 Beispiel $\{wcw^R \mid w \in \{a,b\}^*\} \in \mathscr{L}_{\text{det}}(\mathsf{MV_R}, \mathsf{DL})$

Ein deterministischer $(\mathsf{MV_R}, \mathsf{DL})$-Automat $\mathcal{A}$ kann die angegebene Sprache wie folgt erkennen:

$\mathcal{A}$ bewegt sich mit $\mathsf{MV_R}$-Schritten nach rechts, bis das erste c erreicht wird. $\mathcal{A}$ löscht nun das c mit einer DL-Operation und beginnt damit, die Zeichen links und rechts von der Kopfposition zu vergleichen und dabei zu löschen. Hierbei dürfen nur noch die Zeichen a und b auftreten.

Falls $\mathcal{A}$ nach dem Löschen von zwei zu vergleichenden Zeichen nacheinander die beiden Bandbegrenzungssymbole erreicht (das heißt alle Zeichen gelöscht hat), wird die Eingabe akzeptiert.

Die formale Definition des Automaten lautet

$$\mathcal{A} = \langle \{s_0, \ldots, s_5\}, \{a,b,c\}, \rhd, \lhd, \{\mathsf{MV_R}, \mathsf{DL}\}, \delta, s_0, \{s_5\} \rangle,$$

wobei die Überführungsfunktion δ wie folgt definiert ist:

Zustand	Eingabe	Folgezustand	Operation
s_0	a	s_0	$\mathsf{MV_R}$
s_0	b	s_0	$\mathsf{MV_R}$
s_0	c	s_1	$\mathsf{DL_L}$
s_0	$\rhd$	s_0	$\mathsf{MV_R}$
s_0	$\lhd$	undefiniert	
s_1	a	s_2	$\mathsf{DL_R}$
s_1	b	s_3	$\mathsf{DL_R}$
s_1	c	undefiniert	
s_1	$\rhd$	s_4	$\mathsf{MV_R}$
s_1	$\lhd$	undefiniert	
s_2	a	s_1	$\mathsf{DL_L}$
s_2	b	undefiniert	
s_2	c	undefiniert	
s_2	$\rhd$	undefiniert	
s_2	$\lhd$	undefiniert	

(Fortsetzung der Tabelle)

Zustand	Eingabe	Folgezustand	Operation
s_3	a	undefiniert	
s_3	b	s_1	$\mathsf{DL_L}$
s_3	c	undefiniert	
s_3	$\rhd$	undefiniert	
s_3	$\lhd$	undefiniert	
s_4	a	undefiniert	
s_4	b	undefiniert	
s_4	c	undefiniert	
s_4	$\rhd$	undefiniert	
s_4	$\lhd$	s_5	$\mathsf{MV_L}$
s_5	a	undefiniert	
s_5	b	undefiniert	
s_5	c	undefiniert	
s_5	$\rhd$	undefiniert	
s_5	$\lhd$	undefiniert	

Die Zustände s_2 bzw. s_3 werden dabei erreicht, wenn $\mathcal{A}$ zuvor im Zustand s_1 ein a bzw. ein b gelöscht hat. Im Zustand s_4 wird schließlich überprüft, ob die Eingabe komplett gelöscht wurde. Falls dies der Fall ist, akzeptiert $\mathcal{A}$ die Eingabe im Zustand s_5. □

Das nichtdeterministische Äquivalent zum vorangegangenen Beispiel ist die Menge der Palindrome gerader Länge über dem Alphabet $\{a, b\}$, ein klassisches Beispiel für eine kontextfreie Sprache, die nicht deterministisch kontextfrei ist:

3.2.4 Beispiel $\{ww^R \mid w \in \{a,b\}^*\} \in \mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$

Ein nichtdeterministischer $(\mathsf{MV_R}, \mathsf{DL})$-Automat $\mathcal{A}$ kann die Menge der Palindrome wie folgt erkennen:

$\mathcal{A}$ bewegt sich mit $\mathsf{MV_R}$-Bewegungen nach rechts und rät, wann die Mitte der Eingabe erreicht ist. Ab diesem Zeitpunkt vergleicht $\mathcal{A}$ analog zu Beispiel 3.2.3 die Zeichen links und rechts vom Kopf und akzeptiert die Eingabe, falls durch das paarweise Löschen alle Zeichen gelöscht werden.

Formal definieren wir den Automaten $\mathcal{A}$ und seine Überführungsfunktion wie folgt:

$$\mathcal{A} = \langle \{s_0, \ldots, s_6\}, \{a, b, c\}, \rhd, \lhd, \{\mathsf{MV_R}, \mathsf{DL}\}, \delta, s_0, \{s_6\} \rangle.$$

Zustand	Eingabe	(Folgezustand, Operation)
s_0	a	$(s_1, \mathsf{MV_R}),\ (s_2, \mathsf{MV_R})$
s_0	b	$(s_1, \mathsf{MV_R}),\ (s_2, \mathsf{MV_R})$
s_0	$\rhd$	$(s_0, \mathsf{MV_R}),\ (s_2, \mathsf{MV_R})$
s_0	$\lhd$	$(s_6, \mathsf{MV_L})$
s_1	a	$(s_1, \mathsf{MV_R}),\ (s_2, \mathsf{MV_R})$
s_1	b	$(s_1, \mathsf{MV_R}),\ (s_2, \mathsf{MV_R})$
s_1	$\rhd$	$\emptyset$
s_1	$\lhd$	$\emptyset$
s_2	a	$(s_3, \mathsf{DL_R})$
s_2	b	$(s_4, \mathsf{DL_R})$
s_2	$\rhd$	$(s_5, \mathsf{MV_R})$
s_2	$\lhd$	$\emptyset$
s_3	a	$(s_2, \mathsf{DL_L})$
s_3	b	$\emptyset$
s_3	$\rhd$	$\emptyset$
s_3	$\lhd$	$\emptyset$
s_4	a	$\emptyset$
s_4	b	$(s_2, \mathsf{DL_L})$
s_4	$\rhd$	$\emptyset$
s_4	$\lhd$	$\emptyset$
s_5	a	$\emptyset$
s_5	b	$\emptyset$
s_5	$\rhd$	$\emptyset$
s_5	$\lhd$	$(s_6, \mathsf{MV_L})$
s_6	a	$\emptyset$
s_6	b	$\emptyset$
s_6	$\rhd$	$\emptyset$
s_6	$\lhd$	$\emptyset$

Beim erfolgreichen Raten der Mitte eines Palindroms geschieht Folgendes:

- Im Falle des leeren Wortes bewegt sich $\mathcal{A}$ vom linken Bandbegrenzungssymbol nach rechts, nimmt erneut den Zustand s_0 an und akzeptiert die Eingabe nach Erreichen des rechten Bandbegrenzungssymbols im Zustand s_6.
- Bei Palindromen der Länge 2 (also aa und bb) bewegt sich $\mathcal{A}$ nach rechts und nimmt den Zustand s_2 an. Nach dem Löschen der beiden Eingabezeichen erreicht $\mathcal{A}$ das linke Bandbegrenzungssymbol im Zustand s_2, überprüft im Zustand s_5, dass die Eingabe komplett gelöscht wurde und akzeptiert im Zustand s_6.

- Bei den Palindromen $a_1a_2a_3a_4$ der Länge 4 nimmt $\mathcal{A}$ zunächst ein weiteres Mal den Zustand s_0 an, bewegt sich dann in den Zustand s_2 auf das Zeichen a_2 und löscht die zusammengehörigen Zeichen a_2 und a_3 sowie a_1 und a_4. Anschließend wird auch hier das Wort durch Annehmen der Zustände s_5 und s_6 akzeptiert.
- Bei Palindromen der Länge $2n$ mit $n \geq 3$ wechselt $\mathcal{A}$ vom ersten Eingabezeichen in den Zustand s_1 und bewegt sich darin bis zum n-ten Zeichen, indem es im Zustand s_2 das paarweise Löschen von Zeichen beginnt.

□

Die folgende Sprache ist die erste in diesem Abschnitt, die nicht kontextfrei ist und somit nicht von einem Kellerautomaten erkannt werden kann. Das in dem Beispiel verwendete Modell des (ER, DL)-Automaten kann somit Sprachen erkennen, die nicht kontextfrei sind. Später wird sich jedoch auch zeigen, dass kontextfreie Sprachen existieren, die nicht von einem (ER, DL)-Automaten erkannt werden können.

3.2.5 Beispiel $\{a^{2^n} \mid n \in \mathbb{N}_0\} \in \mathscr{L}_{\text{det}}(\mathsf{ER}, \mathsf{DL})$

Ein deterministischer (ER, DL)-Automat $\mathcal{A}$ kann L wie folgt erkennen:

$\mathcal{A}$ bewegt sich wiederholt über die Eingabe von Bandbegrenzungssymbol zu Bandbegrenzungssymbol und löscht dabei jedes zweite Zeichen auf dem Band – während für die restlichen Symbole eine ER-Operation verwendet wird. Auf diese Weise wird die Eingabe schrittweise halbiert. $\mathcal{A}$ akzeptiert schließlich genau dann, wenn die Länge der Eingabe nach jedem Halbierungsschritt gerade ist (bzw. 1 im letzten Schritt).

Die formale Definition des Automaten $\mathcal{A}$ und der Überführungsfunktion kann wie folgt realisiert werden:

$$\mathcal{A} = \langle \{s_0, \ldots, s_5\}, \{a, b, c\}, \rhd, \lhd, \sqcup, \{\mathsf{ER}, \mathsf{DL}\}, \delta, s_0, \{s_5\} \rangle.$$

Zustand	Eingabe	Folgezustand	Operation
s_0	a	s_1	DL_R
s_0	$\sqcup$	s_1	DL_R
s_0	$\rhd$	s_0	MV_R
s_0	$\lhd$	undefiniert	
s_1	a	s_2	ER_R
s_1	$\sqcup$	s_2	ER_R
s_1	$\rhd$	undefiniert	
s_1	$\lhd$	s_5	MV_L

(Fortsetzung der Tabelle)

Zustand	Eingabe	Folgezustand	Operation
s_2	a	s_3	$\mathsf{DL_R}$
s_2	$\sqcup$	s_3	$\mathsf{DL_R}$
s_2	$\triangleright$	undefiniert	
s_2	$\triangleleft$	s_4	$\mathsf{MV_L}$
s_3	a	s_2	$\mathsf{ER_R}$
s_3	$\sqcup$	s_2	$\mathsf{ER_R}$
s_3	$\triangleright$	undefiniert	
s_3	$\triangleleft$	undefiniert	
s_4	a	undefiniert	
s_4	$\sqcup$	s_4	$\mathsf{ER_L}$
s_4	$\triangleright$	s_0	$\mathsf{MV_R}$
s_4	$\triangleleft$	undefiniert	
s_5	a	undefiniert	
s_5	$\sqcup$	undefiniert	
s_5	$\triangleright$	undefiniert	
s_5	$\triangleleft$	undefiniert	

Im Zustand s_0 wird dabei in jedem Durchgang zunächst das linke Bandbegrenzungssymbol verlassen und nach dem Löschen des ersten Zeichens in den Zustand s_1 verzweigt. Dort wird beim Erreichen des rechten Bandbegrenzungssymbols in den akzeptierenden Zustand s_5 gewechselt. Dies geschieht bei der Eingabe $a^{2^0} = a$ bzw. sobald nur noch ein Leerzeichen nach dem mehrfachen Halbieren der ursprünglichen Eingabe verbleibt.

Falls im Zustand s_1 das Bandende nicht erreicht wird, beginnt das abwechselnde Löschen und Ausradieren der restlichen Zeichen in den Zuständen s_2 und s_3. Ist die Länge der Eingabe gerade, verzweigt $\mathcal{A}$ beim Erreichen des rechten Bandbegrenzungssymbols in den Zustand s_4, in dem der Kopf zurück zum Bandanfang geführt wird. Anschließend beginnt $\mathcal{A}$ mit der nächsten Iteration. □

3.3 Bereits bekannte Ergebnisse

In diesem Abschnitt geben wir die Ergebnisse an, die in den Arbeiten [26, 27, 28, 29, 30, 31, 43, 44, 49] erzielt wurden und die die Grundlagen für die weiteren Betrachtungen sind. Die sich daraus ergebenden Äquivalenzen der insgesamt jeweils 63 Automatenmodelle im nichtdeterministischen und im deterministischen Fall werden in den Tabellen 3.1 bzw. 3.2 zusammengefasst. Die meisten der darin aufgelisteten Äquivalenzen ergeben sich durch die direkte Simulation einer fehlenden Operation mithilfe einer Hintereinanderausführung anderer Operationen. So kann beispielsweise die Gleichheit der Familien $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$ und $\mathscr{L}(\mathsf{MV}, \mathsf{ER_L})$ wie folgt begründet werden:

1. Die Inklusion $\mathscr{L}(\mathsf{MV}, \mathsf{ER_L}) \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{ER})$ gilt trivialerweise, da ein $(\mathsf{MV}, \mathsf{ER})$-Automat über alle Operationen verfügt, die ein $(\mathsf{MV}, \mathsf{ER_L})$-Automat ausführen kann.

2. Die Inklusion $\mathscr{L}(\mathsf{MV}, \mathsf{ER}) \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{ER_L})$ gilt, da die fehlende $\mathsf{ER_R}$-Operation bei einem $(\mathsf{MV}, \mathsf{ER_L})$-Automaten durch die Hintereinanderausführung der Operationen $\mathsf{ER_L}$ sowie zweimal $\mathsf{MV_R}$ simuliert werden kann (vergleiche Abbildung 3.2).

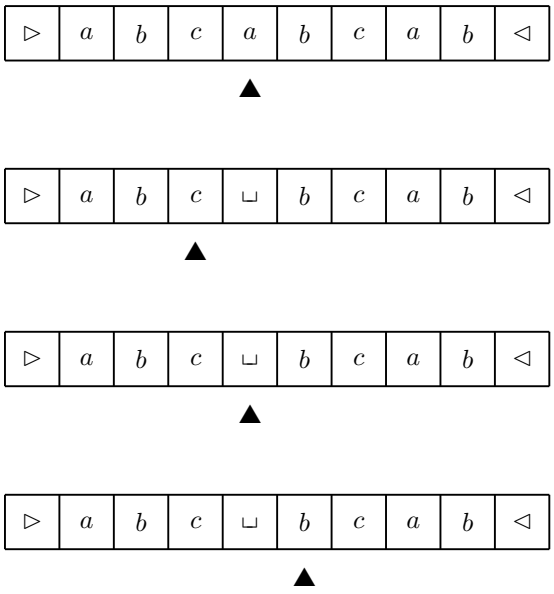


Abbildung 3.2: Simulation der $\mathsf{ER_R}$-Operation durch eine $\mathsf{ER_L}$- und zwei $\mathsf{MV_R}$-Operationen.

Bezeichnung	Äquivalente Modelle
$\mathscr{L}(\mathsf{MV_L})$	$\mathscr{L}(\mathsf{MV_L})$, $\mathscr{L}(\mathsf{ER_L})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_L})$
REG	$\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_L}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_R})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_R})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV_R})$, $\mathscr{L}(\mathsf{ER})$, $\mathscr{L}(\mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}(\mathsf{ER_L}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{ER_R})$, $\mathscr{L}(\mathsf{DL})$, $\mathscr{L}(\mathsf{DL_R})$
$\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$	$\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL_L})$
$\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$	$\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL_L})$
CFL	$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_L})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL_L})$
$\mathscr{L}(\mathsf{ER}, \mathsf{DL})$	$\mathscr{L}(\mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{ER}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{ER}, \mathsf{DL_R})$
$\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$	$\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_R}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER_R}, \mathsf{DL_R})$
$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$	$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_L}, \mathsf{DL_R})$
$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$	$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_L})$
$\mathscr{L}(\mathsf{MV}, \mathsf{DL})$	$\mathscr{L}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV}, \mathsf{DL_R})$
$\mathscr{L}(\mathsf{MV}, \mathsf{ER})$	$\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_L})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_R})$
$\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$	$\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_L}, \mathsf{DL_R})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_R}, \mathsf{DL_L})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER_R}, \mathsf{DL_R})$

Tabelle 3.1: Klassifizierung der nichtdeterministischen Forgetting-Automaten-Modelle.

Bezeichnung	Äquivalente Modelle
$\mathscr{L}_{\det}(\mathsf{MV_L})$	$\mathscr{L}_{\det}(\mathsf{MV_L})$, $\mathscr{L}_{\det}(\mathsf{ER_L})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_L})$
REG	$\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_L}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_R})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV_R})$, $\mathscr{L}_{\det}(\mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER_L}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{ER_R})$, $\mathscr{L}_{\det}(\mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{DL_R})$
$\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL_L})$
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL_L})$
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$	$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_L})$
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL_L})$
$\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL_L})$
$\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL_R})$	$\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL_R})$
$\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_R}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER_R}, \mathsf{DL_R})$
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$	$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_L}, \mathsf{DL_R})$
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_L})$
$\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL_R})$
$\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$	$\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_L})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_R})$
$\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$	$\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_L}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_L}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_L}, \mathsf{DL_R})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_R}, \mathsf{DL_L})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER_R}, \mathsf{DL_R})$

Tabelle 3.2: Klassifizierung der deterministischen Forgetting-Automaten-Modelle.

Die weiteren Ergebnisse – insbesondere die Inklusionsbeziehungen zwischen verschiedenen Sprachfamilien – werden in der Übersicht über die Klassifizierung der nichtdeterministischen und der deterministischen Automatenmodelle in den Abbildungen 3.3 und 3.4 auf Seite 42 dargestellt. Die wichtigsten Ergebnisse diskutieren wir jedoch kurz an dieser Stelle.

Die erste Sprachfamilie oberhalb der Familie der regulären Sprachen wurde durch die Sprache $\{a^n b^n \mid n \in \mathbb{N}\}$ von dieser getrennt; die Inklusion ist durch die mögliche Simulation eines endlichen Automaten klar:

3.3.1 Satz ([29])
$\mathrm{REG} \subset \mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$

Ein $(\mathsf{ER_R}, \mathsf{DL})$-Automat kann durch einen $(\mathsf{MV_R}, \mathsf{DL})$-Automaten simuliert werden, indem die $\mathsf{ER_R}$-Operation durch $\mathsf{MV_R}$ simuliert wird und die Zeichen links vom Kopf als Leerzeichen interpretiert werden. Die Echtheit der Inklusion konnte hier mithilfe der Sprache $\{wcw^R \mid w \in \{a,b\}^*\}$ gezeigt werden:

3.3.2 Satz ([29])
$\mathscr{L}(\mathsf{ER_R}, \mathsf{DL}) \subset \mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$

$(\mathsf{MV_R}, \mathsf{DL})$-Automaten können andererseits durch einen Kellerautomaten simuliert werden; die Sprache

$$\{c^{n_1} a_{i_1} c^{n_1} c^{n_2} a_{i_2} c^{n_2} \cdots c^{n_k} a_{i_k} c^{n_k} s d_{i_k} d_{i_{k-1}} \cdots d_{i_1} \mid k \geq 1, i_j \in \{1,2\}, n_j > 0 \quad \forall 1 \leq j \leq k\}$$

trennt $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$ von der Familie der kontextfreien Sprachen:

3.3.3 Satz ([29])
$\mathscr{L}(\mathsf{MV_R}, \mathsf{DL}) \subset \mathrm{CFL}$

Das wohl bedeutendste Ergebnis ist die Charakterisierung der Familie der kontextfreien Sprachen:

3.3.4 Satz ([27, 30, 31])
$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}) = \mathscr{L}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL}) = \mathrm{CFL}$

Der Nachweis der Gleichheit mittels doppelter Inklusion kann dabei einerseits durch die Simulation von Forgetting-Automaten durch Kellerautomaten, andererseits durch die rückwärtige Abarbeitung der Rechtsableitung einer kontextfreien Grammatik (in einer speziellen Normalform) durch die Forgetting-Automaten erfolgen.

Oberhalb von CFL liegt die Familie $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$, die Sprachen wie $\{a^n b^n c^n \mid n \in \mathbb{N}_0\}$ oder $\{a^{2^n} \mid n \in \mathbb{N}\}$ enthält:

3.3.5 Satz ([29])
$\mathrm{CFL} \subset \mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$

Am oberen Ende der Hierarchie der Forgetting-Automaten liegt die Familie der (MV, ER, DL)-Automaten, also der Automaten, die über alle sechs möglichen Operationen verfügen. Von den Familien $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$ und $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$ wird sie durch die Sprache $\{wcw \mid w \in \{a, b\}^*\}$ getrennt:

3.3.6 Satz ([29])
$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R}) \subseteq \mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL}) \subset \mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$

Wir kehren nun zu der Betrachtung weiter unten in der Hierarchie liegender Sprachfamilien zurück, die bisher nicht eingeordnet wurden. Geht man vom Modell des $(\mathsf{ER_R}, \mathsf{DL})$-Automaten zu dem des (ER, DL)-Automaten über, ist offenkundig, dass die Berechnungsmächtigkeit mindestens gleich bleibt. Tatsächlich kann ein (ER, DL)-Automat nicht-kontextfreie Sprachen wie $\{a^{2^n} \mid n \in \mathbb{N}\}$ erkennen (siehe Beispiel 3.2.5).

3.3.7 Satz ([29])
$\mathscr{L}(\mathsf{ER_R}, \mathsf{DL}) \subset \mathscr{L}(\mathsf{ER}, \mathsf{DL})$

Während die Echtheit der Inklusion $\mathscr{L}(\mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ offen bleibt, erfolgt die Trennung von der obersten Klasse der Hierarchie durch die Sprache $\{wcw^R \mid w \in \{a, b\}^*\}$:

3.3.8 Satz ([29])
$\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$

Analog zur Hinzunahme der $\mathsf{ER_L}$-Operation bei $(\mathsf{ER_R}, \mathsf{DL})$-Automaten wird auch bei $(\mathsf{MV_R}, \mathsf{DL})$-Automaten durch die Hinzunahme der $\mathsf{MV_L}$-Operation das Erkennen von nicht-kontextfreien Sprachen wie $\{a^{2^n} \mid n \in \mathbb{N}\}$ möglich:

3.3.9 Satz ([29])
$\mathscr{L}(\mathsf{MV_R}, \mathsf{DL}) \subset \mathscr{L}(\mathsf{MV}, \mathsf{DL})$

Das Löschen mittels einer DL-Operation kann durch das Ausradieren mit einer entsprechenden ER-Operation und das Überwandern der entstehenden Leerzeichen mit MV-Schritten simuliert werden. Dadurch ergibt sich die folgende Inklusion:

3.3.10 Satz ([29])
$\mathscr{L}(\mathsf{MV}, \mathsf{DL}) \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{ER})$

Aus der Gleichheit $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}) = \mathrm{CFL}$ folgt bei Hinzunahme der $\mathsf{MV_L}$-Operation direkt die Inklusion $\mathrm{CFL} \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{ER})$. Durch die Zeugensprache $\{a^{2^n} \mid n \in \mathbb{N}\} \in \mathscr{L}(\mathsf{MV}, \mathsf{ER})$ folgt zudem die Echtheit der Inklusion. Für den deterministischen Fall wurde in [49] gezeigt, dass alle deterministisch kontextfreien Sprachen bereits durch deterministische $(\mathsf{MV}, \mathsf{DL})$-Automaten erkannt werden können. Als Trennsprache fungiert erneut $\{a^{2^n} \mid n \in \mathbb{N}\}$.

3.3.11 Satz ([49])
$\mathrm{DCFL} \subset \mathscr{L}_{\mathrm{det}}(\mathsf{MV}, \mathsf{DL})$

Um die Berechnungsmächtigkeit der Forgetting-Automaten nach oben abzuschätzen, wurde in [26] die oberste Klasse $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ unter der Familie der deterministisch kontextsensitiven Sprachen eingeordnet und mittels Diagonalisierung von dieser getrennt:

3.3.12 Satz ([26])
$\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL}) \subset \mathrm{DCSL}$

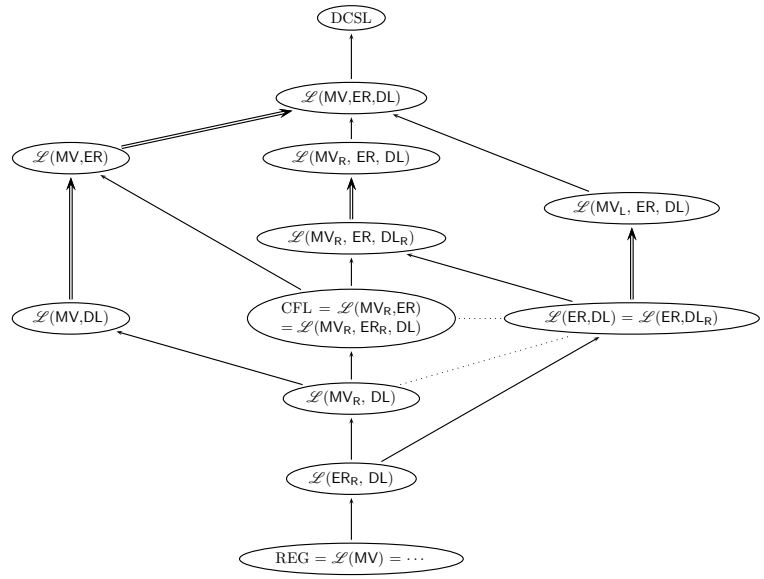


Abbildung 3.3: Die Hierarchie der nichtdeterministischen Forgetting-Automaten auf dem Stand *vor* dieser Arbeit ($\Rightarrow$: Inklusion, $\rightarrow$: echte Inklusion, $\cdots$: unvergleichbar).

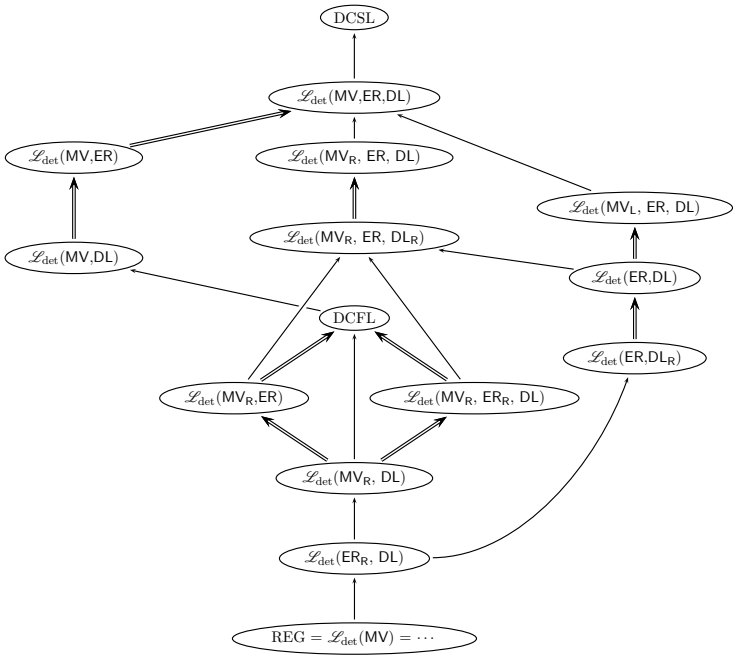


Abbildung 3.4: Die Hierarchie der deterministischen Forgetting-Automaten auf dem Stand *vor* dieser Arbeit.

3.4 Normierung

Für einige Beweise in den folgenden Kapiteln wird es von Vorteil sein, gewisse Annahmen für das Verhalten der Forgetting-Automaten treffen zu können. Daher geben wir in diesem Abschnitt einige Lemmata an, die sich damit beschäftigen, ob und an welcher Position ein Forgetting-Automat hält und welchen Bandinhalt er hinterlässt, falls er akzeptiert.

Wie bereits erwähnt, betrachten wir dabei die Automatenmodelle am unteren Ende der Hierarchie – beziehungsweise die durch sie charakterisierten Sprachfamilien $\mathscr{L}(\mathsf{MV_L})$, $\mathscr{L}_{\text{det}}(\mathsf{MV_L})$ und REG – nicht mehr (siehe Tabellen 3.1 und 3.2 auf Seite 37 f.).

Zunächst wird eine Normalform eingeführt, die die Bandposition beim Halten und die Anzahl der Endzustände betrifft:

3.4.1 Lemma
Bei allen Forgetting-Automaten kann o. B. d. A. vorausgesetzt werden, dass sie auf dem rechten Bandbegrenzungssymbol $\lhd$ halten, wenn sie eine Eingabe akzeptieren. Weiterhin kann angenommen werden, dass es genau einen Endzustand gibt.

Beweis Es sei $\mathcal{A} = \langle S, A, \rhd, \lhd, \sqcup, O, \delta, s_0, F\rangle$ ein beliebiger Forgetting-Automat. Wir simulieren $\mathcal{A}$ mit einem Forgetting-Automaten

$$\mathcal{B} = \langle S \cup \{s', s'', s'''\}, A, \rhd, \lhd, \sqcup, O, \delta', s_0, \{s'''\}\rangle$$

mit $\mathscr{L}(\mathcal{B}) = \mathscr{L}(\mathcal{A})$, der – falls er akzeptiert – auf dem rechten Bandbegrenzungssymbol hält. Zunächst simuliert $\mathcal{B}$ den Automaten $\mathcal{A}$ direkt. Sobald $\mathcal{A}$ einen akzeptierenden Zustand $s \in F$ annimmt und somit hält, geht $\mathcal{B}$ in einen neuen Zustand s' über, in dem er sich zum rechten Bandbegrenzungssymbol $\lhd$ bewegt und dort, mithilfe eines weiteren Hilfszustandes s'', schließlich im (einzigen) Endzustand s''' hält. Die für die Rechtsbewegung im Zustand s' verwendete Operation X_R ist dabei

$$X_R = \begin{cases} \mathsf{MV_R} & \text{falls } \mathsf{MV_R} \in O, \\ \mathsf{ER_R} & \text{sonst.} \end{cases}$$

Man beachte, dass bei den betrachteten Automatenmodellen, die mächtiger als endliche Automaten sind, stets die $\mathsf{MV_R}$- oder die $\mathsf{ER_R}$-Operation zur Verfügung steht (siehe Tabellen 3.1 und 3.2 auf Seite 37 f.).

Für die formale Definition der Überführungsfunktion bedeutet dies:

$$\begin{aligned} &\delta'(s,a) \ni (t,o) && \text{falls } (t,o) \in \delta(s,a), t \notin F \\ &\delta'(s,a) \ni (s',o) && \text{falls } (t,o) \in \delta(s,a), t \in F \end{aligned}$$

(Fortsetzung der Definition)

$$\begin{aligned}
\delta'(s', a) &= \{(s', X_R)\} && \forall a \in A \cup \{\sqcup\} \\
\delta'(s', \rhd) &= \{(s', \mathsf{MV_R})\} \\
\delta'(s', \lhd) &= \{(s'', \mathsf{MV_L})\} \\
\delta'(s'', a) &= \{(s''', X_R)\} && \forall a \in A \cup \{\sqcup\} \\
\delta'(s'', \rhd) &= \{(s''', \mathsf{MV_R})\} \\
\delta'(s''', a) &= \emptyset && \forall a \in A \cup \{\rhd, \lhd, \sqcup\}
\end{aligned}$$

Dabei ist $\mathcal{B}$ genau dann deterministisch, wenn $\mathcal{A}$ es ist. □

Wir haben uns davon überzeugt, dass wir für alle Forgetting-Automaten voraussetzen können, dass sie – falls sie akzeptieren – auf dem rechten Bandbegrenzungssymbol halten. Steht eine DL-Operation zur Verfügung, können wir die Endkonfiguration noch weiter vereinfachen:

3.4.2 Lemma
Bei allen Forgetting-Automaten, die über eine DL-Operation verfügen, kann o. B. d. A. vorausgesetzt werden, dass sie die Eingabe komplett löschen und auf dem rechten Bandbegrenzungssymbol $\lhd$ halten, wenn sie eine Eingabe akzeptieren. Außerdem kann angenommen werden, dass es genau einen Endzustand gibt.

Beweis Es sei $\mathcal{A} = \langle S, A, \rhd, \lhd, \sqcup, O, \delta, s_0, F\rangle$ ein beliebiger Forgetting-Automat. Wir simulieren $\mathcal{A}$ – analog zu Lemma 3.4.1 – mit einem Forgetting-Automaten

$$\mathcal{B} = \langle S \cup \{s', s'', s''', s''''\}, A, \rhd, \lhd, \sqcup, O, \delta', s_0, \{s''''\}\rangle$$

mit $\mathscr{L}(\mathcal{B}) = \mathscr{L}(\mathcal{A})$, der – falls er akzeptiert – die Eingabe komplett löscht und auf dem rechten Bandbegrenzungssymbol hält. $\mathcal{B}$ simuliert zunächst $\mathcal{A}$ und löscht die restliche Eingabe, sobald $\mathcal{A}$ einen akzeptierenden Zustand annimmt. Dies wird wie folgt realisiert:

$$X_L = \begin{cases} \mathsf{DL_L} & \text{falls } \mathsf{DL_L} \in O, \\ \mathsf{ER_L} & \text{sonst.} \end{cases}$$

sowie

$$\begin{aligned}
\delta'(s, a) &\ni (t, o) && \text{falls } (t, o) \in \delta(s, a), t \notin F \\
\delta'(s, a) &\ni (s', o) && \text{falls } (t, o) \in \delta(s, a), t \in F \\
\delta'(s', \lhd) &= \{(s', \mathsf{MV_L})\} \\
\delta'(s', a) &= \{(s', X_L)\} && \forall a \in A \cup \{\sqcup\} \\
\delta'(s', \rhd) &= \{(s'', \mathsf{MV_R})\}
\end{aligned}$$

(Fortsetzung der Definition)

$$\begin{aligned}
\delta'(s'', a) &= \{(s'', \mathsf{DL_R})\} && \forall a \in A \cup \{\sqcup\} \\
\delta'(s'', \lhd) &= \{(s''', \mathsf{MV_L})\} \\
\delta'(s''', \rhd) &= \{(s'''', \mathsf{MV_R})\} \\
\delta'(s'''', a) &= \emptyset && \forall a \in A \cup \{\rhd, \lhd, \sqcup\}
\end{aligned}$$

Die Operationen X_L (also $\mathsf{DL_L}$ oder $\mathsf{ER_L}$) und $\mathsf{DL_R}$ stehen dabei allen betreffenden Automatenmodellen zur Verfügung (siehe Tabellen 3.1 und 3.2 auf Seite 37 f.). Zudem gilt, dass $\mathcal{B}$ genau dann deterministisch ist, wenn $\mathcal{A}$ es ist. □

Die einzigen Automatenmodelle, die nicht über eine DL-Operation verfügen, sind die der $(\mathsf{MV_R}, \mathsf{ER})$- und der $(\mathsf{MV}, \mathsf{ER})$-Automaten. Für diese können wir selbstverständlich nicht annehmen, dass sie ihre Eingabe komplett löschen, falls sie sie akzeptieren. Wir können hier aber das komplette Ausradieren voraussetzen:

3.4.3 Lemma
Bei $(\mathsf{MV_R}, \mathsf{ER})$- und $(\mathsf{MV}, \mathsf{ER})$-Automaten kann o. B. d. A. vorausgesetzt werden, dass sie die Eingabe komplett ausradieren und auf dem rechten Bandbegrenzungssymbol $\lhd$ halten, wenn sie eine Eingabe akzeptieren.

Beweis Analog zum Beweis von Lemma 3.4.2 kann ein simulierender Automat $\mathcal{B}$ die Eingabe zunächst mit $\mathsf{ER_L}$ bis zum linken und anschließend mit $\mathsf{ER_R}$ bis zum rechten Bandbegrenzungssymbol ausradieren. □

Für einige der weiteren Betrachtungen ist es wichtig zu wissen, ob wir von den deterministischen Automatenmodellen annehmen können, dass ihre Berechnungen stets endlich sind, das heißt, dass sie nie in einer Endlosschleife münden.

Eine Endlosschleife tritt bei einer Berechnung auf, wenn eine identische Konfiguration ein zweites Mal erreicht wird, das heißt, wenn ein Konfigurationsübergang $vsw \vdash^* vsw$ erfolgt. In den folgenden Lemmata wird für die jeweiligen Automatenmodelle durch unterschiedliche Methoden sichergestellt, dass eine solche Wiederholung einer Konfiguration nicht auftritt. Somit kann angenommen werden, dass ein entsprechender Automat stets hält.

3.4.4 Lemma
Zu jedem deterministischen $(\mathsf{ER_R}, \mathsf{DL})$-Automaten existiert ein äquivalenter deterministischer $(\mathsf{ER_R}, \mathsf{DL})$-Automat, der auf allen Eingaben hält.

Beweis Es sei ein beliebiger deterministischer ($\mathsf{ER_R}$, DL)-Automat $\mathcal{A}$ gegeben. Nach Lemma 3.4.2 können wir annehmen, dass $\mathcal{A}$ auf dem leeren Band hält, falls er die Eingabe akzeptiert.

Wir konstruieren einen haltenden deterministischen ($\mathsf{ER_R}$, DL)-Automaten $\mathcal{B}$ mit $L(\mathcal{B}) = L(\mathcal{A})$ wie folgt:

Zunächst beginnt $\mathcal{B}$ mit der direkten Simulation von $\mathcal{A}$. Dabei gilt generell, dass, falls $\mathcal{A}$ in einem nicht-akzeptierenden Zustand hält, $\mathcal{B}$ die restliche Eingabe löscht und ebenfalls in einem nicht-akzeptierenden Zustand hält. Somit bleibt zu klären, wie Schleifen verhindert werden können.

Da mit den Operationen $\mathsf{DL_L}$ und $\mathsf{DL_R}$ das Eingabeband stets verkürzt wird, kann $\mathcal{A}$ mit diesen Operationen nicht zu einer bereits vorher aufgetretenen Konfiguration gelangen, das heißt in eine Schleife laufen. Mit der $\mathsf{ER_R}$-Operation können bereits ausradierte Felder ungelöscht nach rechts verlassen werden; eine Linksbewegung, die die Eingabe nicht verkürzt, kann jedoch nur auf dem rechten Bandbegrenzungssymbol ausgeführt werden.

$\mathcal{B}$ simuliert daher die Aufeinanderfolge von endlich vielen, nicht in eine Schleife führenden $\mathsf{ER_R}$- und $\mathsf{MV_L}$-Schritten am rechten Bandende in einer Operation. Da die Anzahl solcher $\mathsf{ER_R}$-Schritte durch die Zahl der Zustände von $\mathcal{A}$ beschränkt ist, können diese Folgen von Schritten direkt an der Überführungsfunktion abgelesen werden. Falls $\mathcal{A}$ mit den $\mathsf{ER_R}$-Schritten hingegen in eine Schleife läuft, löscht $\mathcal{B}$ sofort alle verbliebenen Eingabesymbole und hält in einem nicht-akzeptierenden Zustand.

Nachdem alle Leerzeichen gelöscht wurden und das linke Bandbegrenzungssymbol $\triangleright$ erreicht ist, hält $\mathcal{B}$. Wie zuvor nimmt $\mathcal{B}$ dabei einen nicht-akzeptierenden Zustand an, falls $\mathcal{A}$ in eine Schleife läuft (durch abwechselnde $\mathsf{MV_L}$- und $\mathsf{MV_R}$-Schritte auf den beiden Bandbegrenzungssymbolen) oder nach endlich vielen Schritten in einem nicht-akzeptierenden Zustand hält. Falls $\mathcal{A}$ jedoch einen akzeptierenden Zustand erreicht und hält, geschieht dies auch bei $\mathcal{B}$.

Somit akzeptiert $\mathcal{B}$ die Eingabe genau dann, wenn $\mathcal{A}$ sie akzeptiert. □

Aus dem vorangegangenen Beweis ergibt sich für die Berechnung eines deterministischen ($\mathsf{ER_R}$, DL)-Automaten:

1. Wir können annehmen, dass die Berechnung terminiert, das heißt, dass der Automat stets hält.
2. Wir können annehmen, dass die Eingabe (bis auf die Bandbegrenzungssymbole $\triangleright$ und $\triangleleft$) am Ende der Berechnung komplett gelöscht ist.

3.4.5 Lemma
Zu jedem deterministischen ($\mathsf{MV_R}$, DL)-Automaten existiert ein äquivalenter deterministischer ($\mathsf{MV_R}$, DL)-Automat, der auf allen Eingaben hält.

Beweis Erneut sei ein beliebiger deterministischer $(\mathsf{MV_R}, \mathsf{DL})$-Automat gegeben, der gemäß Lemma 3.4.2 – falls er akzeptiert – auf dem leeren Band hält und den wir mit einem haltenden Automaten des gleichen Typs simulieren wollen. Analog zum Beweis von Lemma 3.4.4 können die restlichen Symbole auf dem Band gelöscht werden, nachdem das rechte Bandbegrenzungssymbol $\lhd$ erreicht wird. In diesem Fall handelt es sich jedoch um Eingabesymbole und nicht um Leerzeichen. Sobald alle Eingabesymbole gelöscht sind und das linke Bandbegrenzungssymbol $\rhd$ erreicht ist, hält der simulierende Automat und akzeptiert genau dann, wenn auch der zu simulierende Automat die Eingabe akzeptiert. □

3.4.6 Lemma
Zu jedem deterministischen $(\mathsf{MV_R}, \mathsf{ER})$-Automaten existiert ein äquivalenter deterministischer $(\mathsf{MV_R}, \mathsf{ER})$-Automat, der auf allen Eingaben hält.

Beweis Es sei ein beliebiger deterministischer $(\mathsf{MV_R}, \mathsf{ER})$-Automat $\mathcal{A}$ gegeben. Nach Lemma 3.4.3 können wir annehmen, dass $\mathcal{A}$ auf dem komplett ausradierten Band hält, falls er die Eingabe akzeptiert.

Zunächst stellen wir die Überlegung an, in welchen Situationen bei einem deterministischen $(\mathsf{MV_R}, \mathsf{ER})$-Automaten eine Endlosschleife auftreten, das heißt, wann ein Konfigurationsübergang $vsw \vdash^* vsw$ erfolgen kann.

In einer Schleife kann kein Ausradieren eines Eingabezeichens (das heißt eines noch nicht ausradierten Zeichens) mehr stattfinden. Es sei daher

$$x = x_1x_2x_3, \text{ mit } x_1 \in A \cup \{\rhd, \sqcup\}, x_2 \in \sqcup^* \text{ und } x_3 \in \{\sqcup, \lhd\},$$

der Teil des Bandes, den $\mathcal{A}$ während eines Schleifendurchlaufs betritt. Wir unterscheiden nun zwei Fälle:

1. x ist an beiden Seiten durch ein Zeichen begrenzt, das *kein* Leerzeichen ist, das heißt $x = x_1x_2x_3 \in (A \cup \{\rhd\})\sqcup^*\lhd$.
2. x ist an mindestens einer Seite durch ein Leerzeichen begrenzt, das heißt $x_1 = \sqcup \vee x_3 = \sqcup$.

Im ersten Fall kann $\mathcal{A}$ in eine Schleife unbeschränkter Länge laufen, das heißt, x_2 kann beliebig groß sein. Bei der Bewegung über x_2 kann $\mathcal{A}$ beliebig oft denselben Zustand annehmen, durch das Erreichen von x_1 bzw. x_3 kann die Bewegungsrichtung jedoch wieder umgekehrt werden.

Im zweiten Fall kann die Anzahl der Schritte, die $\mathcal{A}$ in eine Richtung läuft, nicht beliebig groß sein. Da sich $\mathcal{A}$ hier über eine Folge von Leerzeichen bewegt und schließlich auch auf einem Leerzeichen die Bewegungsrichtung umkehrt, kann die Anzahl der Schritte nicht größer als die Anzahl seiner Zustände sein.

Wir konstruieren nun einen haltenden deterministischen ($\mathsf{MV_R}$, ER)-Automaten $\mathcal{B}$ mit $L(\mathcal{B}) = L(\mathcal{A})$ wie folgt:

Für den ersten Fall kann der simulierende Automat $\mathcal{B}$ die auftretende Schleife dadurch erkennen, dass er sich den Zustand merkt, in dem sich $\mathcal{A}$ beim Erreichen eines Eingabesymbols oder des linken Bandbegrenzungssymbols befindet, bevor dieses nach rechts verlassen wird. Falls $\mathcal{A}$ anschließend über ein Teilwort $\sqcup^*\triangleleft$ nach rechts und wieder zurück läuft, kann $\mathcal{B}$ beim Erreichen des Eingabezeichens bzw. des Bandbegrenzungssymbols (x_1 in obiger Notation) den Zustand mit dem zuvor gespeicherten vergleichen und damit im Falle einer Übereinstimmung das Auftreten einer Schleife erkennen. Da nicht mehr als die (endlich vielen) Zustände von $\mathcal{A}$ gespeichert werden müssen, kann dieser Prozess wiederholt werden, bis $\mathcal{A}$ entweder einen bereits zuvor angenommenen Zustand auf x_1 erreicht oder den Bandinhalt durch eine ER-Operation auf x_1 (für den Fall $x_1 \in A$) verändert. Falls sich ein Zustand wiederholt, kann $\mathcal{B}$ an dieser Stelle halten und somit die Schleife verhindern. In diesem Fall akzeptieren sowohl $\mathcal{B}$ als auch $\mathcal{A}$ die Eingabe nicht.

Für den zweiten Fall kann $\mathcal{B}$ mit der Information über die endlich vielen Teilwörter x ausgestattet werden, auf denen $\mathcal{A}$ in eine Schleife läuft. Bevor $\mathcal{B}$ eine $\mathsf{ER_L}$-Operation auf einem Leerzeichen oder dem rechten Bandbegrenzungssymbol direkt simuliert, lassen wir $\mathcal{B}$ zunächst mit weiteren $\mathsf{ER_L}$-Operationen nach links laufen, um zu testen, ob eines der ihm bekannten, zu einer Schleife führenden Teilwörter auftritt. Da es nur endlich viele solcher Wörter gibt, ist auch die maximale Länge l dieser Wörter endlich. $\mathcal{B}$ muss daher höchstens l Schritte nach links (und wieder zurück) laufen – bis ein Eingabezeichen erreicht wird oder l Leerzeichen überschritten werden – um zu wissen, ob der nächste Schritt in eine Schleife führt.

Durch die Festlegung, das Hineinlaufen in eine Schleife am rechten Rand der entsprechenden Teilwörter zu testen und Felder links von der aktuellen Position zu überprüfen, kann hier mithilfe der $\mathsf{MV_R}$-Operation sichergestellt werden, dass keine Eingabezeichen verändert werden müssen. Falls $\mathcal{A}$ in eine Schleife läuft, kann $\mathcal{B}$ halten und somit die Schleife verhindern.

Insgesamt akzeptiert $\mathcal{B}$ daher genau dann, wenn auch $\mathcal{A}$ die Eingabe akzeptiert. □

3.4.7 Lemma
Zu jedem deterministischen ($\mathsf{MV_R}$, $\mathsf{ER_R}$, DL)-Automaten existiert ein äquivalenter deterministischer ($\mathsf{MV_R}$, $\mathsf{ER_R}$, DL)-Automat, der auf allen Eingaben hält.

Beweis Es sei ein beliebiger deterministischer ($\mathsf{MV_R}$, $\mathsf{ER_R}$, DL)-Automat – der gemäß Lemma 3.4.2 auf dem leeren Band hält, falls er akzeptiert –

gegeben, den wir mit einem haltenden Automaten des gleichen Typs simulieren wollen. Analog zum Beweis der Lemmata 3.4.4 und 3.4.5 können die restlichen Symbole auf dem Band gelöscht werden, nachdem das rechte Bandbegrenzungssymbol erreicht wird. Sobald alle Eingabesymbole gelöscht sind und das linke Bandbegrenzungssymbol erreicht ist, hält der simulierende Automat und akzeptiert genau dann, wenn auch der zu simulierende Automat die Eingabe akzeptiert. □

3.4.8 Lemma
Zu jedem deterministischen (ER, DL)-Automaten existiert ein äquivalenter deterministischer (ER, DL)-Automat, der auf allen Eingaben hält.

Beweis Es sei ein beliebiger deterministischer (ER, DL)-Automat $\mathcal{A}$ gegeben. Nach Lemma 3.4.2 können wir annehmen, dass $\mathcal{A}$ auf dem leeren Band hält, falls er die Eingabe akzeptiert.

Auf ähnliche Weise wie im Beweis von Lemma 3.4.6 betrachten wir das erneute Erreichen einer identischen Konfiguration. Es sei $x = x_1x_2x_3$, mit $x_1 \in \{\rhd, \sqcup\}$, $x_2 \in \sqcup^*$ und $x_3 \in \{\sqcup, \lhd\}$, der Teil des Bandes, den $\mathcal{A}$ während eines Schleifendurchlaufs betritt. Wir unterscheiden erneut zwei Fälle:

1. x ist an beiden Seiten durch ein Bandbegrenzungssymbol begrenzt, das heißt $x_1 = \rhd \wedge x_3 = \lhd$.
2. x ist an mindestens einer Seite durch ein Leerzeichen begrenzt, das heißt $x_1 = \sqcup \vee x_3 = \sqcup$.

Im ersten Fall kann $\mathcal{A}$ in eine Schleife unbeschränkter Länge laufen, im zweiten Fall nicht.

Analog zum Beweis von Lemma 3.4.6 kann daher ein haltender deterministischer Automat $\mathcal{B}$ mit $L(\mathcal{B}) = L(\mathcal{A})$ konstruiert werden. Für den ersten Fall werden die an einem Begrenzungssymbol auftretenden Zustände mitprotokolliert, bis sich ein Zustand wiederholt oder ein DL-Schritt ausgeführt wird. Für den zweiten Fall kann $\mathcal{B}$ das Erreichen einer Schleife auf den endlich vielen dazu führenden Teilwörtern vermeiden. □

3.4.9 Lemma
Zu jedem deterministischen (MV_L, ER, DL)-Automaten existiert ein äquivalenter deterministischer (MV_L, ER, DL)-Automat, der auf allen Eingaben hält.

Beweis Ein (MV_L, ER, DL)-Automat hat im Vergleich zu einem (ER, DL)-Automaten lediglich einen Vorteil: Ein einzelnes Eingabezeichen kann unverändert nach links verlassen werden.

Der Beweis von Lemma 3.4.8 muss deshalb lediglich dahingehend modifiziert werden, dass das Zeichen x_3 nun auch ein Eingabezeichen sein kann. □

Kapitel 4

Die Berechnungsmächtigkeit der Forgetting-Automaten

Nachdem die Hierarchie der Familien der von Forgetting-Automaten akzeptierten Sprachen (auf dem Stand vor dieser Arbeit) in Kapitel 3 vorgestellt wurde, befassen wir uns in diesem Kapitel eingehend mit der Berechnungsmächtigkeit der verschiedenen Automatenmodelle, das heißt der genaueren Eingrenzung der durch die Modelle beschriebenen Sprachfamilien. Grundsätzlich ergeben sich dabei drei mögliche Arten von Ergebnissen:

1. Die Charakterisierung – das heißt die Gleichheit – zweier gegebener Sprachfamilien $\mathscr{L}_1$ und $\mathscr{L}_2$.

   Zumeist geschieht der Nachweis der Gleichheit durch den Beweis der doppelten Inklusion, also durch den Nachweis von $\mathscr{L}_1 \subseteq \mathscr{L}_2$ und $\mathscr{L}_1 \supseteq \mathscr{L}_2$.

2. Die Inklusion einer gegebenen Sprachfamilie in einer anderen.

   Hier wird in der Regel ein Modell durch ein anderes simuliert und – im Falle einer echten Inklusion – eine Zeugensprache angegeben, die zeigt, dass keine Gleichheit der Sprachfamilien bestehen kann.

3. Die Unvergleichbarkeit zweier Sprachfamilien $\mathscr{L}_1$ und $\mathscr{L}_2$.

   Gilt weder $\mathscr{L}_1 \subseteq \mathscr{L}_2$ noch $\mathscr{L}_1 \supseteq \mathscr{L}_2$, sind die beiden Sprachfamilien unvergleichbar.

Um die Berechnungsmächtigkeit der Forgetting-Automaten genauer zu untersuchen, betrachten wir mögliche Charakterisierungen, Inklusionen und Unvergleichbarkeiten der beschriebenen Sprachfamilien sowohl untereinander als auch im Vergleich zu anderen wohlbekannten Modellen der theoretischen Informatik.

4.1 Vergleich mit anderen Modellen

In diesem Abschnitt vergleichen wir die verschiedenen Modelle der Forgetting-Automaten mit aus der Literatur bekannten Automaten- und Grammatikmodellen bzw. Sprachfamilien wie den linearen und metalinearen Sprachen, den deterministisch linearen Sprachen und 1-Zähler-Automaten.

Zunächst betrachten wir die Klassen zwischen den Familien der regulären und der kontextfreien Sprachen genauer und vergleichen sie mit Familien der linearen bzw. metalinearen Sprachen.

4.1.1 Satz
LIN $\subset$ METALIN $\subset \mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$

Beweis

Zu METALIN $\subseteq \mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$:

Gegeben sei eine k-lineare Grammatik $\mathcal{G}$. Ein $(\mathsf{MV_R}, \mathsf{DL})$-Automat kann mithilfe von $\mathsf{MV_R}$-Schritten die Positionen raten, an denen die k linearen Teilpfade der Ableitung schließlich enden, das heißt an denen das jeweilige Nichtterminalsymbol durch eine Zeichenkette ersetzt wird, die ausschließlich aus Terminalsymbolen besteht. An diesen Stellen können die k linearen Ableitungen gemäß den Produktionsregeln der Grammatik sukzessive von innen nach außen (durch Löschen von Symbolen mit $\mathsf{DL_L}$ bzw. $\mathsf{DL_R}$) abgearbeitet werden.

Nach dem Erreichen eines der k Nichtterminalsymbole, die im ersten Ableitungsschritt der Grammatik aus dem Startsymbol erzeugt werden können, bewegt sich der Automat erneut mit $\mathsf{MV_R}$-Schritten nach rechts, um die nächste Position zu raten. Dabei merkt er sich die k Nichtterminalsymbole und kann nach dem Löschen aller Zeichen feststellen, ob sich die gespeicherte Zeichenkette aus Nichtterminalen aus dem Startsymbol der Grammatik ableiten lässt.

Zu METALIN $\neq \mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$:

Die Sprache $L = \left\{w \in \{a,b\}^* \;\middle|\; |w|_a = |w|_b\right\} \in \mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$ ist nicht metalinear. In Beispiel 3.2.2 wurde gezeigt, wie ein $(\mathsf{ER_R}, \mathsf{DL})$-Automat L erkennen kann. Auf analoge Weise kann auch ein $(\mathsf{MV_R}, \mathsf{DL})$-Automat L erkennen; hier werden lediglich – anstelle von Leerzeichen – die jeweils überzähligen Symbole links vom Kopf gespeichert. □

4.1.2 Lemma
LIN und METALIN sind unvergleichbar mit $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$.

Beweis

„$\not\subseteq$“ Die Sprache $\left\{wcw^R \;\middle|\; w \in \{a,b\}^*\right\} \notin \mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ (siehe [29]) ist eine (sogar deterministisch) lineare Sprache.

„$\not\supseteq$“ Die kontextfreie Sprache $\left\{w \in \{a,b\}^* \;\middle|\; |w|_a = |w|_b\right\}$ ist nicht metalinear, kann aber von einem $(\mathsf{ER_R}, \mathsf{DL})$-Automaten $\mathcal{A}$ erkannt werden, wie bereits in Beispiel 3.2.2 gezeigt wurde. □

Im deterministischen Fall vergleichen wir die Familien der von $(\mathsf{ER_R}, \mathsf{DL})$- und von $(\mathsf{MV_R}, \mathsf{DL})$-Automaten erkannten Sprachen ebenfalls mit den entsprechenden deterministisch linearen Sprachen, die durch deterministische 1-Turn-Kellerautomaten charakterisiert werden.

4.1.3 Satz
$\mathrm{DetLIN} \subset \mathscr{L}_{\mathrm{det}}(\mathsf{MV_R}, \mathsf{DL})$

Beweis Es sei ein deterministischer 1-Turn-Kellerautomat $\mathcal{A}$ gegeben. Zunächst wandeln wir $\mathcal{A}$ durch die in [19, Theorem 5.7.1] beschriebene Methode in eine lineare Grammatik $G = \langle N, T, S, P\rangle$ um. Wie in [21] erläutert wurde, führt diese Konstruktion zu einer linearen LR(1)-Grammatik.

Die Verarbeitung eines gegebenen Eingabewortes a durch einen deterministischen $(\mathsf{MV_R}, \mathsf{DL})$-Automaten $\mathcal{B}$ gestalten wir in zwei Phasen: In der ersten Phase simuliert $\mathcal{B}$ den Kellerautomaten $\mathcal{A}$, bis dieser erstmalig eine *Pop-Operation* ausführt, das heißt eine Zustandsüberführung, bei der er das oberste Kellersymbol löscht (also das leere Wort auf den Keller schreibt). In der zweiten Phase werden die Regeln der Grammatik sukzessive angewandt und – gemäß den gegebenen linearen Produktionen – durch Löschen des Eingabewortes von innen nach außen simuliert.

$\mathcal{B}$ kann den ersten Teil der Simulation des Kellerautomaten $\mathcal{A}$ korrekt ausführen, da er dabei lediglich $\mathsf{MV_R}$-Operationen ausführen und nur das jeweils oberste Kellersymbol als Zustandskomponente speichern muss. Sobald $\mathcal{A}$ die erste Pop-Operation ausführt, geht $\mathcal{B}$ zur zweiten Phase der Simulation über und wendet die Ableitungsregeln der Grammatik G in umgekehrter Richtung an. Aus der Konstruktion der Grammatik G folgt, dass die erste Pop-Operation des Kellerautomaten genau an der Position des Eingabewortes auftritt, an der im letzten Ableitungsschritt der linearen Grammatik das Nichtterminalsymbol in Terminalsymbole umgewandelt wird. Ist diese Position erreicht, kann $\mathcal{B}$ schrittweise das Nichtterminalsymbol bestimmen, welches jeweils in der vorangegangenen Satzform im Ableitungsprozess der erzeugenden Grammatik auftritt; aufgrund der Linearität existiert in jeder Satzform jeweils nur ein Nichtterminal und infolge der LR(1)-Bedingung ist das Nichtterminal in der vorherigen Satzform eindeutig bestimmt.

Um die LR(1)-Bedingung ausnutzen zu können, liest $\mathcal{B}$ das folgende Terminalsymbol mithilfe einer $\mathsf{MV_R}$- und einer anschließenden $\mathsf{DL_L}$-Operation; das

gelesene Symbol $a \in T$ speichert $\mathcal{B}$ in seinen Zuständen ab. Da die rechten Seiten der Produktionen in P maximal die Länge 2 haben (vergleiche die Konstruktion von G in [21]), gibt es nur Produktionen der Formen

$$A \to B, \qquad A \to a'B \quad \text{und} \quad A \to Ba$$

(mit $A, B \in N$ und $a, a' \in T$). $\mathcal{B}$ kennt somit den Teil $a'Ba$ der aktuellen Satzform $ua'Baw_1$ und kann daher aufgrund der LR(1)-Bedingung das vorangegangene Nichtterminal A eindeutig bestimmen, für das je nach Form der anzuwendenden Produktionsregel gilt (vergleiche Definition 2.5.1):

$$ua'Aaw_1 \Rightarrow ua'Baw_1, \quad uAaw_1 \Rightarrow ua'Baw_1 \text{ oder } ua'Aw_1 \Rightarrow ua'Baw_1.$$

Auf diese Weise kann $\mathcal{B}$ schrittweise die passenden Symbole löschen, die (mögliche) Ableitung der Grammatik G zurückverfolgen und schließlich die Eingabe akzeptieren, falls das Startsymbol S erreicht wird.

Folglich gilt DetLIN $\subseteq \mathscr{L}_{\text{det}}(\mathsf{MV_R}, \mathsf{DL})$. Die Inklusion ist echt, da die Sprache $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\} \in \mathscr{L}_{\text{det}}(\mathsf{MV_R}, \mathsf{DL})$ nicht (deterministisch) linear ist – vergleiche Beispiel 3.2.2. □

Beim Vergleich der Sprachfamilien zwischen REG und DCFL mit der Familie der deterministisch linearen Sprachen ergeben sich ähnliche Ergebnisse wie im nichtdeterministischen Fall. Wie auch im nichtdeterministischen Fall ist die Familie der von $(\mathsf{ER_R}, \mathsf{DL})$-Automaten erkannten Sprachen unvergleichbar mit der Familie der entsprechenden linearen Sprachen:

4.1.4 Lemma
DetLIN ist unvergleichbar mit $\mathscr{L}_{\text{det}}(\mathsf{ER_R}, \mathsf{DL})$.

Beweis

„$\not\subseteq$“ Die Sprache $\{wcw^R \mid w \in \{a,b\}^*\} \notin \mathscr{L}_{\text{det}}(\mathsf{ER_R}, \mathsf{DL})$ (siehe [29]) ist eine deterministisch lineare Sprache.

„$\not\supseteq$“ Die Sprache $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\} \in \mathscr{L}_{\text{det}}(\mathsf{ER_R}, \mathsf{DL})$ ist nicht (deterministisch) linear – vergleiche Beispiel 3.2.2. □

In [29] wurde das interessante Ergebnis erzielt, dass $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ mit der Familie der kontextfreien Sprachen übereinstimmt. Entfernen wir die $\mathsf{MV_R}$-Operation, erhalten wir eine andere wichtige Sprachfamilie, nämlich die Familie der von (unidirektionalen) 1-*Zähler-Automaten* erkannten Sprachen.

Ein 1-Zähler-Automat (1CA, *one-way one-counter automaton*) ist ein Kellerautomat, der in seinem Keller außer dem Kellerendesymbol $\perp$ nur noch *ein* weiteres Symbol verwenden darf. Die im Keller gespeicherte Information

kann daher mit der Länge der gespeicherten unären Zeichenkette gleichgesetzt werden. Der Keller entspricht in diesem Fall also einfach einem *Zähler*, in dem Zahlen aus $\mathbb{N}_0$ abgespeichert werden. Weiterführende Informationen über Zählerautomaten können beispielsweise dem Lehrbuch [2] oder den Originalarbeiten [5, 8, 9, 40] entnommen werden.

Interessanterweise stimmt das Modell des 1-Zähler-Automaten in der Berechnungsstärke mit dem Modell des $(\mathsf{ER_R}, \mathsf{DL})$-Automaten überein:

4.1.5 Satz
$\mathscr{L}(\mathsf{ER_R}, \mathsf{DL}) = \mathscr{L}(1\mathrm{CA})$

Beweis

„$\subseteq$“ Da $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL}) = \mathscr{L}(\mathsf{ER_R}, \mathsf{DL_L})$ gilt ($\mathsf{DL_R}$ kann in diesem Fall durch die Hintereinanderausführung von $\mathsf{DL_L}$ und $\mathsf{ER_R}$ ersetzt werden, siehe Kapitel 3), können wir bei der Simulation eines $(\mathsf{ER_R}, \mathsf{DL})$-Automaten davon ausgehen, dass lediglich die Operationen $\mathsf{DL_L}$ und $\mathsf{ER_R}$ verwendet werden.

Des Weiteren gehen wir nach Lemma 3.4.4 davon aus, dass ein gegebener $(\mathsf{ER_R}, \mathsf{DL})$- bzw. $(\mathsf{ER_R}, \mathsf{DL_L})$-Automat eine Eingabe durch das Annehmen eines Endzustandes auf dem rechten Bandbegrenzungssymbol bei komplett gelöschter Eingabe akzeptiert.

Es sei $\mathcal{A}$ nun ein beliebiger $(\mathsf{ER_R}, \mathsf{DL_L})$-Automat mit den genannten Eigenschaften. Ein 1-Zähler-Automat $\mathcal{B}$ kann $\mathcal{A}$ wie folgt simulieren:

Jede Operation von $\mathcal{A}$ wird durch eine Folge von Operationen von $\mathcal{B}$ simuliert. Dabei dient der Zähler von $\mathcal{B}$ zum Mitzählen der während der Simulation links vom Lesekopf befindlichen Leerzeichen auf dem Eingabeband des Forgetting-Automaten $\mathcal{A}$.

Da $\mathcal{A}$ jedes Element seiner Eingabe im unausradierten Zustand nur einmal lesen kann (alle möglichen Operationen löschen die Eingabe oder radieren sie aus), kann er durch einen unidirektional arbeitenden Automaten simuliert werden, falls dieser stets die Position des ersten bisher ungelesenen Eingabezeichens und die Anzahl der momentan auf dem Band befindlichen Leerzeichen mitverfolgen kann. Ersteres wird von $\mathcal{B}$ durch die Position seines eigenen Lesekopfes bewerkstelligt, die Anzahl der Leerzeichen – wie oben erwähnt – durch den Zählerstand.

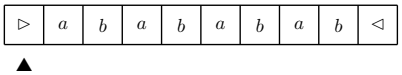
Der Lesekopf von $\mathcal{A}$ befindet sich während der Berechnung stets auf einer der folgenden Positionen (siehe auch Abbildung 4.1):

(a) linkes Bandbegrenzungssymbol,

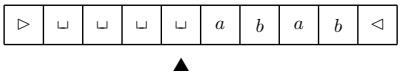
(b) das am weitesten rechts liegende Leerzeichen,

(c) das am weitesten links liegende Eingabezeichen,

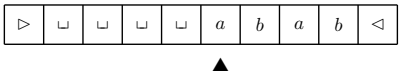
(d) rechtes Bandbegrenzungssymbol.



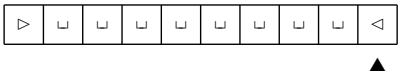
(a) Der Lesekopf steht auf dem linken Bandbegrenzungssymbol.



(b) Der Lesekopf steht auf dem am weitesten rechts liegenden Leerzeichen.



(c) Der Lesekopf steht auf dem am weitesten links liegenden Eingabezeichen.



(d) Der Lesekopf steht auf dem rechten Bandbegrenzungssymbol.

Abbildung 4.1: Die möglichen Positionen des Lesekopfes von $\mathcal{A}$ während einer Berechnung.

Da sich der Lesekopf von $\mathcal{B}$ während der Simulation jedoch immer auf einem Eingabesymbol befindet, muss er die Fälle (a), (b) und (d) mithilfe von λ-Übergängen bewerkstelligen. Zu diesem Zweck stellt $\mathcal{B}$ bei jedem Simulationsschritt fest, ob sich $\mathcal{A}$ im nächsten Schritt auf einem Eingabezeichen (also auf dem am weitesten links liegenden) befinden wird oder nicht; dies kann anhand der zu simulierenden Operation und des Zählerstandes ermittelt werden, wie im Folgenden gezeigt wird. $\mathcal{B}$ merkt sich diese Tatsache ($\mathcal{A}$ liest ein ausradiertes Zeichen oder ein Eingabesymbol) mithilfe seiner Zustände und handelt im nächsten Schritt entsprechend. Dabei kann Fall (a) anhand des Zählerstandes erkannt werden, Fall (d) ist separat zu behandeln (siehe unten).

Die Simulation der Operation von $\mathcal{A}$ geschieht im Einzelnen wie folgt:

$\mathsf{ER_R}$ Simuliert $\mathcal{B}$ das Lesen eines Eingabezeichens, geschieht Folgendes:

$\mathcal{B}$ erhöht den Zählerstand um eins und simuliert im nächsten Schritt wiederum das Lesen eines Eingabezeichens.

Simuliert $\mathcal{B}$ das Lesen eines Leerzeichens, geschieht Folgendes:

Mithilfe eines λ-Überganges simuliert $\mathcal{B}$ den Zustandsübergang von $\mathcal{A}$ und simuliert im nächsten Schritt das Lesen eines Eingabezeichens. Der Zählerstand bleibt dabei unverändert.

$\mathsf{DL_L}$ Simuliert $\mathcal{B}$ das Lesen eines Eingabezeichens, geschieht Folgendes:

Das Eingabezeichen wird gelesen; der Zählerstand bleibt unverändert. Ist der Zählerstand größer als null, simuliert $\mathcal{B}$ im nächsten Schritt das Lesen eines Leerzeichens. Ist der Zählerstand jedoch gleich null,

stößt $\mathcal{A}$ im nächsten Schritt auf das linke Bandbegrenzungssymbol und bewegt sich von dort zurück auf das erste Symbol der Eingabe. $\mathcal{B}$ führt die Simulation dieser beiden Schritte in einer Operation durch und simuliert im darauf folgenden Schritt wiederum das Lesen eines Eingabezeichens.

Simuliert $\mathcal{B}$ das Lesen eines Leerzeichens, geschieht Folgendes:

Mithilfe eines λ-Überganges simuliert $\mathcal{B}$ den Zustandsübergang von $\mathcal{A}$ und vermindert den Zählerstand um eins. Anschließend testet $\mathcal{B}$ den Zählerstand auf null und legt den Simulationsmodus für den nächsten Schritt entsprechend fest: Ist der Zählerstand null ($\mathcal{A}$ hat in diesem Fall das letzte verbliebene Leerzeichen gelöscht und wird mit den nächsten Schritten auf das linke Bandbegrenzungssymbol und von dort zurück auf das erste Eingabesymbol gelangen), simuliert $\mathcal{B}$ zunächst die folgende Operation auf dem linken Bandbegrenzungssymbol $\triangleright$ mit und simuliert im nächsten Schritt wieder das Lesen eines Eingabezeichens. Ist der Zählerstand größer als null ($\mathcal{A}$ steht nach diesem Schritt erneut auf einem Leerzeichen), simuliert $\mathcal{B}$ im nächsten Schritt abermals das Lesen eines Leerzeichens.

Schließlich bleibt noch das Verhalten von $\mathcal{B}$ zu klären, welches für die Simulation des Erreichens des rechten Bandbegrenzungssymbols durch $\mathcal{A}$ nötig ist. Da die Eingabe von $\mathcal{B}$ nicht durch Bandbegrenzungssymbole eingefasst ist, konstruieren wir $\mathcal{B}$ derart, dass das Eingabeende mithilfe von nichtdeterministischen Schritten geraten und die Simulation der von $\mathcal{A}$ beim Auftreffen auf das rechte Bandbegrenzungssymbol ausgeführten Schritte mithilfe von λ-Übergängen realisiert wird. Zu diesem Zweck führen wir von jedem bisherigen Zustand einen zusätzlichen λ-Übergang in eine „Test-Routine“ ein, in deren (neu hinzuzufügenden) Zuständen nur noch λ-Übergänge möglich sind. Auf diese Weise kann $\mathcal{B}$ die Eingabe nicht komplett lesen (und akzeptieren), falls das Eingabeende falsch geraten wird. Wird das Ende jedoch korrekt geraten, führt $\mathcal{B}$ die im Folgenden beschriebenen Schritte zur Simulation des Verhaltens von $\mathcal{A}$ aus und nimmt schließlich genau dann einen akzeptierenden Zustand an, wenn auch $\mathcal{A}$ einen akzeptierenden Zustand erreicht.

Trifft $\mathcal{A}$ auf das rechte Bandbegrenzungssymbol, ist der verbliebene Bandinhalt von der Form $\triangleright\sqcup^k\triangleleft$, wobei $k \in \mathbb{N}_0$. $\mathcal{A}$ bewegt sich zunächst in jedem Fall ein Feld nach links; auf dem letzten der k Leerzeichen stehend, kann $\mathcal{A}$ lediglich die noch verbliebene Zeichenkette $\sqcup^k$ abbauen. Die möglichen Operationen von $\mathcal{A}$ werden nun von $\mathcal{B}$ wie folgt (ausschließlich mit λ-Übergängen) simuliert:

$\mathsf{ER_R}$ Durch $\mathsf{ER_R}$ ändert $\mathcal{A}$ den Inhalt des bereits ausradierten Elementes nicht; durch die Rechtsbewegung erreicht $\mathcal{A}$ zunächst das rechte

Bandbegrenzungssymbol und bewegt sich von dort im nächsten Schritt wieder auf das ursprüngliche Element zurück. $\mathcal{B}$ simuliert dies durch den aus den zwei Schritten resultierenden Zustandsübergang.

DL_L Durch DL_L verkürzt $\mathcal{A}$ das Band von $\triangleright\sqcup^k\triangleleft$ auf $\triangleright\sqcup^{k-1}\triangleleft$; der Lesekopf steht anschließend wiederum auf dem letzten Leerzeichen. $\mathcal{B}$ simuliert dies durch Vermindern des Zählers. Das Erreichen des linken Bandbegrenzungssymbols – und damit das komplette Löschen der Eingabe – erkennt $\mathcal{B}$ durch Prüfen des Zählers auf null.

Schließlich akzeptiert $\mathcal{B}$ die Eingabe beim Herunterfallen vom Band genau dann, wenn auch $\mathcal{A}$ – nach dem kompletten Löschen der Eingabe – auf dem rechten Bandbegrenzungssymbol hält und akzeptiert.

„$\supseteq$“ Es sei $\mathcal{A}$ ein beliebiger 1-Zähler-Automat. Nach [18] können wir annehmen, dass dieser λ-frei ist, das heißt keine λ-Übergänge ausführt. $\mathcal{A}$ macht folglich pro Eingabezeichen nur einen Überführungsschritt, bei dem ein neuer Zustand angenommen und eine Zeichenkette auf den Keller geschrieben wird. An der Überführungsfunktion ist dabei die längste Zeichenkette ablesbar, die bei einer Überführung auf den Keller geschrieben werden kann. Ist k die Länge dieser Zeichenkette, so kann die Länge des Kellerinhalts, das heißt des Zählerstandes, in jedem Schritt maximal um k erhöht werden. Diese Tatsache nutzen wir aus, um $\mathcal{A}$ mithilfe eines (ER_R, DL)-Automaten $\mathcal{B}$ zu simulieren.

Um die Funktion des Zählers von $\mathcal{A}$ zu simulieren, muss $\mathcal{B}$ sowohl den Zählerstand während der Berechnung als auch die Information, ob der Zählerstand null oder größer als null ist, mitführen. Da der Zählerstand jedoch die Länge der bereits gelesenen Eingabe überschreiten kann, kann $\mathcal{B}$ den Zählerstand nicht direkt durch ausradierte Zeichen repräsentieren. $\mathcal{B}$ speichert deshalb in seinen Zuständen den Zählerstand modulo k ab und verwendet jeweils ein ausradiertes Eingabezeichen für die Darstellung von k. Tritt ein „Überlauf“ des internen Zählerstandes ein (die Erhöhung des Zählerstandes führt zu einem internen Zählerstand größer oder gleich k), radiert $\mathcal{B}$ ein weiteres Eingabezeichen aus; tritt ein „Unterlauf“ ein (die Verminderung des Zählerstandes führt zu einem internen Zählerstand kleiner als null), löscht $\mathcal{B}$ ein Leerzeichen aus. Um dabei einen Zählerstand von $l \cdot k$ mit $l > 0$ vom Zählerstand null unterscheiden und somit den Nulltest von $\mathcal{A}$ simulieren zu können, führt $\mathcal{B}$ zudem in seinen Zuständen die Information mit, ob sich mindestens ein ausradiertes Zeichen auf dem Band befindet. Diese Information bezeichnen wir im Folgenden als Variable X, deren Wert WAHR (es befinden sich ausradierte Zeichen auf dem Band) oder FALSCH sein kann. Zu Beginn der Simulation ist X auf FALSCH gesetzt; nach dem Löschen eines Leerzeichens wird der Wert jeweils neu bestimmt (siehe unten). Den Nulltest von $\mathcal{A}$ kann $\mathcal{B}$ nun dadurch simulieren, dass er überprüft, ob X den Wert FALSCH hat und der interne Zählerstand null ist.

Die Zustandsüberführungsfunktion von $\mathcal{A}$ ist von der Form:

$$\delta : S \times A \times \Gamma \to S \times \Gamma^{\leq k}$$

Durch die Simulation des Zustandes, das sequentielle Lesen der Eingabe (wie bei $\mathcal{A}$) und die Mitführung des Zählerstandes sowie der Variablen X kann $\mathcal{B}$ die Zustandsüberführungsfunktion δ direkt nachbilden. Einzig die korrekte Simulation des Zählerstandes und die Bestimmung des Wertes von X sind noch genauer zu beschreiben:

Tritt durch die Veränderung des Zählerstandes kein Über- oder Unterlauf des internen Zählers ein, geht $\mathcal{B}$ lediglich durch eine $\mathsf{DL_R}$-Operation zum Lesen des nächsten Eingabezeichens über und führt die interne Zählerstandsänderung aus.

Falls ein Überlauf des internen Zählers eintritt, erhöht $\mathcal{B}$ die Anzahl der Leerzeichen auf dem Band durch das Ausführen einer $\mathsf{ER_R}$-Operation und speichert den internen Zählerstand modulo k reduziert ab. Da sich nun in jedem Fall mindestens ein Leerzeichen auf dem Band befindet, wird X auf WAHR gesetzt.

Tritt ein Unterlauf des internen Zählers ein, werden durch zwei aufeinanderfolgende $\mathsf{DL_L}$-Operationen sowohl das gelesene Eingabezeichen als auch das infolge des Unterlaufes zu löschende Leerzeichen entfernt. Anschließend befindet sich der Lesekopf von $\mathcal{B}$ auf dem am weitesten rechts liegenden Leerzeichen (falls zuvor mindestens zwei Leerzeichen auf dem Band standen) oder auf dem linken Bandbegrenzungssymbol. Im ersten Fall belässt $\mathcal{B}$ den Wert von X bei WAHR und bewegt sich mit einer $\mathsf{ER_R}$-Operation (diese entspricht hier $\mathsf{MV_R}$, da sich $\mathcal{B}$ schon auf einem Leerzeichen befindet) zum nächsten zu lesenden Eingabezeichen. Im zweiten Fall setzt $\mathcal{B}$ den Wert von X auf FALSCH und bewegt sich mit der obligatorischen $\mathsf{MV_R}$-Bewegung auf das erste Eingabezeichen. In beiden Fällen wird zudem der interne Zählerstand modulo k aktualisiert.

Auf die beschriebene Weise kann $\mathcal{B}$ jeden Schritt von $\mathcal{A}$ simulieren und nimmt schließlich genau dann auf dem rechten Bandbegrenzungssymbol einen akzeptierenden Zustand an, wenn $\mathcal{A}$ nach Lesen des letzten Eingabezeichens in einem akzeptierenden Zustand hält. □

Um den Satz auch für den deterministischen Fall beweisen zu können, benötigen wir das folgende Lemma. Bei einem deterministischen 1-Zähler-Automaten können wir zwar nicht auf λ-Übergänge verzichten, aber eine lineare obere Schranke für die Häufigkeit ihrer Verwendung angeben.

4.1.6 Lemma
Zu jedem deterministischen 1-Zähler-Automaten $\mathcal{A}$ existiert ein deterministischer 1-Zähler-Automat $\mathcal{B}$ mit der Eigenschaft, dass nach dem Lesen von n Eingabezeichen der Zählerstand maximal $k \cdot n$ (für ein festes $k \in \mathbb{N}$) beträgt, sodass $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{B})$ gilt.

Beweis Ist bei Automat $\mathcal{A}$ – ausgehend von einem bestimmten Zustand – eine Folge von λ-Übergängen möglich, kann diese entweder

- in eine Schleife von Zuständen $s_1, s_2, \ldots, s_l, s_1, \ldots$ münden oder
- letztlich in einen Zustand übergehen, für den wiederum eine Zustandsüberführung für ein Eingabezeichen definiert ist,

da im deterministischen Fall in einem Zustand nicht beide Arten der Überführung möglich sind. Wir betrachten nun nur $\delta(\,.\,, \lambda, x)$ mit $x \neq \perp$ (das heißt nur die Zustandsüberführungen, die einen nichtleeren Keller betreffen) und suchen nach den zwei genannten Fällen von Übergangsfolgen. Da die Zustandsüberführungsfunktion von $\mathcal{A}$ deterministisch ist, können wir sie leicht auf solche Schleifen untersuchen.

Für die Erkennung einer nicht endenden Wiederholung von λ-Übergängen ist es wesentlich, ob der Zählerstand im Verlauf eines Schleifendurchlaufs $s_1, s_2, \ldots, s_l, s_1$ mindestens gleich bleibt oder vermindert wird. Wird der Zählerstand vermindert, führt die Wiederholung der Zustände nicht zu einer Endlosschleife des Automaten, da letztlich der Zählerstand null wird und ein anderer Zustandsübergang greift. Wird der Zählerstand jedoch erhöht oder bleibt er gleich, können wir einen bestimmten Zustandsübergang entfernen, das heißt auf undefiniert setzen und somit die Berechnung terminieren lassen, ohne die akzeptierte Sprache des Automaten zu verändern: Wir betrachten einen (nicht notwendigerweise eindeutigen) Zustand s_i, $1 \leq i \leq l$, bei dem der Zählerstand beim einmaligen Durchlaufen der Schleife minimal ist. Dann können wir den Übergang von s_i nach s_{i+1} herausnehmen. Ein Zählerstand von null kann sich nach dem Erreichen von s_i nicht mehr einstellen (siehe Abb. 4.2), wenn er bis dahin nicht aufgetreten ist. Daher kann $\mathcal{A}$ die Schleife nicht mehr verlassen und somit keine weiteren Eingabezeichen lesen.

Durch die Entfernung der genannten Zustandsübergänge in $\mathcal{A}$ erhalten wir den Automaten $\mathcal{B}$ mit den gewünschten Eigenschaften und können die obere Schranke k für die Erhöhung des Zählerstandes pro Eingabezeichen bestimmen. Die Länge einer Folge von λ-Übergängen – bzw. die Anzahl der Zeichen, die während dieser auf den Keller geschrieben werden – kann anhand der Zustandsüberführungsfunktion von $\mathcal{B}$ ermittelt werden. Machen wir dies für alle möglichen Zustände des Automaten, erhalten wir auf diese Weise die Zahl k als maximale Anzahl von Zeichen, die während einer Folge

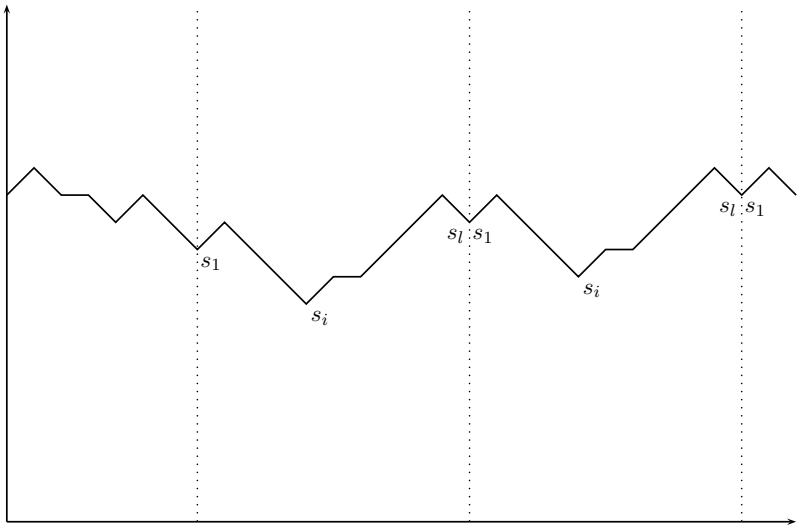


Abbildung 4.2: Veränderung des Zählerstandes während einer Folge von λ-Übergängen.

von λ-Übergängen auf den Keller geschrieben werden kann (das heißt als maximale Erhöhung des Zählerstandes in einer Folge), plus der maximalen Anzahl von Zeichen, die bei einem anschließenden „normalen“ Übergang (kein λ-Übergang) geschrieben werden kann. □

4.1.7 Satz
$\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL}) = \mathscr{L}_{\det}(1\mathrm{CA})$

Beweis

„$\subseteq$“ Die Simulation eines deterministischen $(\mathsf{ER_R}, \mathsf{DL_L})$-Automaten $\mathcal{A}$ durch einen 1-Zähler-Automaten $\mathcal{B}$ erfolgt im Wesentlichen wie im nichtdeterministischen Fall. Lediglich das Verhalten beim Auftreffen des zu simulierenden Automaten auf das rechte Bandbegrenzungssymbol kann nicht analog behandelt werden, da das Eingabeende im vorherigen Beweis nichtdeterministisch „geraten“ wurde; auf die dort beschriebene Verarbeitung der restlichen Leerzeichen können wir im deterministischen Fall jedoch verzichten, indem wir schon während des ersten Teils der Berechnung das Ergebnis des zweiten Teils mitberechnen.

$\mathcal{A}$ hat beim Auftreffen auf das rechte Bandbegrenzungssymbol lediglich eine unäre Eingabe zu verarbeiten, die sich links vom Lesekopf befindet. Mit den ihm zur Verfügung stehenden Operationen $\mathsf{ER_R}$ und $\mathsf{DL_L}$ kann $\mathcal{A}$ lediglich

testen, ob das verbliebene Wort in einer gewissen regulären Sprache enthalten ist. Dies kann folgendermaßen begründet werden:

1. Diese Sprache ist regulär, da jede unäre kontextfreie Sprache regulär ist und $\mathscr{L}_{\text{det}}(\mathsf{ER_R}, \mathsf{DL}) \subset \text{CFL}$ gilt.

2. Das Verhalten von $\mathcal{A}$ kann durch einen endlichen Automaten simuliert werden. $\mathsf{ER_R}$ verändert die Eingabe nicht und führt (mit anschließender $\mathsf{MV_L}$-Operation) lediglich zu einer Zustandsänderung, die in einem Schritt simuliert werden kann; $\mathsf{DL_L}$ führt zum „Verbrauchen“ eines Leerzeichens. Im Wesentlichen wird also die Zeichenkette nur einmal von rechts nach links gelesen.

   Um dies zu verdeutlichen, betrachten wir ein Beispiel. Nehmen wir an, dass

   $$\mathcal{A} = \langle \{s_0, \ldots, s_6\}, A, \rhd, \lhd, \sqcup, \{\mathsf{ER_R}, \mathsf{DL_L}\}, \delta, s_0, \{s_6\} \rangle$$

   gilt und die Überführungsfunktion δ auf den Zeichen $\rhd$, $\sqcup$ und $\lhd$ wie folgt definiert ist:

Zustand	Eingabe	Folgezustand	Operation
s_0	$\rhd$	s_1	$\mathsf{MV_R}$
s_0	$\sqcup$	undefiniert	
s_0	$\lhd$	s_5	$\mathsf{MV_L}$
s_1	$\rhd$	undefiniert	
s_1	$\sqcup$	s_4	$\mathsf{DL_L}$
s_1	$\lhd$	s_3	$\mathsf{MV_L}$
s_2	$\rhd$	undefiniert	
s_2	$\sqcup$	s_0	$\mathsf{ER_R}$
s_2	$\lhd$	undefiniert	
s_3	$\rhd$	undefiniert	
s_3	$\sqcup$	s_2	$\mathsf{DL_L}$
s_3	$\lhd$	undefiniert	
s_4	$\rhd$	s_6	$\mathsf{MV_R}$
s_4	$\sqcup$	s_5	$\mathsf{DL_L}$
s_4	$\lhd$	undefiniert	
s_5	$\rhd$	s_6	$\mathsf{MV_R}$
s_5	$\sqcup$	s_1	$\mathsf{DL_L}$
s_5	$\lhd$	undefiniert	
s_6	$\rhd$	undefiniert	
s_6	$\sqcup$	undefiniert	
s_6	$\lhd$	undefiniert	

Erreicht $\mathcal{A}$ das rechte Bandbegrenzungssymbol im Zustand s_1 mit drei verbliebenen Leerzeichen auf dem Band, durchläuft er die folgenden Konfigurationen:

$$\begin{aligned}
&\rhd \sqcup\sqcup\sqcup s_1 \lhd \\
\vdash\ &\rhd \sqcup\sqcup s_3 \sqcup \lhd \\
\vdash\ &\rhd \sqcup s_2 \sqcup \lhd \\
\vdash\ &\rhd \sqcup\sqcup s_0 \lhd \\
\vdash\ &\rhd \sqcup s_5 \sqcup \lhd \\
\vdash\ &\rhd s_1 \sqcup \lhd \\
\vdash\ &s_4 \rhd \lhd \\
\vdash\ &\rhd s_6 \lhd
\end{aligned}$$

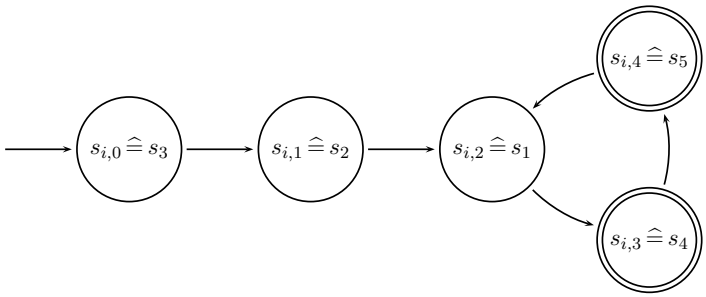
$\mathcal{A}$ akzeptiert also die Eingabe in diesem Fall. Sind stattdessen beim Erreichen des rechten Bandbegrenzungssymbols fünf Leerzeichen verblieben, findet – ebenfalls ausgehend vom Zustand s_1 – die folgende weitere Berechnung statt:

$$\begin{aligned}
&\rhd \sqcup\sqcup\sqcup\sqcup\sqcup s_1 \lhd \\
\vdash\ &\rhd \sqcup\sqcup\sqcup\sqcup s_3 \sqcup \lhd \\
\vdash\ &\rhd \sqcup\sqcup\sqcup s_2 \sqcup \lhd \\
\vdash\ &\rhd \sqcup\sqcup\sqcup\sqcup s_0 \lhd \\
\vdash\ &\rhd \sqcup\sqcup\sqcup s_5 \sqcup \lhd \\
\vdash\ &\rhd \sqcup\sqcup s_1 \sqcup \lhd \\
\vdash\ &\rhd \sqcup s_4 \sqcup \lhd \\
\vdash\ &\rhd s_5 \sqcup \lhd \\
\vdash\ &s_1 \rhd \lhd
\end{aligned}$$

In diesem Fall hält $\mathcal{A}$ im Zustand s_1 und akzeptiert die Eingabe somit nicht. Allgemein lässt sich feststellen, dass ausgehend von einer Konfiguration $\rhd \sqcup^n \lhd$ ein akzeptierender Zustand erreicht wird, falls

$$n \in \{3, 4, 6, 7, 9, 10, \ldots\} = 3\mathbb{N} \cup (3\mathbb{N} + 1)$$

ist. Ein DFA $\mathcal{C}_1$, der das Verhalten des $(\mathsf{ER_R}, \mathsf{DL_L})$-Automaten nach dem Erreichen des rechten Bandbegrenzungssymbols simuliert, kann daher wie folgt angegeben werden:



Damit kehren wir zur Betrachtung des allgemeinen Falls zurück. Anstatt die Berechnung auf der unären Eingabe am Ende auszuführen (ist s_i der Zustand, in dem die Berechnung beginnt, so sei das Verhalten durch einen DFA $\mathcal{C}_i$ gegeben), führt $\mathcal{B}$ alle möglichen Berechnungen der Automaten $\mathcal{C}_i$ in jedem Schritt im Voraus parallel aus und nimmt genau dann einen akzeptierenden Zustand an, wenn er in einen Zustand s_i wechselt und der DFA $\mathcal{C}_i$ den jeweils vorliegenden Zählerstand zum gegebenen Zeitpunkt akzeptiert.

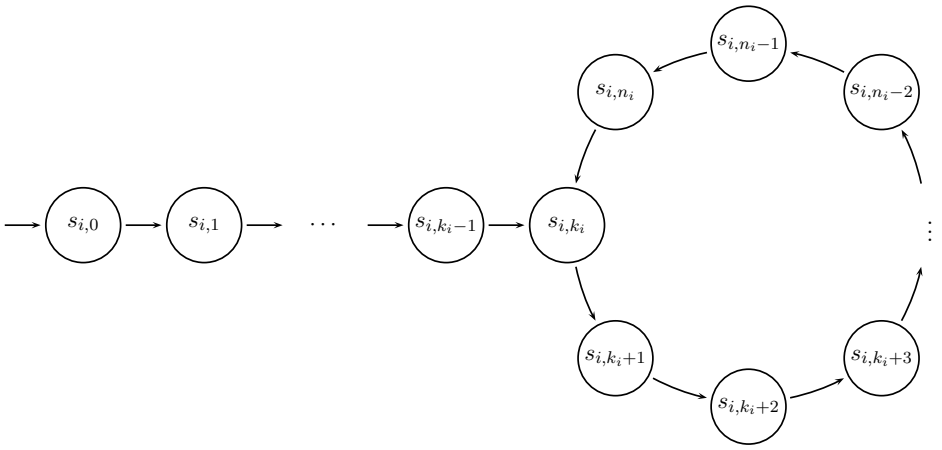


Abbildung 4.3: Der DFA $\mathcal{C}_i$.

Um die Berechnung des endlichen Automaten $\mathcal{C}_i$ jeweils für den aktuellen Zählerstand ausführen zu können, muss $\mathcal{B}$ aufgrund der einfachen Struktur eines unären DFA (siehe Abb. 4.3) nicht den tatsächlichen Zählerstand verwenden – das Herunterzählen des Zählers bis null würde den Zählerstand „vernichten“. Ausgehend vom Startzustand $s_{i,0}$ folgt $\mathcal{B}$ bei jeder Zählererhöhung lediglich der entsprechenden Kante zum Folgezustand. Bei Verminderungen des Zählerstandes wird entsprechend der Vorgängerzustand angenommen; einzig im Zustand s_{i,k_i} ist der korrekte Vorgängerzustand dabei zunächst unklar. Um zu überprüfen, ob sich der DFA $\mathcal{C}_i$ im Zustand s_{i,k_i}

befindet, nachdem er k_i oder $k_i + l \cdot (n_i - k_i + 1)$, mit $l \geq 1$, Zeichen gelesen hat (d. h. null oder bereits $l \geq 1$ Schleifendurchläufe absolviert hat), vermindert $\mathcal{B}$ den tatsächlichen Zählerstand um k_i und führt einen Nulltest aus. Ist der Zählerstand nun null, wird die Simulation von $\mathcal{C}_i$ im Zustand s_{i,k_i-1} fortgesetzt, anderenfalls im Zustand s_{i,n_i}. Anschließend erhöht $\mathcal{B}$ den Zählerstand wieder um k_i und hat auf diese Weise den ursprünglichen Zählerstand wiederhergestellt.

Durch die beschriebene Vorausberechnung wechselt $\mathcal{B}$ bei jedem Übergang zum nächsten Eingabezeichen genau dann in einen Endzustand, wenn $\mathcal{A}$ beim Auftreffen auf das rechte Bandbegrenzungssymbol im nächsten Schritt – mit den auf dem Band befindlichen Leerzeichen links vom Kopf – schließlich akzeptieren wird. Somit gilt $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{B})$.

„$\supseteq$“ Nach Lemma 4.1.6 existiert eine obere Schranke k für die Erhöhung des Zählerstandes pro Eingabezeichen. Mit dieser Konstante verläuft der Beweis ähnlich zum nichtdeterministischen Fall.

Da hier – im Gegensatz zum nichtdeterministischen Fall – jedoch Folgen von λ-Übergängen auftreten können, müssen diese vom $(\mathsf{ER_R}, \mathsf{DL})$-Automaten (der keine λ-Übergänge verwenden kann) in einem Schritt vorweggenommen werden. Aufgrund der deterministischen Überführungsfunktion des Zähler-Automaten kann die Verarbeitung eines Eingabezeichens und die anschließende Folge von λ-Übergängen vom $(\mathsf{ER_R}, \mathsf{DL})$-Automaten in einem Schritt erledigt werden, falls die entstehenden Konfigurationen während der Abarbeitung der λ-Übergänge im Voraus bekannt sind. Da die Erhöhung des Zählerstandes pro Eingabezeichen nach Lemma 4.1.6 durch k beschränkt, die Verminderung pro Eingabezeichen aber nicht a priori beschränkt ist (der Automat kann mit λ-Übergängen beliebig hohe Zählerstände löschen), simuliert der deterministische $(\mathsf{ER_R}, \mathsf{DL})$-Automat jeweils nur k Schritte des Zähler-Automaten in einer Operation und kann dadurch das Vermindern des Zählers korrekt simulieren. Dies geschieht – wie im nichtdeterministischen Fall – durch das Vermindern des internen Zählers und das Ausführen von $\mathsf{DL_L}$-Operationen. Da Eingabe und Zustand jeweils bekannt sind, muss nur noch die Kenntnis über das Ergebnis des Nulltests erlangt werden.

Bei einer Erhöhung des Zählerstandes ist das Ergebnis des Nulltests klar; bei einer Verminderung des Zählerstandes kann anhand der Variablen X und des Zählers modulo k (vgl. Satz 4.1.5) im Voraus erkannt werden, ob nach einer Folge von k Schritten ein Unterlauf des internen Zählers auftritt und der Zählerstand null erreicht wird oder nicht. Da bei einem Unterlauf des internen Zählers eine $\mathsf{DL_L}$-Operation ausgeführt werden muss, kann der $(\mathsf{ER_R}, \mathsf{DL})$-Automat auch diese – zunächst nicht beschränkten – Folgen von λ-Übergängen simulieren, ohne selbst λ-Übergänge verwenden zu müssen.

□

Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ sind also offensichtlich echte Obermengen der Familie der regulären Sprachen. Typische Beispiele für die Trennung sind beispielsweise die Sprachen $\{a^n b^n \mid n \in \mathbb{N}\}$ (siehe Beispiel 3.2.1) und $\left\{w \in \{a,b\}^* \,\middle|\, |w|_a = |w|_b\right\}$ (siehe Beispiel 3.2.2).

4.2 Die Beziehungen zwischen den Modellen der Forgetting-Automaten

In diesem Abschnitt vergleichen wir hauptsächlich die – nach den bereits gefundenen Charakterisierungen verbliebenen – von Forgetting-Automaten erkannten Sprachfamilien untereinander. Dabei ergeben sich einerseits weitere Charakterisierungen, die die Anzahl der tatsächlich unterschiedlich mächtigen Automatenmodelle weiter reduzieren. Andererseits betrachten wir Inklusionen und Unvergleichbarkeiten, die die Beziehungen zwischen den verschiedenen Modellen genauer beleuchten.

4.2.1 Lemma
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$ ist unvergleichbar mit $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$.

Beweis

„$\not\subseteq$“ Die kontextfreie Sprache $L_1 = \{wcw^R \mid w \in \{a,b\}^*\} \in \mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$ kann nicht von einem deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten akzeptiert werden (siehe [28]). Ein deterministischer $(\mathsf{MV_R}, \mathsf{DL})$-Automat kann L_1 gemäß Beispiel 3.2.3 akzeptieren.

„$\not\supseteq$“ Die Sprache $L_2 = \{a^{2^n} \mid n \in \mathbb{N}\} \in \mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ kann nicht von einem deterministischen $(\mathsf{MV_R}, \mathsf{DL})$-Automaten akzeptiert werden, da L_2 nicht kontextfrei und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$ eine Teilmenge von DCFL ist. Ein deterministischer $(\mathsf{ER}, \mathsf{DL})$-Automat kann L_2 jedoch gemäß Beispiel 3.2.5 akzeptieren. □

Mit den Sprachen L_1 und L_2 aus dem vorangegangenen Beweis können wir zudem folgern:

4.2.2 Korollar
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ und DCFL sind jeweils unvergleichbar mit $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$.

4.2.3 Satz
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL}) \subset \mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$

Beweis Die Inklusion $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL}) \subseteq \mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$ ergibt sich durch die Simulation von DL durch ER und das Ignorieren der entstehenden Leerzeichen (siehe auch Abschnitt 3.3). Wir zeigen nun die Echtheit der Inklusion.

In [49] wurde gezeigt, dass die Sprache

$$L = \{a^{n_1}ba^{n_1}a^{n_2}ba^{n_2}\cdots a^{n_l}ba^{n_l}cb^l \mid l, n_1, \ldots, n_l \in \mathbb{N}\}$$

nicht von einem deterministischen ($\mathsf{MV_R}$, DL)-Automaten erkannt werden kann.

Ein deterministischer ($\mathsf{MV_R}$, ER)-Automat $\mathcal{A}$ kann L wie folgt akzeptieren: $\mathcal{A}$ bewegt sich zunächst nach rechts bis zum ersten b; dabei verwendet er bei allen Feldern, mit Ausnahme des zweiten, $\mathsf{MV_R}$-Schritte und radiert lediglich das zweite Feld als Trennfeld aus. Anschließend radiert er durch Rechts- und Linksbewegungen mit ER-Schritten abwechselnd jeweils ein a auf der rechten und linken Seite der erreichten Position aus. Um das zu Beginn ausradierte Trennfeld erkennen zu können, führt $\mathcal{A}$ das Ausradieren der Felder auf der linken Seite mit einer $\mathsf{ER_L}$-Operation aus und bewegt sich mit $\mathsf{MV_R}$ vom erreichten Feld zurück nach rechts. Auf diese Weise kann $\mathcal{A}$ erkennen, wann er das Trennfeld erreicht und somit das Ausradieren des ersten Feldes verhindern. $\mathcal{A}$ bewegt sich anschließend nach rechts und radiert zwei weitere a aus.

Zu diesem Zeitpunkt hat $\mathcal{A}$ den ersten Block $a^{n_1}ba^{n_1}$ in $a\sqcup^{2n_1}$ umgewandelt und kann zur Verarbeitung des nächsten Blocks übergehen. Durch Iteration des beschriebenen Vorgangs wird somit jedes Teilwort $a^{n_i}ba^{n_i}$ in $a\sqcup^{2n_i}$ umgewandelt.

In Abbildung 4.4 wird das Band während der Berechnung des Automaten $\mathcal{A}$ bei der Verarbeitung des Wortes $a^4ba^7ba^8ba^5cb^3 = a^4ba^4a^3ba^3a^5ba^5cb^3$ dargestellt. Jede Zeile repräsentiert den Inhalt des Bandes, nachdem ein weiteres Feld ausradiert wurde.

Wenn $\mathcal{A}$ das Zeichen c erreicht, hat er jeweils ein a für jeden Block $a^{n_1}ba^{n_1}$ auf der linken Seite des Kopfes übrig behalten und kann diese a nun durch abwechselnde Links- und Rechtsbewegungen mit den b auf der rechten Seite des Kopfes vergleichen. $\mathcal{A}$ akzeptiert schließlich die Eingabe, falls die Eingabe am Ende dieses Vergleichs komplett ausradiert wurde. □

4.2.4 Satz
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL}) \subset \mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$

Beweis Analog zum Beweis von Satz 4.2.3 kann ein deterministischer ($\mathsf{MV_R}$, $\mathsf{ER_R}$, DL)-Automat $\mathcal{A}$ in jedem Block $a^{n_i}ba^{n_i}$ die Anzahl der Zeichen links und rechts vom Markierungszeichen b vergleichen und ein a pro Block für den späteren Vergleich mit den b am Eingabeende übrig lassen.

Da in diesem Fall keine $\mathsf{ER_L}$-Operation zur Verfügung steht – dafür jedoch die Operationen $\mathsf{DL_L}$ und $\mathsf{DL_R}$ – können hier die restlichen Zeichen eines Blocks gelöscht statt ausradiert werden. Jeder Block $a^{n_i}ba^{n_i}$ wird von $\mathcal{A}$ also

Abbildung 4.4: Beispiel-Abarbeitung des Wortes $a^4ba^7ba^8ba^5cb^3 = a^4ba^4a^3ba^3a^5ba^5cb^3$ durch einen deterministischen $(\mathsf{MV_R}, \mathsf{ER})$-Automaten.

in ein einzelnes a umgeschrieben. Schließlich kann $\mathcal{A}$ nach dem Erreichen von c die verbliebenen Zeichen links und rechts vom Kopf durch abwechselndes Löschen vergleichen und die Eingabe im Erfolgsfall akzeptieren. □

Die beiden folgenden Sätze betreffen deterministische (ER, DL)-Automaten, für die in den Sätzen 3.3.7 und 4.1.5 bereits nachgewiesen wurde, dass sie mächtiger als deterministische 1-Zähler-Automaten sind:

4.2.5 Satz
$\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL}) = \mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL}_\mathsf{R})$

Beweis In [29] wurde nachgewiesen, dass im nichtdeterministischen Fall die Inklusion $\mathscr{L}(\mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}(\mathsf{ER}, \mathsf{DL}_\mathsf{R})$ gilt. Der dort angegebene Beweis verwendet jedoch Nichtdeterminismus, um die fehlende DL_L-Operation durch einen $(\mathsf{ER}, \mathsf{DL}_\mathsf{R})$-Automaten zu simulieren und ist daher nicht auf den deterministischen Fall übertragbar.

Die bei einem $(\mathsf{ER}, \mathsf{DL}_\mathsf{R})$-Automaten fehlende DL_L-Operation kann allerdings sowohl im nichtdeterministischen als auch im deterministischen Fall einfach durch die Hintereinanderausführung von ER_L und DL_R simuliert werden, da sich links vom Kopf kein Eingabezeichen befinden kann (siehe Abbildung 4.5). Wird die DL_L-Operation auf dem ersten Zeichen neben dem linken Bandbegrenzungssymbol ausgeführt und zieht sie somit einen MV_R-Schritt nach sich, wird dies entsprechend durch die Hintereinanderausführung von ER_L, MV_R und DL_R simuliert.

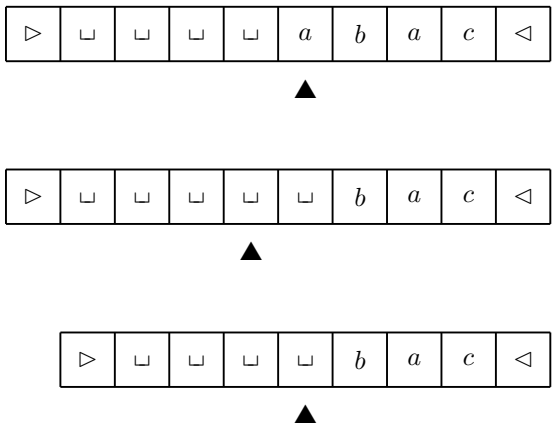


Abbildung 4.5: Simulation der DL_L-Operation durch eine ER_L- und eine DL_R-Operation.

Daher gilt $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL}) = \mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL}_\mathsf{R})$. □

4.2.6 Satz
$\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL}) \subset \mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$

Beweis Der Beweis aus [29] gilt auch für den deterministischen Fall, da die Zeugensprache $\{wcw^R \mid w \in \{a,b\}^*\} \notin \mathscr{L}(\mathsf{ER}, \mathsf{DL})$ auch von einem deterministischen $(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$-Automaten akzeptiert werden kann (durch Suchen des Symbols c und abwechselndes Löschen der benachbarten Zeichen auf der linken und rechten Seite). □

Vergleichen wir die Modelle der $(\mathsf{ER}, \mathsf{DL})$-Automaten und der $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automaten miteinander, stellen wir fest, dass $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automaten durch die zusätzliche $\mathsf{MV_L}$-Operation lediglich in der Lage sind, das jeweils erste bisher nicht gelöschte oder ausradierte Zeichen nach links unverändert zu verlassen.

Ein $(\mathsf{ER}, \mathsf{DL})$-Automat kann ein Eingabezeichen nach links nur durch $\mathsf{ER_L}$ oder $\mathsf{DL_L}$ verlassen. Erfordert die Überprüfung eines Teilwortes v mehrfache Hin- und Rückbewegungen auf v, kann ein $(\mathsf{ER}, \mathsf{DL})$-Automat diese Bewegungen nicht ausführen, ohne weitere Zeichen rechts von v auszuradieren oder zu löschen. Dies ist der Fall, wenn eine nicht-reguläre Eigenschaft überprüft werden muss; da ein $(\mathsf{ER}, \mathsf{DL})$-Automat jedes Zeichen im unausradierten Zustand nur ein einziges Mal zu lesen vermag, kann eine derartige Eigenschaft allerdings nur die Länge eines Teilwortes betreffen. Eine solche Eigenschaft ist beispielsweise die, dass die Länge eines bestimmten Teilwortes eine Zweierpotenz sein muss.

Die Trennung von $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ kann daher durch die markierte Konkatenation zweier geeigneter nicht-regulärer Sprachen erfolgen.

4.2.7 Satz
$\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL}) \subset \mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$

Beweis Als Trennsprache sei die folgende Sprache aus $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ gegeben:

$$L := \left\{ a^{2^n} c a^m b^m \;\middle|\; n, m \in \mathbb{N} \right\}$$

Ein deterministischer $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automat kann zunächst durch schrittweise Halbierung der Länge überprüfen, ob die Länge der ersten Zeichenkette der Form a^* (bis zum Trennzeichen c) eine Zweierpotenz ist; dabei ist die $\mathsf{MV_L}$-Operation entscheidend dafür, dass der Automat imstande ist, das Trennzeichen c unverändert zu verlassen. Anschließend kann der Rest der Eingabe auf die Form $a^m b^m$ überprüft werden.

Um zu zeigen, dass $L \notin \mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ gilt, nehmen wir zunächst das Gegenteil an und betrachten zu diesem Zweck einen deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten $\mathcal{A} = \langle S, A, \rhd, \lhd, \sqcup, O, \delta, s_0, F \rangle$ mit $L(\mathcal{A}) = L$.

Es sei $s = |S|$ die Anzahl der Zustände von $\mathcal{A}$. Wir betrachten ein Eingabewort $w = a^{2^{2^n}} ca^n b^n \in L$ mit $n > 2^s$ und die Berechnung von $\mathcal{A}$ auf w. Dabei unterscheiden wir zunächst die folgenden Fälle:

1. $\mathcal{A}$ erreicht eine Schleife der Form $\triangleright \sqcup^k x a^i c a^n b^n \triangleleft \vdash^+ \triangleright \sqcup^k x a^i c a^n b^n \triangleleft$

   In diesem Fall erreicht $\mathcal{A}$ niemals das c und akzeptiert daher w genau dann, wenn er $wb = a^{2^{2^n}} ca^n b^{n+1} \notin L$ akzeptiert.

   Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

2. $\mathcal{A}$ erreicht eine Schleife der Form $\triangleright \sqcup^k x a^i c a^n b^n \triangleleft \vdash^+ \triangleright \sqcup^k x a^j c a^n b^n \triangleleft$ mit $i > j$ und $i - j \leq s$

   In diesem Fall erreicht $\mathcal{A}$ das Trennzeichen c in einer Konfiguration $\triangleright \sqcup^{k'} x' c a^n b^n \triangleleft$ mit $x' \in S$ und $k' < s$. $\mathcal{A}$ akzeptiert daher w genau dann, wenn er $a^{i-j} w \notin L$ akzeptiert.

   Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

3. $\mathcal{A}$ erreicht eine Schleife der Form $\triangleright \sqcup^k x a^i c a^n b^n \triangleleft \vdash^+ \triangleright \sqcup^l x a^j c a^n b^n \triangleleft$ mit $k < l, i > j$ und $k + i \geq l + j$

   Wir betrachten die minimale Anzahl an Konfigurationsänderungen, für die die obige Bedingung gilt. Dann löscht $\mathcal{A}$ in jedem Schleifendurchlauf $0 \leq i + k - (j + l) \leq s$ Zeichen und bewegt sich in Richtung des Trennsymbols c nach rechts. An diesem Punkt können erneut drei verschiedene Fälle unterschieden werden:

   (a) $\mathcal{A}$ erreicht das erste b, bevor er $\triangleright$ erreicht (oder er erreicht $\triangleright$ überhaupt nicht mehr)

   In diesem Fall erreicht $\mathcal{A}$ eine Schleife der Form

   $$\triangleright \sqcup^{k'} y a^{i'} b^n \triangleleft \vdash^+ \triangleright \sqcup^{l'} y a^{j'} b^n \triangleleft$$

   mit $i' > j'$. $\mathcal{A}$ akzeptiert daher w genau dann, wenn er

   $$a^{2^{2^n} - (l'-k')(i-j)} c a^{n+(l-k)(i'-j')} b^n \notin L$$

   akzeptiert, da er bei der Berechnung auf beiden Wörtern das erste b in der gleichen Konfiguration erreicht:

   Das erste Teilwort $a^{2^{2^n}}$ wird um ein Vielfaches von $i-j$ verkürzt. $\mathcal{A}$ erreicht c daher in demselben Zustand wie bei der Berechnung auf w. Durch die Verkürzung werden dabei $(l'-k')(l-k)$ weniger Leerzeichen links vom Kopf stehen gelassen.

   Das Teilwort a^n wird um ein Vielfaches von $i' - j'$ verlängert; daher erreicht $\mathcal{A}$ das erste b in demselben Zustand wie bei der Berechnung auf w. Durch die Verlängerung werden hier auf dem Band $(l-k)(l'-k')$ zusätzliche Leerzeichen stehen gelassen.

Insgesamt erreicht $\mathcal{A}$ folglich bei beiden Eingabewörtern das erste b in demselben Zustand und mit derselben Anzahl an Leerzeichen links vom Kopf.

Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

(b) $\mathcal{A}$ erreicht im weiteren Verlauf weder ein b noch $\triangleright$

In diesem Fall akzeptiert $\mathcal{A}$ w genau dann, wenn er das Wort $wb = a^{2^{2^n}} ca^n b^{n+1} \notin L$ akzeptiert.

Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

(c) $\mathcal{A}$ erreicht $\triangleright$, bevor er das erste b erreicht (oder er erreicht überhaupt kein b mehr)

In diesem Fall kann $\mathcal{A}$ höchstens s Zeichen in der Umgebung des ersten b löschen oder ausradieren, bevor er eine Schleife erreicht, die zum linken Bandbegrenzungssymbol zurückführt. Nachdem das linke Bandbegrenzungssymbol $\triangleright$ wieder erreicht wird, müssen die Fälle 1 bis 3 iteriert betrachtet werden.

Die Fälle 1, 2, 3 (a) und 3 (b) müssen dabei nur nach höchstens s Iterationen von Fall 3 (c) betrachtet werden, d. h. nach höchstens s Hin- und Rückbewegungen zwischen dem linken Bandbegrenzungssymbol $\triangleright$ und dem jeweils ersten a in Schleifen, die mindestens eine ER-Operation beinhalten.

— Falls nach einem bestimmtem Zeitpunkt das am weitesten links liegende a nicht mehr erreicht wird (dies entspricht Fall 1), gilt die obige Begründung immer noch, da $\mathcal{A}$ nach wie vor w genau dann akzeptiert, wenn er auch wb akzeptiert.

Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

— Falls $\mathcal{A}$ weniger als s Leerzeichen auf dem Band übrig lässt, bevor er das erste b erreicht (für maximal s Iterationen entspricht dies Fall 2), kann das erste Teilwort w wie folgt erweitert werden:

Ein Eingabewort führt $\mathcal{A}$ in denselben Zustand – bevor die Schleife betreten wird, die die Eingabe analog zu Fall 2 verkürzt – falls es in der Menge $N := \{a^\tau ca^n b^n \mid \tau \in \alpha\mathbb{N} + \beta\}$ (mit $\alpha, \beta \in \mathbb{N}$) enthalten ist. Dabei ist $\alpha\mathbb{N} + \beta$ die Menge der Lösungen der simultanen Kongruenzen, die durch die Hin- und Herbewegungen definiert werden, die $\mathcal{A}$ jeweils in denselben Zuständen vom linken Bandbegrenzungssymbol $\triangleright$ zum ersten unausradierten Zeichen und zurück führen. Da eine Lösung existiert (nämlich für w), existieren nach dem Chinesischen Restsatz sogleich unendlich viele Lösungen.

Da maximal s Iterationen durchgeführt werden, ist α kleiner als 2^{2^n} und es existiert ein Wort $a^{n'} ca^n b^n \in N$ derart, dass n'

keine Zweierpotenz ist und $\mathcal{A}$ das Wort $a^{n'}ca^nb^n \notin L$ genau dann akzeptiert, wenn er w akzeptiert.
Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

— Falls $\mathcal{A}$ das erste b nach maximal s Iterationen von Fall 3 (c) erreicht – dies entspricht Fall 3 (a) – dann existiert wiederum eine Menge $N' := \{a^\tau ca^nb^n \mid \tau \in \alpha'\mathbb{N} + \beta'\}$ von Eingabewörtern, die $\mathcal{A}$ in denselben Zustand führen, in dem die Schleife begonnen wird, die zum ersten b führt. Daher existieren zwei Zahlen $n' \in \mathbb{N}_0$ und $n'' \in \mathbb{N}$ derart, dass $\mathcal{A}$ das Wort $a^{2^{2^n}-n'}ca^{n+n''}b^n \notin L$ genau dann akzeptiert, wenn er w akzeptiert.
Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

— Falls $\mathcal{A}$ nach maximal s Iterationen von Fall 3 (c) zu der Situation aus Fall 3 (b) gelangt, das heißt nie wieder das linke Bandbegrenzungssymbol $\triangleright$ oder ein b erreicht, gilt wiederum, dass $\mathcal{A}$ das Wort $wb \notin L$ genau dann akzeptiert, wenn er w akzeptiert.
Dies ist ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

— Falls Schleifendurchläufe wie in Fall 3 (c) iteriert werden, bis das erste b erreicht wird, tritt eine Folge $z_1, z_2, \ldots$ von Zuständen auf, in denen $\mathcal{A}$ das jeweils erste unausradierte a erreicht. Aufgrund der Wahl von n existieren p und q mit $p < q$ so, dass $z_p = z_q$ gilt.
Falls ausschließlich ER-Operationen in den Schleifen verwendet werden, die von $\triangleright$ zu einem a und zurück führen und die nach dem Auftreten von z_p auftreten, existieren $n', n'' \in \mathbb{N}$ derart, dass $\mathcal{A}$ die gleiche Konfiguration auf dem ersten b erreicht, wenn er w bzw. $a^{2^{2^n}+n'}ca^{n-n''}b^n \notin L$ verarbeitet.
Falls jedoch DL-Operationen in diesen Schleifendurchläufen auftreten, existieren $n', n'' \in \mathbb{N}$ so, dass die gleiche Konfiguration auf dem ersten b erreicht wird, wenn $\mathcal{A}$ w bzw. $a^{2^{2^n}+n'}ca^{n+n''}b^n \notin L$ verarbeitet.
Dies ist abermals ein Widerspruch zur Annahme $L(\mathcal{A}) = L$.

Schließlich haben wir in allen Fällen einen Widerspruch zu der am Beginn getroffenen Annahme $L(\mathcal{A}) = L$ erlangt. Somit folgt die Trennung der Sprachfamilien $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$. □

Während die Inklusion $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R}) \subseteq \mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$ trivial ist, wurde die Frage, ob eine echte Inklusion oder Gleichheit herrscht, in [29] offen gelassen. Wir geben hier eine Antwort auf diese Frage und zeigen, dass das Fehlen der $\mathsf{DL_L}$-Operation keinen Unterschied bei der Berechnungsmächtigkeit ergibt:

4.2.8 Satz

$$\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R}) = \mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$$
$$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R}) = \mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$$

Beweis Ein $(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$-Automat $\mathcal{B}$ kann einen $(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$-Automaten $\mathcal{A}$ wie folgt simulieren: Um die $\mathsf{DL_L}$-Operation zu simulieren, die dem $(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL_R})$-Automaten nicht zur Verfügung steht, unterscheidet $\mathcal{B}$ während der Simulation zwei verschiedene Modi, die wir als Modus 1 und Modus 2 bezeichnen.

In Modus 1 simuliert $\mathcal{B}$ die Operationen $\mathsf{MV_R}$, $\mathsf{ER_L}$, $\mathsf{ER_R}$ und $\mathsf{DL_R}$ direkt und verbleibt in Modus 1. Die Operation $\mathsf{DL_L}$ hingegen wird durch das Ausführen von $\mathsf{ER_L}$ und das Annehmen von Modus 2 simuliert. Dadurch verbleibt ein zusätzliches Leerzeichen auf der rechten Seite des Kopfes, das beim späteren Wiederannehmen von Modus 1 gelöscht werden muss.

In Modus 2 simuliert $\mathcal{B}$ das Verhalten von $\mathcal{A}$ wie folgt:

$\mathcal{A}$	$\mathcal{B}$	
führt aus	führt aus	nimmt Modus an
$\mathsf{MV_R}$	$\mathsf{MV_R}$, $\mathsf{DL_R}$	1
$\mathsf{ER_L}$	$\mathsf{DL_R}$, $\mathsf{ER_L}$	1
$\mathsf{ER_R}$	$\mathsf{DL_R}$, $\mathsf{MV_R}$	1
$\mathsf{DL_L}$	$\mathsf{DL_R}$, $\mathsf{ER_L}$	2
$\mathsf{DL_R}$	$\mathsf{DL_R}$, $\mathsf{DL_R}$	1

Auf diese Weise kann $\mathcal{B}$ immer zur direkten Simulation in Modus 1 zurückkehren oder den Zustand beibehalten, dass genau ein zusätzliches Leerzeichen rechts vom Kopf vorhanden ist.

Wenn beispielsweise $(s', \mathsf{ER_R}) \in \delta_{\mathcal{A}}(s, w_i)$ gilt, führt der Konfigurationsübergang

$$\triangleright w_1 \cdots w_{i-1} s w_i \cdots w_n \triangleleft \;\vdash_{\mathcal{A}}\; \triangleright w_1 \cdots w_{i-1} \sqcup s' w_{i+1} \cdots w_n \triangleleft$$

in Modus 2 der Simulation zu den Transitionen

$$\begin{aligned} \triangleright w_1 \cdots w_{i-1}(s,2) w_i \sqcup w_{i+1} \cdots w_n \triangleleft \;&\vdash_{\mathcal{B}}\; \triangleright w_1 \cdots w_{i-1}(\tilde{s},2) \sqcup w_{i+1} \cdots w_n \triangleleft \\ &\vdash_{\mathcal{B}}\; \triangleright w_1 \cdots w_{i-1} \sqcup (s',1) w_{i+1} \cdots w_n \triangleleft . \end{aligned}$$

Dabei ist (s,i), $i \in \{1,2\}$, der Zustand von $\mathcal{B}$ beim Simulieren des Zustands s von $\mathcal{A}$ in Modus i. Der Zustand $\tilde{s} \notin S_{\mathcal{A}}$ ist ein für die Simulation benötigter Zwischenzustand.

Auf diese Weise akzeptiert $\mathcal{B}$ schließlich die Eingabe genau dann, wenn $\mathcal{A}$ akzeptiert. □

Eine weitere Beziehung zwischen Sprachfamilien, die in [29] offen gelassen wurde, ist die zwischen $\mathscr{L}(\mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$. Wir zeigen hier die folgende Inklusion:

4.2.9 Satz
$\mathscr{L}(\mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{ER})$

Beweis Ein (MV, ER)-Automat $\mathcal{B}$ kann folgendermaßen einen (ER, DL_L)-Automaten $\mathcal{A}$ – nach Abschnitt 3.3 gilt $\mathscr{L}(\mathsf{ER}, \mathsf{DL}) = \mathscr{L}(\mathsf{ER}, \mathsf{DL}_\mathsf{L})$ – simulieren: Im Wesentlichen simuliert $\mathcal{B}$ ER durch MV und DL durch ER, das heißt, er radiert die Felder, die $\mathcal{A}$ ausradiert, nicht aus, sondern bewegt sich nur nach rechts. Die Felder, die $\mathcal{A}$ löscht, radiert $\mathcal{B}$ aus und betrachtet diese Felder später als bereits gelöscht.

Um den (ER, DL)-Automaten korrekt simulieren zu können, ist es notwendig, stets die „kritische“ Position c zu kennen, bis zu der $\mathcal{A}$ die Eingabe bereits ausradiert hat. Diese Kenntnis ist nötig, um zu wissen, ob $\mathcal{A}$ zu einem Zeitpunkt der Simulation ein Leerzeichen oder ein Eingabezeichen liest. Zu diesem Zweck wird das Symbol an der Position c durch Ausradieren markiert und zudem die Information abgespeichert, dass für diesen Fall das am weitesten rechts liegende Leerzeichen auf dem Band die Position c repräsentiert und nicht – wie sonst – ein gelöschtes Symbol. Ob ein Zeichen das am weitesten rechts liegende Leerzeichen auf dem Band ist, kann dabei wie folgt festgestellt werden: Der (MV, ER)-Automat $\mathcal{B}$ bewegt sich beim Erreichen eines Leerzeichens mit MV_R-Schritten nach rechts, bis das nächste Leerzeichen oder das rechte Bandbegrenzungssymbol erreicht wird. Im ersten Fall ist das ursprüngliche Leerzeichen nicht das am weitesten rechts liegende Leerzeichen, im zweiten Fall schon.

$\mathcal{B}$ simuliert nun $\mathcal{A}$ auf die oben beschriebene Weise. Falls $\mathcal{A}$ sich dabei von der Position c nach links bewegt (d. h. über die bereits ausradierten Zeichen am Bandanfang läuft), rät $\mathcal{B}$, ob $\mathcal{A}$ zu dieser Position zurückkehren wird, ohne ein Zeichen zu löschen oder nicht. Betrachten wir die Teilberechnungen, die $\mathcal{A}$ ausgehend von einem Zustand s_i und einer Bandbeschriftung der Form $\rhd\sqcup^* xy \cdots \lhd$ auf dem Symbol $x \in A$ mit einer ER_L-Operation (bzw. DL_L-Operation) beginnt und die ansonsten nur aus ER-Operationen auf Leerzeichen bestehen, so erhalten wir das Verhalten eines 2NFA, der seine Berechnung auf dem vorletzten (bzw. letzten) Leerzeichen der unären Eingabe $w \in \{\sqcup\}^*$ (mit Begrenzungssymbolen $\rhd$ und y, siehe Abbildung 4.6) startet.

Da dieser 2NFA auch durch einen DFA repräsentiert werden kann, simuliert $\mathcal{B}$ das Verhalten eines solchen DFA $\mathcal{C}_i$ – jeweils ein Automat für jeden möglichen Ausgangszustand s_i, siehe Abbildung 4.3 auf Seite 64 – beim Erreichen des Begrenzungssymbols y im Voraus und bewegt sich lediglich im entsprechenden Zustand einen Schritt nach rechts. Die Simulation von $\mathcal{C}_i$ kann

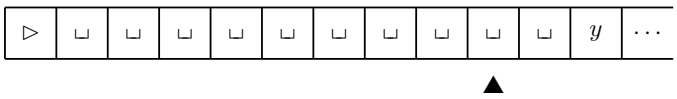


Abbildung 4.6: Der vom 2NFA betretene Teil des Bandes.

wiederum dadurch erreicht werden, dass $\mathcal{B}$ den Kanten des unären Automaten folgt – in Richtung der Kanten oder umgekehrt (vergleiche Satz 4.1.5). Ausgehend vom Zustand s_{i,k_i} überprüft $\mathcal{B}$ beim Zurückgehen, ob sein Kopf auf Position k_i steht: $\mathcal{B}$ läuft k_i Felder nach links, testet, ob das linke Bandbegrenzungssymbol erreicht wird, und läuft anschließend wieder k_i Felder nach rechts. Auf diese Weise kann $\mathcal{B}$ stets den Zustand des DFA korrekt mitsimulieren, während er sich auf dem Band hin- und herbewegt.

Falls $\mathcal{B}$ jedoch rät, dass $\mathcal{A}$ mindestens ein Symbol löschen wird, bevor er zur aktuellen Position zurückkehrt, markiert $\mathcal{B}$ die Position c (er radiert sie aus), führt die Simulation fort und hält in einem nichtakzeptierenden Zustand an, falls er falsch geraten hat, das heißt, falls $\mathcal{A}$ zur aktuellen Position zurückkehrt, ohne ein Symbol gelöscht zu haben. Sobald $\mathcal{A}$ aber das erste Mal ein Symbol links von Position c löscht, betrachtet $\mathcal{B}$ einfach das am weitesten rechts liegende Eingabesymbol links von Position c als gelöscht. (Ob ein Symbol das am weitesten rechts liegende Eingabesymbol ist, kann auf bereits beschriebene Weise überprüft werden.)

Auf diese Weise können die Berechnungen auf der Folge von Leerzeichen am Bandanfang korrekt simuliert werden und $\mathcal{B}$ akzeptiert die Eingabe schließlich genau dann, wenn $\mathcal{A}$ akzeptiert. □

Das vorangegangene Ergebnis kann durch Hinzunahme der $\mathsf{MV_L}$-Operation noch erweitert werden:

4.2.10 Satz
$\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{ER})$

Beweis Während $\mathsf{MV_L}$ auf bereits ausradierten Symbolen äquivalent zu $\mathsf{ER_L}$ ist, erlaubt es einem $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL_L})$-Automaten, das am weitesten links liegende noch nicht ausradierte Eingabesymbol nach links zu verlassen, ohne es auszuradieren oder zu löschen.

Der Beweis von Satz 4.2.9 kann daher wie folgt erweitert werden:

- $\mathsf{MV_L}$ auf einem Eingabezeichen wird durch $\mathsf{MV_L}$ simuliert. Eine andere Behandlung ist nicht nötig, da sich die Position c nicht ändert.
- $\mathsf{MV_L}$ auf einem Leerzeichen wird – da äquivalent – wie $\mathsf{ER_L}$ behandelt, das heißt ebenfalls durch $\mathsf{MV_L}$ simuliert.

Auf diese Weise kann ein (MV, ER)-Automat auch einen (MV_L, ER, DL)-Automaten korrekt simulieren. □

Die Sätze 4.2.9 und 4.2.10 können auch auf eine deterministische Version erweitert werden:

4.2.11 Satz
$\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$

Beweis Der Beweis von Satz 4.2.9 kann wie folgt modifiziert werden, um den deterministischen Fall abzudecken: Falls sich der Automat $\mathcal{A}$ von Position c aus nach links bewegt, kann $\mathcal{B}$ hier nicht raten, ob $\mathcal{A}$ mindestens ein Zeichen löschen wird, bevor er zurückkehrt – dies ist jedoch auch nicht nötig. Betrachten wir den DFA $\mathcal{C}_i$, der aus einer ausschließlichen Bewegung über die Leerzeichen am Bandanfang mit ER-Schritten resultiert, so ergeben sich im deterministischen Fall zwei Möglichkeiten:

1. Der Zustand von $\mathcal{C}_i$ repräsentiert genau einen Zustand des 2DFA oder
2. der Zustand von $\mathcal{C}_i$ repräsentiert die leere Menge von Zuständen.

Im ersten Fall wird $\mathcal{A}$ zur Position c zurückkehren, ohne zwischendurch ein Symbol gelöscht zu haben. $\mathcal{B}$ kann daher direkt den entsprechenden Zustand annehmen. Im zweiten Fall wird $\mathcal{A}$ entweder halten oder eine DL-Operation ausführen. $\mathcal{B}$ kann daher Position c durch Ausradieren markieren und mit der direkten Simulation der Schritte von $\mathcal{A}$ auf den Leerzeichen fortfahren.
□

In [29] wurde gezeigt, dass die Sprache $\left\{wcw^R \mid w \in \{a,b\}^*\right\}$ nicht durch einen (MV_L, ER, DL)-Automaten erkannt werden kann. Zusammen mit den vorangegangenen Sätzen 4.2.9 bis 4.2.11 können wir daher eine echte Inklusion folgern, sowohl für den deterministischen als auch den nichtdeterministischen Fall:

4.2.12 Korollar

$$\mathscr{L}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL}) \subset \mathscr{L}(\mathsf{MV}, \mathsf{ER})$$
$$\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL}) \subset \mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$$

Für eine Übersicht der Klassifikation der Forgetting-Automaten im nichtdeterministischen bzw. deterministischen Fall siehe Abbildung 4.7 bzw. 4.8. Im deterministischen Fall ist als Vorgriff bereits die echte Inklusion von $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$ in $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$ eingetragen, die sich in Kapitel 5 als Folgerung aus der Trennung für unäre Sprachen ergibt.

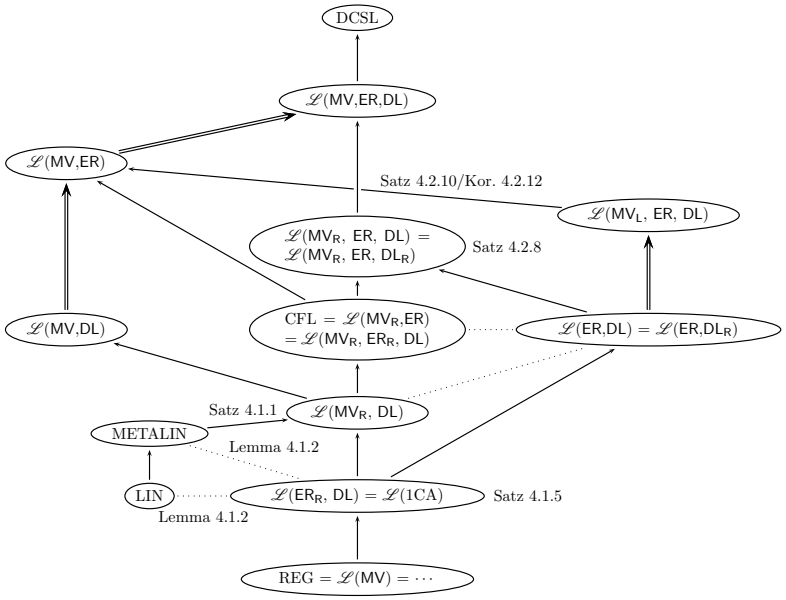


Abbildung 4.7: Die Hierarchie der nichtdeterministischen Forgetting-Automaten (⇒: Inklusion, →: echte Inklusion, ⋯: unvergleichbar).

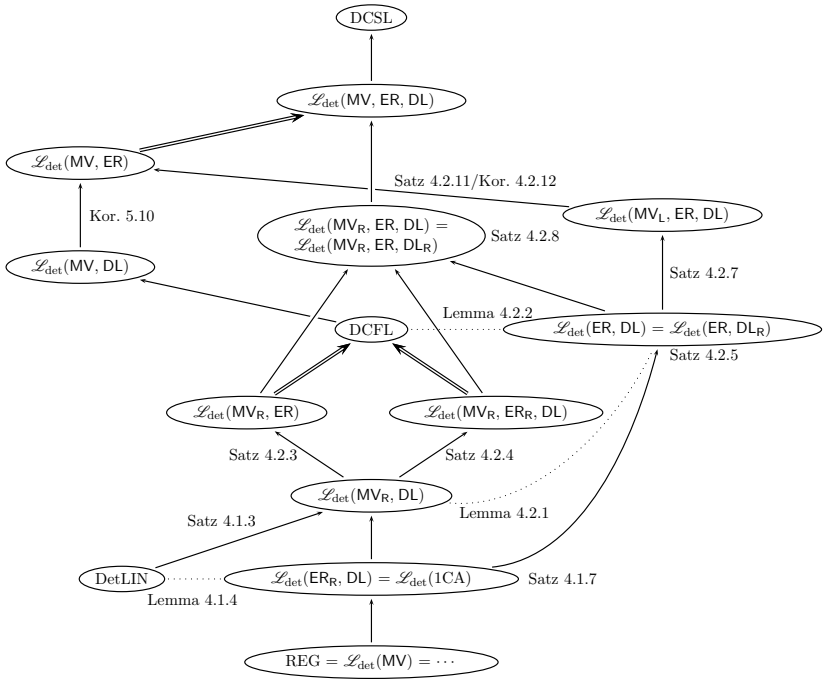


Abbildung 4.8: Die Hierarchie der deterministischen Forgetting-Automaten.

4.3 Offene Fragen

Im nichtdeterministischen Fall sind die folgenden verbliebenen offenen Fragen von besonderem Interesse:

- Ist $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$ eine echte Teilmenge von $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$ oder stimmen die beiden Sprachfamilien überein? (Für den deterministischen Fall ergibt sich in Kapitel 5 die strikte Inklusion als Folgerung aus der Trennung für unäre Sprachen.)
- Eng verbunden mit der ersten Frage ist die folgende: Ist $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$ eine Obermenge von CFL, das heißt, kann jede kontextfreie Sprache von einem (MV, DL)-Automaten erkannt werden? Falls $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$ gleich $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$ ist, kann diese Frage bejaht werden, da $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$ eine Obermenge von CFL ist.

Als offene Frage bezüglich der deterministischen Automatenmodelle verbleibt im Wesentlichen die folgende:

- Sind $\mathscr{L}_{\mathrm{det}}(\mathsf{MV}_\mathsf{R}, \mathsf{ER})$ und $\mathscr{L}_{\mathrm{det}}(\mathsf{MV}_\mathsf{R}, \mathsf{ER}_\mathsf{R}, \mathsf{DL})$ echte Teilmengen von DCFL? Im nichtdeterministischen Fall stimmen die entsprechenden Sprachfamilien $\mathscr{L}(\mathsf{MV}_\mathsf{R}, \mathsf{ER})$ und $\mathscr{L}(\mathsf{MV}_\mathsf{R}, \mathsf{ER}_\mathsf{R}, \mathsf{DL})$ mit der Familie der kontextfreien Sprachen überein. Im deterministischen Fall ist nur bekannt, dass die Sprachfamilien jeweils eine Teilmenge von DCFL sind.

Am oberen Ende der Hierarchie der Forgetting-Automaten bleibt schließlich – sowohl im deterministischen als auch im nichtdeterministischen Fall – die Frage, ob die Familien der von (MV, ER)- und (MV, ER, DL)-Automaten erkannten Sprachen übereinstimmen. Das Modell des (MV, ER)-Automaten hat sich als erstaunlich mächtig erwiesen und es ist noch nicht klar, ob es eine Sprache gibt, die von einem (MV, ER, DL)-Automaten erkannt werden kann, von einem (MV, ER)-Automaten jedoch nicht.

Kapitel 5

Die Berechnungsmächtigkeit im Fall unärer Sprachen

In diesem Kapitel betrachten wir die Berechnungsmächtigkeit der verschiedenen Modelle von Forgetting-Automaten für den Fall, dass die Eingabe stets unär ist, das heißt, dass das Eingabealphabet nur aus einem einzigen Symbol besteht. Ohne Beschränkung der Allgemeinheit nehmen wir dabei an, dass das Eingabealphabet die Menge $A = \{a\}$ ist.

Forgetting-Automaten auf unären Sprachen

Als Erstes betrachten wir dabei die Sprachfamilien am unteren Ende der Hierarchie. Hier fallen die Familien ab der Familie der regulären Sprachen bis einschließlich der Familie der kontextfreien Sprachen zusammen, da diese im unären Fall übereinstimmen:

5.1 Lemma

$$\begin{aligned}\mathrm{REG}^u &= \mathscr{L}^u(\mathsf{ER_R}, \mathsf{DL}) = \mathscr{L}^u(\mathsf{MV_R}, \mathsf{DL}) = \mathscr{L}^u(\mathsf{MV_R}, \mathsf{ER}) \\ &= \mathscr{L}^u(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL}) = \mathrm{CFL}^u\end{aligned}$$

Während im allgemeinen (nicht-unären) Fall noch immer unbekannt ist, ob die Inklusion $\mathrm{CFL} \subseteq \mathscr{L}(\mathsf{MV}, \mathsf{DL})$ gilt, können wir im unären Fall das folgende Resultat beweisen:

5.2 Satz

$\mathrm{REG}^u = \mathrm{CFL}^u \subset \mathscr{L}^u(\mathsf{MV}, \mathsf{DL})$

Beweis Da ein $(\mathsf{MV}, \mathsf{DL})$-Automat offensichtlich mindestens die Fähigkeiten eines endlichen Automaten hat, ist damit auch klar, dass die Inklusion $\mathrm{REG}^u \subseteq \mathscr{L}^u(\mathsf{MV}, \mathsf{DL})$ gilt.

Um die Echtheit der Inklusion zu zeigen, betrachten wir die nicht-reguläre Sprache $L := \{a^{2^n} \mid n \in \mathbb{N}\}$. Ein $(\mathsf{MV}, \mathsf{DL})$-Automat $\mathcal{A}$ kann L analog zu Beispiel 3.2.5 erkennen, indem er die Eingabe in Bewegungen von Bandbegrenzungssymbol zu Bandbegrenzungssymbol schrittweise halbiert. □

Zwei weitere Inklusionen von Sprachfamilien, deren Beziehungen im nicht-unären Fall noch nicht geklärt sind, sind die folgenden (Gleichheit oder Echtheit der Inklusionen bleiben offen):

5.3 Satz

$$\mathscr{L}^u(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}^u(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$$
$$\mathscr{L}^u_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}^u_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$$

Beweis Ein $(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$-Automat $\mathcal{B}$ kann einen $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automaten $\mathcal{A}$ wie folgt simulieren: Zunächst läuft $\mathcal{B}$ mit $\mathsf{MV_R}$-Schritten bis zum rechten Bandbegrenzungssymbol $\lhd$. Anschließend simuliert $\mathcal{B}$ alle Schritte von $\mathcal{A}$ in umgekehrter Richtung, das heißt $\mathsf{MV_R}$ statt $\mathsf{MV_L}$, $\mathsf{ER_L}$ statt $\mathsf{ER_R}$ usw. Auf diese Weise akzeptiert $\mathcal{B}$ eine Eingabe genau dann, wenn auch $\mathcal{A}$ sie akzeptiert. □

Ob die Familie $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ unter Spiegelung abgeschlossen ist, ist bisher nicht bekannt (siehe Kapitel 6). Falls der Abschluss gilt, folgt das vorangegangene Ergebnis auch für den allgemeinen Fall.

Das folgende Resultat gilt im nicht-unären Fall nicht, da dort die Sprache $\left\{wcw^R \,\middle|\, w \in \{a,b\}^*\right\}$ als Gegenbeispiel herangezogen werden kann:

5.4 Satz

$$\mathscr{L}^u(\mathsf{MV}, \mathsf{DL}) \subseteq \mathscr{L}^u(\mathsf{ER}, \mathsf{DL})$$
$$\mathscr{L}^u_{\det}(\mathsf{MV}, \mathsf{DL}) \subseteq \mathscr{L}^u_{\det}(\mathsf{ER}, \mathsf{DL})$$

Beweis Ein $(\mathsf{ER}, \mathsf{DL})$-Automat $\mathcal{B}$ kann einen $(\mathsf{MV}, \mathsf{DL})$-Automaten $\mathcal{A}$ wie folgt simulieren: Da der $(\mathsf{MV}, \mathsf{DL})$-Automat $\mathcal{A}$ nicht über eine ER-Operation verfügt und somit keine Leerzeichen auf das Band schreiben kann, liest er im unären Fall während der Berechnung – außer den Bandbegrenzungssymbolen $\rhd$ und $\lhd$ – stets das gleiche Zeichen a.

$\mathcal{B}$ simuliert daher $\mathsf{MV_L}$ bzw. $\mathsf{MV_R}$ durch $\mathsf{ER_L}$ bzw. $\mathsf{ER_R}$ und verhält sich beim Lesen von $\sqcup$ und a gleich. □

Um eine Inklusion von $\mathscr{L}^u(\mathsf{MV}, \mathsf{DL})$ in DCSL^u zu zeigen, vergleichen wir zunächst $(\mathsf{MV}, \mathsf{DL})$-Automaten mit Zweiwege-1-Zähler-Automaten. Nachdem wir uns in Kapitel 4 bereits mit den unidirektionalen 1-Zähler-Automaten beschäftigt haben, erweitern wir dieses Modell nun um die Möglichkeit, den Kopf in beide Richtungen über die Eingabe zu bewegen. Anders als beim

Übergang vom NFA zum 2NFA wird hier die Berechnungsmächtigkeit des Automaten erhöht. Ein *(beschränkter) Zweiwege-1-Zähler-Automat* (2CA, „*two-way one-counter automaton*“) ist ein Automat, der sich bidirektional über ein mit Bandbegrenzungssymbolen eingefasstes Eingabeband bewegen kann und der über einen Zähler verfügt, dessen Stand während jeder möglichen Berechnung nie die Länge der gegebenen Eingabe überschreitet. Der Automat kann in jedem Berechnungsschritt testen, ob der Zähler *leer* (das heißt der Wert gleich null) ist und den Zählerstand anschließend um eins erhöhen, vermindern oder unverändert lassen. Die Eingabe wird akzeptiert, falls der Automat in einem Endzustand hält.

Wir werden uns zudem mit einer weiteren Form der Zähler-Automaten beschäftigen. Mit dem Modell des *beschränkten Zweiwege-2-Zähler-Automaten* (2-2CA, „*two-way two-counter automaton*“) wird der Zweiwege-1-Zähler-Automat um einen weiteren (durch die Eingabelänge beschränkten) Zähler erweitert. Die Beschränkung des Zählers auf die Länge der Eingabe stellt dabei eine echte Einschränkung dar. Zähler-Automaten mit zwei unbeschränkten Zählern können bereits Turingmaschinen simulieren und somit die rekursiv aufzählbaren Sprachen erkennen, wie in [40] gezeigt wurde. Automaten mit beschränkten Zählern sind dagegen weitaus weniger mächtig.

5.5 Satz
$\mathscr{L}^u(\mathsf{MV}, \mathsf{DL}) \subseteq \mathscr{L}^u(\text{2CA})$

Beweis Ein 2CA $\mathcal{B}$ kann einen (MV, DL)-Automaten $\mathcal{A}$ im unären Fall wie folgt simulieren: Um den (MV, DL)-Automaten $\mathcal{A}$ korrekt zu simulieren, muss der Zähler-Automat einerseits die Kopfposition von $\mathcal{A}$ verfolgen und andererseits die Länge der verbliebenen unären Eingabe speichern können. Zu diesem Zweck setzt $\mathcal{B}$ seine eigene Kopfposition und den Zählerstand folgendermaßen ein: Der Zählerstand speichert während der Simulation stets die Position des Kopfes von $\mathcal{B}$ – wobei die Position des linken Bandbegrenzungssymbols mit 0 und die Position des ersten Eingabezeichens mit 1 bezeichnet wird – minus der Anzahl der von $\mathcal{A}$ bereits gelöschten Zeichen. Mit anderen Worten: Die Anzahl der bereits gelöschten Zeichen wird durch die Position des Kopfes von $\mathcal{B}$ minus des Zählerstandes repräsentiert.

$\mathcal{B}$ simuliert die Schritte von $\mathcal{A}$ daher wie in der folgenden Tabelle dargestellt:

$\mathcal{A}$	$\mathcal{B}$	
	Kopfbewegung	Zähleränderung
$\mathsf{MV_L}$	links	-1
$\mathsf{MV_R}$	rechts	$+1$
$\mathsf{DL_L}$	keine	-1
$\mathsf{DL_R}$	rechts	± 0

$\mathcal{B}$ kann durch den Nulltest feststellen, ob $\mathcal{A}$ das linke Bandbegrenzungssymbol erreicht hat. Das rechte Bandbegrenzungssymbol hingegen markiert sowohl für $\mathcal{A}$ als auch für $\mathcal{B}$ das Ende der Eingabe.

Das Vermindern des Zählers wird folglich nur ausgeführt, wenn sich der Kopf von $\mathcal{A}$ nicht auf dem linken Bandbegrenzungssymbol befindet und der Zähler somit einen positiven Wert enthält.

Auf diese Weise akzeptiert $\mathcal{B}$ die Eingabe genau dann, wenn $\mathcal{A}$ sie akzeptiert. □

Wir können nun auch eine echte Inklusion folgern:

5.6 Korollar
$\mathscr{L}^u(\mathsf{MV}, \mathsf{DL}) \subset \mathrm{DCSL}^u$

Beweis Nach Satz 5.5 ist $\mathscr{L}^u(\mathsf{MV}, \mathsf{DL})$ eine Teilmenge von $\mathscr{L}^u(2\mathrm{CA})$. Die Familie $\mathscr{L}^u(2\mathrm{CA})$ wiederum ist nach der Zähler-Hierarchie der unären Zweiwege-Zähler-Automaten (siehe [41]) eine echte Teilmenge der Familie NL^u der von nichtdeterministischen Turingmaschinen mit logarithmisch beschränktem Arbeitsband erkannten unären Sprachen. Diese Turingmaschinen verfügen über ein Eingabeband, welches sie nur lesen, aber nicht beschreiben können. Zusätzlich steht ihnen ein Arbeitsband zur Verfügung, dessen Länge sich nach der Länge der Eingabe richtet. NL^u ist schließlich eine Teilmenge von DCSL^u (siehe zum Beispiel [56, Seite 413]).

Es gilt also: $\mathscr{L}^u(\mathsf{MV}, \mathsf{DL}) \subseteq \mathscr{L}^u(2\mathrm{CA}) \subset \mathrm{NL}^u \subseteq \mathrm{DCSL}^u$. □

Mit dem Modell des Zweiwege-2-Zähler-Automaten können wir die Trennung von DCSL^u auch für $\mathscr{L}^u(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$ erreichen und damit das vorangegangene Ergebnis verbessern. Die echte Inklusion $\mathscr{L}^u(\mathsf{MV}, \mathsf{DL}) \subset \mathrm{DCSL}^u$ folgt dann zwar auch direkt aus dem neuen Ergebnis, die Beweise erscheinen jedoch für sich genommen von einem gewissen Interesse zu sein, weshalb wir hier beide separat angeben.

5.7 Satz
$\mathscr{L}^u(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}^u(2\text{-}2\mathrm{CA})$

Beweis Ein Zweiwege-2-Zähler-Automat $\mathcal{B}$ kann einen $(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$-Automaten $\mathcal{A}$ im unären Fall wie folgt simulieren. Um die Arbeitsweise des $(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$-Automaten $\mathcal{A}$ korrekt nachbilden zu können, muss der Zähler-Automat Folgendes gewährleisten:

1. Die Kopfposition von $\mathcal{A}$ muss stets verfolgt werden können bzw. bekannt sein.

2. Die Anzahl der Leerzeichen am Bandanfang muss gespeichert werden.

3. Die Anzahl der bereits gelöschten Zeichen muss beachtet werden.

$\mathcal{B}$ bewegt seinen Kopf während der Simulation zunächst genau so, wie auch $\mathcal{A}$ den Kopf bewegt. Sobald Zeichen gelöscht werden, kann $\mathcal{B}$ allerdings nicht mehr die exakt gleiche Kopfposition beibehalten, da die genaue Position der bereits gelöschten Zeichen nicht abgespeichert werden kann. Daher bewegt $\mathcal{B}$ seinen Kopf stets derart, dass dessen Abstand zum rechten Bandbegrenzungssymbol der gleiche ist, den auch der Kopf von $\mathcal{A}$ zur rechten Begrenzung hat. Auf diese Weise markiert das rechte Bandbegrenzungssymbol sowohl für $\mathcal{A}$ als auch für $\mathcal{B}$ das Ende der Eingabe.

Den ersten Zähler setzt $\mathcal{B}$ ein, um dem Abstand des Kopfes von $\mathcal{A}$ zum linken Bandbegrenzungssymbol mitzuzählen. Erreicht der Zählerstand null, weiß $\mathcal{B}$, dass $\mathcal{A}$ das linke Bandbegrenzungssymbol erreicht hat – auch wenn dies für seinen eigenen Kopf noch nicht gilt.

Im zweiten Zähler speichert $\mathcal{B}$ schließlich noch den Abstand des Kopfes von $\mathcal{A}$ zur Position c, bis zu der $\mathcal{A}$ die Eingabe bereits ausradiert hat (vergleiche Satz 4.2.9). Da sich der Kopf von $\mathcal{A}$ sowohl links als auch rechts von dieser Position c befinden kann, speichert $\mathcal{B}$ den Abstand im zweiten Zähler und merkt sich zusätzlich ein Vorzeichen für den Zählerstand (das heißt, die Tatsache, ob sich der Kopf links oder rechts von Position c befindet) als Zustandskomponente. Dabei bezeichnet das positive Vorzeichen eine Kopfposition rechts von c.

Ein Zählerstand von null im zweiten Zähler entspricht somit der Tatsache, dass $\mathcal{A}$ auf dem am weitesten rechts liegenden Leerzeichen steht. Bewegt sich $\mathcal{A}$ von diesem auf das am weitesten links liegende Eingabezeichen, bemerkt $\mathcal{B}$ dies durch eine Rechtsbewegung bei Zählerstand null und speichert das positive Vorzeichen in seinen Zuständen ab. Die umgekehrte Bewegung wird entsprechend durch das Erreichen des Zählerstandes null – bei zuvor abgespeichertem positiven Vorzeichen – bemerkt.

Die Simulation der Operationen von $\mathcal{A}$ geschieht daher wie folgt:

$\mathcal{A}$	$\mathcal{B}$		
	Kopfbewegung	Änderung Zähler 1	Änderung Zähler 2
$\mathsf{MV_L}$	links	-1	-1
$\mathsf{ER_L}$ auf a	links	-1	-1
$\mathsf{ER_L}$ auf $\sqcup$	links	-1	-2
$\mathsf{ER_R}$ auf a	rechts	$+1$	-1
$\mathsf{ER_R}$ auf $\sqcup$	rechts	$+1$	± 0
$\mathsf{DL_L}$ auf a	keine	-1	-1
$\mathsf{DL_L}$ auf $\sqcup$	keine	-1	± 0
$\mathsf{DL_R}$ auf a	rechts	± 0	-1
$\mathsf{DL_R}$ auf $\sqcup$	rechts	± 0	± 0

Das Vermindern des ersten Zählers wird nur ausgeführt, wenn sich der Kopf von $\mathcal{A}$ nicht auf dem linken Bandbegrenzungssymbol befindet und der Zähler somit einen positiven Wert enthält. Im Falle des zweiten Zählers bezeichnet das Vermindern jedoch die normale Minus-Operation, wenn man das Vorzeichen einbezieht. Steht der Zähler beispielsweise auf dem Wert $+1$ und wird um zwei vermindert, führt dies zum Ergebnis -1, das heißt zum gleichen Stand des eigentlichen Zählers, aber zu einem Vorzeichenwechsel in der Zustandskomponente.

Auf diese Weise akzeptiert $\mathcal{B}$ die Eingabe genau dann, wenn $\mathcal{A}$ akzeptiert. □

Wir folgern daher:

5.8 Korollar
$\mathscr{L}^u(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL}) \subset \mathrm{DCSL}^u$

Beweis Die Familie $\mathscr{L}^u$(2-2CA) ist nach der bereits erwähnten Zähler-Hierarchie der unären Zweiwege-Zähler-Automaten aus [41] eine echte Teilmenge der Familie NL^u.

Somit gilt: $\mathscr{L}^u(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}^u(\text{2-2CA}) \subset \mathrm{NL}^u \subseteq \mathrm{DCSL}^u$. □

Im deterministischen Fall können wir ein stärkeres Trennungsergebnis für $\mathscr{L}^u_{\mathrm{det}}(\mathsf{MV}, \mathsf{DL})$ als 5.6 angeben, indem wir einen $(\mathsf{MV}, \mathsf{DL})$-Automaten durch eine 2-Register-Maschine simulieren.

Eine *2-Register-Maschine* („*two-register machine*“) besteht aus einer endlichen Zustandskontrolle und zwei Zählern, die hier Register genannt werden. Die Werte dieser Register können, wie üblich, in einem Schritt um eins erhöht bzw. vermindert werden, oder sie können unverändert bleiben. Die Maschine erhält eine natürliche Zahl als Eingabe im ersten Register und hat – im Gegensatz zu den übrigen hier betrachteten Automatenmodellen – kein Eingabeband. Sie arbeitet deterministisch und akzeptiert einen Eingabewert durch das Annehmen eines akzeptierenden Zustands.

5.9 Satz
$\mathscr{L}^u_{\mathrm{det}}(\mathsf{MV}, \mathsf{DL}) \subset \mathscr{L}^u_{\mathrm{det}}(\mathsf{MV}, \mathsf{ER})$

Beweis Die Inklusion $\mathscr{L}^u_{\mathrm{det}}(\mathsf{MV}, \mathsf{DL}) \subseteq \mathscr{L}^u_{\mathrm{det}}(\mathsf{MV}, \mathsf{ER})$ gilt, da in diesem Fall DL-Operationen einfach durch entsprechende ER-Operationen simuliert und die entstehenden Leerzeichen ignoriert werden können (siehe Abschnitt 3.3 bzw. [29]).

Eine 2-Register-Maschine $\mathcal{B}$ kann einen deterministischen $(\mathsf{MV}, \mathsf{DL})$-Automaten $\mathcal{A}$ im unären Fall wie folgt simulieren: $\mathcal{B}$ erhält die Länge der Eingabe von $\mathcal{A}$ als Eingabewert in seinem ersten Register. Während der Simulation

speichert $\mathcal{B}$ im ersten Register die Anzahl der Eingabezeichen, die sich unter dem Kopf von $\mathcal{A}$ und rechts von ihm befinden. Im zweiten Register werden entsprechend die Zeichen links vom Kopf mitgezählt. MV-Operationen kann $\mathcal{B}$ daher durch die entsprechende Verlagerung des Zählerstandes simulieren. Bewegt sich $\mathcal{A}$ beispielsweise ein Zeichen nach links, erhöht $\mathcal{B}$ den Wert des ersten Registers und vermindert den Wert des zweiten Registers jeweils um eins. DL_L-Operationen (bzw. DL_R-Operationen) kann $\mathcal{B}$ durch das Vermindern des zweiten (bzw. des ersten) Registers simulieren.

In [25] wurde bewiesen, dass die Menge von natürlichen Zahlen

$$L_2 = \{n^2 \mid n \geq 0\}$$

– die der unären Sprache $L_2' = \{a^{n^2} \mid n \geq 0\}$ entspricht – nicht durch eine deterministische 2-Register-Maschine erkannt werden kann. Ein deterministischer (MV, ER)-Automat $\mathcal{C}$ kann L_2' wie folgt erkennen: $\mathcal{C}$ markiert, beginnend bei 1 und 4, sukzessive die Positionen der Quadratzahlen. Da die Differenz $(i+1)^2 - i^2$ zwischen zwei benachbarten Quadratzahlen gleich $2i + 1$ ist, kann $\mathcal{C}$ die Abstände zwischen zwei Zahlen jeweils nach rechts kopieren und um zwei erhöhen.

Zu diesem Zweck markiert $\mathcal{C}$ die Positionen i^2 und $i^2 + 1$ mit Leerzeichen und radiert die umgebenden Eingabezeichen (das heißt die Zeichen an den Positionen $i^2 - 1$ und $i^2 + 2$, siehe Abbildung 5.1) nicht aus.

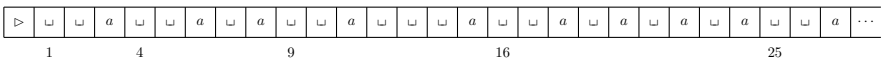


Abbildung 5.1: Der Bandinhalt nach dem Markieren der Position 25.

Die Teilwörter $a␣␣a$ dienen dabei als Begrenzungen, die es $\mathcal{C}$ erlauben, sich auf dem Band hin und her zu bewegen, ohne das Wissen über die aktuelle Position zu verlieren. Beim Kopieren von einem Bereich in den nächsten markiert $\mathcal{C}$ nur jede zweite Position zwischen den Begrenzungen. Auf diese Weise können die Felder zwischen den Quadratzahlen i^2 und $(i+1)^2$ zweimal verwendet werden:

1. Für das „Hineinkopieren“ des Abstands $2(i-1)+1$ von links, mit anschließendem Hinzufügen zweier weiterer Felder. Dabei werden die Positionen $i^2+3, i^2+5, \ldots, (i+1)^2-4, (i+1)^2-2$ verwendet.

2. Für das „Weiterkopieren“ des Abstands $2i+1$ nach rechts. Hierzu werden die Positionen $i^2{+}4, i^2{+}6, \ldots, (i{+}1)^2{-}5, (i{+}1)^2{-}3$ verwendet.

Um den Kopierprozess zu verdeutlichen, wird in Abbildung 5.2 ein Teil des Bandes im Verlauf der Simulation dargestellt. Jede Zeile repräsentiert den Inhalt des Bandes (zwischen den Positionen 15 und 51), nachdem ein weiteres Feld ausradiert wurde.

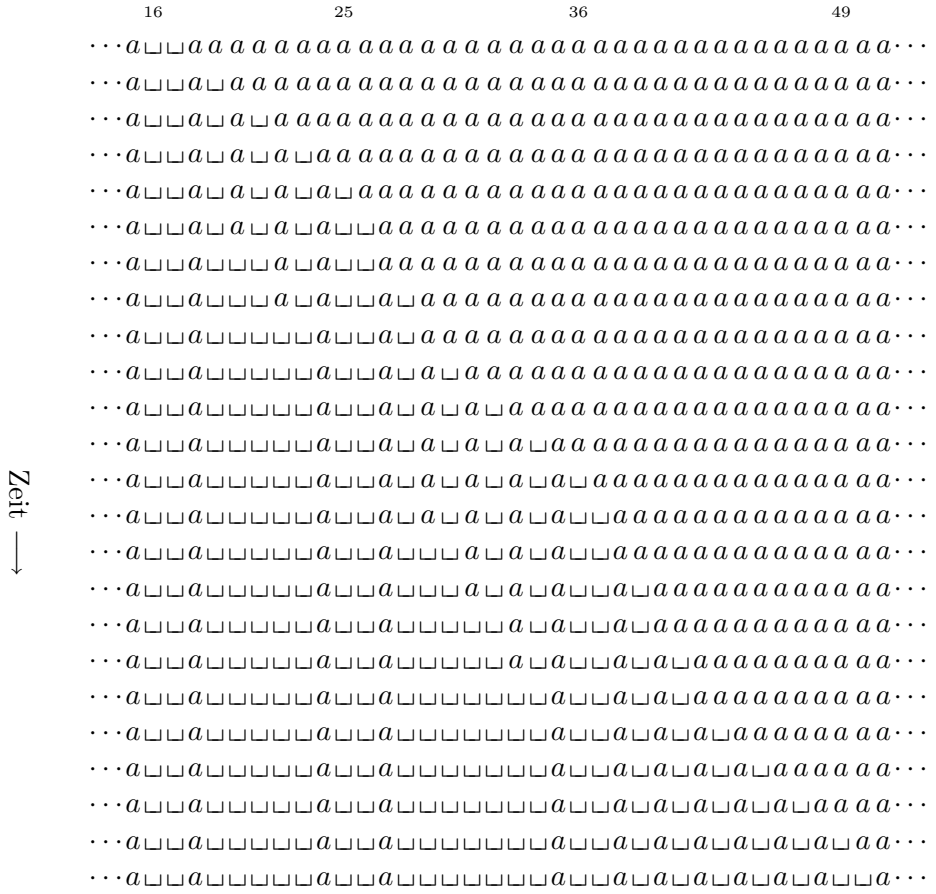


Abbildung 5.2: Eine Folge von Bandinhalten, die während der Berechnung des Automaten $\mathcal{C}$ auftreten.

Auf die beschriebene Weise akzeptiert $\mathcal{C}$ schließlich die Eingabe genau dann, wenn am Ende dieser Konstruktion beim Erreichen des rechten Bandbegrenzungssymbols die Position des Symbols links von $\lhd$ eine Quadratzahl ist. □

Ein offenes Problem aus [29] ist die Frage, ob die Berechnungsmächtigkeit von (MV, ER)-Automaten größer als die von (MV, DL)-Automaten ist. Der

vorangegangene Satz (bzw. das folgende Korollar) löst dieses Problem zumindest teilweise – nämlich für den deterministischen Fall.

5.10 Korollar
$\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL}) \subset \mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$

Beweis Die Inklusion $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL}) \subseteq \mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$ gilt wiederum, da hier DL-Operationen einfach durch entsprechende ER-Operationen simuliert werden können (siehe Abschnitt 3.3 bzw. [29]).

Die Echtheit der Inklusion folgt aus Satz 5.9. □

Im deterministischen Fall können wir zudem auch einen $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automaten durch einen $(\mathsf{MV}, \mathsf{DL})$-Automaten simulieren:

5.11 Satz
$\mathscr{L}^u_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL}) \subseteq \mathscr{L}^u_{\det}(\mathsf{MV}, \mathsf{DL})$

Beweis Ein deterministischer $(\mathsf{MV}, \mathsf{DL})$-Automat $\mathcal{B}$ kann einen deterministischen $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automaten $\mathcal{A}$ im unären Fall wie folgt simulieren:

Um $\mathcal{A}$ korrekt zu simulieren, muss $\mathcal{B}$ die Position c verfolgen, bis zu der $\mathcal{A}$ die Eingabe ausradiert hat (vergleiche Satz 4.2.9). Um das Verhalten von $\mathcal{A}$ auf einer hinreichend langen Eingabe simulieren zu können (die endlich vielen Eingaben bis zu einer bestimmen festen Länge können direkt abgearbeitet werden), betrachten wir Schleifen der Formen

$$\triangleright\sqcup^k xa^n\triangleleft \quad \vdash^* \quad \triangleright\sqcup^k xa^m\triangleleft \qquad (m \leq n) \qquad \text{und} \qquad (1)$$

$$\triangleright\sqcup^k xa^n\triangleleft \quad \vdash^* \quad \triangleright\sqcup^l xa^m\triangleleft \qquad (m < n, k < l), \qquad (2)$$

bei denen $\mathcal{A}$ den gleichen Zustand mehrfach annimmt und endlos läuft (falls $m = n$) oder schließlich das rechte Bandbegrenzungssymbol erreicht. Diese Schleifen bestimmen das langfristige Verhalten des Automaten und machen somit den Kern der Berechnung aus, die es zu simulieren gilt. Schleifen einer möglichen dritten Form

$$\triangleright\sqcup^k xa^n\triangleleft \quad \vdash^* \quad \triangleright\sqcup^l xa^m\triangleleft \qquad (m > n, k > l) \qquad (3)$$

müssen dabei nicht betrachtet werden, da sich $\mathcal{A}$ bis zum Beginn einer solchen Schleife maximal s Schritte (s bezeichne die Anzahl der Zustände von $\mathcal{A}$) vom linken Bandbegrenzungssymbols entfernt haben kann. Läuft $\mathcal{A}$ in eine Schleife der Form (3), erreicht er aufgrund der Bedingung $k > l$ kurz darauf wieder das linke Bandbegrenzungssymbol. Daher ergibt sich – auch bei mehreren Durchläufen dieser Form – bei Eingaben hinreichender Länge irgendwann eine Schleife der Form (1) oder (2). Eine Schleife der Form (3)

kann daher als anfängliche, endliche Teilberechnung angesehen und von der weiteren Betrachtung ausgenommen werden.

$\mathcal{B}$ simuliert daher diese Schleifen, während er mit dem Kopf an Position c stehen bleibt. Zunächst simuliert $\mathcal{B}$ die ersten Schritte bis zum Erreichen der Schleife direkt. Für Form (1) kann $\mathcal{B}$ das Verkürzen der verbleibenden Eingabe aus a^* direkt an Position c ausführen, während für Form (2) die Simulation der Schleife durch eine Rechtsbewegung (Position c folgend) und – falls nötig – durch ein Löschen von entsprechend vielen Symbolen erfolgt.

Um das Verhalten von $\mathcal{A}$ beim Erreichen von $\triangleleft$ simulieren zu können, überprüft $\mathcal{B}$ vor der Simulation eines jeden Schleifendurchlaufs die Distanz zum rechten Bandbegrenzungssymbol. Da $\mathcal{A}$ sich während eines Schleifendurchlaufs vor- und zurückbewegen kann, könnte das Verhalten durch das bloße Löschen der entsprechenden Anzahl von Symbolen vor Ort eventuell nicht korrekt erfasst werden. $\mathcal{B}$ bewegt sich daher zunächst stets eine bestimmte Anzahl von Schritten nach rechts – diese Anzahl ist endlich und durch $|S_{\mathcal{A}}|$ nach oben beschränkt – und überprüft, ob ein weiterer Schleifendurchlauf problemlos simuliert werden kann, das heißt, ob $\mathcal{A}$ das rechte Bandbegrenzungssymbol noch nicht erreicht. $\mathcal{B}$ kann schließlich die letzten Schritte vor dem Erreichen des rechten Bandbegrenzungssymbols direkt simulieren und anschließend ER-Operationen durch entsprechende MV-Operationen simulieren (da der verbliebene Bandinhalt ausschließlich aus Leerzeichen besteht) sowie MV_L- und DL-Operationen unverändert übernehmen.

Auf diese Weise akzeptiert $\mathcal{B}$ genau dann, wenn $\mathcal{A}$ akzeptiert. □

Im Verbindung mit Satz 5.4 erhalten wir somit im deterministischen Fall eine Gleichheit der folgenden drei Sprachfamilien:

5.12 Korollar
$\mathscr{L}^u_{\det}(\mathsf{MV}, \mathsf{DL}) = \mathscr{L}^u_{\det}(\mathsf{ER}, \mathsf{DL}) = \mathscr{L}^u_{\det}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$

Für eine Übersicht der Klassifikation der unären Forgetting-Automaten im nichtdeterministischen bzw. deterministischen Fall siehe Abbildungen 5.3 und 5.4.

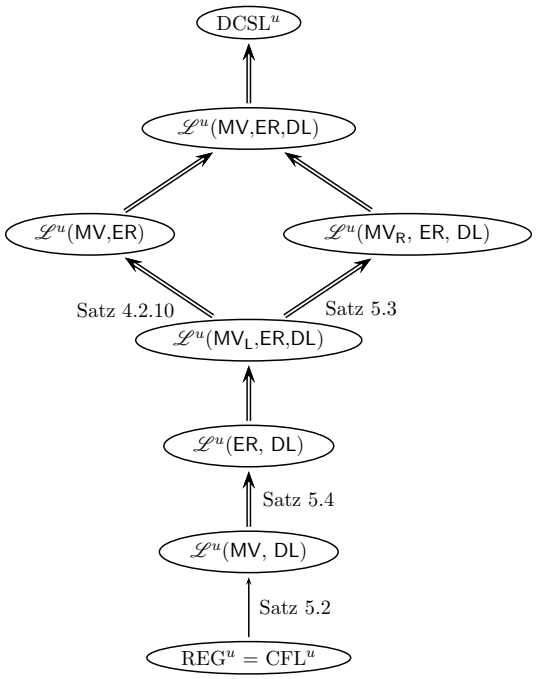


Abbildung 5.3: Die Hierarchie der nichtdeterministischen unären Forgetting-Automaten (⇒: Inklusion, →: echte Inklusion).

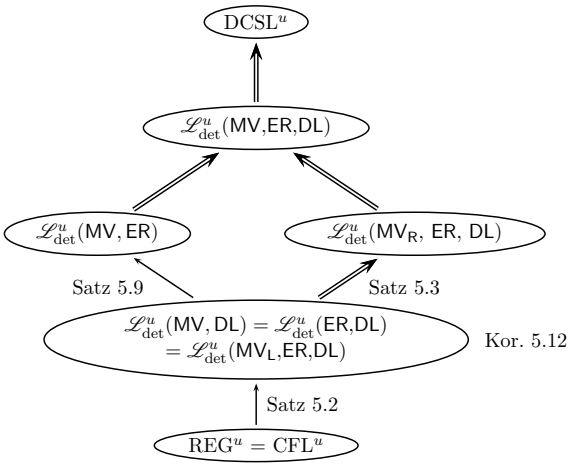


Abbildung 5.4: Die Hierarchie der deterministischen unären Forgetting-Automaten.

Kapitel 6

Abschlusseigenschaften

In diesem Kapitel werden die Abschlusseigenschaften der Familien der von Forgetting-Automaten erkannten Sprachen untersucht. Als Abschlusseigenschaft bezeichnet man die Eigenschaft einer Sprachfamilie, unter einer bestimmten Operation – wie zum Beispiel der mengentheoretischen Vereinigung – abgeschlossen zu sein oder nicht. Mit anderen Worten: Es wird die Frage untersucht, ob die Anwendung einer in der Regel ein- oder zweistelligen Operation auf eine bzw. zwei Sprachen aus einer gegebenen Sprachfamilie $\mathscr{L}$ immer eine Sprache ergibt, die wieder in $\mathscr{L}$ liegt, oder ob sich auch Sprachen ergeben können, die nicht mehr in $\mathscr{L}$ liegen.

Wir betrachten hier die folgenden Operationen:

Vereinigung Gilt $L_1 \cup L_2 \in \mathscr{L}$ für alle $L_1, L_2 \in \mathscr{L}$?

markierte Vereinigung Gilt $\{0\}L_1 \cup \{1\}L_2 \in \mathscr{L}$ für alle $L_1, L_2 \in \mathscr{L}$?

Vereinigung mit regulären Sprachen Gilt $L \cup R \in \mathscr{L}$ für alle $L \in \mathscr{L}$ und $R \in \mathrm{REG}$?

Durchschnitt Gilt $L_1 \cap L_2 \in \mathscr{L}$ für alle $L_1, L_2 \in \mathscr{L}$?

Durchschnitt mit regulären Sprachen Gilt $L \cap R \in \mathscr{L}$ für alle $L \in \mathscr{L}$ und $R \in \mathrm{REG}$?

Komplement Gilt $\overline{L} \in \mathscr{L}$ für alle $L \in \mathscr{L}$?

Spiegelung Gilt $L^R \in \mathscr{L}$ für alle $L \in \mathscr{L}$?

Konkatenation Gilt $L_1L_2 \in \mathscr{L}$ für alle $L_1, L_2 \in \mathscr{L}$?

markierte Konkatenation Gilt $L_1\#L_2 \in \mathscr{L}$ für alle $L_1, L_2 \in \mathscr{L}$ (mit einem Trennzeichen $\#$, das nicht in L_1 oder L_2 vorkommt)?

Iteration Gilt $\bigcup_{i\geq 0} L^i \in \mathscr{L}$ für alle $L \in \mathscr{L}$?

markierte Iteration Gilt $\bigcup_{i\geq 0}(L\#)^i \in \mathscr{L}$ für alle $L \in \mathscr{L}$?

Homomorphismus Gilt $h(L) \in \mathscr{L}$ für alle $L \in \mathscr{L}$ und alle Homomorphismen h?

λ-freier Homomorphismus Gilt $h(L) \in \mathscr{L}$ für alle $L \in \mathscr{L}$ und alle λ-freien Homomorphismen h?

Die Abschlusseigenschaften der Familien der von Forgetting-Automaten erkannten Sprachen

Die Eigenschaften der Sprachfamilien, die mit den Familien der regulären bzw. kontextfreien Sprachen zusammenfallen, sind bereits hinreichend untersucht (siehe z. B. [23]) und müssen daher hier nicht mehr betrachtet werden.

6.1 Satz (Abschluss unter Vereinigung)

(a) Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}(\mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ sind abgeschlossen unter Vereinigung.

(b) Die Sprachfamilien $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ sind nicht abgeschlossen unter Vereinigung.

Beweis

(a) Es sei $\mathscr{L}$ eine der genannten (nichtdeterministischen) Sprachfamilien und es seien die Sprachen L_1 und L_2 aus $\mathscr{L}$ sowie die Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ mit $L(\mathcal{A}_1) = L_1$ und $L(\mathcal{A}_2) = L_2$ gegeben.

Ein nichtdeterministischer Automat $\mathcal{B}$ mit $L(\mathcal{B}) = L_1 \cup L_2$ rät lediglich im ersten Schritt, ob L_1 oder L_2 erkannt werden soll und fährt mit der Simulation von $\mathcal{A}_1$ oder $\mathcal{A}_2$ fort.

(b) Die Sprachen $\{a^n b^n c^m \mid n, m \in \mathbb{N}\}$ und $\{a^n b^m c^m \mid n, m \in \mathbb{N}\}$ sind offensichtlich Zählersprachen, d. h., sie liegen in $\mathscr{L}_{\det}(1\mathrm{CA}) = \mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ und damit auch in den Familien $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$.

Die Vereinigung $\{a^n b^n c^m \mid n, m \in \mathbb{N}\} \cup \{a^n b^m c^m \mid n, m \in \mathbb{N}\}$ liegt jedoch nicht in DCFL ([2, Seite 221]) und somit nicht in den genannten darin enthaltenen Sprachfamilien.

Die Familie $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ ist nicht unter Vereinigung abgeschlossen, da sie – wie in den Sätzen 6.5 und 6.7 gezeigt wird – unter Schnitt

nicht abgeschlossen, unter Komplement jedoch abgeschlossen ist. Wäre $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ unter Vereinigung abgeschlossen, würde durch die Abgeschlossenheit unter Komplement nach dem Satz von De Morgan auch die Abgeschlossenheit unter Schnitt (und somit ein Widerspruch) folgen. □

Wie wir gesehen haben, sind alle Familien der von nichtdeterministischen Modellen erzeugten Sprachen unter Vereinigung abgeschlossen. Vereinfachen wir die Aufgabenstellung durch eine für jede der beiden zu vereinigenden Sprachen jeweils unterschiedliche Markierung des ersten Zeichens, können auch alle deterministischen Modelle diese veränderte Form der Vereinigung erkennen. Diese Erweiterung bezeichnen wir als *markierte Vereinigung*:

6.2 Definition (markierte Vereinigung)
Für zwei Sprachen L_1 und L_2 ist die markierte Vereinigung $L_1 \oplus L_2$ definiert als

$$L_1 \oplus L_2 = \{0x \mid x \in L_1\} \cup \{1x \mid x \in L_2\}.$$

6.3 Satz (Abschluss unter markierter Vereinigung)
Alle Familien von Forgetting-Automaten sind abgeschlossen unter markierter Vereinigung.

Beweis Es seien die Sprachen L_1 und L_2 aus einer Familie $\mathscr{L}(O)$ von Forgetting-Automaten gegeben, wobei O eine Menge möglicher Operationen ist. Es seien ferner die O-Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ mit $L(\mathcal{A}_1) = L_1$ und $L(\mathcal{A}_2) = L_2$ gegeben. Dann kann ein O-Automat $\mathcal{B}$ die markierte Vereinigung $L_1 \oplus L_2$ wie folgt erkennen, wobei wir zwei Fälle unterscheiden:

1. Ist $\mathsf{DL}_\mathsf{R} \in O$, so kann $\mathcal{B}$ das erste Zeichen (das heißt die Markierung) mit einem DL_R-Schritt entfernen und dem ersten Zeichen entsprechend mit der Simulation von $\mathcal{A}_1$ oder $\mathcal{A}_2$ fortfahren.

2. Ist $\mathsf{DL}_\mathsf{R} \notin O$, dies sind die Fälle $O = \{\mathsf{MV}_\mathsf{R}, \mathsf{ER}\}$ und $O = \{\mathsf{MV}, \mathsf{ER}\}$, radiert $\mathcal{B}$ das erste Zeichen zunächst aus und beginnt mit der Simulation von $\mathcal{A}_1$ bzw. $\mathcal{A}_2$ auf der restlichen Eingabe x. Wenn sich $\mathcal{B}$ während dieser Simulation nach links bewegt und ein $\sqcup$ erreicht, läuft er mit einem weiteren ER_L-Schritt nach links und testet, ob er auf das linke Bandbegrenzungssymbol oder ein anderes Symbol (das heißt ein Eingabesymbol oder $\sqcup$) trifft. Im ersten Fall simuliert $\mathcal{B}$ das Verhalten beim Auftreffen auf $\triangleright$ (das Erreichen des ausradierten Symbols entsprach hier bereits dem Erreichen der linken Bandbegrenzung) und bewegt sich mit zwei MV_R-Schritten nach rechts über das ausradierte Markierungssymbol zum ersten Zeichen des zu verarbeitenden Wortes x zurück. Im zweiten Fall bewegt sich $\mathcal{B}$ lediglich mit einem MV_R-Schritt wieder nach rechts und fährt mit der Simulation fort.

Auf diese Weise erkennt $\mathcal{B}$ genau die Sprache $L_1 \oplus L_2$. □

Eine weitere Veränderung bzw. Einschränkung der Vereinigungsoperation ist die *Vereinigung mit regulären Sprachen*. Hier werden nicht zwei Sprachen aus einer Sprachfamilie miteinander vereinigt, sondern nur eine Sprache aus einer gegebenen Sprachfamilie und eine reguläre Sprache. Auch in diesem Fall sind alle Automatenmodelle in der Lage, die daraus entstehenden Sprachen zu akzeptieren.

6.4 Satz (Abschluss unter Vereinigung mit regulären Sprachen) Alle Familien von Forgetting-Automaten sind abgeschlossen unter Vereinigung mit regulären Sprachen.

Beweis Da alle nichtdeterministischen Sprachfamilien nach Satz 6.1 unter Vereinigung abgeschlossen sind und die Familie der regulären Sprachen eine Teilmenge jeder Familie von Forgetting-Automaten ist, sind die nichtdeterministischen Sprachfamilien auch unter Vereinigung mit regulären Sprachen abgeschlossen.

Die Sprachfamilien $\mathscr{L}_{\text{det}}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\text{det}}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\text{det}}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\text{det}}(\mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\text{det}}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ sind nach Satz 6.7 (siehe unten) unter Komplementbildung abgeschlossen; nach Satz 6.6 sind sie außerdem unter Schnitt mit regulären Sprachen abgeschlossen. Nach dem Gesetz von De Morgan und der Abgeschlossenheit der regulären Sprachen unter Komplementbildung ist damit auch für eine Sprache L_1 aus einer der genannten Sprachfamilien $\mathscr{L}$ und für eine reguläre Sprache L_2 die Vereinigung $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$ wieder Element von $\mathscr{L}$.

Für die Sprachfamilien $\mathscr{L}_{\text{det}}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\text{det}}(\mathsf{MV}, \mathsf{ER})$ und $\mathscr{L}_{\text{det}}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ gilt die Abgeschlossenheit aufgrund des Vorhandenseins der $\mathsf{MV_L}$- und der $\mathsf{MV_R}$-Operation: Es seien eine Sprache L_1 aus einer der gegebenen Sprachfamilien und ein entsprechender Forgetting-Automat $\mathcal{A}_1$ mit $L(\mathcal{A}_1) = L_1$ gegeben. Es seien ferner die reguläre Sprache L_2 und ein endlicher Automat $\mathcal{A}_2$ mit $L(\mathcal{A}_2) = L_2$ gegeben. Die Vereinigung $L_1 \cup L_2$ kann von einem Forgetting-Automaten $\mathcal{B}$ (gleichen Typs wie $\mathcal{A}_1$) erkannt werden, indem $\mathcal{B}$ sich zunächst mit $\mathsf{MV_R}$-Schritten einmal von links nach rechts über die Eingabe bewegt und dabei $\mathcal{A}_2$ simuliert. Wird die Eingabe von $\mathcal{A}_2$ akzeptiert, so akzeptiert auch $\mathcal{B}$ die Eingabe an diesem Punkt. Anderenfalls kann sich $\mathcal{B}$ mit $\mathsf{MV_L}$-Schritten wieder zurück zum linken Bandbegrenzungssymbol bewegen und die Berechnung des Automaten $\mathcal{A}_1$ (auf der bisher unveränderten Eingabe) simulieren. □

Im Fall der Schnittmengenbildung sind alle Sprachfamilien zwischen den Familien der regulären und der kontextfreien Sprachen nicht abgeschlossen. Da für die restlichen Sprachfamilien nur sehr wenige Zeugensprachen bekannt

sind, die nicht in ihnen enthalten sind, konnte der Nichtabschluss für diese Familien bisher nicht gezeigt werden.

6.5 Satz (Abschluss unter Schnitt)
Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ sind nicht abgeschlossen unter Schnitt.

Beweis Die Sprachen $\{a^n b^n c^m \mid n, m \in \mathbb{N}\}$ und $\{a^n b^m c^m \mid n, m \in \mathbb{N}\}$ sind offensichtlich Zählersprachen, d. h., sie liegen in $\mathscr{L}_{\det}(1\mathrm{CA}) = \mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ und damit auch in den anderen genannten Sprachfamilien. Die Schnittmenge $\{a^n b^n c^m \mid n, m \in \mathbb{N}\} \cap \{a^n b^m c^m \mid n, m \in \mathbb{N}\} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ist jedoch nicht kontextfrei und liegt somit nicht in den genannten Sprachfamilien.

Zu $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$:

Wir betrachten erneut die Sprache

$$L := \left\{ a^{2^n} c a^m b^m \;\middle|\; n, m \in \mathbb{N} \right\},$$

von der im Beweis von Satz 4.2.7 gezeigt wurde, dass sie nicht von einem deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten erkannt werden kann. L kann als Schnitt der Sprachen

$$L_1 := \left\{ a^n c a^m b^m \;\middle|\; n, m \in \mathbb{N} \right\}$$

und

$$L_2 := \left\{ a^{2^n} c a^m b^l \;\middle|\; n, m, l \in \mathbb{N} \right\}$$

dargestellt werden, die beide in $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ liegen. Im Fall von L_1 kann ein deterministischer $(\mathsf{ER}, \mathsf{DL})$-Automat zunächst alle a und das c löschen und schließlich das restliche Teilwort auf die Form $a^m b^m$ überprüfen (vergleiche Beispiel 3.2.1). Im Fall von L_2 kann sich ein deterministischer $(\mathsf{ER}, \mathsf{DL})$-Automat zunächst mit $\mathsf{ER_R}$-Schritten über das erste Teilwort bewegen und alle Zeichen ab dem Trennzeichen c mit $\mathsf{DL_R}$-Schritten löschen; dabei muss der Automat die Zeichen nach dem Trennzeichen lediglich auf die reguläre Struktur $a^* b^*$ testen. Nach dem Erreichen des rechten Bandbegrenzungssymbols kann der Automat analog zu Beispiel 3.2.5 durch schrittweises Halbieren der verbliebenen Eingabe testen, ob die Länge des ersten Teilwortes eine Zweierpotenz war.

Daher ist $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ nicht unter Schnitt abgeschlossen. □

Untersuchen wir nicht den allgemeinen Fall der Schnittbildung, sondern nur den Schnitt mit regulären Sprachen, so sind, wie auch im Fall der Vereinigung, alle betrachteten Sprachfamilien unter dieser Operation abgeschlossen.

6.6 Satz (Abschluss unter Schnitt mit regulären Sprachen)
Alle Familien von Forgetting-Automaten sind abgeschlossen unter Schnitt mit regulären Sprachen.

Beweis Alle Klassen sowohl der von nichtdeterministischen als auch der von deterministischen (MV, DL)-, (MV, ER)- und (MV, ER, DL)-Automaten erkannten Sprachen sind aufgrund des Vorhandenseins der $\mathsf{MV_L}$- und der $\mathsf{MV_R}$-Operation abgeschlossen (vgl. Satz 6.4). Ein zugehöriger Forgetting-Automat kann zunächst einmal über die gesamte Eingabe laufen und das Enthaltensein in der regulären Sprache überprüfen; anschließend bewegt sich der Automat zurück zum linken Bandbegrenzungssymbol und simuliert den entsprechenden Forgetting-Automaten. Die Eingabe wird schließlich akzeptiert, wenn sowohl der simulierte endliche Automat als auch der simulierte Forgetting-Automat die Eingabe akzeptieren.

Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{ER}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$ sind aufgrund der folgenden Tatsache unter Schnitt mit regulären Sprachen abgeschlossen: Ein zu einer Sprachfamilie gehörender Forgetting-Automat kann parallel zur Verarbeitung eines Wortes einen deterministischen endlichen Automaten simulieren; dabei werden nur Eingabezeichen verarbeitet – das heißt nicht bereits ausradierte und daher mit $\sqcup$ beschriftete Felder – und davon nur solche, die in Folge von Rechtsbewegungen, also durch eine $\mathsf{MV_R}$-, $\mathsf{ER_R}$- oder $\mathsf{DL_R}$-Operation, erreicht werden.

Für die Familien der von (nichtdeterministischen oder deterministischen) ($\mathsf{MV_L}$, ER, DL)-Automaten erkannten Sprachen funktioniert die parallel ablaufende Simulation eines deterministischen endlichen Automaten auf ähnliche Weise: Durch Rechtsschritte erreichte Eingabezeichen werden zunächst jeweils als neue Eingabe des endlichen Automaten interpretiert; wird jedoch ein Eingabezeichen mithilfe eines $\mathsf{MV_L}$-Schrittes verlassen, so merkt sich der Forgetting-Automat dies und bezieht es beim erneuten Erreichen des Zeichens nicht in die Simulation des endlichen Automaten ein. Man beachte, dass jeweils nur ein einziges solches Eingabezeichen auftreten kann, da hier Rechtsbewegungen lediglich mit einer $\mathsf{ER_R}$- oder $\mathsf{DL_R}$-Operation möglich sind und somit keine Eingabezeichen links vom Lesekopf auftreten, die mit $\mathsf{MV_L}$-Bewegungen nach links überwandert werden können. □

Als weitere Anwendung der Lemmata aus Abschnitt 3.4 können wir nun den Abschluss unter Komplement für die entsprechenden deterministischen Modelle zeigen.

6.7 Satz (Abschluss unter Komplement)

(a) Die Sprachfamilien $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ sind abgeschlossen unter Komplement.

(b) Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$ sind nicht abgeschlossen unter Komplement.

Beweis

(a) Nach den Lemmata 3.4.4 bis 3.4.9 können deterministische $(\mathsf{ER_R}, \mathsf{DL})$-, $(\mathsf{MV_R}, \mathsf{DL})$-, $(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$-, $(\mathsf{ER}, \mathsf{DL})$- und $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automaten als haltend angenommen werden. Für eine Sprache L aus einer der Sprachfamilien $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ oder $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ und einen entsprechenden Automaten $\mathcal{A}$ mit $L(\mathcal{A}) = L$ erhalten wir daher den Automaten $\mathcal{B}$ mit $L(\mathcal{B}) = \overline{L}$ aus $\mathcal{A}$ durch Vertauschen von End- und Nichtendzuständen.

(b) Nach Satz 6.1 ist die Familie $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ unter Vereinigung abgeschlossen. Wäre $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ auch unter Komplement abgeschlossen, würde nach dem Satz von De Morgan auch die Abgeschlossenheit unter Schnitt gelten. Da die Familie $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ nach Satz 6.5 jedoch nicht unter Schnitt abgeschlossen ist, kann sie auch unter Komplement nicht abgeschlossen sein.

Analog folgt der Nichtabschluss unter Komplement für $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$. □

6.8 Satz (Abschluss unter Spiegelung)

(a) Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ sind abgeschlossen unter Spiegelung.

(b) Die Sprachfamilien $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ sind nicht abgeschlossen unter Spiegelung.

Beweis

(a) Die Familie $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ stimmt mit $\mathscr{L}(1\text{-CA})$ überein und ist daher unter Spiegelung abgeschlossen (siehe [17]).

Die Familien $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ sind abgeschlossen unter Spiegelung, da die jeweiligen Automatenmodelle zu jeder Operation in eine Richtung auch die entsprechende Operation in die entgegengesetzte

Richtung ausführen können. Für eine Sprache L aus einer der genannten Sprachfamilien und einen entsprechenden Automaten $\mathcal{A}$ mit $L(\mathcal{A}) = L$ erhalten wir daher den Automaten $\mathcal{B}$ mit $L(\mathcal{B}) = L^R$ wie folgt: Zunächst bewegt sich $\mathcal{B}$ nach rechts bis zum rechten Bandbegrenzungssymbol $\triangleleft$ und simuliert anschließend den Automaten $\mathcal{A}$, wobei alle Operationen in umgekehrter Richtung ausgeführt werden. Auf diese Weise akzeptiert $\mathcal{B}$ die Eingabe w genau dann, wenn $\mathcal{A}$ die Eingabe w^R akzeptiert.

(b) $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$: Die Sprache

$$L = \{a0^n1^n2^m \mid n, m \in \mathbb{N}\} \cup \{b0^n1^m2^m \mid n, m \in \mathbb{N}\}$$

kann von einem deterministischen $(\mathsf{ER_R}, \mathsf{DL})$-Automaten $\mathcal{A}$ wie folgt erkannt werden:

Ist das erste Zeichen ein a, löscht $\mathcal{A}$ zunächst das a und bewegt sich anschließend mit $\mathsf{ER_R}$-Operationen über die Nullen bis zur ersten Eins. Durch abwechselnde $\mathsf{DL_L}$- und $\mathsf{DL_R}$-Operationen löscht er dann jeweils eine Eins und ein Leerzeichen, das heißt eine ausradierte Null. Auf diese Weise kann die Anzahl der Nullen und Einsen verglichen werden. Ist das erste Zeichen ein b, löscht $\mathcal{A}$ zunächst alle Nullen und vergleicht anschließend die Anzahl der Einsen und Zweien auf analoge Weise. Die Eingabe wird nicht akzeptiert, falls nach dem ersten Zeichen kein Wort der Form $0^*1^*2^*$ folgt. Auf diese Weise akzeptiert $\mathcal{A}$ genau die angegebene Sprache L.

Die Spiegelung L^R von L ist jedoch nicht deterministisch kontextfrei und liegt damit nicht in $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, da $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ eine Teilmenge von DCFL ist.

Die obige Zeugensprache L kann auch für den Nichtabschluss der Sprachfamilien $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ herangezogen werden, da diese jeweils Obermengen von $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ und Teilmengen von DCFL sind. □

6.9 Satz (Abschluss unter Konkatenation)

(a) Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ sind abgeschlossen unter Konkatenation.

(b) Die Sprachfamilien $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ sind nicht abgeschlossen unter Konkatenation.

Beweis

(a) Es seien im Folgenden zu jeder Sprachfamilie $\mathscr{L}(O)$ stets die Sprachen $L_1, L_2 \in \mathscr{L}(O)$ und die (O)-Automaten $\mathcal{A}_1, \mathcal{A}_2$ mit $L(\mathcal{A}_1) = L_1$ sowie $L(\mathcal{A}_2) = L_2$ gegeben.

Zu $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$:

Wir konstruieren einen Automaten $\mathcal{B}$, der die Konkatenation $L = L_1L_2$ akzeptiert. Der Automat versucht dabei, eine Zerlegung seiner Eingabe w in $w = w_1w_2$ mit $w_1 \in L_1$ und $w_2 \in L_2$ zu finden.

Analog zum Beweis von Lemma 3.4.4 kann das Abarbeiten der verbliebenen unären Eingabe links vom Kopf nach Erreichen des rechten Bandbegrenzungssymbols erledigt werden, ohne Rechtsbewegungen auf dem jeweils letzten Leerzeichen ausführen zu müssen: Eine $\mathsf{ER_R}$-Operation auf dem letzten Leerzeichen kann – inklusive einer darauffolgenden $\mathsf{MV_L}$-Operation auf dem rechten Bandbegrenzungssymbol – direkt simuliert werden, ohne den Kopf bewegen zu müssen. Weiterhin können $\mathsf{DL_R}$-Operationen samt einer $\mathsf{MV_L}$-Operation mithilfe einer $\mathsf{DL_L}$-Operation simuliert werden, die die zwei Zustandswechsel in einem Schritt zusammenfasst; hierdurch wird in beiden Fällen die Eingabe um ein Zeichen verkürzt und der Kopf des Automaten steht erneut auf dem letzten der verbliebenen Leerzeichen. Die $\mathsf{DL_L}$-Operationen hingegen können direkt simuliert werden.

Ein nichtdeterministischer $(\mathsf{ER_R}, \mathsf{DL})$-Automat kann daher raten, dass sich der Kopf momentan an der Konkatenationsstelle (das heißt auf dem letzten Zeichen des Wortes $w_1 \in L_1$) befindet. In diesem Fall löscht er das Zeichen unter dem Kopf und die verbliebenen Zeichen links vom Kopf mit $\mathsf{DL_L}$-Schritten und überprüft – falls er die Eingabe w_1 akzeptiert – die restliche Eingabe auf das Enthaltensein in L_2.

Somit akzeptiert $\mathcal{B}$ genau die Sprache $L = L_1L_2$.

Zu $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$:

Nach Lemma 3.4.5 kann ein $(\mathsf{MV_R}, \mathsf{DL})$-Automat nach Erreichen des rechten Bandbegrenzungssymbols $\lhd$ die restliche Eingabe in einem Zug löschen.

Der simulierende Automat $\mathcal{B}$ verhält sich daher wie folgt:

1. Er beginnt zunächst mit der Simulation von $\mathcal{A}_1$.
2. Er rät während der Simulation die Konkatenationsstelle.
3. Schließlich löscht er die restliche Eingabe des ersten Wortes gemäß dem bereits erwähnten Lemma 3.4.5.

Anschließend kann $\mathcal{B}$ mit der Simulation von $\mathcal{A}_2$ auf der verbliebenen Eingabe fortfahren.

Zu $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$:

Ein $(\mathsf{MV}, \mathsf{ER})$-Automat $\mathcal{B}$ mit $L(\mathcal{B}) = L_1 L_2$ kann wie folgt konstruiert werden:

1. $\mathcal{B}$ bewegt sich zunächst mit MV_R-Schritten nach rechts und rät ein beliebiges Feld, dessen Inhalt er sich merkt und anschließend ausradiert. $\mathcal{B}$ unterteilt die Eingabe w damit in zwei Teile w_1 (dessen letztes Zeichen das zuvor ausradierte ist) und w_2, für die der Automat nacheinander das Enthaltensein von w_1 in L_1 und das Enthaltensein von w_2 in L_2 testet.
2. $\mathcal{B}$ kehrt zum Bandanfang zurück und simuliert das Verhalten von $\mathcal{A}_1$. Während der Simulation von $\mathcal{A}_1$ dient das am weitesten rechts liegende Leerzeichen als Markierung für das letzte Zeichen von w_1. Betritt $\mathcal{B}$ dabei ein Leerzeichen, bestimmt er zunächst, ob es sich um das als Begrenzung dienende Symbol handelt:
   - $\mathcal{B}$ läuft nach rechts, bis das nächste Leerzeichen oder (das eigentliche Bandbegrenzungssymbol) $\lhd$ erreicht wird. Im ersten Fall handelte es sich bei dem zuvor erreichten Symbol folglich nicht um das am weitesten rechts liegende, im zweiten Fall ist $\mathcal{B}$ dagegen bereits auf dem am weitesten rechts liegenden Leerzeichen gestartet.
   - $\mathcal{B}$ läuft nun nach links bis zum Erreichen des nächsten Leerzeichens zurück und ist damit wieder am Ausgangspunkt angelangt (ohne den Bandinhalt zu verändern). Entspricht das gelesene Leerzeichen dem letzten Zeichen von w_1, verhält sich $\mathcal{B}$ entsprechend – auch im Falle, dass der nächste Schritt nach rechts führt und somit das Verhalten von $\mathcal{A}_1$ auf $\lhd$ simuliert werden muss. Um $\mathcal{A}_1$ korrekt zu simulieren, merkt sich $\mathcal{B}$ zudem, ob $\mathcal{A}_1$ dieses Feld im Laufe der Simulation bereits ausradiert hat oder nicht.
3. Wird die Eingabe w_1 von $\mathcal{A}_1$ akzeptiert, stellt $\mathcal{B}$ sicher, dass w_1 komplett ausradiert wird, indem er das am weitesten rechts liegende Leerzeichen sucht und mit ER_L-Operationen bis zum linken Begrenzungssymbol läuft. Im Anschluss daran bewegt sich $\mathcal{B}$ zum ersten Zeichen von w_2.
4. $\mathcal{B}$ simuliert nun das Verhalten von $\mathcal{A}_2$ auf w_2. Dabei dient das am weitesten links liegende, nicht ausradierte Eingabesymbol als Markierung für das erste Zeichen von w_2. Betritt $\mathcal{B}$ bei der Simulation ein Eingabezeichen, bestimmt er wie folgt, ob es sich um das als Begrenzung dienende Symbol handelt:
   - $\mathcal{B}$ läuft nach links, bis er das nächste Eingabezeichen oder das linke Bandbegrenzungssymbol $\rhd$ erreicht. Im ersten Fall handelt es sich

bei dem zuvor erreichten Symbol folglich nicht um das Bandbegrenzungssymbol, im zweiten Fall schon.
- $\mathcal{B}$ läuft nun nach rechts bis zum Erreichen des nächsten Eingabezeichens zurück und hat damit den Ausgangspunkt wieder erreicht. Ist das gelesene Eingabezeichen das erste Zeichen von w_2, führt $\mathcal{B}$ ein gegebenenfalls zu simulierendes Ausradieren des Feldes nicht aus. Stattdessen merkt $\mathcal{B}$ sich das Ausradieren des Feldes in einer Zustandskomponente und weiß im Falle einer Linksbewegung, dass anschließend das Verhalten von $\mathcal{A}_2$ auf $\triangleright$ zu simulieren ist.

Schließlich akzeptiert $\mathcal{B}$ die Eingabe w genau dann (in einem der nichtdeterministischen Versuche), wenn $w_1 \in L_1$ sowie $w_2 \in L_2$ und somit $w \in L_1 L_2$ gelten.

Zu $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$:

Wir konstruieren einen $(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$-Automaten $\mathcal{B}$ mit $L(\mathcal{B}) = L_1 L_2$ wie folgt:

$\mathcal{B}$ versucht, eine Zerlegung der Eingabe w in $w = w_1 w_2$ mit $w_1 \in L_1$ und $w_2 \in L_2$ zu finden. $\mathcal{B}$ beginnt mit der Simulation von $\mathcal{A}_1$ und rät beim Erreichen eines beliebigen Eingabesymbols, dass das erreichte Zeichen $x \in A$ das erste Zeichen des zweiten Wortes w_2 ist und damit als rechtes Bandbegrenzungssymbol für $\mathcal{A}_1$ fungiert.

Damit das geratene Symbol während der Simulation nicht schon zuvor als Eingabezeichen interpretiert wird (dies könnte zu einer fehlerhaften Simulation von $\mathcal{A}_1$ führen), merkt sich der Automat während der ersten Phase (das heißt bis zum Raten der Konkatenationsstelle) beim Erreichen eines Eingabesymbols und dem anschließenden Verlassen durch $\mathsf{MV_L}$ stets, dass das nächste Erreichen eines Eingabesymbols nicht zum Raten der Konkatenationsstelle führen darf.

Sobald $\mathcal{B}$ also rät, das Bandbegrenzungssymbol für $\mathcal{A}_1$ erreicht zu haben, verlässt er dieses entsprechend mit einer $\mathsf{MV_L}$-Operation und führt die restliche Simulation von $\mathcal{A}_1$ auf dem verbliebenen Bandinhalt der Form $\triangleright \sqcup^* x$ fort.

Sobald $\mathcal{A}_1$ einen akzeptierenden Zustand erreicht, löscht $\mathcal{B}$ alle verbliebenen Leerzeichen und fährt mit der Simulation von $\mathcal{A}_2$ fort. Auf diese Weise akzeptiert $\mathcal{B}$ genau die Konkatenation der Sprachen L_1 und L_2.

Zu $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$:

Analog zu $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$ kann das am weitesten rechts liegende Leerzeichen als Bandbegrenzungssymbol für das Teilwort w_1 dienen. Hier muss sich der Automat während der Simulation von $\mathcal{A}_1$ als Zustandskomponente lediglich drei (statt zwei) verschiedene Möglichkeiten für dieses Feld merken:

1. Das Zeichen ist als das ursprünglich dort vorgefundene Zeichen zu interpretieren, das heißt, das gespeicherte Zeichen ist zu verwenden.
2. Das Zeichen wurde bereits ausradiert und ist damit tatsächlich als Leerzeichen zu interpretieren.
3. Das Zeichen wurde bereits gelöscht, das heißt, es ist als das rechte Bandbegrenzungssymbol zu interpretieren.

Als Vereinfachung der Schritte 3 und 4 kann hier sogar das bereits überprüfte Teilwort komplett gelöscht statt ausradiert werden.

(b) Zu $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$:

Die Sprache

$$L = \{a0^n1^n2^m \mid n, m \in \mathbb{N}_0\} \cup \{0^n1^m2^m \mid n, m \in \mathbb{N}_0\}$$

kann von einem deterministischen $(\mathsf{ER_R}, \mathsf{DL})$-Automaten erkannt werden (vergleiche den Beweis von Satz 6.8).

Die Konkatenation $a^*L = \{a^n \mid n \in \mathbb{N}_0\}L$ der regulären Sprache $\{a^n \mid n \in \mathbb{N}_0\}$ und der deterministisch kontextfreien Sprache L ist hingegen nicht mehr deterministisch kontextfrei (siehe zum Beispiel [23]) und liegt damit nicht in $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, da $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ eine Teilmenge von DCFL ist.

Für $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ kann ebenso die obige Zeugensprache L für den Nichtabschluss herangezogen werden, da diese Sprachfamilien jeweils Obermengen von $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ und Teilmengen von DCFL sind.

Zu $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$:

Im Beweis von Satz 4.2.7 wurde gezeigt, dass die Sprache

$$\left\{a^{2^n}ca^mb^m \,\middle|\, n, m \in \mathbb{N}\right\} = \left\{a^{2^n} \,\middle|\, n \in \mathbb{N}\right\}\left\{ca^mb^m \,\middle|\, m \in \mathbb{N}\right\}$$

nicht von einem deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten erkannt werden kann. Da jedoch sowohl $\left\{a^{2^n} \,\middle|\, n \in \mathbb{N}\right\}$ als auch $\left\{ca^mb^m \,\middle|\, m \in \mathbb{N}\right\}$ von deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten erkannt werden können (vergleiche Beispiele 3.2.5 und 3.2.1), ist $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ nicht unter Konkatenation abgeschlossen. □

6.10 Satz (Abschluss unter markierter Konkatenation)

(a) Die Sprachfamilien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$,

$\mathscr{L}_{\det}(\mathsf{MV_R},\ \mathsf{ER},\ \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_L},\ \mathsf{ER},\ \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_L},\ \mathsf{ER},\ \mathsf{DL})$, $\mathscr{L}(\mathsf{MV},\ \mathsf{ER},\ \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV},\ \mathsf{ER},\ \mathsf{DL})$ sind abgeschlossen unter markierter Konkatenation.

(b) Die Familie $\mathscr{L}_{\det}(\mathsf{ER},\ \mathsf{DL})$ ist nicht abgeschlossen unter markierter Konkatenation.

Beweis

(a) Die Sprachfamilien $\mathscr{L}(\mathsf{ER_R},\ \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R},\ \mathsf{DL})$, $\mathscr{L}(\mathsf{MV},\ \mathsf{ER})$, $\mathscr{L}(\mathsf{MV_L},\ \mathsf{ER},\ \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV},\ \mathsf{ER},\ \mathsf{DL})$ sind nach Satz 6.9 abgeschlossen unter Konkatenation. Der Abschluss unter markierter Konkatenation folgt jeweils analog zu dem dort angegebenen Beweis. Im Falle von $\mathscr{L}(\mathsf{ER_R},\ \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV_R},\ \mathsf{DL})$ löscht der entsprechende Automat lediglich zusätzlich das Markierungssymbol an der Konkatenationsstelle. Bei $\mathscr{L}(\mathsf{MV_L},\ \mathsf{ER},\ \mathsf{DL})$ kann das Markierungssymbol mit $\mathsf{MV_L}$ verlassen und – nach dem Akzeptieren – das erste Wort inklusive des Markierungssymbols gelöscht werden. Im Falle von $\mathscr{L}(\mathsf{MV},\ \mathsf{ER})$ und $\mathscr{L}(\mathsf{MV},\ \mathsf{ER},\ \mathsf{DL})$ können die entsprechenden Automaten sowohl $\mathsf{MV_L}$- als auch $\mathsf{MV_R}$-Bewegungen ausführen und daher das Markierungssymbol unverändert stehen lassen.

Analog sind die Familien $\mathscr{L}(\mathsf{MV},\mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV},\mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV},\mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV},\mathsf{ER},\mathsf{DL})$ aufgrund des Vorhandenseins der $\mathsf{MV_L}$- und der $\mathsf{MV_R}$-Operation unter markierter Konkatenation abgeschlossen.

Es seien im Folgenden stets die Sprachen L_1 und L_2 aus der betreffenden Sprachfamilie $\mathscr{L}$ sowie die Automaten $\mathcal{A}_1$ und $\mathcal{A}_2$ mit $L(\mathcal{A}_1) = L_1$ und $L(\mathcal{A}_2) = L_2$ gegeben.

Zu $\mathscr{L}_{\det}(\mathsf{ER_R},\ \mathsf{DL})$:

Nach dem Beweis von Lemma 3.4.4 können wir annehmen, dass $\mathcal{A}_1$ nach dem Erreichen des Bandbegrenzungssymbols die auf dem Band verbliebenen Symbole in einem Zug von rechts nach links löscht. Wir konstruieren daher den Automaten $\mathcal{B}$, der die markierte Konkatenation von L_1 und L_2 erkennt, wie folgt: Zunächst wird der Automat $\mathcal{A}_1$ simuliert; dabei dient das Markierungssymbol als rechtes Bandbegrenzungssymbol für $\mathcal{A}_1$, es wird beim Erreichen gelöscht. Anschließend wird $\mathcal{A}_2$ auf der verbliebenen Eingabe simuliert. Somit wird die Eingabe genau dann von $\mathcal{B}$ akzeptiert, wenn das erste Teilwort in L_1 und das zweite Teilwort in L_2 liegt.

Zu $\mathscr{L}_{\det}(\mathsf{MV_R},\ \mathsf{DL})$:

Nach dem Beweis von Lemma 3.4.5 können wir auch für einen deterministischen $(\mathsf{MV_R},\mathsf{DL})$-Automaten annehmen, dass er nach dem Erreichen

des Bandbegrenzungssymbols die auf dem Band verbliebenen Symbole in einem Zug von rechts nach links löscht. Daher erfolgt die Konstruktion von $\mathcal{B}$ analog zu der für $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$.

Zu $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$:

Der Automat $\mathcal{B}$ beginnt mit der direkten Simulation von $\mathcal{A}_1$. Sobald die Markierungsstelle erreicht wird, merkt sich $\mathcal{B}$ den Zustand von $\mathcal{A}_1$, der sich bis dahin bei der Verarbeitung des ersten Wortes w_1 ergab. Dann bewegt sich $\mathcal{B}$ mit $\mathsf{MV_R}$ über die Markierung hinweg und verarbeitet zunächst das zweite Wort komplett. Dabei dient das Markierungssymbol als linkes Bandbegrenzungssymbol; es kann mittels der $\mathsf{MV_R}$-Operation stets wieder unverändert nach rechts verlassen werden.

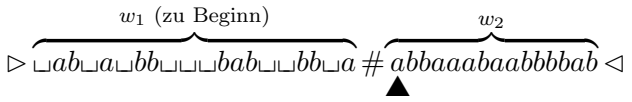


Abbildung 6.1: Beginn der Berechnung auf w_2.

Falls $\mathcal{A}_2$ die Eingabe akzeptiert, läuft $\mathcal{B}$ nach links, radiert das Markierungssymbol aus und nimmt die Simulation von $\mathcal{A}_1$ wieder auf. Dabei wird der zuvor gespeicherte Zustand von $\mathcal{A}_1$ wiederverwendet. Bei der weiteren Verarbeitung des ersten Wortes werden $\mathsf{MV_R}$- und $\mathsf{ER_R}$-Operationen vermieden, indem das Verhalten bei der Bewegung über die Leerzeichen rechts von der aktuellen Kopfposition parallel durch einen unären DFA $\mathcal{C}$ simuliert wird.

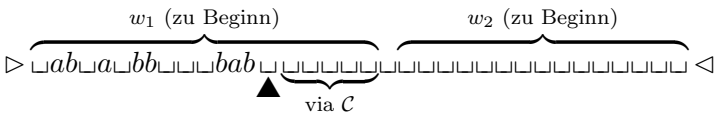


Abbildung 6.2: Fortsetzung der Berechnung auf w_1 mithilfe des DFA $\mathcal{C}$.

Analog zu Satz 4.1.5 kann der sich ergebende Folgezustand direkt am DFA abgelesen werden; in diesem Fall handelt es sich im Gegensatz zum dortigen Beweis allerdings um eine Folge von Leerzeichen, die am

rechten Rand des Bandes auftritt. Schließlich akzeptiert $\mathcal{B}$ die Eingabe, falls nach $\mathcal{A}_2$ nun auch $\mathcal{A}_1$ die Eingabe akzeptiert.

Zu $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$:

Analog zu $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ bzw. $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$ – hier mit Lemma 3.4.7.

Zu $\mathscr{L}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$:

Die Konstruktion von $\mathcal{B}$ erfolgt hier ähnlich zu der von $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$: Zunächst wird $\mathcal{A}_1$ bis zum Erreichen des Markierungssymbols direkt simuliert, der Zustand von $\mathcal{A}_1$ abgespeichert und mit der Simulation von $\mathcal{A}_2$ auf dem zweiten Wort fortgefahren. Falls $\mathcal{A}_2$ akzeptiert, kann dieser (gemäß Lemma 3.4.2) das zweite Teilwort komplett löschen. Somit kann $\mathcal{B}$ mithilfe des abgespeicherten Zustandes die direkte Simulation von $\mathcal{A}_1$ fortführen und schließlich akzeptieren, falls $\mathcal{A}_1$ und $\mathcal{A}_2$ akzeptieren.

Zu $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER}, \mathsf{DL})$:

Analog zum nichtdeterministischen Fall.

Zu $\mathscr{L}_{\det}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$:

Zunächst simuliert $\mathcal{B}$ den Automaten $\mathcal{A}_1$, wobei das Markierungssymbol als rechtes Bandbegrenzungssymbol für $\mathcal{A}_1$ fungiert. Da hier eine $\mathsf{MV_L}$-Operation zur Verfügung steht, kann das Markierungssymbol während der Simulation immer wieder unverändert verlassen werden. Falls $\mathcal{A}_1$ akzeptiert, können wir nach Lemma 3.4.2 annehmen, dass das erste Teilwort komplett gelöscht ist. $\mathcal{B}$ kann somit das Markierungssymbol löschen, mit der Simulation von $\mathcal{A}_2$ fortfahren und schließlich genau dann akzeptieren, wenn $\mathcal{A}_1$ und $\mathcal{A}_2$ akzeptieren.

(b) Zu $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$:

Im Beweis von Satz 4.2.7 wurde gezeigt, dass die Sprache

$$\{a^{2^n} c a^m b^m \mid n, m \in \mathbb{N}\} = \{a^{2^n} \mid n \in \mathbb{N}\}\{c\}\{a^m b^m \mid m \in \mathbb{N}\}$$

nicht von einem deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten erkannt werden kann. Da jedoch sowohl $\{a^{2^n} \mid n \in \mathbb{N}\}$ als auch $\{a^m b^m \mid m \in \mathbb{N}\}$ von deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten erkannt werden können (vergleiche Beispiele 3.2.5 und 3.2.1), ist $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ nicht unter markierter Konkatenation abgeschlossen. □

Lediglich für die Sprachfamilie $\mathscr{L}(\mathsf{ER}, \mathsf{DL})$ wurde der Abschluss unter markierter Konkatenation hier nicht gezeigt. Bei $(\mathsf{ER}, \mathsf{DL})$-Automaten ergibt sich die Schwierigkeit bei einer Simulation des ersten Automaten $\mathcal{A}_1$ durch die möglichen „Längenüberprüfungen“ auf der verbliebenen Zeichenkette aus

Leerzeichen, nachdem die Konkatenationsstelle erreicht wurde. Das Markierungssymbol kann ohne eine $\mathsf{MV_L}$-Operation nicht unverändert verlassen werden – der Automat $\mathcal{A}_1$ kann jedoch beispielsweise noch $\log_2 |w_1|$ weitere Durchläufe über die verbliebenen Leerzeichen machen, um zu testen, ob die Länge von w_1 eine Zweierpotenz ist. Dadurch ergeben sich für den simulierenden Automaten $\mathcal{B}$ im Falle des Ausradierens oder Löschens von weiteren Eingabezeichen neben dem ursprünglichen Markierungssymbol unbeschränkt viele zu merkende Zeichen, was die Möglichkeiten der endlichen Zustandskontrolle übersteigt.

Für den deterministischen Fall konnte daher in Satz 4.2.7 nachgewiesen werden, dass die markierte Konkatenation bestimmter nicht-regulärer Sprachen nicht von einem deterministischen $(\mathsf{ER}, \mathsf{DL})$-Automaten erkannt werden kann. Im nichtdeterministischen Fall kann ein entsprechender Automat jedoch die nötige Anzahl an Hin- und Herbewegungen auf dem ersten Teilwort raten und in entsprechendem Abstand zum Markierungssymbol das erste Mal umkehren. Auf diese Weise kann das Löschen bzw. Ausradieren der ersten Symbole des zweiten Teilwortes vermieden und – in manchen Fällen – die getrennte Simulation zweier nichtdeterministischer $(\mathsf{ER}, \mathsf{DL})$-Automaten durchgeführt werden. Ob auf diese Weise jedoch stets die markierte Konkatenation zweier gegebener Sprachen aus $\mathscr{L}(\mathsf{ER}, \mathsf{DL})$ akzeptiert werden kann, bleibt offen.

6.11 Satz (Abschluss unter Iteration)

(a) Die Familien $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ sind abgeschlossen unter Iteration.

(b) Die Familien $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ sind nicht abgeschlossen unter Iteration.

Beweis

(a) Zu $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV_L}, \mathsf{ER}, \mathsf{DL})$:

Analog zum Abschluss unter Konkatenation (Satz 6.9) kann sukzessive jeweils ein Teilwort geraten und verarbeitet werden, bis das Eingabewort komplett verbraucht ist.

Zu $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$:

Analog zum Abschluss unter Konkatenation werden sukzessive Teilwörter $w_1, \ldots, w_n$ geraten und das Enthaltensein eines jeden Teilwortes w_i in L überprüft. Die Überprüfung der Teilwörter w_1 und w_n erfolgt dabei genauso wie für die beiden Teilwörter in Satz 6.9. Für die Überprüfung der Teilwörter w_i, $i = 2, \ldots, n-1$, dient dabei sowohl das am weitesten links liegende, nicht ausradierte Eingabesymbol als Markierung für das

erste Zeichen von w_i als auch das am weitesten rechts liegende Leerzeichen als Markierung für das letzte Zeichen von w_i:

$$\triangleright \underbrace{\sqcup\sqcup\sqcup\sqcup\sqcup}_{w_1}\underbrace{\sqcup\sqcup\sqcup}_{w_2}\cdots\underbrace{\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup}_{w_{i-1}}\underbrace{bbbababa\sqcup}_{w_i} abaababababbbababb\triangleleft$$

Auf diese Weise gilt $w \in L(\mathcal{B})$ genau dann, wenn $w \in L^*$.

Zu $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$:

Analog zu $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$ kann die Vorgehensweise aus Satz 6.9 iteriert werden.

(b) Zu $\mathscr{L}_{\det}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$:

Im Beweis von Satz 6.9 wurde bereits angegeben, dass die Sprache

$$L = \{a0^n1^n2^m \mid n, m \in \mathbb{N}_0\} \cup \{0^n1^m2^m \mid n, m \in \mathbb{N}_0\}$$

in $\mathscr{L}_{\det}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$ enthalten ist. Die Iteration L^* ist jedoch nicht deterministisch kontextfrei (siehe [23]) und liegt damit nicht in der Familie $\mathscr{L}_{\det}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$, da diese in DCFL enthalten ist.

Der Beweis für $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER}_\mathsf{R}, \mathsf{DL})$ erfolgt analog, da diese Sprachfamilien Obermengen von $\mathscr{L}_{\det}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$ sind (und damit L enthalten) und ebenfalls Teilmengen von DCFL sind (und daher L^* nicht enthalten). □

6.12 Satz (Abschluss unter markierter Iteration)
Die Familien $\mathscr{L}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ sind abgeschlossen unter markierter Iteration.

Beweis Die Sprachfamilien $\mathscr{L}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}(\mathsf{MV}, \mathsf{ER})$, $\mathscr{L}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$ und $\mathscr{L}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$ sind nach Satz 6.11 unter Iteration abgeschlossen und daher auf analoge Weise auch unter markierter Iteration abgeschlossen.

Im Falle der Familien $\mathscr{L}_{\det}(\mathsf{ER}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER}_\mathsf{R}, \mathsf{DL})$ und $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{L}, \mathsf{ER}, \mathsf{DL})$ kann die Vorgehensweise für markierte Konkatenation aus Satz 6.10 iteriert werden.

Zu $\mathscr{L}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{ER}, \mathsf{DL})$:

Die entsprechenden Automaten können in diesen Fällen sowohl MV_L- als auch MV_R-Bewegungen ausführen und daher die Markierungssymbole unverändert stehen lassen. Demzufolge können die einzelnen Teilwörter hintereinander abgearbeitet und die gesamte Eingabe kann akzeptiert werden, wenn alle Teilwörter akzeptiert werden.

Zu $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER})$:

Es sei $\mathcal{A}$ mit $L(\mathcal{A}) = L$ gegeben. Da $\mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER}) = \mathscr{L}_{\det}(\mathsf{MV}_\mathsf{R}, \mathsf{ER}_\mathsf{L})$ gilt, können wir annehmen, dass $\mathcal{A}$ ein $(\mathsf{MV}_\mathsf{R}, \mathsf{ER}_\mathsf{L})$-Automat ist. Wir konstruieren einen deterministischen $(\mathsf{MV}_\mathsf{R}, \mathsf{ER})$-Automaten $\mathcal{B}$, der die markierte Iteration von L erkennt, indem er sukzessive $\mathcal{A}$ auf jedem der markierten Teilwörter $w_1, \ldots, w_n$ simuliert und sie anschließend komplett ausradiert.

Zunächst simuliert $\mathcal{B}$ den Automaten $\mathcal{A}$ bis zum Erreichen des ersten Markierungssymbols direkt. Da $\mathcal{B}$ das Markierungssymbol nur mit ER_L nach links verlassen kann, wird die weitere Abarbeitung wie folgt verändert: Um das mehrfache Erreichen des Bandbegrenzungssymbols von $\mathcal{A}$ zu vermeiden, simuliert $\mathcal{B}$ das Verhalten bei der Bewegung über die Leerzeichen rechts von der aktuellen Kopfposition parallel durch einen unären DFA (vergleiche Satz 6.10). Eine Schleife wird dabei nicht verhindert – dies ist allerdings auch nicht nötig, da der Automat somit die Eingabe nicht akzeptiert. Man beachte, dass Lemma 3.4.6 nicht ohne Einschränkung anwendbar ist, da in dessen Beweis das mehrfache Erreichen des rechten Bandbegrenzungssymbols nicht ausgeschlossen werden kann.

Falls $\mathcal{A}$ schließlich die Eingabe akzeptiert, radiert $\mathcal{B}$, falls noch nicht geschehen, das erste Wort komplett mit ER_L-Operationen aus und bewegt sich mit MV_R-Schritten zum nächsten Eingabesymbol, das heißt dem ersten Symbol von w_2.

$\mathcal{B}$ beginnt nun mit der Simulation von $\mathcal{A}$ auf w_2. Dabei können ER_L-Operationen auf bereits ausradierten Feldern sowie MV_R-Operationen stets direkt ausgeführt werden. Eine ER_L-Operation auf einem Eingabesymbol hingegen erfordert die Überprüfung, ob dieses Symbol das am weitesten links liegende Eingabesymbol auf dem Band ist, um das Simulieren des linken Bandbegrenzungssymbols für $\mathcal{A}$ korrekt zu ermöglichen. Für diese Überprüfung wird die ER_L-Operation wie folgt simuliert: $\mathcal{B}$ bewegt sich mit ER_L-Operationen bis zu $\triangleright$ oder zum nächsten Eingabesymbol nach links und anschließend mit MV_R-Operationen zurück nach rechts. Da die ursprüngliche Position dabei in der Regel nicht wiedergefunden werden kann, muss das Verhalten von $\mathcal{A}$ auf andere Weise korrekt simuliert werden, wie es nachfolgend beschrieben wird.

Sobald das erste Symbol von w_2 ausradiert wurde, wird erneut ein unärer DFA verwendet, um das Verhalten von $\mathcal{A}$ auf den Leerzeichen am Anfang des

zweiten Wortes zu simulieren. Wir bezeichnen diesen DFA mit $\mathcal{B}_\triangleright$ und die Anzahl der Leerzeichen am Anfang des Wortes mit $n_\triangleright$. Da $\mathcal{B}_\triangleright$ nur endlich viele Zustände hat, kann er die beliebig groß werdende Länge $n_\triangleright$ natürlich nicht abspeichern, aber das Verhalten von $\mathcal{A}$ auf dem Bandanfang $\triangleright\sqcup^{n_\triangleright}$ beschreiben. Im Folgenden werden wir noch zwei weitere Längen n_l und n_r betrachten, für die das Verhalten von $\mathcal{A}$ durch unäre DFAs $\mathcal{B}_l$ und $\mathcal{B}_r$ beschrieben wird (siehe Abbildung 6.3).

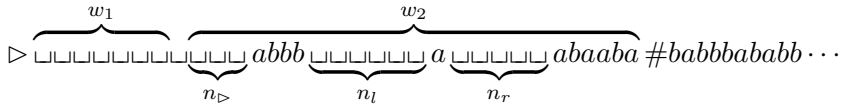


Abbildung 6.3: Skizze der Längen $n_\triangleright$, n_l und n_r bei Abarbeitung des Teilwortes w_2.

Um das Verhalten von $\mathcal{A}$ beim Verschmelzen von zwei Blöcken von Leerzeichen (durch das Ausradieren eines dazwischenliegenden Leerzeichens) zu bewerkstelligen, speichert $\mathcal{B}$ stets mithilfe des unären DFA $\mathcal{B}_r$ die Länge $n_r \geq 0$ des Blocks von Leerzeichen, auf dem er sich momentan befindet bzw. der rechts vom Kopf liegt (falls $\mathcal{B}$ mit einer $\mathsf{ER_L}$-Operation ein Eingabesymbol erreicht). Führt $\mathcal{A}$ eine $\mathsf{ER_L}$-Operation auf einem Eingabesymbol aus, simuliert $\mathcal{B}$ diese wie folgt: Zunächst führt $\mathcal{B}$ ebenfalls eine $\mathsf{ER_L}$-Operation aus. Falls dabei ein Eingabesymbol erreicht wird (das heißt keine Blöcke verschmolzen werden), sind keine weiteren Maßnahmen nötig. Falls jedoch ein Leerzeichen erreicht wird, führt $\mathcal{B}$ weitere $\mathsf{ER_L}$-Operationen aus und zählt die Anzahl n_l der Leerzeichen – mithilfe des unären DFA $\mathcal{B}_l$ und daher mit endlich vielen Zuständen – bis (a) zum nächsten Eingabesymbol oder (b) dem linken Bandbegrenzungssymbol.

(a) Wird ein Eingabesymbol erreicht, simuliert $\mathcal{B}$ das Verhalten von $\mathcal{A}$ auf dem zusammengefügten Block aus Leerzeichen mithilfe von $\mathcal{B}_l$ und $\mathcal{B}_r$. Anschließend führt er die direkte Simulation von $\mathcal{A}$ an der aktuellen Position (falls $\mathcal{A}$ den Block auf der linken Seite verlässt) oder dem nächsten, weiter rechts gelegenem Eingabesymbol (falls $\mathcal{A}$ den Block auf der rechten Seite verlässt) fort.

Man beachte, dass im Falle des Verlassens auf der rechten Seite kein weiterer Block von Leerzeichen rechts vom Kopf liegen kann. Für einen solchen Block wäre die Länge n_r nicht bekannt und könnte ohne das Ausradieren eines weiteren Eingabezeichens auch nicht bestimmt werden. Da nur $\mathsf{MV_R}$- und $\mathsf{ER_L}$-Operationen zur Verfügung stehen, kann

allerdings beim Erreichen eines Eingabesymbols mit $\mathsf{MV_R}$ kein Leerzeichen rechts vom Kopf mehr existieren.

(b) Wird das linke Bandbegrenzungssymbol erreicht und somit ein Block aus Leerzeichen mit dem Block am Anfang von w_2 verschmolzen, kann $\mathcal{B}$ bis zum ersten Eingabesymbol nach rechts laufen und das Verhalten von $\mathcal{A}$ mithilfe der DFAs $\mathcal{B}_\triangleright$ und $\mathcal{B}_r$ weitersimulieren.

Sobald das nächste Markierungssymbol erreicht wird, stellt der unäre DFA $\mathcal{B}_r$ sicher, dass das Symbol während der Verarbeitung von w_2 nur ein einziges Mal betreten werden muss (wie schon für w_1 geschehen).

Falls $\mathcal{A}$ schließlich die Eingabe w_2 akzeptiert, kann die gesamte Prozedur für die restlichen Teilwörter w_3 bis w_n iteriert werden. □

Nur für die Familien der von deterministischen bzw. nichtdeterministischen (ER, DL)- und ($\mathsf{MV_R}$, ER, DL)-Automaten akzeptierten Sprachen wurde der Abschluss unter markierter Iteration hier nicht gezeigt. Für das erstgenannte Automatenmodell gelten die gleichen Anmerkungen wie bei dem Abschluss unter markierter Konkatenation. Im Falle der ($\mathsf{MV_R}$, ER, DL)-Automaten lässt sich zudem die für die markierte Konkatenation verwendete Methode nicht auf markierte Iteration übertragen.

6.13 Satz (Abschluss unter Homomorphismus)
(a) Die Familie $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ ist abgeschlossen unter Homomorphismus.

(b) Die Familien $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$ sind nicht abgeschlossen unter λ-freiem Homomorphismus.

Die Familie $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$ ist nicht abgeschlossen unter Homomorphismus.

Beweis

(a) Da $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ mit $\mathscr{L}$(1-CA) übereinstimmt, ist $\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$ als *volle AFL* unter Homomorphismus abgeschlossen (siehe [17]).

(b) Zu $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$:

Der Homomorphismus $h : \{a, a', b, c\} \to \{a, b, c\}$, mit $h(a) = h(a') = a$, $h(b) = b$, $h(c) = c$, bildet $\{a^n b^n c^m \mid n, m \in \mathbb{N}\} \cup \{a'^n b^m c^m \mid n, m \in \mathbb{N}\}$ auf $\{a^n b^n c^m \mid n, m \in \mathbb{N}\} \cup \{a^n b^m c^m \mid n, m \in \mathbb{N}\}$ ab.

Während $\{a^n b^n c^m \mid n, m \in \mathbb{N}\} \cup \{a'^n b^m c^m \mid n, m \in \mathbb{N}\}$ von einem deterministischen ($\mathsf{ER_R}$, DL)-Automaten erkannt werden kann (vergleiche Satz 6.8), ist $\{a^n b^n c^m \mid n, m \in \mathbb{N}\} \cup \{a^n b^m c^m \mid n, m \in \mathbb{N}\}$ nicht deterministisch kontextfrei und liegt damit nicht in $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$.

Für den Nichtabschluss der Familien $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$, $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$ und $\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$ können wir ebenfalls die obige Zeugensprache betrachten, da die genannten Sprachfamilien jeweils Obermengen von $\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$ und Teilmengen von DCFL sind.

Zu $\mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$:

Wir betrachten die folgende Sprache:

$$L_1 := \{a\} \cup \{aba^2ba^4ba^6b \cdots ba^{2i} \mid i \geq 1\} \in \mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL}).$$

L_1 kann von einem $(\mathsf{MV}, \mathsf{DL})$-Automaten $\mathcal{A}_1$ wie folgt erkannt werden: $\mathcal{A}_1$ bewegt sich in mehreren Durchgängen vom linken zum rechten Bandbegrenzungssymbol und zurück. In einem Durchgang löscht $\mathcal{A}_1$ dabei das Teilwort aba am Bandanfang und jeweils zwei a in jedem weiteren aus a bestehenden (und durch b begrenzten) Teilwort. $\mathcal{A}_1$ akzeptiert die Eingabe, wenn die Eingabe auf diese Weise schließlich bis zu dem Wort a verkürzt werden kann.

Der Homomorphismus $h : \{a, b\} \to \{a\}$, mit $h(a) = h(b) = a$, bildet L_1 auf $\{a^{n^2} \mid n \geq 1\} \notin \mathscr{L}_{\det}(\mathsf{MV}, \mathsf{DL})$ ab (siehe Satz 5.9).

Zu $\mathscr{L}_{\det}(\mathsf{ER}, \mathsf{DL})$:

Ein deterministischer $(\mathsf{ER}, \mathsf{DL})$-Automat $\mathcal{A}_2$ kann die Sprache

$$L_2 = \left\{a^{2^n} d^n c a^m b^m \;\middle|\; n, m \in \mathbb{N}\right\}$$

folgendermaßen erkennen: $\mathcal{A}_2$ bewegt sich mit abwechselnden $\mathsf{DL_R}$- und $\mathsf{ER_R}$-Schritten bis zum ersten d und halbiert damit die Länge des ersten Teilwortes der Form a^*. Anschließend löscht $\mathcal{A}_2$ das erreichte d in einer $\mathsf{DL_L}$-Operation und bewegt sich mit $\mathsf{ER_L}$-Schritten zurück zum linken Bandbegrenzungssymbol. $\mathcal{A}_2$ iteriert nun den Halbierungsvorgang und erreicht schließlich – bei einer korrekten Eingabe – nach dem Löschen des letzten verbliebenen Leerzeichens am Bandanfang das Zeichen c. Auf diese Weise kann $\mathcal{A}_2$ sowohl überprüfen, dass die Länge des ersten Teilwortes eine Zweierpotenz 2^n (für ein $n \in \mathbb{N}$) ist, als auch testen, dass das zweite Teilwort die Form d^n hat.

Schließlich kann $\mathcal{A}_2$ das erreichte c löschen, das restliche Teilwort auf die Form $a^m b^m$ überprüfen (vergleiche Beispiel 3.2.1) und die Eingabe genau dann akzeptieren, wenn sie in L_2 liegt.

Wenden wir auf L_2 den Homomorphismus $h : \{a, b, c, d\} \to \{a, b, c\}$ mit $h(a) = a$, $h(b) = b$, $h(c) = c$ und $h(d) = \lambda$ an, erhalten wir hingegen die Sprache

$$\left\{a^{2^n} c a^m b^m \;\middle|\; n, m \in \mathbb{N}\right\},$$

von der im Beweis von Satz 4.2.7 gezeigt wurde, dass sie nicht von einem deterministischen (ER, DL)-Automaten erkannt werden kann. Daher ist $\mathscr{L}_{\text{det}}(\mathsf{ER}, \mathsf{DL})$ nicht unter Homomorphismus abgeschlossen. □

Der Nichtabschluss von $\mathscr{L}_{\text{det}}(\mathsf{ER}, \mathsf{DL})$ wurde hier nur für allgemeinen Homomorphismus gezeigt, nicht jedoch für λ-freien Homomorphismus. Vermutlich ist $\mathscr{L}_{\text{det}}(\mathsf{ER}, \mathsf{DL})$ auch unter λ-freiem Homomorphismus nicht abgeschlossen – da jedoch keine geeigneten Zeugensprachen außerhalb von $\mathscr{L}_{\text{det}}(\mathsf{ER}, \mathsf{DL})$ bekannt sind, konnte der Nachweis bisher nicht erbracht werden.

Eine Übersicht der Abschlusseigenschaften der Sprachfamilien bis einschließlich der Familie der kontextfreien Sprachen wird in Tabelle 6.1 dargestellt. Dabei symbolisiert ein Pluszeichen, dass eine Familie unter der entsprechenden Operation abgeschlossen ist; ein Minuszeichen symbolisiert entsprechend den Nichtabschluss unter einer Operation. Im Falle eines Fragezeichens ist der entsprechende Abschluss oder Nichtabschluss bisher nicht geklärt. Es fällt dabei auf, dass die Familien der von den deterministischen bzw. den nichtdeterministischen Modellen akzeptierten Sprachen die gleichen Abschlusseigenschaften haben wie die Familien, die durch deterministische bzw. nichtdeterministische Kellerautomaten charakterisiert werden.

	Vereinigung	markierte Vereinigung	Vereinigung mit regulären Sprachen	Schnitt	Schnitt mit regulären Sprachen	Komplement	Spiegelung	Konkatenation	markierte Konkatenation	Iteration	markierte Iteration	(λ-freier) Homomorphismus
REG	+	+	+	+	+	+	+	+	+	+	+	+
$\mathscr{L}(\mathsf{ER_R}, \mathsf{DL})$	+	+	+	−	+	−	+	+	+	+	+	+
$\mathscr{L}_{\det}(\mathsf{ER_R}, \mathsf{DL})$	−	+	+	−	+	+	−	−	+	−	+	−
$\mathscr{L}(\mathsf{MV_R}, \mathsf{DL})$	+	+	+	−	+	−	?	+	+	+	+	?
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{DL})$	−	+	+	−	+	+	−	−	+	−	+	−
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER})$	−	+	+	−	+	+	−	−	+	−	+	−
$\mathscr{L}_{\det}(\mathsf{MV_R}, \mathsf{ER_R}, \mathsf{DL})$	−	+	+	−	+	+	−	−	+	−	+	−
$\mathrm{CFL} = \mathscr{L}(\mathsf{MV_R}, \mathsf{ER})$	+	+	+	−	+	−	+	+	+	+	+	+
DCFL	−	+	+	−	+	+	−	−	+	−	+	−

Tabelle 6.1: Übersicht der Abschlusseigenschaften der Sprachfamilien bis einschließlich CFL.

Literaturverzeichnis

[1] Aho, A. V. Indexed grammars – an extension of context-free grammars. *Journal of the ACM 15* (1968), S. 647–671.

[2] Balke, L. und Böhling, K. H. *Einführung in die Automatentheorie und Theorie formaler Sprachen*, Band 90 der *Reihe Informatik*. BI Wissenschaftsverlag, 1993.

[3] Chomsky, N. Three models for the description of language. *IRE Transactions on Information Theory 2* (1956), S. 113–124.

[4] Chomsky, N. On certain formal properties of grammars. *Information and Control 2* (1959), S. 137–167.

[5] Chrobak, M. Variations on the technique of Ďuriš and Galil. *Journal of Computer and System Sciences 30* (1985), S. 77–85.

[6] Chytil, M. P., Plátek, M. und Vogel, J. A note on the Chomsky hierarchy. *Bulletin of the EATCS 27* (1985), S. 23–30.

[7] Dassow, J. und Păun, G. *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, 1989.

[8] Ďuriš, P. und Galil, Z. Fooling a two way automaton or one pushdown store is better than one counter for two way machines. *Theoretical Computer Science 21* (1982), S. 39–53.

[9] Fischer, P. C. Turing machines with restricted memory access. *Information and Control 9* (1966), S. 364–379.

[10] Giammarresi, D. und Restivo, A. Two-dimensional languages. In *Handbook of Formal Languages*, Band 3, *Beyond Words*. Springer-Verlag, 1997, S. 215–267.

[11] Glöckler, J. Forgetting automata and unary languages. In *Conference on Implementation and Application of Automata 2006* (CIAA 2006), Band 4094 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 186–197.

[12] Glöckler, J. Closure properties of the families of languages accepted by forgetting automata. In *Mathematical and Engineering Methods in Computer Science 2007* (MEMICS 2007), Masaryk University, Brno, Institute of Computer Science, S. 51–58.

[13] Glöckler, J. Forgetting automata and unary languages. *International Journal of Foundations of Computer Science 18* (2007), S. 813–827.

[14] Glöckler, J. A taxonomy of deterministic forgetting automata. In *Developments in Language Theory 2008* (DLT 2008), Band 5257 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 371–382.

[15] Gonczarowski, J. und Shamir, E. Pattern selector grammars and several parsing algorithms in the context-free style. *Journal of Computer and System Sciences 30* (1985), S. 249–273.

[16] Greibach, S. und Hopcroft, J. Scattered context grammars. *Journal of Computer and System Sciences 3* (1969), S. 233–247.

[17] Greibach, S. A. An infinite hierarchy of context-free languages. *Journal of the ACM 16* (1969), S. 91–106.

[18] Greibach, S. A. Erasable context-free languages. *Information and Control 29* (1975), S. 301–326.

[19] Harrison, M. A. *Introduction to Formal Language Theory.* Addison-Wesley Publishing Company, 1978.

[20] Higuera, C. D. L. und Oncina, J. Inferring deterministic linear languages. In *Computational Learning Theory 2002* (COLT 2002), Band 2375 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 185–200.

[21] Holzer, M. und Lange, K.-J. On the complexities of linear LL(1) and LR(1) grammars. In *Fundamentals of Computation Theory 1993* (FCT 1993), Band 710 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 299–308.

[22] Hopcroft, J. E. und Ullman, J. D. *Formal Languages and their Relation to Automata.* Addison-Wesley Publishing Company, 1969.

[23] Hopcroft, J. E. und Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[24] Ibarra, O. H. Simple matrix languages. *Information and Control 17* (1970), S. 359–394.

[25] Ibarra, O. H. und Trân, N. Q. A note on simple programs with two variables. *Theoretical Computer Science 112* (1993), S. 391–397.

[26] Jančar, P. Nondeterministic forgetting automata are less powerful than deterministic linear bounded automata. *Acta Mathematica et Informatica Universitatis Ostraviensis 1* (1993), S. 67–74.

[27] Jančar, P., Mráz, F. und Plátek, M. Characterization of context-free languages by erasing automata. In *Mathematical Foundations of Computer Science 1992* (MFCS 1992), Band 629 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 305–314.

[28] Jančar, P., Mráz, F. und Plátek, M. Forgetting automata and the Chomsky hierarchy. In *Conference on Current Trends in Theory and Practice of Informatics 1992* (SOFSEM 1992), Masaryk University, Brno, Institute of Computer Science, S. 41–44.

[29] Jančar, P., Mráz, F. und Plátek, M. A taxonomy of forgetting automata. In *Mathematical Foundations of Computer Science 1993* (MFCS 1993), Band 711 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 527–536.

[30] Jančar, P., Mráz, F. und Plátek, M. A taxonomy of forgetting automata. Report 101, Department of Computer Science, Charles University, Prague, 1993.

[31] Jančar, P., Mráz, F. und Plátek, M. Forgetting automata and context-free languages. *Acta Informatica 33* (1996), S. 409–420.

[32] Jančar, P., Mráz, F., Plátek, M. und Vogel, J. Restarting automata. In *Fundamentals of Computation Theory 1995* (FCT 1995), Band 965 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 283–292.

[33] Jančar, P., Mráz, F., Plátek, M. und Vogel, J. On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics 4* (1999), 287–312.

[34] Jiřička, P. und Král, J. Deterministic forgetting planar automata are more powerful than non-deterministic finite-state planar automata. In *Developments in Language Theory 1999* (DLT 1999), World Scientific, S. 71–80.

[35] Jurdziński, T., Otto, F., Mráz, F. und Plátek, M. On left-monotone deterministic restarting automata. In *Developments in Language Theory 2004* (DLT 2004), Band 3340 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 249–260.

[36] Jurdziński, T., Otto, F., Mráz, F. und Plátek, M. Deterministic two-way restarting automata and Marcus contextual grammars. *Fundamenta Informaticae 64*, 1-4 (2005), S. 217–228.

[37] Kleijn, H. C. M. und Rozenberg, G. Context-free like restrictions on selective rewriting. *Theoretical Computer Science 16* (1981), S. 237–269.

[38] Marcus, S. Contextual grammars. *Revue Roumaine de Mathématiques Pures et Appliquées 14* (1969), S. 1525–1534.

[39] Maurer, H. A. und Kuich, W. Tuple languages. In *ACM International Computing Symposium* (ICS 1970), German Chapter of the ACM, S. 882–891.

[40] Minsky, M. Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. *Annals of Mathematics 74* (1961), S. 437–455.

[41] Monien, B. Two-way multihead automata over a one-letter alphabet. *RAIRO – Theoretical Informatics and Applications 14* (1980), 67–82.

[42] Mráz, F. und Plátek, M. Erasing automata recognize more than context-free languages. In *Conference on Current Trends in Theory and Practice of Informatics 1991* (SOFSEM 1991), Masaryk University, Brno, Institute of Computer Science, S. 251–255.

[43] Mráz, F. und Plátek, M. A remark about forgetting automata. In *Conference on Current Trends in Theory and Practice of Informatics 1993* (SOFSEM 1993), Masaryk University, Brno, Institute of Computer Science, S. 63–66.

[44] Mráz, F. und Plátek, M. Erasing automata recognize more than context-free languages. *Acta Mathematica et Informatica Universitatis Ostraviensis 3* (1995), S. 77–85.

[45] Mráz, F., Plátek, M. und Procházka, M. On special forms of restarting automata. *Grammars 2* (1999), S. 223–233.

[46] Mráz, F., Plátek, M. und Vogel, J. Restarting automata with rewriting. In *Conference on Current Trends in Theory and Practice of Informatics 1996* (SOFSEM 1996), Band 1175 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 401–408.

[47] Niemann, G. und Otto, F. Restarting automata, Church-Rosser languages, and representations of r.e. languages. In *Developments in Language Theory 1999* (DLT 1999), World Scientific, S. 103–114.

[48] PĂUN, G. *Marcus Contextual Grammars*, Band 67 der *Studies in Linguistics and Philosophy*. Kluwer Academic Publishers, 1997.

[49] PLÁTEK, M. UND VOGEL, J. Deterministic list automata and erasing graphs. *The Prague bulletin of mathematical linguistics 45* (1986), 27–50.

[50] PRŮŠA, D. *Two-dimensional Languages*. PhD thesis, Charles University, Prague, 2004.

[51] RABIN, M. O. UND SCOTT, D. Finite automata and their decision problems. *IBM Journal of Research and Development 3* (1959), S. 114–125.

[52] RYTTER, W. On the recognition of context-free languages. In *Computation Theory 1985* (SCT 1985), Band 208 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 318–325.

[53] SHEPHERDSON, J. C. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development 3* (1959), S. 198–200.

[54] SIROMONEY, R. UND KRITHIVASAN, K. Parallel context-free languages. *Information and Control 24* (1974), S. 155–162.

[55] VON BRAUNMÜHL, B. UND VERBEEK, R. Finite change automata. In *Theoretical Computer Science, Fourth GI-Conference* (1979), Band 67 der *Lecture Notes in Computer Science*, Springer-Verlag, S. 91–100.

[56] WAGNER, K. UND WECHSUNG, G. *Computational Complexity*. VEB Deutscher Verlag der Wissenschaften, 1986.