Beschreibungskomplexität von Restart-Automaten

Jens Reimann

2.3.2007

Danksagungen

An dieser Stelle möchte ich mich herzlich bei allen bedanken, die durch ihre Unterstützung, Hilfe und Geduld zur vorliegenden Arbeit und ihrer Fertigstellung beigetragen haben.

Besonderer Dank gilt meinem Betreuer Herrn Prof. Dr. Martin Kutrib, der mir in vielen Situationen durch wertvolle Diskussionen, kritische Anmerkungen und Ratschläge geholfen hat.

Ebenso danke ich meinem Kollegen Herrn Dipl.-Math. Jens Glöckler für die anregenden Diskussionen, so manchen kritischen Hinweis und nicht zuletzt für die Mühe des Korrekturlesens.

Des Weiteren möchte ich mich bei Herrn Prof. Dr. Friedrich Otto, Herrn Dr. Frantisek Mráz und Herrn Dr. Andreas Malcher für alle Anregungen, Auskünfte und interessierten Gespräche bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Umfeld der vielfältigen Mechanismen zur Beschreibung formaler Sprachen ist das Modell des Restart-Automaten (engl. *restarting automata*) ein relativ neues. Sein gedanklicher Vorgänger, das Modell der Vergessenden Automaten (*forgetting automata*), wurde zu Beginn der Neunziger Jahre von Jančar, Mráz, Plátek und Vogel ([13]) eingeführt. Motiviert wurden beide Systeme von der linguistischen Satzanalyse-Technik *analysis by reduction*, deren Ziel es ist, durch gezielte Reduktionen eine Vereinfachung der Satzstruktur zu bewirken und somit schrittweise auf eine Form hinzuführen, in der die syntaktische Korrektheit des verbleibenden Satzes unmittelbar erkannt werden kann.

Im Allgemeinen besteht ein Restart-Automat aus einer – über eine endliche Zustandsmenge gesteuerte – Schreib- und Leseeinheit (Schreib-/Lesefenster), die auf ein endliches, zu Beginn der Abarbeitung mit dem Eingabewort beschriebenes Arbeitsband zugreift. Das Schreib-/Lesefenster ist dabei von fester Größe. Ferner muss der Automat das Arbeitsband mit jeder Schreiboperation verkürzen, wobei jeweils nur Stellen aus dem entsprechend anliegenden Fensterinhalt entfernt werden dürfen. Den Abschluss eines der Grundidee entsprechenden Reduktionsschritts bildet die Ausführung des namensgebenden Restarts. Hierbei wird die Fensterposition wieder auf den Beginn des (reduzierten) Wortes zurückgesetzt und der Automat nimmt erneut seinen Startzustand an. Ist die Wortstruktur im Laufe der Verarbeitung hinreichend vereinfacht worden, so akzeptiert oder verwirft der Automat das Eingabewort in Übereinstimmung mit der Korrektheit des reduzierten Wortes. Wenn nicht anders erwähnt, arbeiten Restart-Automaten stets nichtdeterministisch – die von ihnen erkannte Sprache besteht aus Wörtern, zu denen eine Reduktionsfolge existiert, an deren Ende der Automat das Restwort akzeptiert.

Bei den – Mitte der Neunziger Jahre – zuerst betrachteten Restart-Automaten ([14]) waren zunächst nur Rechtsbewegungen des Fensters und das Löschen (durch Herausschneiden) beliebiger Stellen des Fensterinhalts definiert. Außerdem sollte nach jedem Löschschritt umgehend ein Neustart erfolgen. Mit einer zusätzlichen Einschränkung der Arbeitsweise (der sogenannten *Monotonie*) konnte die Charakterisierung von deterministisch kontextfreien Sprachen durch solche deterministischen, monotonen Restart-Automaten gezeigt werden.

Im Laufe der darauf folgenden Betrachtungen wurde dieses Grundmodell um einige Berechnungs-Ressourcen erweitert und es kamen auch neue Einschränkungen hinzu. So betrachtete man in [15] die Möglichkeit, statt des Löschens ein (noch immer verkürzendes) Umschreiben des Fensterinhalts zuzulassen. Dieser Vorgang wurde in den nachfolgenden Untersuchungen dahingehend

erweitert, dem Automaten auch das Schreiben von Nicht-Eingabezeichen zu gestatten, worüber (z. B. in [16]) die Charakterisierung von kontextfreien Sprachen (durch monotone Restart-Automaten, die Sonderzeichen schreiben können) nachgewiesen werden konnte.

Des Weiteren untersuchte man die Möglichkeit, den Restart-Automaten nach einem Umschreibe-Vorgang und vor dem entsprechenden Neustart weitere Leseoperationen ausführen zu lassen. Auch wurden die Auswirkungen einiger Veränderungen des Monotoniebegriffs studiert. In [30] wurde die Charakterisierung der Church-Rosser Sprachen durch deterministische Restart-Automaten, die Sonderzeichen schreiben können, sowie in [19] die Charakterisierung der wachsend kontextsensitiven Sprachen durch sogenannte *schwach monotone*, Sonderzeichen schreibende Restart-Automaten nachgewiesen.

Wie in [27] und auch im dritten Kapitel der vorliegenden Arbeit nachgewiesen, finden sich sogar Einschränkungen an Restart-Automaten, die eine Charakterisierung der regulären Sprachen ergeben.

Resultat all dieser Betrachtungen ist eine sehr enge Vernetzung der Restart-Automaten und des kontextsensitiven Teils der Chomsky Hierarchie. Darüber hinaus ergibt sich durch die verschiedenen Kombinationen von Arbeitsressourcen und Einschränkungen eine Fülle von Teilhierarchien.

Die unterschiedlichen Arbeitsweisen bedingen, dass in den Hierarchien der von unterschiedlichen Restart-Automaten erkannten Sprachklassen sehr oft entweder eine echte Inklusion vorliegt oder Unvergleichbarkeit gilt. Auch im Vergleich zu bereits bekannten Hierarchien, wie zum Beispiel der der linearen Sprachfamilien, herrscht zwischen beliebigen Paaren einzelner Sprachklassen vorwiegend Unvergleichbarkeit.[1] Auf Grund dieser und einiger weiterer Eigenschaften, wie z. B. der effizienten Entscheidbarkeit des Wortproblems, wird das Gesamtmodell der Restart-Automaten zu einem interessanten Beschreibungssystem formaler Sprachen.

In diesem Sinne liegt auch die Frage nach der Effizienz der gelieferten Beschreibungen in unmittelbarer Nähe. Eine solche Fragestellung wurde erstmalig 1971 von Meyer und Fischer in [26] untersucht, was einen Meilenstein, wenn nicht sogar den Grundstein der Geschichte der Beschreibungskomplexität darstellt:
Bereits mit der Einführung der nichtdeterministischen endlichen Automaten (NFAs) in [36] bewiesen Rabin und Scott im Jahre 1959, dass es zu jedem NFA mit n Zuständen einen äquivalenten, deterministischen endlichen Automaten (DFA) gibt und dass dieser nicht mehr als 2^n Zustände hat. Die Frage,

[1] Mit Ausnahme des Vergleichs zu Restart-Automaten-Sprachklassen zu denen Inklusionen vorliegen. Im obigen Beispiel (der Hierarchie der linearen Sprachfamilien) ergeben sich letztere aus den Charakterisierungen der regulären bzw. kontextfreien Sprachen.

ob es Sprachen gibt, für die der DFA zwingend von einer solchen Größe sein muss, konnten Meyer und Fischer für beliebige n positiv beantworten.

Ebenso fanden sie eine Familie von regulären Sprachen mit der Eigenschaft, dass die jeweils durch den Wert n gegebene Sprache von einem deterministischen Kellerautomaten (DPDA) mit $O(n)$ Zuständen und einem Zeichensatz von $O(n)$ Kellersymbolen erkannt werden kann, der minimale DFA aber stets 2^{2^n} Zustände besitzt. Auch hier war die Zahl der maximal benötigten Zustände schon vorher bekannt: sie entstammte einem Beweis zur Entscheidbarkeit der Regularität von DPDA-Sprachen, den Stearns bereits 1967 in [42] geführt hatte.

Ein ganz neues Phänomen trat jedoch bei der Beschreibung regulärer Sprachen durch kontextfreie Grammatiken auf: Meyer und Fischer wiesen mit Hilfe einer von Hartmanis in [9] verwendeten Codierungstechnik von Turingmaschinen-Sprachen nach, dass die Größe des zu einer kontextfreien Grammatik äquivalenten DFAs von keiner rekursiven Funktion beschränkt sein kann. In der Folge wurden von Valiant ([43]) und Schmidt und Szymanski ([41]) ähnliche Ergebnisse bei der Beschreibung kontextfreier Sprachen nachgewiesen: so ist sowohl der Größenzuwachs beim Wechsel von kontextfreien zu eindeutigen, kontextfreien Grammatiken, als auch beim Wechsel von eindeutigen, kontextfreien Grammatiken zu deterministischen Kellerautomaten durch keine rekursive Funktion beschränkt.

Anders als im Fall der Regularität deterministisch kontextfreier Sprachen kann jedoch z. B. die deterministische Kontextfreiheit kontextfreier Sprachen nicht allgemein entschieden werden. In seiner Arbeit [10] erweiterte Hartmanis die vorigen Ergebnisse auf Sprachen, zu deren Beschreibungsumfang auch die Größe des Nachweises des Enthaltenseins im weniger mächtigen System gezählt wird. Darüber hinaus stellte er eine vereinheitlichte Beweistechnik zum Nachweis dieser sogenannten nichtrekursiven Trade-offs auf.

Beeinflusst durch diese Arbeiten erschienen bis zum heutigen Tage viele Veröffentlichungen zur Effizienz verschiedenster Beschreibungssysteme. So befasst sich z. B. [6] mit der Beschreibung alternierender Automaten und zeigt unter anderem, dass DFAs die Erkennung von durch alternierende endliche Automaten (AFAs) beschriebenen Sprachen mit höchstens doppelt exponentiell vielen Zuständen bewältigen können, diese Zustandszahl jedoch im Extremfall auch nicht unterschritten werden kann.

In [7] griff Chrobak die Unterschiede der Beschreibungseffizienz von NFAs und DFAs erneut auf und diskutierte ihre Veränderung bezüglich der Beschreibung von unären Sprachen. Er wies einen maximalen – und im Extremfall auch optimalen – Trade-off von $O(e^{\Theta(\sqrt{n \log n})})$ nach.

Weitere Veröffentlichungen befassen sich mit der Beschreibungskomplexität von Zellularautomaten ([23]) oder der von Operationen auf regulären Spra-

chen (z. B. [44]). Für einen Überblick über Ergebnisse zur Beschreibungskomplexität aus diesen und weiteren Bereichen sei auf [8] verwiesen.

Ziel der vorliegenden Arbeit ist es nun, die Beschreibungseffizienz von Restart-Automaten näher zu beleuchten. Verglichen werden sollen dabei die in verschiedenem Umfang mit Berechnungsressourcen ausgestatteten Modelle untereinander sowie in Bezug auf gängige Beschreibungssysteme der eingangs erwähnten, charakterisierten Sprachklassen.

Nach einer Einführung in die Grundbegriffe im Rahmen des nachfolgenden Kapitels beschäftigen wir uns zunächst mit der Beschreibungskomplexität regulärer Sprachen und führen Vergleiche zu den bekannten Modellen DFA und NFA durch. Im vierten Kapitel wird sich die Betrachtung auf weniger eingeschränkte Typen von Restart-Automaten konzentrieren und Bezüge zu anderen Beschreibungssystemen der kontextfreien, deterministisch kontextfreien, Church-Rosser und wachsend kontextsensitiven Sprachen herstellen.

2 Grundlagen

Wir bezeichnen die Menge der natürlichen Zahlen mit $\mathbb{N}$ und fügen den Index 0 an, wenn die Null ebenfalls enthalten ist. Die Potenzmenge einer endlichen Menge M bezeichnen wir mit 2^M. Eine endliche, nichtleere Menge A von Zeichen nennen wir *Alphabet* und für die Menge der nichtleeren Wörter über A schreiben wir A^+. Kommt das leere Wort λ zu dieser Menge hinzu, so stellen wir das Ergebnis durch A^* dar.

Für die Länge eines Wortes $w \in A^*$ schreiben wir abkürzend $|w|$, wobei $|\lambda| = 0$ gilt. Die Menge aller Wörter der Länge $n \in \mathbb{N}_0$ sei durch A^n bezeichnet. Ebenso bezeichne $A^{\leq n}$ die Menge aller Wörter, deren Länge n nicht übersteigt.

Für die Spiegelung $u_n \cdots u_1$ eines Wortes $w = u_1 \cdots u_n$ schreiben wir w^R. Die Mengeninklusion bezeichnen wir mit $\subseteq$ bzw. $\subset$, falls eine echte Inklusion vorliegt.

Ferner sei $(\mathbb{B}, \wedge, \vee, \neg)$ die Schaltalgebra über der Menge $\mathbb{B} = \{0, 1\}$. Von dieser werden wir in Kapitel 3 vor allem die beiden Absorptionseigenschaften $(x \wedge y) \vee x = x$ und $(x \vee y) \wedge x = x$ (für beliebige $x, y \in \mathbb{B}$) verwenden, weshalb sie hier noch einmal explizit genannt werden.

Restart-Automaten

Das Konzept der Restart-Automaten ist im Laufe der zurückliegenden Jahre stets erweitert worden, was manchmal auch eine veränderte Beschreibung mit sich brachte. In der angegebenen Definition folgen wir im Wesentlichen der Beschreibung aus [33].

Es sei T ein Bandalphabet und $\rhd$ bzw. $\lhd$ die Markierung des linken bzw. rechten Bandendes. Das Schreib-/Lesefenster der im Folgenden definierten Automaten bewegt sich stets zwischen diesen Markierungen, so dass wir die Menge der möglichen Fensterinhalte eines Fensters der Größe k als

$$W_k = (\rhd T^{\leq k-1}) \cup T^{\leq k} \cup (T^{\leq k-1} \lhd) \cup (\rhd T^{\leq k-2} \lhd)$$

bezeichnen.

Definition 2.1 Ein System $\mathcal{M} = \langle S, A, T, \rhd, \lhd, s_0, k, \delta \rangle$, in dem S eine endliche, nicht leere Menge von Zuständen, A ein Eingabe- und T ein Bandalphabet ist, $\rhd, \lhd \notin T$ das linke bzw. rechte Bandbegrenzungszeichen, $s_0 \in S$ der Startzustand, k die Fenstergröße und δ die partielle Überführungsfunktion von $S \times W_k$ in die (endlichen) Teilmengen von

$$\Big((S \times (W_k \cup \{\mathsf{MVR}\})) \cup \{\mathsf{Restart}, \mathsf{Accept}\} \Big),$$

nennt man einen *Restart-Automaten*, wenn für alle Paare $(s', v_1 \cdots v_j) \in \delta(s, u_1 \cdots u_i)$ mit $u_1, \ldots, u_i, v_1, \ldots, v_j \in T \cup \{\rhd, \lhd\}$ die folgenden drei Eigenschaften gelten:

(i) $j < i$,

(ii) $v_1 = \rhd$ gdw. $u_1 = \rhd$ und

(iii) $v_j = \lhd$ gdw. $u_i = \lhd$.

Es sei s ein Zustand von $\mathcal{M}$ und $u \in W_k$ ein Fensterinhalt. Dann kann die Überführungsfunktion δ die folgenden Schritte liefern:

- Eine *Rechtsbewegung* (MVR-Schritt) ist von der Form $(s', \mathsf{MVR}) \in \delta(s, u)$ und versetzt die Position des Schreib-/Lesefensters um ein Zeichen nach rechts. Gleichzeitig nimmt $\mathcal{M}$ den Zustand s' an. Eine solche Bewegung ist nur möglich, wenn u noch vom rechten Bandbegrenzungszeichen verschiedene Zeichen enthält ($u \neq \lhd$).
- Ein *Umschreiben des Fensterinhaltes* (Rewrite-Schritt) hat die Form $(s', v) \in \delta(s, u)$ und veranlasst $\mathcal{M}$, den Zustand s' anzunehmen und an Stelle des Fensterinhalts u die Zeichenkette $v \in W_k$ auf das Band

zu schreiben. Das Band wird dabei nach Eigenschaft (i) entsprechend verkürzt. Am Ende des Rewrite-Schritts wird die Position des Schreib-/Lesefensters auf die rechtsbenachbart liegenden Zeichen gesetzt. Falls u mit dem Zeichen $\lhd$ endet, so gilt dies nach (iii) auch für v und das Fenster wird in diesem Fall auf $\lhd$ verschoben.

- Ein *Neustart* (Restart-Schritt) (Restart $\in \delta(s, u)$) setzt die Fensterposition wieder auf die mit dem linken Bandbegrenzungssymbol beginnenden Stellen zurück und $\mathcal{M}$ wechselt in den Startzustand s_0.
- Der Automat *akzeptiert* und hält (Accept-Schritt), wenn $\delta(s, u)$ das Element Accept enthält.

Um die einzelnen Schritte der Verarbeitung eines Wortes durch einen Restart-Automaten besser beschreiben zu können, geben wir den Gesamtzustand des Systems als Zeichenkette $\rhd usv\lhd$ an und bezeichnen diese als *Konfiguration*. Dabei steht $\rhd uv\lhd$ für den – von Bandbegrenzungszeichen eingeschlossenen – Bandinhalt, wobei $s \in S$ der anliegende Zustand ist und durch seine Position in der Zeichenkette angibt, dass sich das Schreib-/Lesefenster auf den ersten k Zeichen von v befindet (bzw. $v\lhd$ enthält, falls $|v| < k$). Den Übergang zwischen zwei Konfigurationen beschreiben wir durch die Relation $\vdash$ mit $(x, y) \in \vdash$ genau dann, wenn δ einen Schritt enthält, durch den x in y über geht. Die reflexive, transitive Hülle dieser Relation stellen wir durch $\vdash^*$ dar (sie enthält alle Paare (x, y), bei denen y durch eine beliebige Zahl von Schritten aus x hervor geht).

Der Beschreibung der von uns betrachteten Restart-Automaten fehlt zunächst noch die in allen älteren Veröffentlichungen vorkommende Einschränkung der Anzahl der Rewrite-Schritte. Die obige Definition wurde vorerst allgemeiner gehalten, da mittlerweile auch Variationen der Einschränkung der Anzahl der Rewrite-Schritte betrachtet werden (siehe etwa [25]). Die vorliegende Arbeit wird jedoch nur Automaten mit der herkömmlichen Beschränkung auf genau einen Rewrite-Schritt zwischen zwei Restart-Schritten bzw. dem Start der Verarbeitung und dem ersten Restart-Schritt betrachten.

Ausgehend von der Start- bzw. Neustart-Konfiguration $s_0 \rhd w\lhd$ (mit $w \in A^*$ bzw. T^*), kann ein Restart-Automat $\mathcal{M}$ eine Folge von Arbeitsschritten, die mit einem Restart-Schritt endet, durchlaufen. Eine solche Schrittfolge wird *Zyklus* genannt. Die *Berechnung*, die $\mathcal{M}$ auf w durchführt, besteht somit so lange aus Zyklen, bis sie durch einen undefinierten Übergang oder einen Accept-Schritt beendet wird. Die Schrittfolge vom letzten Neustart bis zu diesem Punkt bezeichnen wir lediglich als *Ende der Berechnung*.

Definition 2.2 Ein RRWW-Automat ist ein Restart-Automat, bei dessen Berechnungen jeder Zyklus genau einen Rewrite-Schritt enthalten muss. Ferner darf das Ende jeder Berechnung höchstens einen Rewrite-Schritt enthalten.

Ein RWW-Automat ist ein RRWW-Automat, der nach jedem Rewrite-Schritt umgehend einen Restart-Schritt ausführt.[2]

Ein R(R)W-Automat ist ein R(R)WW-Automat, welcher keine zusätzlichen Bandsymbole schreiben darf ($A = T$).

Ein R(R)-Automat ist ein R(R)WW-Automat, bei dem das veränderte Wort nur durch die Löschung von beliebigen Zeichen aus dem ursprünglichen Fensterinhalt hervor gehen darf.

Ein Restart-Automat heißt *deterministisch*, wenn es zu jedem Zustand s und jedem Fensterinhalt $u \in W_k$ höchstens ein Element in $\delta(s, u)$ gibt.

Eine Folge von Zyklen $C_1, \ldots, C_n$, bei der der entsprechende RRWW-Automat $\mathcal{M}$ nach dem Durchlaufen von C_n das Ende der Berechnung erreicht, wird *monoton* genannt, wenn $\mathcal{M}$ in jedem Zyklus C_i die Bandinschrift genau einmal von $\rhd x_i u_i y_i \lhd$ zu $\rhd x_i v_i y_i \lhd$ verändert und für die rechten Teilwörter die Ungleichung $|y_1| \geq |y_2| \geq \cdots \geq |y_n|$ gilt.

Ein Restart-Automat heißt *monoton*, wenn seine Berechnungen für jedes Eingabewort monoton verlaufen.

Ein Restart-Automat heißt *schwach monoton*, wenn es eine Konstante c gibt, so dass innerhalb jeder Berechnung für alle aufeinander folgenden Zyklen C_i und C_{i+1} die Ungleichung $|y_i| + c \geq |y_{i+1}|$ gilt.

Zur Abkürzung stellen wir dem entsprechenden System den Präfix det- bzw. mon- bzw. wmon- voran.

Ein Wort $w \in A^*$ wird genau dann akzeptiert, wenn es eine Ableitung $s_0 \rhd w \lhd \vdash^*$ Accept gibt. Die von einem Automaten $\mathcal{M}$ erkannte Sprache wird mit $L(\mathcal{M})$ bezeichnet. Die Klasse $\mathcal{L}(\mathsf{X})$ beinhaltet alle von Automaten des jeweiligen Typs X erkannten Sprachen.

Mit obiger Definition ist die Funktionsweise von Restartautomaten vollständig beschrieben. Bevor mit der Vorstellung von weiteren, grundlegenden Konzepten fortgefahren wird, bedarf es noch der folgenden Anmerkung:

In unseren Definitionen haben wir auf die Möglichkeit, dass Restart-Automaten auch Linksbewegungen (MVL-Schritte – siehe z. B. [35]) ausführen dürfen, verzichtet. Generell kann man sich darunter vorstellen, dass beliebig eingeschränkte oder erweiterte Automaten des Grundtyps RR während jedes Zyklus vor und nach dem Rewrite-Schritt auch Linksbewegungen ausführen dürfen. Eine formale Betrachtung hierzu ist jedoch im Rahmen dieser Arbeit nicht notwendig.

Des Weiteren ist es oft aufwändig und mühselig, alle einzelnen Überführungen eines Restartautomaten aufzuzählen. Legt man aber die oben genannten,

[2] Der durch die Überführungsfunktion zugewiesene Zustand hat dann keinerlei Bedeutung – wir werden in unseren Darstellungen stets suggestiv den Startzustand eintragen.

zykelbasierte Arbeitsweise von Restartautomaten zu Grunde, so ergibt sich die folgende, vereinfachende Beschreibung: In jedem Zyklus, d. h. vom ersten Schritt auf Basis der aktuellen Startkonfiguration bis zum Neustart, verhält sich der Automat maximal zweimal wie ein endlicher, unidirektionaler Akzeptor; nämlich vor und evtl. nach dem Ausführen des Rewrite-Schrittes.

Daher wird jeder solcher Zyklus in einer sogenannten Meta-Instruktion beschrieben, also einem Tripel, dessen erste und letzte Komponente einen regulären Ausdruck enthält und in dessen zweiten Komponente die Anwendung des Rewrite-Schrittes beschrieben ist. Für einen Restart-Automaten $\mathcal{M}$ der Fenstergröße k, der über das Bandalphabet T verfügt, ist eine *Meta-Instruktion* von der Form

$$(\rhd X, u \to v, YZ)\,,$$

wobei $Z \in \{\lambda, \lhd\}$ ist und X, Y beliebige reguläre Ausdrücke[3] über T sind bzw. $YZ = \lambda$, wenn schon beim Umschreiben das rechte Bandendesymbol erreicht war oder $\mathcal{M}$ ein R(W)(W)-Automat ist. Die zweite Komponente beschreibt, dass der Fensterinhalt $u \in T^k \cup T^{\leq k-1}\lhd)$ entsprechend der geltenden Einschränkungen in die Bandinschrift $v \in (T^{\leq k-1} \cup (T^{\leq k-2}\lhd)$ umgeschrieben wird.

Anschaulich befindet sich $\mathcal{M}$ nach der Abarbeitung der ersten Komponente in einem Zustand, in dem er den Rewrite-Schritt aus der zweiten Komponente dann ausführen kann, wenn der Fensterinhalt u ist. Nach Ausführung des Rewrite-Schrittes wird das Fenster auf die ersten k (oder weniger) Positionen rechts von v versetzt und $\mathcal{M}$ hat dann die Möglichkeit, einen Restart-Schritt auszuführen, wenn das Restwort bzw. einer dessen Präfixe von der Form YZ ist.

Eigenschaften von Restart-Automaten

Zusätzlich zu den als bekannt vorausgesetzten Sprachklassen der Chomsky-Hierarchie werden ein paar weitere, ebenfalls relativ gebräuchliche Klassen eine Rolle spielen. Da deren exakte Definitionen im Folgenden nicht von Nutzen sind, begnügen wir uns mit einer kurzen Beschreibung. Es stehe

- FIN für die Klasse der endlichen Sprachen,
- REG für die Klasse der regulären Sprachen,
- LIN für die Klasse der linearen Sprachen,

[3] Das bedingt, dass bei der Verwendung von Meta-Instruktionen zur Beschreibung von deterministischen Restartautomaten wegen der Gefahr von Mehrdeutigkeiten Vorsicht geboten ist.

- DCFL für die Klasse der deterministisch kontextfreien Sprachen,
- CFL für die Klasse der kontextfreien Sprachen,
- DCSL für die Klasse der deterministisch kontextsensitiven Sprachen,
- CSL für die Klasse der kontextsensitiven Sprachen,
- CRL für die Klasse der Church-Rosser Sprachen, also der Sprachen, deren (Eingabe-)Wörter – in einer Umgebung von Nicht-Eingabezeichen – durch ein Thue-System mit Church-Rosser Eigenschaft auf jeweils ein bestimmtes Nicht-Eingabezeichen reduziert werden können und
- GCSL für die Klasse der wachsend kontextsensitiven Sprachen, also der Sprachen, zu denen es eine Grammatik mit ausschließlich verlängernden Produktionen gibt.

Schon anhand der Definition 2.1 lässt sich erkennen, dass sämtliche betrachteten Sprachklassen innerhalb der kontextsensitiven Sprachen liegen werden, da es zu jedem Restart-Automaten einen linear beschränkten Automaten gibt, der

- den Fensterinhalt und die Bewegungen des Fensters mittels seiner Zustände und endlich vieler Bewegungen des eigenen Lesekopfes pro Schritt des Restart-Automaten simuliert und
- das Verkürzen des Bandes durch das Überschreiben der entsprechenden Positionen mit einem zusätzlichen Sonderzeichen ausgleicht.

Da kein Restart-Automat auf einem Wort der Länge n mehr als n Zyklen (mit jeweils nicht mehr als n Schritten) durchlaufen kann, gilt die folgende z. B. in [33] nachlesbare Eigenschaft.

Proposition 2.3 *Es seien NP bzw. P die bekannten Komplexitätsklassen. Dann gilt:*

(i) $\mathcal{L}(\mathsf{RRWW}) \subseteq NP \cap CSL$ *und*

(ii) $\mathcal{L}(\mathsf{det\text{-}RRWW}) \subseteq P \cap DCSL$.

Zur Einordnung der von uns betrachteten Automatenklassen bezüglich ihrer Erkennungsmächtigkeit sei im Folgenden eine Sammlung von Ergebnissen aus verschiedenen Arbeiten angeführt.

Proposition 2.4 *Für die von Restart-Automaten erkannten Sprachklassen gilt:*

(i) Charakterisierungen

- $\mathcal{L}(\textsf{det-mon-R(R)(W)(W)}) = DCFL$ *(Jančar et al. [15]),*
- $\mathcal{L}(\textsf{mon-R(R)WW}) = CFL$ *(Jančar et al. [16]),*
- $\mathcal{L}(\textsf{det-R(R)WW}) = CRL$ *(Niemann, Otto [30] und weitere) und*
- $\mathcal{L}(\textsf{mon-R(R)WW}) = CFL$ *(Otto [32] und [33]).*

Es ist offen, ob $\mathcal{L}(\textsf{RWW}) = \mathcal{L}(\textsf{RRWW})$ *gilt.*

(ii) (echte) Inklusionen

- $GCSL \subset \mathcal{L}(\textsf{RWW})$ *(Jurdziński et al. [19]),*
- $CRL \subset GCSL$ *(Buntrock, Otto [5]) und*
- $CFL \subset GCSL$ *(folgt aus obigem, da z. B.* $\{a^{2^n} \mid n \in \mathbb{N}\} \in CRL \setminus CFL$ *ist).*

Ferner ist für $\textsf{X} \in \{\textsf{det}, \textsf{mon}, \lambda\}$*:*

- $\mathcal{L}(\textsf{X-RRW}) \subset \mathcal{L}(\textsf{X-RRWW})$,
- $\mathcal{L}(\textsf{X-RW}) \subset \mathcal{L}(\textsf{X-RRW})$,
- $\mathcal{L}(\textsf{X-RR}) \subset \mathcal{L}(\textsf{X-RRW})$,
- $\mathcal{L}(\textsf{X-R}) \subset \mathcal{L}(\textsf{X-RW})$ *und*
- $\mathcal{L}(\textsf{X-R}) \subset \mathcal{L}(\textsf{X-RR})$.

Eine gute Übersicht hierzu bietet z. B. [32].

Offen ist, ob $\mathcal{L}(\textsf{RRW}) \subseteq \mathcal{L}(\textsf{RWW})$ *gilt.*

(iii) Unvergleichbarkeit

(a) $\mathcal{L}(\textsf{det-R}) \setminus CFL \neq \{\}$*, wofür [15] einen Beleg in Form einer* $\{a^{2^n}\}$*-verwandten Sprache liefert. Ebenso ist*

(b) $\mathcal{L}(\textsf{mon-R}) \setminus CRL \neq \{\}$*, in [34] belegt durch die Kodierung einer Palindromsprache,*

(c) $\mathcal{L}(\textsf{R}) \setminus GCSL \neq \{\}$*, wie – mit etwas Aufwand – in [33] gezeigt,*

(d) $CFL \setminus \mathcal{L}(\textsf{RRW}) \neq \{\}$*, was die Sprache* $\{a^n b^n \mid n \in \mathbb{N}_0\} \cup \{a^n b^m \mid m > n \geq 0\}$ *in [16] belegt, genauso wie*

(e) $CRL \setminus \mathcal{L}(\textsf{RRW}) \neq \{\}$*, wie in [33] angefügt wird. Außerdem kommen hinzu*

(f) $\mathcal{L}(\textsf{mon-RR}) \setminus \mathcal{L}(\textsf{RW}) \neq \{\}$ *und* $\mathcal{L}(\textsf{mon-RW}) \setminus \mathcal{L}(\textsf{RR}) \neq \{\}$*, sowie*

(g) $\mathcal{L}(\textsf{det-RR}) \setminus \mathcal{L}(\textsf{RW}) \neq \{\}$ *und* $\mathcal{L}(\textsf{det-RW}) \setminus \mathcal{L}(\textsf{RR}) \neq \{\}$, *belegt durch Sprachen aus [16] bzw. [32].*

Aus den obigen Resultaten ergeben sich die folgenden Unvergleichbarkeiten, bei denen stets $\textsf{X}, \textsf{Y} \in \{\textsf{R}, \textsf{RR}, \textsf{RW}, \textsf{RRW}, \textsf{RWW}, \textsf{RRWW}\}$, *sowie* $\textsf{Z} \in \{\textsf{R}, \textsf{RR}, \textsf{RW}, \textsf{RRW}\}$ *ist:*

- *(a)+(b): Für alle Paare* $(\textsf{X}, \textsf{Y})$ *ist* $\mathcal{L}(\textsf{det-X})$ *bezüglich Mengeninklusion unvergleichbar zu* $\mathcal{L}(\textsf{mon-Y})$.
- *(a/c)+(d): Bezüglich Mengeninklusion ist die Klasse CFL unvergleichbar zu den Klassen* $\mathcal{L}(\textsf{Z})$.
- *(b/c)+(e): Bezüglich Mengeninklusion ist die Klasse CRL unvergleichbar zu den Klassen* $\mathcal{L}(\textsf{Z})$.
- *(c)+(d/e): Bezüglich Mengeninklusion ist die Klasse GCSL unvergleichbar zu den Klassen* $\mathcal{L}(\textsf{Z})$.
- *(f)+(g): Für alle* $\textsf{p} \in \{\textsf{det-}, \textsf{mon-}, \lambda\}$ *sind die Klassen* $\mathcal{L}(\textsf{p-RR})$ *und* $\mathcal{L}(\textsf{p-RW})$ *unvergleichbar bezüglich Mengeninklusion.*

Diese Ergebnisse werden in Abbildung 1 zusammengefasst dargestellt. Dabei stellt eine durchgezogene Linie – in Pfeilrichtung – eine echte Inklusion dar (dabei wurden einige Inklusionspfeile zur Erhaltung der Übersichtlichkeit grau gefärbt) und die Zickzack-Linie symbolisiert eine nicht notwendigerweise echte Inklusion. Zwischen nicht verbundenen Klassen herrscht Unvergleichbarkeit – mit der Ausnahme des offenen Problems, ob $\mathcal{L}(\textsf{RRW})$ auch in $\mathcal{L}(\textsf{RWW})$ enthalten ist.

Ferner werden die folgenden Eigenschaften von Restart-Automaten des öfteren zum Tragen kommen:

Proposition 2.5 (Jančar et al. – Fehlererhaltung) *Es sei* $\mathcal{M}$ *ein Restart-Automat mit Startzustand* s_0 *und* u, v *seien Wörter über dem Eingabealphabet, so dass gilt:*

$$s_0 \rhd u \lhd \vdash^* s_0 \rhd v \lhd .$$

Dann ist $v \notin L(M)$, *wenn* $u \notin L(M)$.

Für deterministische Restart-Automaten gilt zusätzlich:[4]

[4]Damit ist nicht gesagt, dass alle korrektheitserhaltenden Restart-Automaten deterministisch sein müssen. Vielmehr werden in [25] auch Klassen von nichtdeterministischen Automaten mit geforderter Korrektheitserhaltung betrachtet und von ihren entsprechenden deterministischen Varianten getrennt.

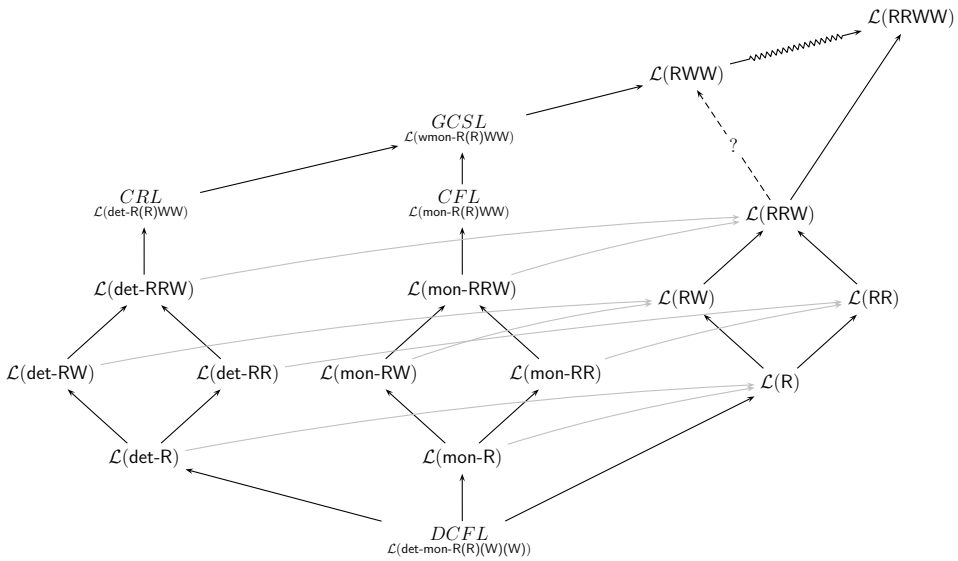

Abbildung 1: Hierarchie der Restart-Automaten-Sprachklassen

Proposition 2.6 (Jančar et al. – Korrektheitserhaltung) *Es sei $\mathcal{M}$ ein deterministischer Restart-Automat mit Startzustand s_0 und u, v seien Wörter über dem Eingabealphabet, so dass gilt:*

$$s_0 \rhd u \lhd \vdash^* s_0 \rhd v \lhd .$$

Dann ist $v \in L(M)$ (genau) dann, wenn $u \in L(M)$.

Beschreibungskomplexität

Im Unterschied zur weitaus verbreiterteren Betrachtung der generativen Mächtigkeit von Restart-Automaten soll in der vorliegenden Arbeit die relative Effizienz dieser Beschreibungssysteme behandelt werden. Eine allgemein gehaltene Vorstellung der Hintergrundidee und der für sinnvolle Folgerungen wesentlichen Voraussetzungen findet sich in [20], was uns in diesem Abschnitt als Vorbild dienen soll.

Gegenstand der Betrachtung ist die Größe der Beschreibung einer Sprache. Wir können letztere z. B. durch eine entsprechend mächtige Grammatik oder einen passenden Restart-Automaten beschreiben. Eine solche Instanz – etwa den mon-RRW-Automaten $\mathcal{D}$ – nennen wir *Beschreiber*. Das *Beschreibungssystem* S – in unserem Beispiel die Menge aller mon-RRW-Automaten – beschreibt somit die Sprachfamilie $\mathcal{L}(S) = \{L(\mathcal{D}) \mid \mathcal{D} \in S\}$.

Zur Notation einer jeden Instanz aus diesen Systemen braucht man jeweils nur endlich viel Platz. Ob dieser in Form einer kodierenden Zeichenkette

oder nur eines genügend großen Stücks Papier vorliegt, soll dabei zunächst keine Rolle spielen. Es ist nur entscheidend, dass es eine einheitliche Form gibt, in der wir kodieren.[5] Um bei der Diskussion solcher Beschreibungssysteme das Auftreten von merkwürdigen Effekten zu vermeiden, wird zusätzlich gefordert, nur rekursive (entscheidbare) Systeme zu betrachten, d. h. von jeder Darstellung soll das Enthaltensein im Beschreibungssystem stets entscheidbar bleiben.[6]

Für die Anwendbarkeit empfiehlt es sich weiterhin, dass es zu jeder festen Beschreibungsgröße nur endlich viele Beschreiber geben darf. Einen Restart-Automaten können wir beispielsweise durch die Tabellierung seiner Überführungsfunktion δ darstellen. Die Größe der Beschreibung hängt damit im Wesentlichen von der Zahl der Zustände multipliziert mit der Zahl der möglichen Fensterinhalte ab und erfüllt das obige Kriterium. Auch wenn das in unserem Beispiel sicherlich möglich ist, folgt damit im allgemeinen noch nicht, dass die Menge der Beschreiber in nach deren Größe geordneter Reihenfolge rekursiv aufzählbar ist.[7] Im Hinblick auf die Konstruktion von Entscheidungsalgorithmen ist dies aber eine sinnvolle, zusätzliche Forderung.

Zur Veranschaulichung sei das in [20] angegebene Beispiel der deterministischen, endlichen Automaten, deren Größe oft als Zahl ihrer Zustände angegeben wird, zitiert. In Kapitel 3 wird dieses Maß ebenfalls zu Grunde liegen. Es führt aber nur dann zu einer endlichen Zahl von Automaten pro Zustandszahl, wenn die Alphabetgröße während der Betrachtung festgehalten wird. Dann lässt sich implizit die Überführungsfunktion tabellieren und die Menge aller Automaten nach aufsteigender Größe sortiert rekursiv aufzählen (bis auf die Verteilung der Endzustände prinzipiell lexikographisch, wenn man die Spalte der Folgezustände quer liest).

Ein solches Größenmaß wird von uns im Folgenden als *sinnvoll* bezeichnet. Die angestellten Betrachtungen finden sich in den folgenden Definitionen wieder:

Definition 2.7 Ein Beschreibungssystem S ist eine rekursive Menge von endlichen Beschreibern, in dem jeder Beschreiber $\mathcal{D} \in S$

(i) eine formale Sprache $L(\mathcal{D})$ beschreibt und

(ii) es eine effektive Prozedur gibt, die aus $\mathcal{D}$ eine Turingmaschine, welche

[5] Im Folgenden identifizieren wir den Beschreiber mit seiner Kodierung.

[6] Da dies jedoch noch nichts über die Kenntnis des entsprechenden Entscheidungsalgorithmus besagt, wird ersatzweise gefordert, dass es einen Algorithmus gibt, der zu jedem Beschreiber effektiv eine Turingmaschine konstruiert, die die beschriebene Sprache erkennt bzw. semi-entscheidet, falls die Sprache nicht rekursiv ist.

[7] Dieses hängt wiederum mit der eventuellen Unkenntnis um den Entscheidungsalgorithmus des Beschreibungssystems zusammen.

$L(\mathcal{D})$ entscheidet (bzw. semi-entscheidet, wenn $L(\mathcal{D})$ nicht kontextsensitiv ist), konstruiert.

Wie bereits angekündigt, bezeichnen wir die Familie der Sprachen, die von den jeweiligen Elementen eines Beschreibungssystems S erkannt werden, mit $\mathcal{L}(S) = \{L(\mathcal{D}) \mid \mathcal{D} \in S\}$. Umgekehrt sei die Menge aller Beschreiber (aus einem System S) einer Sprache L durch $S(L) = \{\mathcal{D} \in S \mid L(\mathcal{D}) = L\}$ angegeben.

Definition 2.8 Es sei S ein Beschreibungssystem. Ein *(sinnvolles) Komplexitätsmaß* für S ist eine totale, rekursive Funktion $c : S \to \mathbb{N}$ mit den folgenden Eigenschaften: Für jedes feste Alphabet A ist die Menge der Beschreiber aus S von Sprachen über A

(i) rekursiv aufzählbar in der Reihenfolge ansteigender Größe und

(ii) enthält stets nur endlich viele Beschreiber gleicher Größe.

Wenn wir nun zwei verschiedene Beschreibungssysteme S_1 und S_2 bezüglich ihrer relativen Effizienz vergleichen wollen, kann dies nur auf Sprachen geschehen, die von beiden Systemen erzeugt werden können. Die implizite Annahme, dass der Schnitt der beiden Klassen $\mathcal{L}(S_1) \cap \mathcal{L}(S_2)$ nicht leer sein darf, wird daher im Folgenden nicht mehr erwähnt. Die Vorgehensweise wird nun sein, jede Sprache durch Beschreiber minimaler Größe aus beiden Systemen darzustellen und nach einer gemeinsamen Schranke zu suchen, die die Größenzunahme beim Wechsel eines beliebigen (minimalen) Beschreibers aus S_1 zu einem äquivalenten Beschreiber aus S_2 nach oben beschränkt. Bei zwei Systemen, von denen keines das jeweilige andere vollständig enthält, ist diese Untersuchung in beide Richtungen möglich. Ein Beispiel hierfür sind die Ergebnisse aus Kapitel 4.2.

Definition 2.9 Es seien S_1 und S_2 zwei Beschreibungssysteme und c ein sinnvolles Komplexitätsmaß für beide. Dann heißt eine Funktion $f : \mathbb{N} \to \mathbb{N}$, $f(n) \geq n$ *obere Schranke* für den Größenzuwachs beim Wechsel von einem beliebigen, minimalen Beschreiber aus S_1 zu einem äquivalenten, minimalen Beschreiber aus S_2, wenn für alle $L \in \mathcal{L}(S_1) \cap \mathcal{L}(S_2)$ gilt:

$$\min\{c(D) \mid D \in S_2(L)\} \leq f(\min\{c(D) \mid D \in S_1\}).$$

Wenn es keine solche, rekursive Funktion gibt, dann sagen wir, der *Trade-off* ist *nichtrekursiv*.

In einem solchen Fall ist der Größenzuwachs sogar unabhängig vom gewählten Maß, da der Unterschied g zwischen zwei verschiedenen, sinnvollen Maßen c_1 und c_2 (für ein System, in dem beide gelten) stets nur rekursiv sein

kann. Dies ist auf Grund der geforderten Eigenschaften durch die Wahl $g(n) = \max\{c_2(\mathcal{D} \mid \mathcal{D} \in S, \text{ mit } c_1(\mathcal{D}) = n\}$ einsichtig: wir können die Beschreiber von S in aufsteigender Größe bezüglich des Maßes c_1 aufzählen und zu jeder festen Größe n das Maximum der endlich vielen Maßzahlen, die uns c_2 von eben diesen Beschreibern liefert, berechnen. Damit ist die Funktion g rekursiv (weil jeder Wert $g(n)$ in endlicher Zeit berechenbar ist).

3 Zustandskomplexität auf regulären Sprachen

Ein im Jahre 2001 veröffentlichter Artikel von Mráz [27] beschäftigt sich mit der Auswirkung der Beschränkung der Fenstergröße von Restart-Automaten auf die (Erkennungs-) Mächtigkeit. Es wird nachgewiesen, dass die Klasse der von R(W)(W)-Automaten mit Fenstergröße 1 (kurz: R(1)-Automaten) erkannten Sprachen mit der der Regulären Sprachen zusammenfällt. Vergleicht man erstere mit den „traditionellen Akzeptoren“ regulärer Sprachen, wie z. B. *deterministischen* bzw. *nichtdeterministischen, endlichen Automaten* (DFAs bzw. NFAs), so fällt ins Auge, dass sich R(1)-Automaten von NFAs nur über die zusätzliche Fähigkeit, das gerade gelesene Zeichen zu löschen und (den Einleseprozess) neu zu starten, unterscheiden.[8] Es stellt sich also die Frage, in wie weit die Hinzunahme von Lösch- und Neustart-Schritten überhaupt eine Neuerung mit sich bringt. In Bezug auf die generative Mächtigkeit liegt die negative Antwort darauf schon vor. Im Folgenden wird sich aber erkennen lassen, dass sich einzelne Sprachen durch bestimmte R(1)-Automaten wesentlich effizienter beschreiben lassen, als durch NFAs und DFAs.

Solche Untersuchungen zur relativen Beschreibungseffizienz verschiedener Akzeptoren von regulären Sprachen legten nicht nur den historischen Grundstein (die frühste, uns bekannte Betrachtung dazu findet sich schon in [42], bei denen deterministische Kellerautomaten zur Erkennung von regulären Sprachen verwendet und in ihrer Effizienz mit deterministischen, endlichen Automaten verglichen wurden), sondern bilden mittlerweile ein dichtes Feld von Resultaten mit aktuellen Veröffentlichungen bis zum heutigen Tag, z. B. [45] und [17].

Gegenstand der Betrachtungen dieses Kapitels wird also die Klasse der regulären Sprachen sein, welche sich durch sogar zwei verschiedene Restart-Automaten-Modelle mit Fenstergröße 1 darstellen lässt. Dazu beginnen wir im nachfolgenden Abschnitt mit der Betrachtung des schon angeführten R(1)-Automaten und der Gegenüberstellung ihrer Ausdruckseffizienz mit der der endlichen Automaten. Im Anschluss daran soll gezeigt werden, dass uns die Klasse der Restartautomaten noch ein weiteres Automatenmodell zum Erkennen der regulären Sprachen bereit hält: die det-RR(1)-Automaten. Die sich daraus ergebenden Vergleichsmöglichkeiten werden auf die darauf folgenden Abschnitte verteilt.

Bevor wir jedoch damit beginnen, legen wir noch eine Forderung für alle in diesem Kapitel betrachteten Restart-Automaten zu Grunde, welche uns stets einen fairen Vergleich zur Arbeitsweise der endlichen Automaten gewährt:

[8] Dies ist nicht ganz präzise: eigentlich müssten die NFAs mit einer beliebigen Anzahl von Startzuständen ausgestattet sein. Eine Anmerkung hierzu und die Festlegung einer einheitlichen Betrachtungsweise findet sich am Ende dieses Abschnitts.

Da ein R(1)-Automat schon auf dem linken Bandbegrenzungszeichen eine nichtdeterministische Wahl seines Folgezustands treffen kann, lenkt der Vergleich zu den restlichen Systemen, die nur über einen Startzustand (der beim Lesen des ersten Eingabezeichens anliegt) verfügen, eventuell von den wesentlichen Einflussfaktoren ab. Bei deterministischen Restart-Automaten, etwa det-R(R)(1), wäre somit der Zustand $\delta(s_0, \triangleright)$ der eigentliche Startzustand. Beides können wir ohne den Verlust von Erkennungsmächtigkeit durch die Festlegung $\delta(s_0, \triangleright) = \{s_0\}$ vereinfachen.[9] Diese Wahl gilt – wie gesagt – für alle Automaten dieses Kapitels und wir werden die besondere Berücksichtigung des linken Bandbegrenzungszeichens somit in allen unseren Betrachtungen außer Acht lassen können.

[9] Es ist anschaulich klar, dass diese Wahl im deterministischen Fall nur einer Umnummerierung der Zustände entspricht und keine Einschränkung bedeutet: der neue wie auch der ursprüngliche Zustand s_0 können während einer entsprechenden Verarbeitung ebenso angenommen werden, wie zuvor. Für den nichtdeterministischen Fall sei vorweg genommen, dass sich die im Folgenden ermittelten Grenzen im Allgemeinen bestenfalls um eins vermindern. Eine genauere Betrachtung dazu findet sich am Ende dieses Kapitels.

3.1 R(1)- und endliche Automaten

Wie schon angekündigt, besteht eine der wesentlichen Grundlagen dieses Abschnitts aus dem folgenden Satz:

Satz 3.1 (Mráz) *Eine Sprache ist genau dann regulär, wenn sie von einem R(1)-Automaten erkannt wird.*

Im dazugehörigen Beweis wird explizit ein äquivalenter DFA angegeben. Um eine gute Einstiegsmöglichkeit in die Arbeitsweise dieser eingeschränkten Automatenmodelle zu haben, schauen wir uns zunächst einmal die entsprechende, in der Darstellungsweise leicht angepasste Konstruktion an:

Es sei $\mathcal{M} = \langle S, A, A, \rhd, \lhd, s_0, 1, \delta \rangle$ ein beliebiger R(W)(W)(1)-Automat. Zur Erinnerung an die Definition sei angemerkt, dass $\mathcal{M}$ auf Grund seiner Fenstergröße keine Zeichen umschreiben, sondern nur löschen kann. Nach einem solchen Schritt muss $\mathcal{M}$ sofort neu starten. Ferner ist $\mathcal{M}$ nichtdeterministisch.

Der dazu äquivalente DFA $\mathcal{M}' = \langle 2^S \cup \{s_a\}, A, s_0, F, \delta' \rangle$ ähnelt der bekannten Potenzmengen-Konstruktion zur Simulation von nichtdeterministischen endlichen Automaten – erweitert um einen Zustand s_a. Dieser ist für den simulierenden DFA dann von Nöten, wenn $\mathcal{M}$ an einer anderen Stelle als dem rechten Wortende akzeptiert und $\mathcal{M}'$ somit noch das Restwort einlesen muss (obwohl schon fest steht, dass das Wort akzeptiert wird). Natürlich ist s_a damit stets ein Endzustand.

Zu einem beliebigen Zustand $S' \in 2^S \cup \{s_a\}$ und einem Zeichen $x \in A$ stellt die Überführungsfunktion δ' die folgenden Arbeitsschritte von $\mathcal{M}$ nach:

Fall 1 Wenn $\mathcal{M}$ zu jedem in S' vorkommenden Zustand auf dem Zeichen x lediglich MVR-Schritte ausführt oder undefiniert ist, überführt δ' in den Zustand S'_x, der alle Nachfolge-Zustände der MVR-Schritte von $\mathcal{M}$ enthält:

$$\delta'(S', x) = S'_x, \text{ mit } S'_x = \{s' \mid \exists s \in S' : \ (s', \mathsf{MVR}) \in \delta(s, x)\}.$$

Fall 2 Falls S' einen Zustand von $\mathcal{M}$ enthält, der durch das Zeichen x einen Accept-Schritt verursacht, so geht S' in den Zustand s_a über, wenn das nicht schon vorher der Fall war:

$$\delta'(S', x) = s_a, \text{ falls entweder } S' = s_a \text{ oder } \exists s \in S' : \ \mathsf{Accept} \in \delta(s, x).$$

Fall 3 Falls keiner der beiden obigen Fälle eintritt, gibt es einen Zustand von $\mathcal{M}$ in S', für den das Zeichen x gelöscht wird. Nach dem sofortigen Neustart, wird $\mathcal{M}$ das nachfolgende Zeichen genau in einem der

Zustände erreichen, aus denen S' besteht. Ebenso kann $\mathcal{M}$ das nachfolgende Zeichen in allen Zuständen erreichen, die auf die Zustände aus S' folgen, in denen das Zeichen x nicht gelöscht wird:

$$\delta'(S', x) = S' \cup S'_x, \text{ wenn } \exists s \in S' : \ (s', \lambda) \in \delta(s, x).$$

S'_x ist hierbei wie in Fall 1 definiert.

Nun muss noch das Verhalten von $\mathcal{M}$ auf dem rechten Bandbegrenzungszeichen simuliert werden. Dieses Zeichen gehört zwar nicht zum Alphabet von $\mathcal{M}'$, jedoch markiert es das Wortende und kann daher durch eine entsprechende Wahl der Endzustände ersetzt werden. Da am Bandende weder MVR- noch Rewrite-Schritte (man beachte, dass $\mathcal{M}$ auf Grund seiner Definition auch keine separaten Restart-Schritte tätigen kann) erfolgen können, muss hier nur beachtet werden, dass sich $\mathcal{M}$ in einem Zustand befinden kann, aus dem er beim Erreichen des rechten Bandendes das Wort noch akzeptiert. Daher ist

$$F = \{s_a\} \cup \{s \in S : \ \mathsf{Accept} \in \delta(s, \triangleleft)\}.$$

Der diesbezügliche Korrektheitsnachweis findet sich in [27]. Das folgende Korollar fasst lediglich die wesentliche Bedeutung, die sich damit für dieses Kapitel ergibt, zusammen.

Obere Schranken

Korollar 3.2 *Zu jedem R(1)-Automaten $\mathcal{M}$ mit n Zuständen gibt es nach der Konstruktion von Mráz einen DFA mit nicht mehr als $2^n + 1$ Zuständen, der $L(\mathcal{M})$ akzeptiert.*

Da wir für deterministische endliche Automaten gefordert haben, dass die Überführungsfunktion stets total ist, kann nach derselben Konstruktion ein NFA mit 2^n Zuständen erhalten werden, wenn man die Übergänge in den *unproduktiven* Zustand $\{\}$ entfernt.

Korollar 3.3 *Zu jedem R(1)-Automaten $\mathcal{M}$ mit n Zuständen gibt es einen NFA mit nicht mehr als 2^n Zuständen, der $L(\mathcal{M})$ akzeptiert.*

Die unmittelbare Frage dazu ist, ob es (von R(1)-Automaten erkannte) Sprachen gibt, zu deren Erkennung eine solche, relative Größe notwendig ist. Wir nähern uns der Antwort schrittweise:

(Wesentliche) untere Schranken

Da man einen R(1)-Automaten, der niemals löscht, stets in einen gleich großen, äquivalenten NFA übersetzen kann, folgt mit Hilfe des bekannten trade-offs zwischen NFA und DFA, dass die optimale untere Schranke des Zustands-trade-offs von R(1)-Automaten zu deterministischen endlichen Automaten schon sehr nah (bis auf einen Zustand) an der gefundenen, oberen Schranke liegt:

Bemerkung 3.4 *Für jedes $n \in \mathbb{N}$ gibt es einen R(1)-Automaten $\mathcal{M}$ mit n Zuständen, so dass jeder DFA, der $L(\mathcal{M})$ erkennt, mindestens 2^n Zustände besitzen muss.*

Um die Optimalität der gefundenen, oberen Schranke auch gegenüber nichtdeterministischen, endlichen Automaten nachzuweisen, benutzen wir eine Technik, mit deren Hilfe schon in verschiedenen Gebieten untere Komplexitätsschranken nachgewiesen wurden. Soweit sich zurückverfolgen lässt, wurde ihre Bezeichnung *Fooling Set* erstmalig von Hromkovič in [12] verwendet. In den Veröffentlichungen von Jirásková (z. B. [18], [17]) werden unter diesem Namen und mit verwandter Grundidee, welche aber auch auf eine frühere Quelle (Birget, [1]) zurückgeht, Komplexitätsschranken bezüglich nichtdeterministischer endlichen Automaten aufgestellt. Die nachfolgende Definition entspricht der letzteren Darstellungsweise.

Definition 3.5 (Fooling Set) Ein *Fooling Set* zu einer regulären Sprache L ist eine nichtleere Menge von Wortpaaren $\{(x_i, y_i) \mid i = 1, \ldots, n\}$, wobei für alle $i, j \in \{1, \ldots, n\}$ die folgenden Eigenschaften gelten:

1. $x_i y_i \in L$ und
2. falls $i \neq j$, dann ist höchstens eines der Worte $x_i y_j$ und $x_j y_i$ aus L.

Die Idee eines Fooling Sets ist es, eine Menge von Wörtern mit jeweils einem ausgezeichneten Prä- und Suffix zu finden, so dass jedes der Präfixe den Automaten in eine bestimmte Menge von möglichen Zuständen bringt, aus dem fast[10] kein Suffix der anderen Wörter mehr akzeptierbar ist. Es ist somit eine Sammlung von Beispielwörtern, deren ausgezeichnetes Präfix einen bestimmten Zustand trifft, der zu allen anderen so erreichten Zuständen nicht äquivalent sein kann. Dieses Ergebnis fasst der folgende Satz zusammen.

Lemma 3.6 (Birget, 1992) *Sei X ein Fooling Set für die reguläre Sprache L. Dann hat jeder NFA, der L erkennt, mindestens $|X|$ Zustände.*

[10] Genauer: es gibt beim paarweisen Vergleich stets eine Kombination aus Präfix und fremdem Suffix, die diese Eigenschaft hat.

Mit dieser Hilfe können wir nun zur Antwort auf die Frage ansetzen, ob man den R(1)-Automaten aus Satz 3.1 durch einen NFA hätte effizienter simulieren können. Die Antwort lautet: im Allgemeinen nicht.

Satz 3.7 *Zu jedem $n \in \mathbb{N}$ gibt es eine Sprache L_n über einem ternären Alphabet, die von einem R(1)-Automaten in n Zuständen akzeptiert wird und zu deren Erkennung jeder NFA wenigstens $2^n - 1$ Zustände benötigt.*

Beweis Die Sprache L_n sei als $L(\mathcal{M}_n)$ über den folgenden R(1)-Automaten beschrieben:

$$\mathcal{M}_n = \langle \{s_0, \ldots, s_{n-1}\}, A, A, \triangleright, \triangleleft, s_0, 1, \delta \rangle \text{ mit } A = \{a, b, c\}, \text{ wobei}$$

die Überführungsfunktion δ durch

$$\begin{aligned}
\delta(s_i, a) &= \{(s_0, \mathsf{MVR}), (s_i, \mathsf{MVR})\} && \text{für } i \in \{1, \ldots, n-1\},\\
\delta(s_j, b) &= \{(s_{(j+1) \bmod n}, \mathsf{MVR})\} && \text{für } j \in \{0, \ldots, n-1\},\\
\delta(s_0, c) &= \{(s_0, \lambda)\},\\
\delta(s_0, \triangleleft) &= \{\mathsf{Accept}\}
\end{aligned}$$

und in allen anderen Fällen als undefiniert festgelegt ist.

Für die Zeichen a und b verhält sich $\mathcal{M}$ wie ein aus [38] entnommener NFA, von dem Salomaa und Yu nachweisen konnten, dass jeglicher äquivalente DFA mindestens 2^n Zustände besitzen muss. Hinzu kommt, dass $\mathcal{M}$ im Zustand s_0 das Zeichen c löschen kann, was bewirkt, dass es nun keinen NFA mehr gibt, der den um diese Übergänge erweiterten Potenzmengen-DFA effizienter beschreiben kann, als dieser selbst.

Zum Nachweis erstellen wir ein Fooling Set aus Paaren $(u_x, v_x w_x)$, deren Teilwörter u_x, v_x und w_x jeweils einer nichtleeren Teilmenge $S_x \subseteq S$ zugeordnet sind (n ist dabei fest):

Für $j \in \{1, \ldots, n-1\}$ sei $S_x = \{s_{i_1}, \ldots, s_{i_j}\}$ und für alle $k \in \{2, \ldots, j\}$ gelte: $i_1, i_k \in \{0, \ldots, n-1\}$ und $i_{k-1} < i_k$.

Um Lemma 3.6 anzuwenden zu können ist noch das Folgende zu zeigen:

i) Es gibt ein Wort u_x, so dass S_x die Menge der möglichen Zustände von $\mathcal{M}_n$ nach der Verarbeitung von u_x ist.

ii) Es gibt ein Teilwort v_x, so dass die Menge der möglichen Zustände von $\mathcal{M}_n$ nach der Verarbeitung von v_x genau dann S_x ist, wenn S_x auch die Menge der möglichen Zustände beim Erreichen des ersten Zeichens von v_x war. Ferner wird die Berechnung von $\mathcal{M}_n$ auf dem Teilwort v_x (verwerfend) abbrechen, wenn in der Menge der möglichen Zustände beim Erreichen des ersten Zeichens von v_x wenigstens ein Zustand aus S_x nicht vorkommt.

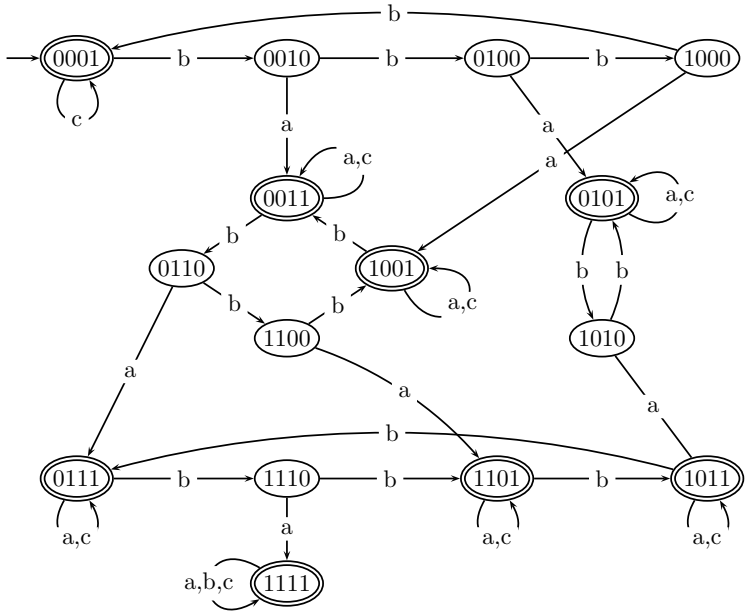

Abbildung 2: der zu $\mathcal{M}$ äquivalente, minimale NFA ($n = 4$)

iii) Es gibt ein Wort w_x, so dass $\mathcal{M}_n$ das auf w_x endende Wort dann akzeptiert, wenn S_x die Menge der möglichen Zustände von $\mathcal{M}_n$ beim Erreichen des ersten Zeichens von w_x ist.

Im Folgenden seien $Q, R \in A^*$ zwei beliebige Zeichenketten.

Als Wörter mit der Eigenschaft i) wählen wir

$$u_x = b^{i_j - i_{j-1}} a b^{i_{j-1} - i_{j-2}} a \dots b^{i_2 - i_1} a b^{i_1}.$$

Um die sich darauf ergebende Abarbeitung besser darstellen zu können, erweitern wir die Konfiguration, indem wir an Stelle des anliegenden Zustands die Menge der möglichen, anliegenden Zustände (bei gleicher Kopfposition und Bandinschrift) schreiben. Zur Unterscheidung von der gewöhnlichen Folgerelation $\vdash$ bzw. ihrer reflexiven, transitiven Hülle $\vdash^*$ verwenden wir nun gleichbedeutend $\models$ bzw. $\models^*$.

Es gilt:

$$\begin{aligned}
\{s_0\}u_xR\lhd &\vDash^* b^{i_j-i_{j-1}}\{s_{i_j-i_{j-1}}\}a\cdots b^{i_1}R\lhd\\
&\vDash b^{i_j-i_{j-1}}a\{s_0,s_{i_j-i_{j-1}}\}b^{i_{j-1}-i_{j-2}}a\cdots b^{i_1}R\lhd\\
&\vDash^* b^{i_j-i_{j-1}}ab^{i_{j-1}-i_{j-2}}a\{s_0,s_{i_{j-1}-i_{j-2}},s_{i_{j-2}-i_{j-3}}\}\cdots b^{i_1}R\lhd\\
&\vdots\\
&\vDash^* b^{i_j-i_{j-1}}a\cdots b^{i_2-i_1}a\{s_0,s_{i_2-i_1},\ldots,s_{i_j-i_1}\}b^{i_1}R\lhd\\
&\vDash u\{s_{i_1},\ldots,s_{i_j}\}R\lhd = u_xS_xR\lhd .
\end{aligned}$$

Als Wörter mit der Eigenschaft ii) wählen wir

$$v_x = b^{n-i_j}cb^{i_j-i_{j-1}}cb^{i_{j-1}-i_{j-2}}c\ldots b^{i_2-i_1}cb^{i_1}.$$

Unter der Voraussetzung, dass $\mathcal{M}$ auf dem Präfix Q nicht löscht, ergibt sich

$$\begin{aligned}
QS_xv_xR\lhd &\vDash^* Qb^{n-i_j}\{s_{n-i_j+i_1},\ldots,s_{n-i_j+i_{j-1}},s_0\}cb^{i_j-i_{j-1}}\cdots b^{i_1}R\lhd\\
&\vDash Qb^{n-i_j}\{s_{n-i_j+i_1},\ldots,s_{n-i_j+i_{j-1}},s_0\}b^{i_j-i_{j-1}}\cdots b^{i_1}R\lhd,
\end{aligned}$$

da der Zustand s_0 in der Menge der annehmbaren Zustände enthalten ist, so dass c gelöscht werden kann. Die Menge der nach der Löschung am nachfolgenden Zeichen anliegenden Zustände bleibt dieselbe, da zu c keine sonstigen Übergänge definiert waren.

Die Verarbeitung durch $\mathcal{M}$ setzt sich nun nach demselben Muster fort:

$$\begin{aligned}
QS_xv_xR\lhd &\vDash^* Qb^{n-i_j}\{s_{n-i_j+i_1},\ldots,s_{n-i_j+i_{j-1}},s_0\}b^{i_j-i_{j-1}}\cdots b^{i_1}R\lhd\\
&\vDash^* Qb^{n-i_{j-1}}\{s_{n-i_{j-1}+i_1},\ldots,s_{n-i_{j-1}+i_{j-2}},s_0,s_{i_j-i_{j-1}}\}cb\cdots b^{i_1}R\lhd\\
&\vDash Qb^{n-i_{j-1}}\{s_{n-i_{j-1}+i_1},\ldots,s_{n-i_{j-1}+i_{j-2}},s_0,s_{i_j-i_{j-1}}\}b\cdots b^{i_1}R\lhd\\
&\vdots\\
&\vDash^* Qb^{n-i_1}\{s_0,s_{i_2-i_1},\ldots,s_{i_j-i_1}\}cb^{i_1}R\lhd\\
&\vDash Qb^{n-i_1}\{s_0,s_{i_2-i_1},\ldots,s_{i_j-i_1}\}b^{i_1}R\lhd\\
&\vDash^* Qb^nS_xR\lhd .
\end{aligned}$$

Analog weisen wir für das gewählte v_x für eine beliebige Menge S_y mit $S_x \setminus S_y \neq \{\}$ nach, dass die Verarbeitung stets verwerfend abbricht. Dazu seien ohne Einschränkung für ein $k \leq j$ die Zustände $s_{i_{k+1}},\ldots,s_{i_j}$ auch in S_y und s_{i_k} nicht. Dann sieht die Verarbeitung durch $\mathcal{M}$ nach obigem Muster

wie folgt aus:

$$\begin{aligned} QS_y v_x R\lhd &\vDash^* Qb^{n-i_j} S_j c b^{i_j - i_{j-1}} \cdots b^{i_1} R\lhd \\ &\vDash^* Qb^{n-i_{j-1}} S_{j-1} c b^{i_{j-1} - i_{j-2}} \cdots b^{i_1} R\lhd \\ &\vdots \\ &\vDash^* Qb^{n-i_k} S_k c b^{i_k - i_{k-1}} \cdots b^{i_1} R\lhd \\ &\vDash \mathsf{Reject}, \end{aligned}$$

wobei nach $s_{i_{k+1}}, \ldots, s_{i_j} \in S_y$ jede der Mengen $S_j, \ldots, S_{k+1}$ den Zustand s_0 enthält, S_k jedoch nicht.

Für die dritte Eigenschaft wählen wir $w_x = b^{n-i_j}$, womit sich

$$\begin{aligned} QS_x w_x\lhd &\vDash^* Qb^{n-i_j}\{s_{n-i_j+i_1}, \ldots, s_{n-i_j+i_{j-1}}, s_0\}\lhd \\ &\vDash \mathsf{Accept} \end{aligned}$$

ergibt, da $\mathcal{M}$ im Zustand s_0 am rechten Bandbegrenzungszeichen akzeptiert (aus allen anderen Zuständen wird verworfen).

Damit ist bewiesen, dass die Paare $(u_x, v_x w_x)$ ein Fooling Set bilden, da

1. nach den Eigenschaften i)-iii) jedes Wort $u_x v_x w_x \in L(\mathcal{M}_n)$ ist und
2. für $S_x \neq S_y$ nach den Eigenschaften i) und ii) mindestens eines der Wörter $u_x v_y w_y$ und $u_y v_x w_x$ nicht in $L(\mathcal{M}_n)$ liegen kann, weil mindestens eine der beiden zugehörigen Zustandsmengen einen Zustand enthält, der der anderen fehlt.

Es gibt $2^n - 1$ Paare in dem so konstruierten Fooling Set, womit die Behauptung gezeigt ist. □

Somit liegen die gefundenen oberen und unteren Schranken der Zustandstrade-offs von R(1)-Automaten zu deterministischen und nichtdeterministischen sehr eng beieinander.

Scharfe untere Schranke

Damit sich beide Schranken exakt treffen, müssen wir sowohl das Erreichen des Sonderzustands s_a, als auch dessen Nicht-Äquivalenz zu allen Zuständen aus dem Potenzmengen-Konstrukt sicherstellen. Zu diesem Zweck erweitern wir das Alphabet, womit Lösch- und Akzeptiervorgänge voneinander unabhängig gemacht werden können.

Satz 3.8 *Zu jedem $n \geq 2$ gibt es eine Sprache L_n über einem vierbuchstabigen Alphabet, die von einem R(1)-Automaten in n Zuständen akzeptiert wird und zu deren Erkennung jeder NFA wenigstens 2^n Zustände benötigt.*

Beweis Wie im vorigen Beweis sei die Sprache L_n als $L(\mathcal{M}_n)$ über den folgenden R(1)-Automaten festgelegt, wobei s_0 und s_{n-1} nicht auf einen Zustand zusammenfallen dürfen:

$$\mathcal{M}_n = \langle \{s_0, \ldots, s_{n-1}\}, A, A, \triangleright, \triangleleft, s_0, 1, \delta \rangle \text{ mit } A = \{a, b, c, d\} \text{ und}$$

$$\begin{aligned}
\delta(s_0, a) &= \{(s_0, \mathsf{MVR}), (s_1, \mathsf{MVR})\}, \\
\delta(s_i, a) &= \{(s_{(i+1) \bmod n}, \mathsf{MVR})\} && \text{für } i \in \{1, \ldots, n-1\}, \\
\delta(s_i, b) &= \{(s_i, \mathsf{MVR})\} && \text{für } i \in \{1, \ldots, n-1\}, \\
\delta(s_0, c) &= \{(s_0, \lambda)\}, \\
\delta(s_{n-1}, d) &= \{\mathsf{Accept}\}
\end{aligned}$$

und δ undefiniert in allen anderen Fällen.

Das Verhalten der Überführungsfunktion auf den Zeichen a und b ist einem, von Leung in [22] verwendeten NFA nachempfunden.

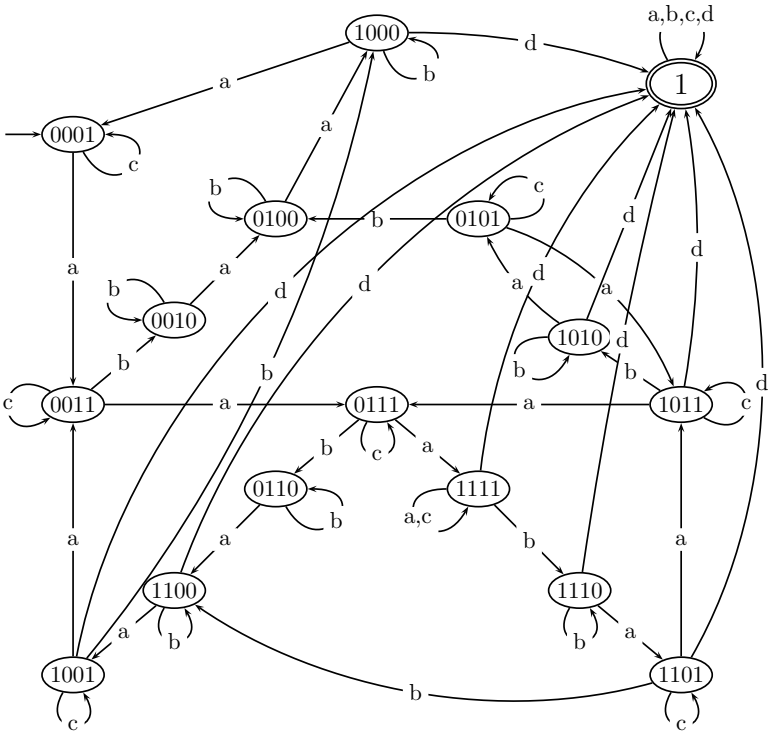

Abbildung 3: der zu $\mathcal{M}$ äquivalente, minimale NFA ($n = 4$)

Ähnlich zum vorigen Beweis erstellen wir nun (für festes n) ein Fooling Set aus Paaren $(u_x, v_x w_x)$, deren Teilwörter u_x, v_x und w_x jeweils einer nichtleeren Teilmenge $S_x \subseteq S$ zugeordnet sind:

Für $j \in \{1, \ldots, n-1\}$ sei $S_x = \{s_{i_1}, \ldots, s_{i_j}\}$ und für alle $k \in \{2, \ldots, j\}$ gelte: $i_1, i_k \in \{0, \ldots, n-1\}$ und $i_{k-1} < i_k$.

Die Vorgehensweise ist wiederum ähnlich zum vorigen Beweis:

i) Es gibt ein Wort u_x, so dass S_x die Menge der möglichen Zustände von $\mathcal{M}_n$ nach der Verarbeitung von u_x ist.

ii) Es gibt ein Teilwort v_x, so dass die Menge der möglichen Zustände von $\mathcal{M}_n$ nach der Verarbeitung von v_x genau dann $\{s_0\}$ ist, wenn S_x die Menge der möglichen Zustände beim Erreichen des ersten Zeichens von v_x war. Ferner wird die Berechnung von $\mathcal{M}_n$ auf dem Teilwort v_x (verwerfend) abbrechen, wenn in der Menge der möglichen Zustände beim Erreichen des ersten Zeichens von v_x wenigstens ein Zustand aus S_x nicht vorkommt.

iii) Es gibt ein Wort w_x, so dass $\mathcal{M}_n$ das auf w_x endende Wort dann akzeptiert, wenn die Menge der möglichen Zustände von $\mathcal{M}_n$ beim Erreichen des ersten Zeichens von w_x den Zustand s_0 enthält.

iv) Es gibt ein Wort v', das stets verworfen wird, wenn es in einem beliebigen Zustand aus S von $\mathcal{M}_n$ erreicht wird.

Im Folgenden seien $Q, R \in A^*$ zwei beliebige Zeichenketten.

Zum Erhalten der Eigenschaft i) wählen wir

$$u_x = a^{n-1}(ba)^{n-1-i_j}a(ba)^{i_j-1-i_{j-1}}a(ba)^{i_{j-1}-1-i_{j-2}}\ldots a(ba)^{i_2-1-i_1}(ab)^{i_1}$$

und weisen

$$\{s_0\}u_xR\lhd \vDash^* u_xS_xR\lhd$$

auf zum vorigen Beweis analoge Weise nach.

Als Zweites sei

$$v_x = \begin{cases} cba^{n-i_j}cba^{i_j-i_{j-1}}cba^{i_{j-1}-i_{j-2}}\ldots cba^{i_3-i_2}c, & \text{falls } i_1 = 0 \text{ und} \\ a^{n-i_j}cba^{i_j-i_{j-1}}cba^{i_{j-1}-i_{j-2}}\ldots cba^{i_2-i_1}c, & \text{falls } i_1 \neq 0. \end{cases}$$

Wir zeigen

$$\begin{aligned} QS_xv_xR\lhd &\vdash^* Qba^{n-i_j}ba^{i_j-i_{j-1}}ba^{i_{j-1}-i_{j-2}}\ldots ba^{i_3-i_2}\{s_0\}R\lhd \text{ bzw.} \\ QS_xv_xR\lhd &\vdash^* Qa^{n-i_j}ba^{i_j-i_{j-1}}ba^{i_{j-1}-i_{j-2}}\ldots ba^{i_2-i_1}\{s_0\}R\lhd \end{aligned}$$

und für $S_x \setminus S_y \neq \{\}$:

$$QS_yv_xR\lhd \vDash^* \mathsf{Reject}.$$

Die Wahl des dritten Teilwortes fällt im Rahmen der beabsichtigten Konkatenation von v_x und w_x stets gleich aus: $w_x = (ab)^{n-1}d$, womit das Erkennen des Wortes folgt.

Analog zum vorigen Beweis sind damit die Fooling Set-Eigenschaften für die so beschriebene, $(2^n - 1)$-elementigen Menge (aus Paaren $(u_x, v_x w_x)$) gezeigt.

Als letztes Paar nehmen wir (u', v') hinzu, wobei $\mathcal{M}_n$ das Wort $u'v'$ schon vor dem Erreichen von v' akzeptiert und v' die geforderte Eigenschaft iv) besitzt. Dies gelingt durch die Wahl $u' = (ab)^{n-1}d$ und $v' = (cba)^n c$.

Die entsprechenden Eigenschaften des so erweiterten Fooling Sets lassen sich leicht zeigen, da $u'v'$ wie gefordert in $L(\mathcal{M}_n)$ liegt; jedoch keines der Wörter $u_x v'$, womit die Behauptung gezeigt ist. □

Die Lücke zwischen oberer und unterer Schranke des Zustands-Zuwachses beim Wechsel von R(1) nach DFA ist damit ebenfalls geschlossen, da jeder entsprechende DFA mit totaler Überführungsfunktion zusätzlich noch einen unproduktiven Zustand benötigt.

Korollar 3.9 *Zu jedem $n \geq 2$ gibt es eine Sprache L_n über einem vierbuchstabigen Alphabet, die von einem R(1)-Automaten in n Zuständen akzeptiert wird und zu deren Erkennung jeder DFA wenigstens $2^n + 1$ Zustände benötigt.*

Die oberen Schranken sind damit – über relativ kleinen Alphabeten – als exakt nachgewiesen. Darüber hinaus wäre sicherlich die Frage nach der Optimalität der zum Erreichen der trade-offs verwendeten Alphabetgrößen interessant. Die Suche nach einer diesbezüglichen Antwort würde uns jedoch zu weit vom beabsichtigten Schwerpunkt der Arbeit weg führen, so dass wir darauf verzichten und uns – wie angekündigt – dem nächsten Beschreibungssystem zuwenden.

3.2 Beschreibung von det-RR(1)-erkannten Sprachen

Im Folgenden untersuchen wir eine weitere Automatenklasse, die die regulären Sprachen beschreibt. Schon die stark eingeschränkte Restartautomatenklasse det-mon-R(2) enthält Automaten, welche nicht-reguläre Sprachen, wie z. B. ein einfache Dyck-Sprache ([27]) oder die Sprache der binären Wörter mit gleicher Anzahl von Nullen und Einsen durch intuitive Lösch-Schritte erkennen können. Daher bleibt nur, eine Fenstergröße von 1 zu wählen. Jedoch könnte dem Automaten gestattet werden, nach dem Löschen noch einen Nachlesevorgang zu starten. Als – wie sich herausstellt – notwendige Abschwächung dazu, wird im Gegenzug eine deterministische Arbeitsweise gefordert (nach unseren Konventionen sprechen wir also von det-RR(1)-Automaten).

Von ihnen wird sich ergeben, dass auch sie genau die Familie der regulären Sprachen erkennen. Interessant ist jedoch vor allem, dass det-RR(1)-Automaten ebenso wenig als Erweiterung nichtdeterministischer, endlicher Automaten gesehen werden können, wie umgekehrt. Dieses wird sich auch direkt auf die Frage nach der Beschreibungskomplexität auswirken – präziser gesagt danach, wieviel Zustände ein simulierender NFA höchstens und (im Extremfall) mindestens benötigt, um einen det-RR(1)-Automaten zu simulieren. Dass das Ergebnis hiervon ein mehr als exponentieller Trade-off ist, zeugt von der Mächtigkeit des Nachlese-Vorgangs.

Im darauf folgenden Abschnitt wird aber klar werden, dass der Nachlese-Vorgang keine Kompensation des Nichtdeterminismus darstellen kann, da wir auch dort Sprachfamilien angeben können, die sich durch einen NFA exponentiell effizienter beschreiben lassen, als durch den jeweils besten det-RR(1)-Automaten.

Wenn wir nun mit diesen Untersuchungen beginnen, wird es zunächst nützlich sein, die Zustandsmenge der betrachteten det-RR(1)-Automaten in zwei Teilmengen aufzuspalten, um damit die im Nachlesevorgang erreichten Zustände leichter beschreiben zu können. Wegen der geforderten Eigenschaft, dass ein Restart-Automat pro Arbeitszyklus höchstens einmal das Band verändern darf, ist diese Trennung auch sinnvoll und impliziert die Disjunktheit der beiden Teilmengen (und die Erreichbarkeit jedes einzelnen Zustandes).

Definition 3.10 Für jeden det-RR(1)-Automaten mit Startzustand s_0 und Überführungsfunktion δ kann man die Menge der erreichbaren Zustände als disjunkte Vereinigung der folgenden beiden Mengen darstellen:

- Die Menge der nach dem (Neu-)Start ohne Rewrite-Schritte erreichbaren Zustände ist definiert als $S = \bigcup_{i=0}^{\infty} S_i$ mit
  $S_0 = \{s_0\}$ und $S_{i+1} = \{s_y \mid s_x \in S_i,\ a \in A,\ \delta(s_x, a) = (s_y, \mathsf{MVR})\}$.

- $S' = \bigcup_{i=0}^{\infty} S'_i$ sei die Menge der Zustände des Nachlesevorgangs, wenn gilt:
  $S'_0 = \{s'_y \mid s_x \in S,\ a \in A,\ \delta(s_x, a) = (s'_y, \lambda)\}$ und
  $S'_{i+1} = \{s'_y \mid s'_x \in S'_i,\ a \in A,\ \delta(s'_x, a) = (s'_y, \mathsf{MVR})\}$.

Mit Hilfe dieser Darstellung wird sich die Konstruktion im nachfolgenden Beweis recht übersichtlich angeben lassen.

Satz 3.11 *Eine Sprache wird genau dann von einem det-RR(1)-Automaten erkannt, wenn sie regulär ist.*

Beweis Wir konstruieren zu einem beliebigen det-RR(1)-Automaten $\mathcal{M} = \langle S \cup S', A, A, \rhd, \lhd, s_0, 1, \delta\rangle$ einen NFA

$$\mathcal{M}' = \left\langle (S \cup S' \cup \{s_a\}) \times 2^{S'}, A, (s_0, \{\}), F, \delta' \right\rangle,$$

dessen Funktionsweise der folgenden Idee entspringt:

- In der ersten Komponente des den Zustand bildenden Tupels werden zunächst die MVR-Schritte von $\mathcal{M}$ nachgestellt.
- Falls ein Accept-Schritt auftritt, wechselt die erste Komponente des Zustandstupels von $\mathcal{M}'$ in den Sonderzustand s_a.
- Tritt ein Rewrite-Schritt (d. h. das Löschen des aktuellen Zeichens) auf, so wird $\mathcal{M}$ entsprechend in den Nachlesevorgang übergehen und – falls dieser in einem Restart-Schritt endet – das hinter dem gelöschten Zeichen liegende Zeichen in demselben Zustand erreichen, wie ehemals das gelöschte Zeichen selbst.

Da der NFA $\mathcal{M}'$ jedes Zeichen nur einmal passieren kann, wird er ab dem ersten Rewrite-Schritt den darauf folgenden Nachlesevorgang, sowie das Weiterlesen nach dem Neustart, parallel simulieren müssen. $\mathcal{M}'$ kann jedoch nicht davon ausgehen, dass der erwartete Restart-Schritt wirklich auftritt, denn $\mathcal{M}$ könnte stattdessen im Nachlesen akzeptieren oder verwerfen. Bei jedem Rewrite-Schritt muss $\mathcal{M}'$ also raten, welcher dieser beiden Fälle eintritt:

- Im Fall, dass $\mathcal{M}$ erfolgreich nachliest und (evtl. erst am rechten Bandbegrenzungszeichen) neu startet, muss die erste Komponente im Zustandstupel von $\mathcal{M}'$ beibehalten und die zweite Komponente um den Startzustand des entsprechenden Nachlesevorgangs erweitert werden.

- Falls $\mathcal{M}$ keinen weiteren Neustart mehr durchführt, darf $\mathcal{M}'$ das Verhalten nach dem Neustart nicht weiter zu simulieren versuchen. Die erste Komponente des Zustandstupels wird hierbei durch den Startzustand des Nachlesevorgangs ersetzt, von dem auf dem Restwort geprüft wird, ob der erwartete Accept-Schritt auch wirklich eintritt.

In jedem dieser Fälle dient die zweite Komponente auch dazu, die jeweils noch ausstehende Überprüfung der erwarteten Restart-Schritte fertig stellen zu können. Zu diesem Zweck sei dies durch eine Menge N, in der die noch nicht in einen Restart-Schritt übergegangenen Nachlese-Zustände gesammelt sind, realisiert. Somit ist es möglich, alle anfallenden Nachlesevorgänge gleichzeitig zu überprüfen:

Beim Lesen eines Zeichens $x \in A$ führt $\mathcal{M}'$ die Verarbeitung der (Menge N in der) zweiten Komponente wie folgt durch:

(a) Jeder Zustand aus N, der auf x einen MVR-Schritt bewirkt, wird durch seinen entsprechenden Nachfolgezustand ersetzt. Insbesondere ist hierbei möglich, dass zwei verschiedene Zustände denselben Nachfolgezustand besitzen.

(b) Jeder Zustand aus N, der auf x einen Restart-Schritt bewirkt, wird aus der Menge entfernt.

(c) Wenn es in N einen Zustand gibt, für den x einen (unerwarteten) Accept-Schritt bewirkt oder es einen Zustand gibt, zu dem für das aktuelle Zeichen x kein Übergang definiert ist, so schlägt die Probe fehl und $\mathcal{M}'$ verwirft die Berechnung.

Um die Überführungsfunktion δ' explizit angeben zu können, definieren wir für ein beliebiges Zeichen $x \in A$ und eine beliebige Menge $N \in 2^{S'}$ die Menge der nachfolgenden Zustände

$$N_x = \{\overline{s}' \mid \exists s' \in N : \ \delta(s', x) = (\overline{s}', \mathsf{MVR})\},$$

entsprechend der Verarbeitungsvorschriften (a) und (b).

Um den Einfluss von Verarbeitungsvorschrift (c) auf fast alle im folgenden anzugebenden Überführungen effizienter angeben zu können, definieren wir zusätzlich das Prädikat P wie folgt:

$$P \iff \forall s' \in N : \ \big(\delta(s', x) = \mathsf{Restart} \ \vee \ \delta(s', x) = (\overline{s}', \mathsf{MVR}) \text{ für ein } \overline{s}' \in S'\big).$$

Erst nach der Aktualisierung der zweiten Komponente des Zustandstupels von $\mathcal{M}'$ werden für dasselbe Zeichen in Bezug auf die erste Komponente die anfangs beschriebenen Veränderungen ausgeführt. Damit haben wir nun alle Ideen, die der Definition von δ' zu Grunde liegen, beschrieben:

Für ein beliebiges Zeichen $x \in A$, eine beliebige Menge $N \in 2^{S'}$, sowie alle Zustände $s \in S$, $s' \in S'$ ist:

1. $\delta'((s, N), x) = \{(\overline{s}, N_x)\}$, wenn $\delta(s, x) = (\overline{s}, \mathsf{MVR})$ und P gelten,
2. $\delta'((s, N), x) = \{(s_a, N_x)\}$, wenn $\delta(s, x) = \mathsf{Accept}$ und P gelten,
3. $\delta'((s, N), x) = \{(s, N_x \cup \{\overline{s}'\}), (\overline{s}', N_x)\}$, wenn $\delta(s, x) = (\overline{s}', \lambda)$ und P gelten,
4. $\delta'((s', N), x) = \{(\overline{s}', N_x)\}$, wenn $\delta(s', x) = (\overline{s}', \mathsf{MVR})$ und P gelten,
5. $\delta'((s', N), x) = \{(s_a, N_x)\}$, wenn $\delta(s', x) = \mathsf{Accept}$ und P gelten,
6. $\delta'((s_a, N), x) = \{(s_a, N_x)\}$, wenn P gilt und
7. δ' undefiniert in allen anderen Fällen.

Dass $\mathcal{M}'$ auch die möglichen Übergänge von $\mathcal{M}$ beim Erreichen des rechten Bandbegrenzungszeichen simulieren kann, wird durch die folgende Festlegung der Endzustände erreicht:

- Es ist $(s_a, N) \in F$ gdw. $\forall s' \in N : \ \delta(s', \lhd) = \mathsf{Restart}$ und
- für $s \in S \cup S'$ ist $(s, N) \in F$ gdw. $\delta(s, \lhd) = \mathsf{Accept}$ und $\forall s' \in N : \delta(s', \lhd) = \mathsf{Restart}$.

Damit ist $\mathcal{M}'$ vollständig definiert. Der Beweis von Korrektheit und Vollständigkeit unserer Konstruktion ergibt sich direkt aus der zu Grunde liegenden Idee. Darüber hinaus findet sich ein detaillierterer Beweis zur generativen Mächtigkeit von det-RR(1)-Automaten im Anschluss an Satz 3.21, mit dessen Hilfe deren Beschreibungseffizienz gegenüber deterministischen, endlichen Automaten präzisiert werden kann. □

Über die Zahl der Zustände eines äquivalenten NFA lässt sich nun die folgende Aussage treffen:

Korollar 3.12 *Zu jedem det-RR(1)-Automaten $\mathcal{M}$ mit $n = |S| + |S'|$ Zuständen (und $|S| > 0$) gibt es einen NFA, der die Sprache $L(\mathcal{M})$ mit höchstens $(2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1}$ Zuständen erkennt, was der asymptotischen Komplexitätsklasse $O(n \cdot 2^n)$ entspricht.*

Beweis Durch die Konstruktion der Zustandsmenge von $\mathcal{M}'$ aus dem vorigen Satz ergibt sich zunächst die obere Schranke $(|S| + |S'| + 1) \cdot 2^{|S'|}$.

Jedoch stellen dabei alle Tupel (s', N), bei denen $s' \in N$ ist, unproduktive Zustände dar. Verfeinert man die Konstruktionsidee dahingehend, dass sie

über δ' nicht erreicht werden können, so kann man ihre Anzahl $|S'| \cdot 2^{|S'|-1}$ von der im Rahmen des vorigen Beweises erreichten Zustandszahl abziehen und erhält

$$(|S| + |S'| + 1) \cdot 2^{|S'|} - |S'| \cdot 2^{|S'|-1} = (2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1}$$

durch $\mathcal{M}'$ annehmbare (produktive) Zustände.

Ferner ist

$$(2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1} = 2 \cdot (|S| + |S'| + 1) \cdot 2^{|S'|-1} = (n+1) \cdot 2^{|S'|} \in O(n \cdot 2^n).$$

□

Untere Schranken

Die obige Grenze ist sogar scharf, wie wir im Folgenden mit Hilfe einer speziellen Folge von Sprachen über einem Alphabet der Größe sieben nachweisen werden. Um zu zeigen, dass det-RR(1)-Automaten auch schon über einem binären Alphabet eine effiziente Sprachbeschreibung liefern können, geben wir zunächst aber ein „kleineres Beispiel“ an:

Beispiel 3.13 *Zu jedem $n \in \mathbb{N}$ gibt es eine Sprache L_n über einem binären Alphabet, so dass es keinen NFA gibt, der L_n mit weniger als 2^n Zuständen erkennt – jedoch kann L_n von einem det-RR(1)-Automaten mit n Zuständen erkannt werden:*

Holzer und Kutrib bewiesen in [11], dass es keinen NFA in weniger als 2^n Zuständen geben kann, welcher die Sprache

$$\{a, b\}^* \setminus \left(\{a, b\}^* a \{a, b\}^{n-1} b \{a, b\}^*\right)$$

akzeptiert.

Ein det-RR(1)-Automat schafft dieses aber in recht einfacher Arbeitsweise in n Zuständen:

- *er läuft im Grundzustand s_0 von links nach rechts über das Wort, überliest dabei alle Zeichen b (und $\triangleright$) und löscht das erste auftretende a.*
- *Im Nachlesevorgang prüft er mit Hilfe der Zustände $s'_1, \ldots, s'_{n-1}$, ob nach $n-1$ weiteren Zeichen ein b folgt.*
- *Falls dies der Fall ist, verwirft der Automat.*
- *Falls das Zeichen nicht b ist (a oder $\triangleleft$), startet der Automat neu.*

- *Falls die Stelle nicht erreicht werden kann, weil das Restwort zu kurz ist, akzeptiert der Automat.*
- *Ferner akzeptiert der Automat alle Wörter über $\{b\}$ (dies gelingt ebenfalls mit Hilfe von s_0).*

Für diesen det-RR(1)-Automaten reichen also n Zustände aus, womit schon für binäre Alphabete ein exponentieller Zustands-Trade-off gegenüber NFAs nachgewiesen ist.

Dass der Trade-off auch echt höher sein kann, zeigen wir nun:

Satz 3.14 *Für jedes $n \geq 2$ gibt es eine Sprache über einem siebenbuchstabigen Alphabet, welche von einem det-RR(1)-Automaten mit $n = |S \cup S'|$ Zuständen, jedoch von keinem NFA mit weniger als $(2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1}$ Zuständen, erkannt werden kann.*

Beweis Für ein beliebiges $0 < m < n$ seien $S = \{s_0, \ldots, s_{m-1}\}$ und $S' = \{s'_0, \ldots, s'_{n-m-1}\}$ die beiden Teilmengen der Zustandsmenge.

Wir erstellen den det-RR(1)-Automaten $\mathcal{M} = \langle S \cup S', A, A, \rhd, \lhd, s_0, 1, \delta \rangle$ über einem siebenbuchstabigen Alphabet ($A = \{a, \ldots, g\}$), um die Definition der Überführungsfunktion so übersichtlich wie möglich halten zu können. Für $i \in \{0, \ldots, m-1\}$, $j \in \{1, \ldots, m-1\}$, $k \in \{0, \ldots, n-m-1\}$ und $l \in \{1, \ldots, n-m-1\}$ sei:

(a) Das Zeichen a bewirkt eine zyklische MVR-Bewegung auf den Zuständen von S, während alle Zustände des Nachlesevorgangs durch das Lesen von a in sich selbst überführt werden:

$$\delta(s_i, a) = (s_{(i+1) \bmod m}, \mathsf{MVR}) \text{ und } \delta(s'_k, a) = (s'_k, \mathsf{MVR}).$$

(b) Wenn $\mathcal{M}$ im Zustand s_0 auf das Zeichen b trifft, so akzeptiert er. In allen anderen Fällen behält $\mathcal{M}$ den Zustand bei und führt einen MVR-Schritt aus:

$$\delta(s_0, b) = \mathsf{Accept},\ \delta(s_j, b) = (s_j, \mathsf{MVR}) \text{ und } \delta(s'_k, b) = (s'_k, \mathsf{MVR}).$$

(c) Für das Zeichen c ist zu keinem Zustand ein Übergang definiert. Dies ist also ein Zeichen, welches in jedem akzeptierten Wort, in dem es vorkommt, von $\mathcal{M}$ nicht erreicht werden darf:

$$\delta(s_i, c) \text{ und } \delta(s'_k, c) \text{ sind undefiniert.}$$

(d) Das Zeichen d wird im Zustand s_0 gelöscht und bewirkt, dass $\mathcal{M}$ den Nachlesevorgang im Zustand s'_0 beginnt. Alle anderen Zustände werden – wie gehabt – überlaufen:

$$\delta(s_0, d) = (s'_0, \lambda),\ \delta(s_j, d) = (s_j, \mathsf{MVR}) \text{ und } \delta(s'_k, d) = (s'_k, \mathsf{MVR}).$$

(e) Das Zeichen e bewirkt – analog zu (a) – eine zyklische MVR-Bewegung auf den Zuständen des Nachlesevorgangs, während alle anderen Zustände in sich selbst überführt werden:

$$\delta(s'_k, e) = (s'_{(k+1) \bmod (n-m)}, \mathsf{MVR}) \text{ und } \delta(s_i, e) = (s_i, \mathsf{MVR}).$$

(f) Wenn $\mathcal{M}$ im Zustand s'_0 auf das Zeichen f trifft, so akzeptiert er – analog zu (b). In allen anderen Fällen behält $\mathcal{M}$ den Zustand bei und führt einen MVR-Schritt aus:

$$\delta(s'_0, f) = \mathsf{Accept},\ \delta(s'_l, f) = (s'_l, \mathsf{MVR}) \text{ und } \delta(s_i, f) = (s_i, \mathsf{MVR}).$$

(g) Das Zeichen g beendet für den Zustand s'_0 den Nachlesevorgang. Alle anderen Zustände werden – wie gehabt – überlaufen:

$$\delta(s'_0, g) = \mathsf{Restart},\ \delta(s'_l, g) = (s'_l, \mathsf{MVR}) \text{ und } \delta(s_i, g) = (s_i, \mathsf{MVR}).$$

Nach der Konstruktion aus Satz 3.11 ergibt sich ein NFA, dessen Zustände eine der drei folgenden Formen annehmen:

1) $(s_i, \{s'_{j_1}, \ldots, s'_{j_k}\})$ mit $i \in \{0, \ldots, m-1\}$, $k \in \{0, \ldots, n-m-1\}$ und $j_1 < \cdots < j_k \in \{0, \ldots, n-m-1\}$.
Es gibt $|S| \cdot 2^{|S'|}$ Zustände dieser Form.

2) $(s'_{j_{k+1}}, \{s'_{j_1}, \ldots, s'_{j_k}\})$ mit $k \in \{0, \ldots, n-m-2\}$, $j_1 < \cdots < j_k \in \{0, \ldots, n-m-1\}$ und $j_{k+1} \in \{0, \ldots, n-m-1\} \setminus \{j_1, \ldots, j_k\}$.
Es gibt $|S'| \cdot 2^{|S'|-1}$ Zustände dieser Form.

3) $(s_a, \{s'_{j_1}, \ldots, s'_{j_k}\})$ mit $k \in \{0, \ldots, n-m-1\}$ und $j_1 < \cdots < j_k \in \{0, \ldots, n-m-1\}$.
Es gibt $2^{|S'|}$ Zustände dieser Form.

Wir zeigen nun im Rahmen der Fooling Set Konstruktion, dass alle diese Zustände erreicht werden (wir beziehen die Präfixe $v^{(1)}$ auf die Zustände der Form 1), usw.) und paarweise nicht äquivalent sind (über entsprechende Suffixe $w^{(1)}$, usw.):

1. Ein Zustand der Form 1) ist erreichbar durch das Wort

$$v^{(1)} = de^{j_k - j_{k-1}} de^{j_{k-1} - j_{k-2}} \cdots de^{j_2 - j_1} de^{j_1} a^i,$$

bzw. a^i für $k = 0$,

2. ein Zustand der Form 2) ist erreichbar durch das Wort

$$v^{(2)} = de^{j_k - j_{k-1}} de^{j_{k-1} - j_{k-2}} \cdots de^{j_2 - j_1} e^{(j_1 - j_{k+1}) \,\mathrm{MOD}(n-m)} de^{j_{k+1}},$$

   bzw. $de^{j_{k+1}}$ für $k = 0$ und

3. ein Zustand der Form 3) ist erreichbar durch das Wort

$$v^{(3)} = de^{j_k - j_{k-1}} de^{j_{k-1} - j_{k-2}} \cdots de^{j_2 - j_1} de^{j_1} b,$$

   bzw. b für $k = 0$.

Da es in $\mathcal{M}$ keinerlei Übergänge am rechten Bandbegrenzungszeichen gibt, ist im Rahmen unserer Konstruktion der Zustand $(s_a, \{\})$ der Form 3) der einzige Endzustand von $\mathcal{M}'$. Im Rahmen der ersten geforderten Fooling Set Eigenschaft verwenden wir zum Erreichen dieses Endzustandes die folgenden Suffixe:

1. Aus einem Zustand der Form 1) wird über das Wort

$$w^{(1)} = e^{n-m-j_k} g e^{j_k - j_{k-1}} g \cdots e^{j_2 - j_1} g a^{m-i} b,$$

   bzw. $a^{m-i}b$ für $k = 0$ der Endzustand erreicht.

2. Aus einem Zustand der Form 2) wird über das Wort

$$w^{(2)} = e^{n-m-j_{k+1}} f e^{(j_{k+1} - j_k) \,\mathrm{MOD}(n-m)} g e^{j_k - j_{k-1}} g \cdots e^{j_2 - j_1} g,$$

   bzw. $e^{n-m-j_{k+1}} f$ für $k = 0$ und

3. aus einem Zustand der Form 3) über das Wort

$$w^{(3)} = e^{n-m-j_k} g e^{j_k - j_{k-1}} g \cdots e^{j_2 - j_1} g c,$$

   bzw. c für $k = 0$ akzeptiert (das Suffix c stellt die zweite Fooling Set Eigenschaft sicher, wie wir gleich sehen werden).

Der Nachweis der zweiten Fooling Set Eigenschaft erfolgt durch Veranschaulichung der Arbeitsweise auf den unterschiedlichen Wortepaaren:

- Für zwei unterschiedliche Zustände x und y der Form 3), welchen die Paare $(v_x^{(3)}, w_x^{(3)})$ bzw. $(v_y^{(3)}, w_y^{(3)})$ zugeordnet sind, sei o. B. d. A. $s'_i \in S'$ ein Zustand, welcher in der zweiten Komponente von x enthalten und in der von y nicht enthalten ist.
  Wenn im Zustand x das Wort $w_y^{(3)}$ verarbeitet wird, so werden nacheinander alle auch in der zweiten Komponente des Zustands y enthaltenen Zustände $s' \in S'$ jeweils in den Zustand s'_{n-m-1} überführt und durch den anschließenden Restart-Schritt aus der Menge entfernt. Auf den Zustand s'_i trifft dies jedoch nicht zu, womit die zweite Komponente des Zustandes, in welchem $\mathcal{M}$ auf das letzte Zeichen c trifft, nicht leer ist. Damit wird das Wort $v_x^{(3)} w_y^{(3)}$ stets verworfen.

- Für einen Zustand x der Form 3) und einen Zustand y der Form 1) oder 2) gilt:
  Im Zustand y wird beim Verarbeiten des Wortes $w_x^{(3)}$ der jeweilige Zustand aus der ersten Komponente nicht mehr nach s_a überführt. Damit ist der entsprechende Übergang zum Zeichen c niemals definiert. Das Wort $v_y^{(1 \text{ bzw. } 2)} w_x^{(3)}$ wird also nicht akzeptiert.

- Für zwei unterschiedliche Zustände x und y der Form 2) können zwei mögliche Fälle vorliegen:

  Fall 1 Die erste Komponente von x und y unterscheidet sich:
  Dann gibt es sowohl aus dem Zustand x für das Wort $w_y^{(2)}$, als auch aus dem Zustand y für das Wort $w_x^{(2)}$ keinen akzeptierenden Übergang. Die erste Komponente der jeweils am Ende der Abarbeitung erreichten Zustände wird also stets ungleich s_a sein. Damit werden $v_x^{(2)} w_y^{(2)}$ und $v_y^{(2)} w_x^{(2)}$ stets verworfen.

  Fall 2 Es gibt o. B. d. A. einen Zustand $s_i' \in S'$, der in der zweiten Komponente von x enthalten und in der von y nicht enthalten ist:
  Wenn also im Zustand x das Wort $w_y^{(2)}$ verarbeitet wird, so erfolgt – wie in Fall 1 nachgewiesen – aus dem aus s_i' hervorgehenden Zustand kein Neustart und die zweite Komponente des Zustandes, in dem sich $\mathcal{M}$ am Ende des Wortes befindet, ist nicht leer. Damit wird das Wort $v_x^{(2)} w_y^{(2)}$ stets verworfen.

- Für zwei unterschiedliche Zustände x und y der Form 1) ergeben sich dieselben zwei Fälle und wir können analog folgern, dass stets wenigstens eines der Wörter $v_x^{(1)} w_y^{(1)}$ und $v_y^{(1)} w_x^{(1)}$ nicht akzeptiert wird.

- Für einen Zustand x der Form 2) und einen Zustand y der Form 1) gilt:
  Im Zustand y liegt in der ersten Komponente ein Zustand ungleich s_a vor. Dieser kann durch das Verarbeiten des Wortes $w_x^{(2)}$ nicht erreicht werden. Das Wort $v_y^{(1)} w_x^{(2)}$ wird also nicht akzeptiert.

Dieser Nachweis der zweiten Fooling Set Eigenschaft schließt den Beweis. Der Fooling Set umfasst damit

$$|S| \cdot 2^{|S'|} + |S'| \cdot 2^{|S'|-1} + 2^{|S'|} = (2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1}$$

Paare, was die obere Schranke des Trade-offs exakt trifft. □

Bemerkung 3.15 *Die Frage nach der kleinstmöglichen Alphabetgröße, so dass sich eine Sprachfamilie mit obigem Zustands-trade-off findet, wurde hier (wie auch im vorigen Kapitel) nicht gestellt. Vielmehr haben wir aus*

Gründen der transparenteren Darstellung gar nicht erst versucht, die Alphabetgröße zu komprimieren.

Obwohl im Rahmen dieser Arbeit nicht weiter darauf eingegangen wird, erscheint auch an dieser Stelle die Frage nach der zur vollen Entfaltung der Beschreibungsmächtigkeit erforderlichen Größe der Parameter durchaus als interessant.

Auf obigen Beweis lässt sich auch die zusätzliche Folgerung stützen, dass R(1)-Automaten in der Beschreibung dieser det-RR(1)-Sprachfamilie gegenüber nichtdeterministischen, endlichen Automaten keinerlei Vorteile zu bieten haben:

Korollar 3.16 *Für jedes $n \in \mathbb{N}$ gibt es eine Sprache über einem siebenbuchstabigen Alphabet, welche von einem det-RR(1)-Automaten mit $n = |S \cup S'|$ Zuständen erkannt wird. Es gibt jedoch keinen R(1)-Automaten mit weniger als $(2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1}$ Zuständen, der diese Sprache erkennt.*

Beweis Wir können uns auf die Zustände konzentrieren, die in einem – wie auch immer gearteten – R(1)-Automaten $\mathcal{M}''$ von den Wörtern $u^{(1)}$ bzw. $u^{(2)}$ bzw. $u^{(3)}$ des vorhergehenden Beweises erreicht werden. Wenn es uns zu zeigen gelingt, dass $\mathcal{M}''$ auf diesen Wörtern kein einziges Zeichen löschen darf, ohne die Unkorrektheitserhaltung zu verletzen, dann erreichen diese Wörter genau die $(2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1}$ Zustände, die im vorigen Beweis als paarweise nicht äquivalent nachgewiesen wurden.

Dazu betrachten wir die Wörter innerhalb der drei Typen in aufsteigender Reihenfolge ihrer Länge:

1. Wir beginnen mit den Wörtern der Länge 1, die vom Typ 1 sind. Dies können d und a sein. Wird dieses Zeichen durch $\mathcal{M}''$ gelöscht, so kann man leicht Suffixe w_d und w_a finden, so dass die Unkorrektheitserhaltung des Wortes dw_d bzw. aw_a verletzt wird (z. B. $w_d = bc$ und $w_a = bc$).

   Im Folgenden lässt sich stets induktiv voraussetzen, dass von der ersten bis zur vorletzten Stelle des betrachteten Wortes bereits gezeigt wurde, dass $\mathcal{M}''$ dort kein Zeichen löschen darf. Der Grund dafür ist, dass alle Präfixe eines Wortes vom Typ 1) selbst wieder vom Typ 1) sind. Wir müssen also nur noch zeigen, dass auch das jeweils letzte Zeichen nicht gelöscht werden darf:

   - Wenn das letzte Zeichen ein a ist und von $\mathcal{M}''$ gelöscht wird, so wird das (nach obigem Beweis) unkorrekte Wort $v^{(1)}w_f^{(1)}$ mit $w_f^{(1)} = e^{m-j_k}g \ldots e^{j_2-j_1}ga^{m-i+1}b$ doch akzeptiert, was die Unkorrektheitserhaltung verletzt.

- Analoge Suffixe findet man auch in den Fällen, in denen das zu löschende Zeichen ein d oder ein e ist.

2. Alle Präfixe von Wörtern des Typs 2) sind entweder selbst vom Typ 2) oder vom Typ 1) ($a^i = \lambda$). So brauchen wir zu jedem dieser Wörter nur die Buchstaben des zweiten Wortteils zu betrachten, die durch den vorigen Fall nicht abgedeckt sind. Der zweite Wortteil hat die Form $e^{(j_1 - j_{k+1}) \,\mathrm{MOD}(n-m)} d e^{j_{k+1}}$, wie man dem vorhergehenden Beweis entnehmen kann.

   Analog zu Obigem lässt sich zeigen, dass weder das d noch eines der e gelöscht werden darf, ohne dass die Unkorrektheitserhaltung verletzt wird.

3. Alle echten Präfixe von Wörtern des Typs 3) sind vom Typ 1), weshalb man hier nur die mögliche Löschung des letzten Zeichens (also b) diskutieren muss:

   Hierzu gibt es keine Suffixe, die gewährleisten, dass das (Gesamt-)Wort ohne das gelöschte Zeichen akzeptiert und mit ihm verworfen wird. Jedoch können nach der zweiten Fooling Set Eigenschaft aus dem vorigen Beweis die dort angegebenen Restwörter $w^{(3)}$ aus keinem der bisher erreichten Zustände akzeptiert werden. Dies bedeutet, dass $\mathcal{M}''$ auch diese Zustände annehmen muss und durch das Löschen des Zeichens b keine Einsparungen erreichen kann.

Wir haben damit gezeigt, dass auch $\mathcal{M}''$ mindestens $(2 \cdot |S| + |S'| + 2) \cdot 2^{|S'|-1}$ paarweise nicht äquivalente Zustände erreichen muss. □

Zum Ende dieses Abschnitts sei noch darauf verwiesen, dass R(1)- und det-RR(1)-Automaten die einzigen, bekannten Restart-Automatentypen sind, die nicht mehr als nur die regulären Sprachen erkennen.

Von den naheliegenden Erweiterungen des Modells seien hier beispielhaft einige erwähnt:

- In [27] wird nachgewiesen, dass es einen RR(1)-Automaten zur Erkennung einer zu $\{a^{2^n} \mid n \in \mathbb{N}\}$ ähnlichen Sprache (aus [15]) gibt.

- Ferner findet sich in ersterer Arbeit auch der Nachweis, dass die von der Grammatik $G = \langle \{S\}, \{a, \overline{a}\}, S, P \rangle$ mit

  $$P = \{(S \to aS\overline{a}), (S \to SS), (S \to \lambda)\}$$

  erzeugte Dyck-Sprache in $\mathcal{L}$(det-(mon-)R(2)) liegt.

- Auch die Hinzunahme von MVL-Schritten zum System der det-RR(1)-Automaten (die kanonische Bezeichnung wäre det-RL(1)) bringt eine Steigerung der Erkennungsmächtigkeit, da sie das Erkennen der Sprache $L = \{a^n b^m \mid n \in \mathbb{N} \wedge (m = n \vee m = n - 1)\}$ ermöglicht. Die Begründung hierzu ist:

  Ein erkennender Automat kann hierbei erst das Eingabewort auf die Form a^*b^* hin prüfen und dabei Modulo 2 mitzählen, ob die Anzahl der a gleich der der b ist. Wenn ja, wird ein b, wenn nein, ein a gelöscht. Auf diese Weise werden stets alternierend ein a und ein b gelöscht und der Automat akzeptiert, wenn das Wort auf a oder ab reduziert werden konnte.

Das Ausweiten der Betrachtung allein auf diese Klassen stellt jedoch schon eine erhebliche Erschwerung dar, da von ihnen derzeit nicht einmal bekannt ist, ob die Regularität ihrer Elemente entscheidbar ist. Aus diesem Grund fehlt allen diesbezüglichen Fragestellungen – wie etwa der Frage, wieviele Zustände ein DFA zur Beschreibung einer regulären, von einem minimalen det-RL(1)-Automaten in n Zuständen akzeptierten Sprache höchstens haben muss – zumindest die gewöhnliche Basis.

3.3 Keine Kompensation des Nichtdeterminismus

Wie wir im vorigen Abschnitt sehen konnten, stellt die Fähigeit zum Löschen und Nachlesen eine in manchen Fällen sehr ausdruckseffiziente Erweiterung der deterministischen, endlichen Automaten dar. Da die Höhe des gefundenen Trade-offs den gewohnten, exponentiellen Trade-off zwischen NFA und DFA sogar noch übersteigt, könnte man erwarten, dass die Ressource Nichtdeterminismus zumindest teilweise kompensiert werden kann.

Interessanterweise ist dies jedoch keineswegs der Fall. Es gibt Sprachen, bei deren Erkennung der kleinstmögliche det-RR(1)-Automat nicht kleiner ist, als ein entsprechender (partieller) DFA. Die Beweisidee dazu lautet demnach: finde eine Sprache, auf der ein det-RR(1)-Automat keinerlei Vorteile aus der Verwendung von Rewrite-Schritten ziehen kann.

Satz 3.17 *Zu jedem $n \in \mathbb{N}$ gibt es eine Sprache über einem ternären Alphabet, welche von einem NFA mit n Zuständen erkannt wird, so dass es keinen det-RR(1)-Automaten gibt, der diese Sprache mit weniger, als $2^n - 1$ Zuständen erkennt.*

Beweis Zu einem beliebigen n sei der folgende NFA gegeben:

$$\mathcal{M} = \langle \{s_0, \ldots, s_{n-1}\}, \{a, b, c\}, s_0, \{s_0\}, \delta \rangle \,,$$

wobei δ genau an den folgenden Stellen definiert ist:

$$\begin{aligned} \delta(s_i, a) &= \{s_0, s_i\} && \text{für } i \in \{1, \ldots, n-1\}, \\ \delta(s_j, b) &= \{s_{(j+1) \bmod n}\} && \text{für } j \in \{0, \ldots, n-1\} \text{ und} \\ \delta(s_0, c) &= \{s_0\}. \end{aligned}$$

Der zu Grunde liegende, auf das Alphabet $\{a, b\}$ eingeschränkte NFA entstammt aus [38], wo gezeigt wurde, dass jeder äquivalente (total definierte) DFA mindestens 2^n Zustände hat.

Mit dem zuletzt aufgeführten Übergang für das Zeichen c hat die Sprache $L(\mathcal{M})$ nun zusätzlich die folgende Eigenschaft:

Behauptung *Für zwei beliebige Wörter $v \in L(\mathcal{M})$ und $w \in \{a, b, c\}^*$ liegt das Wort vcw genau dann in $L(\mathcal{M})$, wenn $w \in L(\mathcal{M})$ ist.*

Diese Eigenschaft ergibt sich daraus, dass $\mathcal{M}$ durch die Verarbeitung des Teilwortes v stets auch den einzigen Endzustand s_0 erreichen muss. Das danach zu lesende Zeichen c belässt $\mathcal{M}$ in diesem Zustand. Da s_0 auch Startzustand ist, werden somit wenigstens die Worte vcw erkannt, für die $w \in L(\mathcal{M})$ ist.

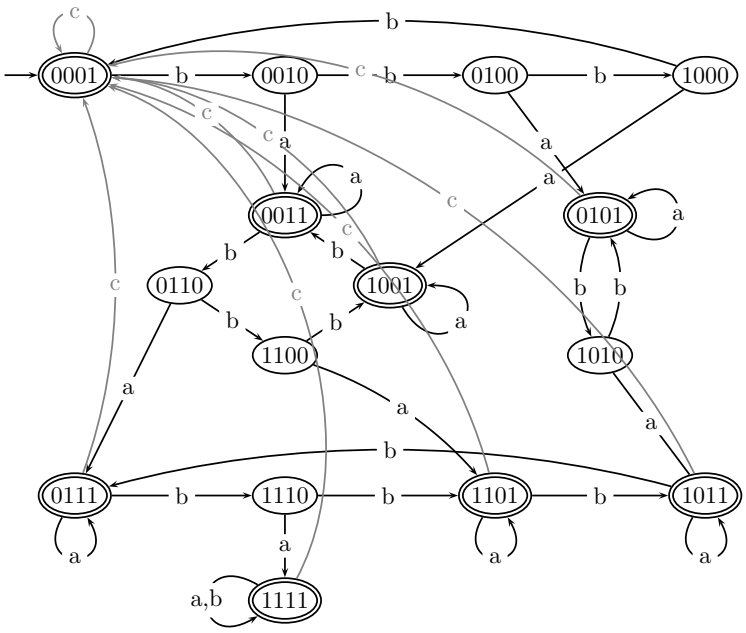

Abbildung 4: der NFA $\mathcal{M}$ für $n = 4$

Beim Lesen des Zeichens c werden jedoch alle anderen durch die Verarbeitung von v erreichbaren Zustände auf einen nicht definierten Übergang führen. Damit ergibt sich, dass keine Wörter vcw erkannt werden in denen $w \notin L(\mathcal{M})$ ist.

Damit haben wir nun die nötigen Voraussetzungen, um zeigen zu können, dass ein zustandsoptimaler det-RR(1)-Automat genau die $2^n - 1$ produktiven Zustände des Potenzmengen-DFAs für $L(\mathcal{M})$ besitzen muss. In einer zu Korollar 3.16 analogen Weise betrachten wir eine Menge von $2^n - 1$ Wörtern, auf denen ein entsprechender DFA ebensoviele, paarweise verschiedene, produktive Zustände annehmen kann:

Im Rahmen der Potenzmengenkonstruktion ergibt sich, dass die Menge der von $\mathcal{M}$ nach der Verarbeitung des Wortes

$$u = b^{i_j - i_{j-1}} a \cdots b^{i_2 - i_1} a b^{i_1} \qquad \text{bzw.} \qquad u = b^{i_1} \text{ für } j = 1$$

annehmbaren Zustände gleich $\{s_{i_1}, \ldots, s_{i_j}\}$, mit $j \in \{1, \ldots, n\}$, $i_1 < \cdots < i_j$ und $i_1, \ldots, i_j \in \{0, \ldots, n-1\}$, ist.

Wiederum sind sämtliche Präfixe eines Wortes u der obigen Form ebenfalls von der Form $b^{i_j - i_{j-1}} a \cdots b^{i_2 - i_1} a b^{i_1}$, weshalb wir wieder induktiv über die Wortlänge argumentieren können.

Im Vergleich zur Vorgehensweise aus Korollar 3.16 ergibt sich ab hier jedoch eine Abweichung: zwar ist jeder $L(\mathcal{M})$ erkennende det-RR(1)-Automat $\mathcal{M}'$ nun auch deterministisch und wir können damit auch die Korrektheitserhaltung voraussetzen, jedoch beschreiben beide Erhaltungssätze nur die Fälle, in denen $\mathcal{M}'$ nach dem Löschen neu startet. Wir müssen also auch die Möglichkeit betrachten, dass $\mathcal{M}'$ noch im Nachlesevorgang akzeptiert oder verwirft. Mit Hilfe der oben bewiesenen Behauptung können wir dieses Verhalten jedoch ausschließen, wie man im Folgenden sieht:

- Das kürzeste, echte Präfix der obigen Form ist b. Dieses Zeichen kann von $\mathcal{M}'$ nicht gelöscht werden, ohne die Korrektheitserhaltung zu verletzen. Als Beleg hierfür dient das Wort $bvcw \in L(\mathcal{M})$ mit $v = a$ und $w \in L(\mathcal{M})$, dessen Suffix vcw nicht in $L(\mathcal{M})$ liegt.

  Falls $\mathcal{M}'$ das Zeichen doch löscht, darf nun kein Restart-Schritt mehr erfolgen. Das Suffix acw muss nun mit Hilfe von deterministischen MVR-Schritten korrekt eingeordnet werden. Dazu aber werden mindestens $2^n + 1$ Nachlese-Zustände benötigt, da es nach [38] keinen DFA geben kann, der das Teilwort w in einer geringeren Anzahl von Zuständen erkennen kann. Also lässt sich durch das Löschen an dieser Stelle nachweislich keine Zustandseinsparung erzielen – das Gegenteil ist der Fall.

- Von jedem Wort der Form $b^{i_j - i_{j-1}} a \cdots b^{i_2 - i_1} a b^{i_1}$ (mit $i_1 \neq 0$) mit längstem, echten Präfix $u = b^{i_j - i_{j-1}} a \cdots b^{i_2 - i_1} a b^{i_1 - 1}$ sei nun vorausgesetzt, dass es keinen ($L(\mathcal{M})$ erkennenden) det-RR(1)-Automaten mit weniger als $2^n - 1$ Zuständen geben kann, der bei der Verarbeitung von u ein Zeichen löscht.

  - Falls $j = 1$ ist, liegt für $v = b^{n - i_1 + 1} a$ das Wort $ubvcw$ in $L(\mathcal{M})$ und das Wort $uvcw$ nicht. Letzteres gilt, da $\mathcal{M}$ beim Lesen des Zeichens c nicht im Zustand s_0 sein kann.

    Damit die Korrektheitserhaltung nicht verletzt wird, muss $\mathcal{M}'$ das Restwort vcw noch im Nachlesen prüfen, was wiederum mehr als $2^n - 1$ Zustände benötigen würde.

  - Falls $j > 1$ ist, kann man aus $i_1 \neq 0$ folgern, dass es stets ein $k \in \{0, \ldots, n-1\}$ gibt, welches einen Zustand s_k bezeichnet, der nicht in der erreichten Zustandsmenge (aus der Potenzmengenkonstruktion) liegt, aber einen Nachbarzustand $s_{(k-1) \bmod n}$ hat, der in der Menge enthalten ist.

    Damit ist für $v = b^{n-k} + 1$ das Wort $ubvcw \in L(\mathcal{M})$ und das Wort $uvcw$ nicht, da der Abstand k dafür sorgt, dass das Zeichen c von $\mathcal{M}$ nur in einem Zustand gelesen werden kann, der nicht s_0 ist, womit $\mathcal{M}'$ das Wort stets verwirft.

- Wenn ein Wort der Form $b^{i_j - i_{j-1}} a \cdots b^{i_2} a$, also mit $i_1 = 0$, vorliegt, können wir ebenfalls voraussetzen, dass alle in seinen echten Präfixen vorkommenden Zeichen schon analog diskutiert wurden. Ferner können wir, weil wir das leere Wort nicht zu betrachten brauchen, davon ausgehen, dass $j > 1$ ist und unser Wort damit auf ein a endet. Ferner ist der Zustand s_0 in der Menge der von $\mathcal{M}$ durch das Lesen von $u = b^{i_j - i_{j-1}} a \cdots b^{i_2}$ erreichbaren Zustände nicht enthalten.

  Deshalb gilt mit $v = cw$ stets, dass $uavcw \in L(\mathcal{M})$ ist und das Wort $uvcw$ nicht.

Damit haben wir gezeigt, dass jeder det-RR(1)-Automat zur Erkennung von $L(\mathcal{M})$ mindestens $2^n - 1$ Zustände benötigt, da er gegenüber einem vergleichbaren DFA keine Vorteile aus den Möglichkeiten des Löschens und Nachlesens ziehen kann. □

Obwohl sich R(1)-Automaten in manchen Fällen als wesentlich effizienter als nichtdeterministische endliche Automaten erwiesen haben, liefert der obige Satz gleichzeitig auch die optimale untere Schranke für den Vergleich zu det-RR(1)-Automaten:

Korollar 3.18 *Zu jedem $n \in \mathbb{N}$ gibt es eine Sprache über einem ternären Alphabet, welche von einem R(1)-Automaten mit n Zuständen erkannt wird, so dass es keinen det-RR(1)-Automaten gibt, der diese Sprache in weniger, als $2^n - 1$ Zuständen erkennt. Ferner gibt es zu jedem R(1)-Automaten mit n Zuständen einen äquivalenten det-RR(1)-Automaten mit höchstens $2^n - 1$ Zuständen.*

Beweis Die untere Schranke ist dieselbe, da sich jeder R(1)-Automat einer bestimmter Zustandszahl wie der oben angegebene NFA verhalten kann. Die obere Schranke ergibt sich wiederum durch die Konstruktion aus 3.1 – ein det-RR(1)-Automat kann gegenüber dem total definierten DFA stets den unproduktiven Zustand und denjenigen zum Fertiglesen des akzeptierten Wortes einsparen. □

3.4 Deterministische endliche und Restart-Automaten

Der letzte Abschnitt dieses Kapitels soll sich nun mit dem noch ausstehenden Vergleich der Beschreibungseffizienz von det-RR(1)- und deterministischen endlichen Automaten auseinander setzen.

Zwar lässt sich nach Korollar 3.12 und Satz 3.14 ein Paar aus oberer und unterer Schranke ableiten, doch diese liegen sehr weit auseinander; nämlich bei $2^{O(n \cdot 2^n)}$ bzw. $\Omega(n \cdot 2^n)$. Die genaue Lage (und eine möglichst scharfe Beschreibung) der Grenze ist damit nach wie vor von Interesse.

Bei der Simulation und damit geeigneten Beschreibung von det-RR(1)-Automaten haben sich die im Folgenden definierten alternierenden endlichen Automaten gegenüber den NFAs als viel präziser herausgestellt und wir werden uns in diesem Kapitel ihre Arbeitsweise als Grundanschauung heranziehen.

Die Historie der alternierenden endlichen Automaten geht zurück auf [6], wobei das Grundkonzept des Alternierens erstmalig in [3] in Zusammenhang mit Automaten gebracht wurde. Die von uns verwendete Definition ist an die von Salomaa und Yu in [39] und [40] angegebene angelehnt.

Definition 3.19 Ein *alternierender endlicher Automat* (AFA) ist ein System $\langle S, A, F, s_0, \delta \rangle$, wobei

- S eine endliche, nichtleere Menge von Zuständen,
- A eine endliche, nichtleere Menge von Eingabezeichen,
- $F \subseteq S$ eine Menge von Endzuständen,
- s_0 der Startzustand und
- δ eine Überführungsfunktion von S nach $\left(A \times \mathbb{B}^S \to \mathbb{B}\right)$ ist.

Für jeden Zustand $s \in S$ ist $\delta(s)$ also eine Funktion von $A \times \mathbb{B}^S$ nach $\mathbb{B}$. Somit impliziert δ also zu jedem Zustand $s \in S$ und Zeichen $a \in A$ eine $|S|$-stellige Schaltfunktion $\delta(s)(a)$. Auf diese Schaltfunktionen wollen wir unsere Anschauung bezüglich der Arbeitsweise von alternierenden endlichen Automaten stützen.

Zur Definition der von einem AFA erkannten Sprache benötigen wir nun noch die folgenden Konstrukte:

a) Zu beliebigen $s \in S$ und booleschem Vektor $z \in \mathbb{B}^S$ schreiben wir die Projektion auf die s-te Komponente von z als: $z(s)$.

b) Die Funktion $\delta_S : \ A \times \mathbb{B}^S \to \mathbb{B}^S$ sei für $z, z' \in \mathbb{B}^S$ und $a \in A$ festgelegt durch:

$$\delta_S(a, z) = z' \ \Leftrightarrow \ \forall s \in S : \ \delta(s)(a)(z) = z'(s).$$

c) Die erweiterte Überführungsfunktion $\Delta : A^* \times \mathbb{B}^S \to \mathbb{B}^S$ sei für $z \in \mathbb{B}^S$, $a \in A$ und $w \in A^*$ rekursiv festgelegt durch:

- $\Delta(\lambda, z) = z$ und
- $\Delta(aw, z) = \delta_S(a, \Delta(w, z))$.

d) Der charakteristische Vektor $f \in \mathbb{B}^S$ von F sei für $s \in S$ festgelegt durch:

$$f(s) = \begin{cases} 1, & \text{falls } s \in F \\ 0, & \text{sonst} \end{cases}.$$

Definition 3.20 Die von einem alternierenden endlichen Automaten $\mathcal{M}$ erkannte Sprache ist $L(\mathcal{M}) = \{w \in A^* \mid \Delta(w, f)(s_0) = 1\}$.

Nun können wir mit Hilfe der disjunkten Zerlegung der Zustandsmenge nach Definition 3.10 einen relativ effizienten, äquivalenten AFA konstruieren. Der Einblick in seine Struktur wird wesentlich zur Bestimmung oberer und unterer Schranken für den Zustands-trade-off zwischen det-RR(1)- und deterministischen endlichen Automaten beitragen.

Satz 3.21 *Zu jedem det-RR(1)-Automaten $\mathcal{M} = \langle S \cup S', A, A, \rhd, \lhd, s_0, 1, \delta \rangle$ gibt es einen AFA $\mathcal{M}'$, der zur Erkennung von $L(\mathcal{M})$ nicht mehr als $|S|+2|S'|$ Zustände benötigt.*

Beweis Wir konstruieren $\mathcal{M}'$ als $\mathcal{M}' = \langle S \cup S' \cup \hat{S}, A, F, s_0, \delta' \rangle$, wobei $\hat{S} = \{\hat{s} \mid s' \in S'\}$ jeweils eine markierte Kopie der Zustände aus S' enthält. Zunächst aber betrachten wir die dabei zu Grunde liegende Idee:

Da jedes Zeichen nur einmal gelesen werden kann, müssen nach jeder Löschung eines Zeichens zwei mögliche Abfolgen der weiteren Verarbeitungen parallel simuliert werden:

1. Zum Einen kann die Verarbeitung im nächsten Zyklus mit dem nachfolgenden Zeichen und dem vor dem Rewrite-Schritt anliegenden Zustand fortfahren (*Fortsetzung*). Dies bedingt jedoch, dass das Nachlesen mit einem Restart-Schritt endet (*Restart*).

2. Zum Anderen kann es sein, dass kein Restart-Schritt mehr erfolgt und $\mathcal{M}$ noch im Nachlesen akzeptiert oder verwirft (*Ende*).

Die Verknüpfung dieser beiden Fälle lässt sich als Schaltfunktion durch den Term

$$Fortsetzung \wedge Restart \vee Ende$$

darstellen. Dies wird sich im Wesentlichen bei der Simulation der Rewrite-Schritte durch die Überführungsfunktion von $\mathcal{M}'$ verwenden lassen.

Zur Trennung des Zweiges *Ende* von dem, den wir *Restart* genannt haben, verwenden wir die kopierten Zustände $\hat{S}$ (im Unterschied zu S', welches *Restart* beschreibt), da beide Zweige unterschiedliche Überprüfungen des Nachlesevorgangs darstellen sollen.

Da im Rahmen der Abarbeitung des $Fortsetzung$-Zweiges weitere Rewrite-Schritte auftreten können, wird die Struktur der parallelen Abarbeitungen mit der Zeit die Form

$$\big((Fortsetzung \wedge Restart_m \vee Ende_m) \wedge \ldots\big) \wedge Restart_1 \vee Ende_1$$

annehmen. Die innersten Ausdrücke stellen hierbei die in der Verarbeitung durch $\mathcal{M}$ als letztes erreichten Nachlese-Vorgänge dar und man kann sehen, dass ihre Auswertung nur dann eine Rolle spielt, wenn jeder frühere Vorgang $Restart_k$ (für $1 \leq k < m$) erfolgreich endet.

Nach diesen Vorüberlegungen definieren wir die Überführungsfunktion δ' für beliebiges $z \in \mathbb{B}^S$ komponentenweise. Zunächst sollen für $s_x, s_y \in S$ bzw. $s'_x, s'_y \in S'$ und $a \in A$ die MVR-Schritte von $\mathcal{M}$ wie folgt nachgestellt werden:

$$\begin{array}{lll} \delta'(s_x)(a)(z) = z(s_y) & \text{gdw.} & \delta(s_x, a) = (s_y, \mathsf{MVR}), \\ \delta'(s'_x)(a)(z) = z(s'_y) & \text{und} & \\ \delta'(\hat{s}_x)(a)(z) = z(\hat{s}_y) & \text{gdw.} & \delta(s'_x, a) = (s'_y, \mathsf{MVR}). \end{array}$$

Entsprechend der Idee der parallelen Simulation der seriellen Ausführung der Zyklen von $\mathcal{M}$ übersetzen wir (für $s \in S$, $s' \in S'$ und $a \in A$) den Rewrite-Schritt zu:

$$\delta'(s)(a)(z) = z(s) \wedge z(s') \vee z(\hat{s}) \quad \text{gdw.} \quad \delta(s, a) = (s', \lambda),$$

wobei zu dieser Idee gehört, dass das Ende des Zyklus wie folgt ausgewertet wird:

$$\begin{array}{lll} \delta'(s)(a)(z) = 1 & \text{gdw.} & \delta(s, a) = \mathsf{Accept}, \\ \delta'(s')(a)(z) = 1 & \text{gdw.} & \delta(s', a) = \mathsf{Restart} \text{ und} \\ \delta'(\hat{s})(a)(z) = 1 & \text{gdw.} & \delta(s', a) = \mathsf{Accept}. \end{array}$$

Dass die Umsetzung dieser Übergänge auch am Wortende – genauer: am rechten Bandbegrenzungszeichen – gelingt, kann auch für den AFA $\mathcal{M}'$ wiederum durch die entsprechende Festlegung der Endzustände sicher gestellt werden. Gemäß Obigem gelte für $s \in S$ und $s' \in S'$:

$$\begin{array}{lll} s \in F & \text{gdw.} & \delta(s, \lhd) = \mathsf{Accept}, \\ s' \in F & \text{gdw.} & \delta(s', \lhd) = \mathsf{Restart} \text{ und} \\ \hat{s} \in F & \text{gdw.} & \delta(s', \lhd) = \mathsf{Accept}. \end{array}$$

Wir müssen nun zeigen, dass durch diese Konstruktion wirklich die Gleichheit von $L(\mathcal{M}')$ und $L(\mathcal{M})$ gilt. Dazu betrachten wir zunächst die Verarbeitung von Wörtern, auf denen $\mathcal{M}$ keinen weiteren Arbeitszyklus erreichen wird. Dabei gelte stets $u = u_1 \ldots u_l \in A^+$ und $v \in A^*$:

Fall 1a Das Wort uv wird beim Erreichen des letzten Zeichens von u akzeptiert:
Dann gilt für $\mathcal{M}'$:

$$\begin{aligned}\Delta'(uv,f)(s_0) &= \Delta'(u_1 \dots u_l v, f)(s_0)\\ &= \delta'_S\left(u_1, \delta'_S(u_2, \dots \delta'_S(u_l, \Delta'(v,f))\dots\right)(s_0)\\ &= \delta'(s_0)(u_1)\left(\delta'_S(u_2, \dots \delta'_S(u_l, \Delta'(v,f))\dots\right)\\ &= \delta'_S\left(u_2, \dots \delta'_S(u_l, \Delta'(v,f))\dots\right)(s_{i_2}),\end{aligned}$$

wenn $\delta(s_0, u_1) = (s_{i_2}, \mathsf{MVR})$ ist, was wir nach der Voraussetzung annehmen können.
Analog gibt es ein $s_{i_l} \in S$, so dass die weitere Entwicklung auf die folgende Form führt:

$$\begin{aligned}\Delta'(uv,f)(s_0) &= \delta'_S(u_l, \Delta'(v,f))(s_{i_l})\\ &= \delta'(s_{i_l})(u_l)(\Delta'(v,f)) = 1,\end{aligned}$$

da $\delta(s_{i_l}, u_l) = \mathsf{Accept}$ ist.
Somit wird uv auch von $\mathcal{M}'$ akzeptiert.

Fall 1b Das Wort u wird beim Erreichen des rechten Bandbegrenzungszeichens akzeptiert:

$$\begin{aligned}\Delta'(u,f)(s_0) &= \Delta'(u_1 \dots u_l, f)(s_0) = \dots\\ &= \delta'_S(u_l, \Delta'(\lambda,f))(s_{i_l})\\ &= \delta'(s_{i_l})(u_l)(\Delta'(\lambda,f)) = \Delta'(\lambda,f)(s_{i_{l+1}})\\ &= f(s_{i_{l+1}}) = 1,\end{aligned}$$

da wegen $\delta(s_{i_{l+1}}, \lhd) = \mathsf{Accept}$ auch $s_{i_{l+1}} \in F$ ist.
Somit wird u auch von $\mathcal{M}'$ akzeptiert.

Fall 2a Das Wort uv wird erkannt, indem $\mathcal{M}$ das k-te Zeichen von u löscht und im anschließenden Nachlesen auf dem letzten Zeichen von u akzeptiert:

$$\begin{aligned}\Delta'(uv,f)(s_0) &= \dots = \delta'_S\left(u_k, \dots \delta'_S(u_l, \Delta'(v,f))\dots\right)(s_{i_k})\\ &= \delta'(s_{i_k})(u_k)\left(\delta'_S(u_{k+1}, \dots)\right)\\ &= \delta'_S(u_{k+1}, \dots)(s_{i_k})\\ &\quad \wedge \delta'_S(u_{k+1}, \dots)(s'_{i_{k+1}}) \vee \delta'_S(u_{k+1}, \dots)(\hat{s}_{i_{k+1}}),\end{aligned}$$

wenn $\delta(s_{i_k}, u_k) = (s'_{i_{k+1}}, \lambda)$ ist.
Die im Zuge des Nachlesens erfolgenden MVR-Schritte sehen in der

Entwicklung von $\Delta'(uv, f)(s_0)$ stets wie folgt aus:

$$\begin{aligned}
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \wedge \delta'_S(u_{k+1}, \dots)(s'_{i_{k+1}}) \vee \delta'_S(u_{k+1}, \dots)(\hat{s}_{i_{k+1}}) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \\
&\quad \wedge \delta'(s'_{i_{k+1}})(u_{k+1})(\delta'_S(u_{k+2}, \dots) \vee \delta'(\hat{s}_{i_{k+1}})(u_{k+1})(\delta'_S(u_{k+2}, \dots) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \wedge \delta'_S(u_{k+2}, \dots)(s'_{i_{k+2}}) \vee \delta'_S(u_{k+2}, \dots)(\hat{s}_{i_{k+2}}),
\end{aligned}$$

wenn $\delta(s'_{i_{k+1}}, u_{k+1}) = (s'_{i_{k+2}}, \mathsf{MVR})$ ist.
In der weiteren Ableitung ergibt sich:

$$\begin{aligned}
\Delta'(uv, f)(s_0) = \cdots &= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \\
&\quad \wedge \delta'_S(u_l, \Delta'(v, f))(s'_{i_l}) \vee \delta'_S(u_l, \Delta'(v, f))(\hat{s}_{i_l}) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \\
&\quad \wedge \delta'(s'_{i_l})(u_l)(\Delta'(v, f)) \vee \delta'(\hat{s}_{i_l})(u_l)(\Delta'(v, f)) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \wedge 0 \vee 1 = 1,
\end{aligned}$$

da $\delta(s'_{i_l}, u_l) = \mathsf{Accept}$ ist (dann ist $\delta(s'_{i_l}, u_l) = \mathsf{Restart}$ nicht möglich, da $\mathcal{M}$ deterministisch ist).
Somit wird uv auch von $\mathcal{M}'$ akzeptiert.

Fall 2b Das Wort u wird erkannt, indem $\mathcal{M}$ das k-te Zeichen von u löscht und im anschließenden Nachlesen auf dem rechten Bandbegrenzungszeichen akzeptiert:
Analog zu den obigen Fällen gilt:

$$\begin{aligned}
\Delta'(uv, f)(s_0) = \cdots &= \delta'_S(u_{k+1}, \dots \delta'_S(u_l, \Delta'(\lambda, f)) \dots)(s_{i_k}) \\
&\quad \wedge \delta'_S(u_l, \Delta'(\lambda, f))(s'_{i_l}) \vee \delta'_S(u_l, \Delta'(\lambda, f))(\hat{s}_{i_l}) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \\
&\quad \wedge \delta'(s'_{i_l})(u_l)(\Delta'(\lambda, f)) \vee \delta'(\hat{s}_{i_l})(u_l)(\Delta'(\lambda, f)) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \wedge \Delta'(\lambda, f)(s'_{i_{l+1}}) \vee \Delta'(\lambda, f)(\hat{s}_{i_{l+1}}) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \wedge f(s'_{i_{l+1}}) \vee f(\hat{s}_{i_{l+1}}) \\
&= \delta'_S(u_{k+1}, \dots)(s_{i_k}) \wedge 0 \vee 1 = 1,
\end{aligned}$$

da mit $\delta(s'_{i_{l+1}}, \triangleleft) = \mathsf{Accept}$ auch $\hat{s}_{i_{l+1}} \in F$ und $s'_{i_{l+1}} \notin F$ gilt.
Somit wird u auch von $\mathcal{M}'$ akzeptiert.

Das Verwerfen eines Wortes im Falle eines undefinierten Übergangs lässt sich ebenfalls in obige vier Fälle unterteilen, wenn kein weiterer Arbeitszyklus erreicht werden darf. Der Beweis, dass $\mathcal{M}'$ genau dieselben Wörter verwirft, erfolgt analog.

Als letztes bleibt zu zeigen, wie die Abarbeitung von Wörtern, auf denen $\mathcal{M}$ mehrere Arbeitszyklen durchlaufen muss, über eine (Un-)Korrektheitserhaltung im Sinne von 2.6 auf eine der oben beschriebenen zurückgeführt werden kann:

Fall 3a Ein Arbeitszyklus von $\mathcal{M}$ auf dem Wort uv bestehe darin, dass das erste gelöschte Zeichen das k-te Zeichen von u ist und der dazugehörige Restart-Schritt auf dem letzten Zeichen von u erfolgt.
Dann gilt für $\mathcal{M}'$:

$$\begin{aligned}
\Delta'(uv,f)(s_0) &= \cdots = \delta'_S\left(u_k, \ldots \delta'_S(u_l, \Delta'(v,f))\ldots\right)(s_{i_k}) \\
&= \cdots = \delta'_S(u_{k+1},\ldots)(s_{i_k}) \\
&\qquad \wedge \delta'_S(u_{k+1},\ldots)(s'_{i_{k+1}}) \vee \delta'_S(u_{k+1},\ldots)(\hat{s}_{i_{k+1}}) \\
&= \cdots = \delta'_S(u_{k+1},\ldots)(s_{i_k}) \\
&\qquad \wedge \delta'_S(u_l, \Delta'(v,f))(s'_{i_l}) \vee \delta'_S(u_l, \Delta'(v,f))(\hat{s}_{i_l}) \\
&= \delta'_S(u_{k+1},\ldots)(s_{i_k}) \\
&\quad \wedge \delta'(s'_{i_l})(u_l)(\Delta'(v,f)) \vee \delta'(\hat{s}_{i_l})(u_l)(\Delta'(v,f)) \\
&= \delta'_S(u_{k+1},\ldots)(s_{i_k}) \wedge 1 \vee 0 = \delta'_S(u_{k+1},\ldots)(s_{i_k}),
\end{aligned}$$

wenn $\delta(s'_{i_l}, u_l) = \mathsf{Restart}$ ist.
Wegen $\delta(s_{i_{k-1}}, u_{k-1}) = (s_{i_k}, \mathsf{MVR})$ gilt jedoch im Folgenden:

$$\begin{aligned}
\Delta'(uv,f)(s_0) &= \delta'(s_{i_{k-1}})(u_{k-1})\left(\delta'_S(u_{k+1},\ldots\delta'_S(u_l,\Delta'(v,f))\ldots)\right) \\
&= \delta'_S\left(u_{k-1},\ldots\delta'_S(u_{k+1},\ldots\delta'_S(u_l,\Delta'(v,f))\ldots)\right)(s_{i_{k-1}}) \\
&= \cdots = \delta'_S\left(u_1,\ldots\delta'_S(u_{k-1},\delta'_S(u_{k+1},\ldots))\ldots\right)(s_0) \\
&= \Delta'(u_1\ldots u_{k-1}u_{k+1}\ldots u_l v, f)(s_0).
\end{aligned}$$

Somit wird uv von $\mathcal{M}'$ genau dann akzeptiert, wenn das um das gelöschte Zeichen u_k verkürzte Wort $u_1\ldots u_{k-1}u_{k+1}\ldots u_l v$ von $\mathcal{M}'$ akzeptiert wird.

Fall 3b Ein Arbeitszyklus von $\mathcal{M}$ auf dem Wort u bestehe darin, dass das erste gelöschte Zeichen das k-te Zeichen von u ist und der dazugehörige Restart-Schritt auf dem rechten Bandbegrenzungszeichen erfolgt.

Dann gilt für $\mathcal{M}'$:

$$\begin{aligned}
\Delta'(u,f)(s_0) &= \cdots = \delta'_S\left(u_k, \ldots \delta'_S(u_l, \Delta'(\lambda, f)) \ldots\right)(s_{i_k}) \\
&= \cdots = \delta'_S(u_{k+1}, \ldots)(s_{i_k}) \\
&\qquad \wedge \delta'_S(u_{k+1}, \ldots)(s'_{i_{k+1}}) \vee \delta'_S(u_{k+1}, \ldots)(\hat{s}_{i_{k+1}}) \\
&= \cdots = \delta'_S(u_{k+1}, \ldots)(s_{i_k}) \\
&\qquad \wedge \delta'_S(u_l, \Delta'(\lambda, f))(s'_{i_l}) \vee \delta'_S(u_l, \Delta'(\lambda, f))(\hat{s}_{i_l}) \\
&= \delta'_S(u_{k+1}, \ldots)(s_{i_k}) \\
&\quad \wedge \delta'(s'_{i_l})(u_l)(\Delta'(\lambda, f)) \vee \delta'(\hat{s}_{i_l})(u_l)(\Delta'(\lambda, f)) \\
&= \cdots = \delta'_S(u_{k+1}, \ldots)(s_{i_k}) \wedge f(s'_{i_{l+1}}) \vee f(\hat{s}_{i_{l+1}}) \\
&= \delta'_S(u_{k+1}, \ldots)(s_{i_k}) \wedge 1 \vee 0 = \delta'_S(u_{k+1}, \ldots)(s_{i_k}),
\end{aligned}$$

weil $\delta(s'_{i_{l+1}}, \lhd) = \mathsf{Restart}$ ist.
Wegen $\delta(s_{i_{k-1}}, u_{k-1}) = (s_{i_k}, \mathsf{MVR})$ gilt jedoch analog zu Obigem:

$$\Delta'(u,f)(s_0) = \cdots = \Delta'(u_1 \ldots u_{k-1} u_{k+1} \ldots u_l, f)(s_0).$$

Somit wird u von $\mathcal{M}'$ genau dann akzeptiert, wenn das verkürzte Wort $u_1 \ldots u_{k-1} u_{k+1} \ldots u_l$ von $\mathcal{M}'$ akzeptiert wird.

Damit ist die Gleichheit $L(\mathcal{M}') = L(\mathcal{M})$ bewiesen. □

Im Folgenden wird uns der obige Beweis wichtiger sein als die bewiesene Aussage, wenn wir uns die Implikationen, welche sich aus der Beschreibung von δ' ergeben, genauer anschauen.

Obere Schranke

Wenn wir die Auswertung des Terms $\Delta'(u,f)(s_0)$ derart durchführen, dass in allen vorkommenden Termen stets dasselbe, noch nicht verarbeitete Restwort vorliegt, kehren wir damit zurück zu der am Anfang des Beweises angegebenen Vorstellung der parallelen Verarbeitung. Befinden sich alle Verarbeitungszweige auf demselben Anfang des noch nicht gelesenen Restwortes, so entspricht dies einem erreichbaren Gesamtzustand des Systems. Dies bietet uns die gewünschte Vergleichsmöglichkeit zum Modell des DFA und wir werden unsere Betrachtung in dieser Richtung fortführen:

Um alle auftretenden Verarbeitungszweige parallel weiterverfolgen zu können, müsste eigentlich vorausgesetzt sein, dass alle betreffenden Übergänge definiert sind. Wenn dies jedoch bei einer Auswertung nicht der Fall sein sollte, wird der enstehende boolesche Gesamtterm entsprechend nur ein Teil des im Folgenden ermittelten Ausdrucks darstellen. Wir werden aber sehen, dass genau diese Eigenschaft schon für unsere Zwecke ausreicht.

Generell bezeichnen wir mit u_{k_j} die jeweils gelöschten Zeichen aus u und mit $i_{j,l}$ die der Menge S (für $j = 0$) bzw. S' entsprechenden Indizes für die vorkommenden Zustände.

Es gilt:

$$\begin{aligned}\Delta'(u,f)(s_0) &= \cdots = \delta'_S(u_{k_1}, \dots \Delta'(\lambda, f) \dots)(s_{i_{0,1}}) \\ &= \delta'_S(u_{k_1+1}, \dots)(s_{i_{0,1}}) \\ &\quad \wedge \delta'_S(u_{k_1+1}, \dots)(s'_{i_{1,0}}) \vee \delta'_S(u_{k_1+1}, \dots)(\hat{s}_{i_{1,0}}).\end{aligned}$$

Unter der oben begründeten Annahme, dass alle erforderlichen Übergänge definiert sind, werden alle diese Terme parallel weiter entwickelt.

$$\begin{aligned}\dots &= \delta'_S(u_{k_2}, \dots)(s_{i_{0,2}}) \wedge \delta'_S(u_{k_2}, \dots)(s'_{i_{1,1}}) \vee \delta'_S(u_{k_2}, \dots)(\hat{s}_{i_{1,1}}) \\ &= \Big(\delta'_S(u_{k_2+1}, \dots)(s_{i_{0,2}}) \wedge \delta'_S(u_{k_2+1}, \dots)(s'_{i_{2,1}}) \vee \delta'_S(u_{k_2+1}, \dots)(\hat{s}_{i_{2,1}})\Big) \\ &\quad \wedge \delta'_S(u_{k_2+1}, \dots)(s'_{i_{1,2}}) \vee \delta'_S(u_{k_2+1}, \dots)(\hat{s}_{i_{1,2}})\end{aligned}$$

Nach m Löschvorgängen ergibt sich:

$$\begin{aligned}\dots = \Big(\dots \big(&\delta'_S(u_{k_m+1}, \dots)(s_{i_{0,m}}) \\ &\wedge \delta'_S(u_{k_m+1}, \dots)(s'_{i_{m,m-1}}) \vee \delta'_S(u_{k_m+1}, \dots)(\hat{s}_{i_{m,m-1}})\big) \\ &\wedge \delta'_S(u_{k_m+1}, \dots)(s'_{i_{m-1,m}}) \vee \delta'_S(u_{k_m+1}, \dots)(\hat{s}_{i_{m-1,m}})\big) \\ &\quad \vdots \\ &\wedge \delta'_S(u_{k_m+1}, \dots)(s'_{i_{1,m}}) \vee \delta'_S(u_{k_m+1}, \dots)(\hat{s}_{i_{1,m}}).\end{aligned}$$

Da es jeweils höchstens $|S'|$ verschiedene Werte $i_{j,l}$ mit $j \neq 0$ gibt, ist $\Delta'(u,f)(s_0)$ stets ein boolescher Ausdruck in höchstens $1 + 2|S'|$ Variablen.

Um zu zeigen, dass die Auswahl an paarweise ungleichwertigen Termen weitaus eingeschränkter ist, abstrahieren wir für ein festes m vom Ausdruck $\delta'_S(u_{k_m+1}, \dots)(s'_{i_{j,m}})$ zur booleschen Variablen q'_j (ebenso schreiben wir $\hat{q}_j$ für $\delta'_S(u_{k_m+1}, \dots)(\hat{s}_{i_{j,m}})$ und q anstatt von $\delta'_S(u_{k_m+1}, \dots)(s'_{i_{0,m}})$).

Damit ergibt sich der folgende Sachverhalt:

Lemma 3.22 *Falls die beiden booleschen Variablen q'_j und q'_l bzw. $\hat{q}_j$ und $\hat{q}_l$ übereinstimmen, sind die Terme*

$$\big(\dots (q \wedge q'_m \vee \hat{q}_m) \wedge \cdots \wedge q'_{j+1} \vee \hat{q}_{j+1}\big) \wedge q'_j \vee \hat{q}_j) \wedge \cdots \wedge q'_l \vee \hat{q}_l) \wedge \cdots \wedge q'_1 \vee \hat{q}_1$$

und

$$\big(\dots (q \wedge q'_m \vee \hat{q}_m) \wedge \cdots \wedge q'_{j+1} \vee \hat{q}_{j+1}\big) \wedge q'_{j-1} \vee \hat{q}_{j-1}) \wedge \cdots \wedge q'_l \vee \hat{q}_l) \wedge \cdots \wedge q'_1 \vee \hat{q}_1$$

äquivalent (für beliebige Positionen $1 \leq j < l \leq m$).

Beweis Die Gleichheit der zwei Wertepaare ergibt nur vier verschiedene Fälle, die sich mühelos nachprüfen lassen. Wir sollten durch diese Abstraktion jedoch nicht die Bedeutung in Bezug auf die Arbeitsweise des AFA $\mathcal{M}'$ aus den Augen verlieren:

Fall 1 Es seien $q'_j = q'_l = 0$ und $\hat{q}_j = \hat{q}_l = 0$.
Das Einsetzen von 0 an der Stellen q'_l und $\hat{q}_l$ absorbiert in beiden Termen sämtliche vorderen Stellen. Es ergibt sich jeweils der vereinfachte Term:

$$(\dots(\hat{q}_{l-1} \wedge q'_{l-2} \vee \hat{q}_{l-2}) \wedge \dots \wedge q'_1 \vee \hat{q}_1.$$

Dies entspricht in Bezug auf den simulierten det-RR(1)-Automaten $\mathcal{M}$ einem undefinierten Übergang während des Nachlesevorgangs, womit die beiden Terme

$$\delta'_S(u_{k_m+1}, \dots)(s'_{i_{j,m}}) = \delta'(s'_{i_{j,m}})(u_{k_m+1})(\dots)$$

und

$$\delta'_S(u_{k_m+1}, \dots)(\hat{s}_{i_{j,m}}) = \delta'(\hat{s}_{i_{j,m}})(u_{k_m+1})(\dots)$$

unabhängig vom Restwort den Wert 0 annehmen.

Fall 2 Für $q'_j = q'_l = 1$ und $\hat{q}_j = \hat{q}_l = 0$ ergibt sich nach den Gesetzen der booleschen Algebra, dass der Gesamtterm mit Ausnahme der Stelle j bzw. im ersten Term auch l erhalten bleibt. Der vereinfachte Term lautet in beiden Fällen:

$$\big(\dots(q \wedge q'_m \vee \hat{q}_m) \wedge \dots \wedge q'_{l+1} \vee \hat{q}_{l+1}) \wedge q'_{l-1} \vee \hat{q}_{l-1}) \wedge \dots \wedge q'_1 \vee \hat{q}_1.$$

Dies entspricht einem Restart-Übergang von $\mathcal{M}$. Das Nachlesen wird beendet und damit ist ein Akzeptieren im Nachlesevorgang nicht mehr möglich.

Fall 3 Für $q'_j = q'_l = 0$ und $\hat{q}_j = \hat{q}_l = 1$ gilt:
Durch $q'_l = 0$ bleiben vom Gesamtterm jeweils nur die rechts von der Stelle l liegenden Terme erhalten. Der vereinfachte Term lautet in beiden Fällen:

$$(\dots(q'_{l-1} \wedge \hat{q}_{l-1} \wedge q'_{l-2} \vee \hat{q}_{l-2}) \wedge \dots \wedge q'_1 \vee \hat{q}_1.$$

Dies entspricht einem Accept-Übergang von $\mathcal{M}$ während des Nachlesevorgangs (wodurch der nächste Arbeitszyklus nicht mehr erreicht werden kann).

Fall 4 Als letztes seien $q'_j = q'_l = 1$ und $\hat{q}_j = \hat{q}_l = 1$.
Das Einsetzen von 1 an der Stellen q'_l und $\hat{q}_l$ absorbiert wiederum in

beiden Termen sämtliche vorderen Stellen und es ergibt sich dasselbe Ergebnis wie in Fall 3:

$$(\dots(q'_{l-1} \wedge \hat{q}_{l-1} \wedge q'_{l-2} \vee \hat{q}_{l-2}) \wedge \dots \wedge q'_1 \vee \hat{q}_1.$$

Anders als in den vorigen Fällen gibt es hier keine Entsprechung zur Verarbeitungsweise von $\mathcal{M}$, da es nach Konstruktion von $\mathcal{M}'$ und dem Determinismus von $\mathcal{M}$ keinen Fall gibt, in dem sowohl

$$\delta'_S(u_{k_m+1}, \dots)(s'_{i_{j,m}}) = \delta'(s'_{i_{j,m}})(u_{k_m+1})(\dots)$$

als auch

$$\delta'_S(u_{k_m+1}, \dots)(\hat{s}_{i_{j,m}}) = \delta'(\hat{s}_{i_{j,m}})(u_{k_m+1})(\dots)$$

den Wert 1 annehmen können.
Dieser Fall ist lediglich im Rahmen unserer Abstraktion entstanden.

Da sich in allen vier Fällen jeweils äquivalente Gesamtterme ergeben, ist die Behauptung bewiesen. □

Mit Hilfe dieses Lemmas wird es nun recht einfach sein, eine obere Schranke für den Zustands-trade-off gegenüber deterministischen endlichen Automaten anzugeben.

Korollar 3.23 *Der AFA $\mathcal{M}'$ aus Satz 3.21 impliziert einen DFA mit höchstens $O(|S \cup S'|!)$ Zuständen, welcher die Sprache $L(\mathcal{M})$ erkennt.*

Beweis Bei der Auswertung von $\Delta'(u, f)(s_0)$ ergibt sich zu jedem Wort u ein boolescher Ausdruck, der genau dem von u erreichten Zustand in einem DFA entspricht. Wir zeigen nun, dass die Zahl der paarweise inäquivalenten Ausdrücke bzw. Zustände wie behauptet nach oben beschränkt ist, indem wir zunächst eine einheitliche Entwicklungsvorschrift angeben:

1. Es werden alle auftretenden Teilterme gleichzeitig um eine Überführung (unter Verarbeitung genau eines Zeichens) weiterentwickelt.
2. Nach jedem Schritt wird zuerst geprüft, ob der Gesamtterm nach Lemma 3.22 verkürzt werden kann, und dies gegebenenfalls durchgeführt.
3. Ebenso wird nach jedem Schritt, in dem Teilterme in 0 oder 1 übergehen, eine Vereinfachung des Gesamtausdruckes nach den Regeln der booleschen Algebra durchgeführt.

Der so entstandene Term für $\Delta'(u, f)(s_0)$ hat dann nach der Verarbeitung des letzten Zeichens (und der Ersetzung von $\Delta'(\lambda, f)$ durch f) stets genau eine der drei folgenden Formen:

- $(\ldots(f(s_{i_0}) \wedge f(s'_{i_m}) \vee f(\hat{s}_{i_m})) \wedge \cdots \wedge f(s'_{i_1}) \vee f(\hat{s}_{i_1})$,
- $(\ldots(f(s'_{i_m}) \vee f(\hat{s}_{i_m})) \wedge \cdots \wedge f(s'_{i_1}) \vee f(\hat{s}_{i_1})$ oder
- $(\ldots(f(\hat{s}_{i_m})) \wedge \cdots \wedge f(s'_{i_1}) \vee f(\hat{s}_{i_1})$,

wobei nach Lemma 3.22 für die Gesamtlänge nur die Werte $0 \leq m \leq |S'|$ in Frage kommen. Somit ergeben sich höchstens

$$(|S|+2) \cdot \sum_{m=0}^{|S'|} \frac{|S'|!}{(|S'|-m)!}$$

paarweise nicht äquivalente Ausdrücke.

Da S mindestens einen Zustand enthalten muss, kann man diese Zahl nach oben mit

$$(|S|+2) \cdot \sum_{m=0}^{|S'|} \frac{|S'|!}{(|S'|-m)!} = (|S|+2) \cdot O\left((|S'|+1)!\right) = O\left(|S \cup S'|!\right)$$

abschätzen. □

Wir werden im Folgenden zeigen, dass diese obere Schranke relativ scharf ist und betrachten dazu eine Familie von det-RR(1)-Automaten, die zu den erkennenden DFAs einen trade-off der Größenordnung der Fakultätsfunktion in Bezug auf die Zahl der benötigten Zustände hat.

Untere Schranke

Satz 3.24 *Für jedes $n \in \mathbb{N}$ gibt es eine Sprache über einem ternären Alphabet, welche von einem det-RR(1)-Automaten mit $n = |S \cup S'|$ Zuständen, jedoch von keinem DFA mit weniger als $\Omega((|S \cup S'|-1)!)$ Zuständen, erkannt werden kann.*

Beweis Für $n \in \mathbb{N}$ sei $L = L(\mathcal{M})$ die Sprache, die vom det-RR(1)-Automaten $\mathcal{M} = \left\langle \{s_0\} \cup \{s'_0, \ldots, s'_{n-2}\}, \{a,b,c\}, \{a,b,c\}, \triangleright, \triangleleft, s_0, 1, \delta \right\rangle$ mit n Zuständen erkannt wird, wobei die Überführungsfunktion δ genau in den folgenden Fällen definiert ist:

- $\delta(s_0, a) = (s'_0, \lambda)$ und $\delta(s_0, x) = (s_0, \mathsf{MVR})$ für $x \in \{b, c, \triangleright\}$,
- $\delta(s'_i, a) = (s'_i, \mathsf{MVR})$ für $i \in \{0, \ldots, n-2\}$,
- $\delta(s'_i, b) = (s'_{(i+1)\ MOD\ (n-1)}, \mathsf{MVR})$ für $i \in \{0, \ldots, n-2\}$,
- $\delta(s'_0, c) = \mathsf{Restart}$ und $\delta(s'_i, c) = (s'_i, \mathsf{MVR})$ für $i \in \{1, \ldots, n-2\}$ und

- $\delta(s'_0, \lhd) = \mathsf{Accept}$.

Zur besseren Argumentation nutzen wir die Konstruktion aus Satz 3.21 und betrachten die vom äquivalenten AFA $\mathcal{M}'$ induzierte Überführungsfunktion Δ', welche sich aus der folgenden Übersetzung ergibt:

In $\mathcal{M}'$ ist:

$$\begin{aligned}
\delta'(s_0)(a)(z) &= z(s_0) \wedge z(s'_0) \vee z(\hat{s}_0)\\
\delta'(s'_i)(a)(z) &= z(s'_i)\\
\delta'(\hat{s}_i)(a)(z) &= z(\hat{s}_i)\\
\\
\delta'(s_0)(b)(z) &= z(s_0)\\
\delta'(s'_i)(b)(z) &= z(s'_{(i+1) \bmod (n-1)})\\
\delta'(\hat{s}_i)(b)(z) &= z(\hat{s}_{(i+1) \bmod (n-1)})\\
\\
\delta'(s_0)(c)(z) &= z(s_0)\\
\delta'(s'_0)(c)(z) &= 1\\
\delta'(\hat{s}_0)(c)(z) &= 0\\
\delta'(s'_j)(c)(z) &= z(s'_j)\\
\delta'(\hat{s}_j)(c)(z) &= z(\hat{s}_j)
\end{aligned}$$

für $i \in \{0, \ldots, n-2\}$, $j \in \{1, \ldots, n-2\}$ und $F = \{\hat{s}'_0\}$.

Wir geben nun Präfixe $u_{(i)}$ an, so dass sich während der Auswertung von $\Delta'(u_{(i)}v, f)(s_0)$ unabhängig vom Restwort $v \in A^*$ ein Term der Form

$$\begin{aligned}
\Delta'(u_{(i)}v, f)(s_0) = \big(\ldots(\Delta'(v, f)(s_0) \wedge \Delta'(v, f)(s'_{i_k}) \vee \Delta'(v, f)(\hat{s}_{i_k}))\\
\wedge \ldots \wedge \Delta'(v, f)(s'_{i_1}) \vee \Delta'(v, f)(\hat{s}_{i_1})
\end{aligned}$$

ergibt, wobei $(i)_{1,\ldots,k}$ eine endliche, möglicherweise leere Folge von Indizes aus $\{0, \ldots, n-2\}$ ist, so dass für $k, l \in \{1, \ldots, n-1\}$ und $k \neq l$ stets $i_k \neq i_l$ gilt. Wir wählen:

$$u_{(i)} = ab^{(i_1 - i_2) \bmod (n-1)} \cdots ab^{(i_{k-1} - i_k) \bmod (n-1)} ab^{i_k}.$$

Es ist offensichtlich, dass diese Wahl von $u_{(i)}$ das Gewünschte leistet – z. B. entsteht der Term an der l-ten Stelle (von rechts bzw. außen gesehen) durch den l-ten Restart-Schritt von $\mathcal{M}$ (auf dem l-ten Buchstaben a) und der entsprechend im Nachlesen angenommene Zustand s'_0 geht durch die danach zu lesenden Zeichen $b^{(i_l - i_{l+1}) \bmod (n-1)} a \cdots b^{(i_{k-1} - i_k) \bmod (n-1)} ab^{i_k}$ in den Zustand $s'_{(0 + i_l - i_{l+1} + \ldots - i_{k-1} + i_k) \bmod (n-1)} = s'_{i_k}$ über.

Man kann sehen, dass auf diese Weise genau

$$\sum_{i=0}^{n-1} \frac{(n-1)!}{(n-1-i)!} \in \Omega((n-1)!)$$

verschiedene Wörter $u_{(i)}$ angegeben werden.

Von ihnen zeigen wir nun, dass von zwei unterschiedlichen Wörtern $u_{(i)}$ und $u_{(j)}$ erzeugte Terme nicht äquivalent sind, da es stets ein Suffix $v_{(i)(j)}$ gibt, so dass gilt:

$$\Delta'(u_{(i)}v_{(i)(j)}, f)(s_0) = 1 \quad \text{und} \quad \Delta'(u_{(j)}v_{(i)(j)}, f)(s_0) = 0.$$

Man beachte, dass diese Eigenschaft wesentlich schwächer ist, als die für die Elemente eines Fooling Sets geltende: sie hängt notwendig von der Kenntnis des Unterschiedes zwischen den Indexfolgen $(i)_{1\ldots k}$ und $(j)_{1\ldots m}$ ab[11]. Wir unterscheiden die beiden Fälle, dass sich $(i)_{1\ldots k}$ und $(j)_{1\ldots m}$ an einer (ersten) Stelle unterscheiden bzw. $(i)_{1\ldots k}$ mehr Elemente hat, als $(j)_{1\ldots m}$.

Fall 1 Es gilt $i_1 = j_1, \ldots, i_{l-1} = j_{l-1}$ und $i_l \neq j_l$ für $k \geq m \geq l \geq 1$:

Wir wählen: $v_{(i)(j)} = b^{n-1-i_1} c b^{(i_1-i_2) \,\mathrm{MOD}(n-1)} \cdots c b^{(i_{l-1}-i_l) \,\mathrm{MOD}(n-1)} a$.

Damit wird das Wort $u_{(i)}v_{(i)(j)}$ von $\mathcal{M}$ nach l Rewrite-Schritten (und $l-1$ Restart-Schritten) im l-ten Nachlesevorgang am rechten Bandbegrenzungszeichen akzeptiert und das Wort $u_{(j)}v_{(i)(j)}$ nach l Rewrite-Schritten (und $l-1$ Restart-Schritten) im l-ten Nachlesevorgang am rechten Bandbegrenzungszeichen verworfen (da nicht der Zustand s_0' anliegt).

Fall 2 Es gilt $i_1 = j_1, \ldots, i_{l-1} = j_{l-1}$ und $k \geq l > m \geq 0$:

Mit dem oben gewählten $v_{(i)(j)}$ wird das Wort $u_{(j)}v_{(i)(j)}$ von $\mathcal{M}$ nach $l-1$ Rewrite-Schritten (und $l-1$ Restart-Schritten) am rechten Bandbegrenzungszeichen verworfen, da $\delta(s_0, \lhd)$ nicht definiert ist.

Damit ist gezeigt, dass jeder DFA zum Erkennen von $L(\mathcal{M})$ (durch die Wörter $u_{(i)}$) in wenigstens $\Omega((n-1)!)$ paarweise nicht äquivalente Zustände gelangen kann. □

[11] Einen Fooling Set dieser Größe wird es nach Satz 3.11 sowieso nicht geben können.

3.5 Zusammenfassung

Die Ergebnisse der vorigen Abschnitte lassen sich in folgendem Bild vereinfacht zusammenfassen:

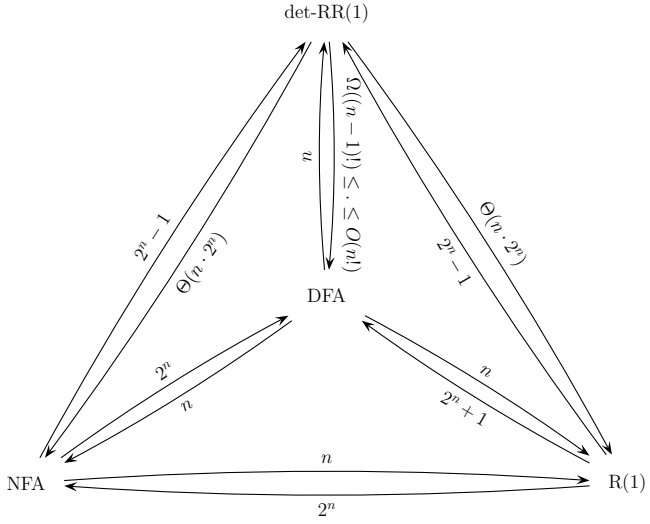

Abbildung 5: Übersicht der Ergebnisse

Die jeweiligen Ausdrücke in der Variablen n auf den Pfeilen stellen dabei die Kosten da, die bei einem Wechsel des Beschreibungssystems in Pfeilrichtung im schlimmsten Fall entstehen können. Einzelne Ausdrücke bedeuten, dass diese Kosten exakt angegeben werden können (Ungleichungen stellen die gefundenen oberen und unteren Schranken dar).

Das Ergebnis von Satz 3.21 findet sich hier nicht wieder – ebensowenig, wie die Angabe der für die jeweilige Schranke nötigen Alphabetgröße.

Der eingezeichnete Zustands-Trade-off beim Wechsel von NFA nach R(1) lässt sich leicht mit einer unären Ein-Wort-Sprache ($\{a^{n-1}\}$) belegen. Ebenso dient die unäre Sprache, bei der alle Wortlängen durch die Zahl n teilbar sind ($\{a^{k \cdot n} \mid k \in \mathbb{N}_0\}$), als Beleg für die trade-offs von DFA zu den beiden neuen Restart-Automaten-Klassen.

Die Klasse der det-R(1)-Automaten, die trivialerweise ebenfalls die Klasse der Regulären Sprachen beschreibt, wurde hier nicht betrachtet, da man leicht sehen kann, dass sich jeder det-R(1)-Automat durch einen DFA mit partieller Überführungsfunktion (und umgekehrt) ohne Vergrößerung der Zustandsmenge beschreiben lässt.

Zum Abschluss sei noch an die zu Beginn des Kapitels getroffene Vereinbarung bezüglich der Überführung $\delta(s_0, \triangleright)$ erinnert. Die gefundenen Ergebnisse lassen sich wie folgt auf den allgemeinen Fall übertragen:

- Der Originalfassung von Satz 3.1 geht bereits von einem beliebigem R(1)-Automaten aus, weshalb die obere Schranke der Trade-offs beim Wechsel von R(1)-Automaten zu einem beliebigen anderen der verglichenen Automaten unverändert bestehen bleibt (die unteren Schranken sind von der Änderung sowieso nicht betroffen).

- Die trivialen Trade-offs beim Wechsel von NFA bzw. DFA zu R(1)-Automaten bleiben ebenfalls unverändert.

- Obere und untere Schranke des Trade-offs beim Wechsel von det-RR(1)-Automaten zu R(1)-Automaten können sich höchstens um einen Zustand nach unten verändern, falls es nicht möglich sein sollte, den Beweis von Korollar 3.16 anzupassen. Grund dafür ist, dass ein allgemeiner R(1)-Automat $\mathcal{M}$ mit n Zuständen stets durch einen R(1)-Automaten $\mathcal{M}'$ mit $n+1$ Zuständen und der Festlegung $\delta(s'_0, \triangleright) = \{s'_0\}$ simuliert werden kann: $\mathcal{M}'$ holt die Überführungen $\delta(s_0, \triangleright)$ von $\mathcal{M}$ auf dem ersten Eingabezeichen nach und sorgt insbesondere dafür, dass letzteres nicht gelöscht wird. Die Überführungsfunktion von $\mathcal{M}$ wird nur um die Behandlung von s'_0 erweitert, woraus auch folgt, dass dieser Zustand nach dem Lesen des ersten Zeichens nicht wieder angenommen werden kann.

4 RRWW-Automaten und ihre Subklassen

In diesem Kapitel werden sich unsere Untersuchungen auf die Automatentypen zu höher in der Chomsky-Hierarchie stehenden Sprachklassen konzentrieren. Die hier jeweils entscheidenden Ressourcen, wie z. B. Nichtdeterminismus oder die Verwendung von Sonderzeichen liefern jedoch in Verbindung mit dem restlichen Restartautomaten eine so große Erkennungsmächtigkeit und Beschreibungseffizienz, dass alle gefundenen Ergebnisse in Form von Nachweisen nichtrekursiver Trade-offs vorliegen. Es ist also oft nachweisbar, dass ein System mit einer zusätzlichen Ressource manche Sprachen beliebig effizienter beschreibt.

Die im vorigen Kapitel essentielle Fenstergröße wird jedoch hierbei keine für uns interessante Ressource sein, da in allen Konstruktionen schon geringe Fenstergrößen zur Erkennung der verwendeten Beispielsprachen ausreichen.[12]

Wir beginnen das Kapitel mit einer Einführung in eine der Standard-Techniken zum Nachweis nichtrekursiver Trade-offs und einigen naheliegenden Resultaten. Hiervon wird auch die Ordnung und Abfolge der einzelnen Abschnitte geprägt sein: auf die zunächst angeführten Ergebnisse und Beweise wird im Folgenden des Öfteren zurückgegriffen werden.

[12] In [27] werden in Bezug auf die Fenstergröße unendliche Hierarchien von R(R)(W)-Automatenklassen nachgewiesen. Für diejenigen Restartautomaten, welche über Sonderzeichen verfügen, scheint dies jedoch nicht zuzutreffen.

4.1 RRWW-Automaten und naheliegende Subklassen

Im Laufe der Zeit wurden in einigen Veröffentlichungen Existenznachweise nichtrekursiver Trade-offs geführt. In seiner Arbeit [20] weist Kutrib darauf hin, dass die Nachweise oft einer von nur zwei verschiedenen, grundsätzlichen Vorgehensweisen folgen und stellt für beide ein vereinheitlichtes Schema vor. Das nachfolgend aufgeführte Schema entspricht dabei der Vorgehensweise, die z. B. Hartmanis in [10] wählte.

Satz 4.1 (Kutrib) *Es seien S_1 und S_2 zwei Beschreibungssysteme für rekursive (entscheidbare) Sprachen. Wenn es ein System S_3 gibt, so dass zu jedem beliebigen $\mathcal{M} \in S_3$*

(i) zu einer Sprache $L_{\mathcal{M}}$ ein Beschreiber aus S_1 effektiv konstruierbar ist,

(ii) während in S_2 genau dann ein Beschreiber für $L_{\mathcal{M}}$ existiert, wenn eine Eigenschaft P für die Sprache $L(\mathcal{M})$ nicht gilt und

(iii) die Eigenschaft P nicht semi-entscheidbar für Sprachen mit Beschreibern aus S_3 ist,

dann ist der Trade-off zwischen S_1 und S_2 nichtrekursiv.

Der Widerspruchsbeweis hierzu verwendet die Eigenschaften (i) und (ii), um aus der Annahme eines rekursiven Trade-offs die Semi-Entscheidbarkeit der Eigenschaft P zu folgern:

- Wenn es zu einer Sprache L mit Beschreiber $\mathcal{D} \in S_1$ einen Beschreiber aus S_2 gibt, dessen Größe zunächst als rekursiv beschränkt angenommen wird, ist die Menge aller nicht größeren Beschreibungssysteme aus S_2 stets endlich.
- Dann simuliert eine Turingmaschine das Verhalten von $\mathcal{D}$ und allen diesen Beschreibern so lange auf der (alphabetisch) sortierten Liste aller Wörter, bis von jedem Beschreiber aus S_2 feststeht, dass er nicht äquivalent zu $\mathcal{D}$ ist. Man kann erkennen, dass die Turingmaschine genau dann hält, wenn es (in der betrachteten Menge) keinen zu $\mathcal{D}$ äquivalenten Beschreiber gibt.
- In der Verallgemeinerung erhält man die rekursiv aufzählbare Menge

  $$R = \{\mathcal{D} \mid \mathcal{D} \in S_1 \text{ und } L(\mathcal{D}) \text{ hat keinen Beschreiber in } S_2\}$$

  und auf Grund der Voraussetzungen lässt sich zur Sprache $L_{\mathcal{M}}$ ein Beschreiber $D_{\mathcal{M}} \in S_1$ konstruieren und von diesem semi-entscheiden, ob $D_{\mathcal{M}} \in R$ ist. Damit aber ist die Eigenschaft P ebenfalls semi-entscheidbar, was somit der Voraussetzung widerspricht.

Die Anwendbarkeit des obigen Beweisschemas stützt sich auf die Existenz eines geeigneten Beschreibungssystems S_3, welches passende Nicht-semi-Entscheidbarkeiten liefert. Aus diesem Grund wird oft die Menge der Turingmaschinen betrachtet. Entscheidend dafür ist jedoch, dass es gelingt, einen tragenden Zusammenhang zwischen diesen rekursiv aufzählbaren Sprachen und den betrachteten Systemen S_1 und S_2 zu schaffen, denn letztere liegen im Allgemeinen in der Chomsky-Hierarchie viel weiter unten.

Eine Möglichkeit hierzu wurde jedoch bereits in [9] aufgezeigt, indem Turing-Berechnungen in Grammatiken niedrigen Typs kodiert wurden.

Eine solche, häufig verwendete Kodierung ergibt sich aus der Betrachtung aller akzeptierenden Berechnungen einer (deterministischen) Turingmaschine: Die Zahl der während der akzeptierenden Verarbeitung eines Wortes angenommenen Konfigurationen ist endlich und man fügt alle diese Konfigurationen zu einem Wort einer neuen Sprache zusammen. Zur Bezeichnung dieser neuen Sprache wird oft der Name *valid computations* verwendet.

Wir folgen der Vorgehensweise aus [20] und betrachten ohne Einschränkung nur deterministische Ein-Band- und Ein-Kopf-Turingmaschinen $\mathcal{M}$ mit den folgenden Eigenschaften:

- Das Bandalphabet sei T, das Eingabealphabet $A \subset T$, das Leerzeichen $\sqcup \in T$, die Zustandsmenge S, der Startzustand $s_0 \in S$ und die Menge der Endzustände $F \subseteq S$,
- $\mathcal{M}$ hält nur nach einer ungeraden Zahl von Schritten, mindestens aber nach drei und
- $\mathcal{M}$ akzeptiert durch Halten und schreibt niemals ein $\sqcup$.

Für die Beschreibung der Konfigurationen von $\mathcal{M}$ legen wir fest:

- Eine Konfiguration w_i ($i \in \{1, \ldots, 2n\}$ für ein $n \geq 2$) ist von der Form $w_i \in T^*ST^*$ mit $w_i = t_1 \cdots t_j s t_{j+1} \cdots t_l$ (für beliebige $0 \leq j \leq l$) bzw. $w_1 = s_0$, falls das leere Wort anliegt, wobei die vorkommenden $t_1, \ldots, t_l \in T \setminus \{\sqcup\}$ genau die vom Leerzeichen verschiedenen Zeichen der Bandinschrift sind. Das Auftreten des Zustands $s \in S$ zwischen zwei Zeichen t_j und t_{j+1} bedeutet, dass der Kopf der Turingmaschine über dem Zeichen t_{j+1}, also an der Stelle $j + 1$ steht. Analog dazu befindet sich der Kopf von $\mathcal{M}$ über dem ersten Zeichen, wenn s links von t_1 steht und auf dem ersten Leerzeichen hinter dem Wort, wenn s rechts von t_l steht.
- Die Startkonfiguration w_1 ist von der Form s_0A^* und die Endkonfiguration aus $(T \setminus \{\sqcup\})^*F(T \setminus \{\sqcup\})^*$.

Bei der Vereinbarung der folgenden Benennungen halten wir uns ebenfalls an die „üblichen Konventionen“ und bezeichnen für $n \geq 2$ die Menge der Wörter $w_1\$w_2^R\$w_3\$w_4^R \dots w_{2n-1}\$w_{2n}^R$ als $\mathsf{VALC}_A(\mathcal{M})$, wobei $\{\$\} \cap (S \cup T) = \emptyset$ ist und für alle $i \in \{1, \dots, 2n-1\}$ gilt, dass w_{i+1} die Folgekonfiguration von w_i (bezüglich der Überführung durch $\mathcal{M}$) ist.

Die Sprache $\mathsf{INVALC}_A(\mathcal{M})$ bezeichne das Komplement von $\mathsf{VALC}_A(\mathcal{M})$ bezüglich $S \cup T \cup \{\$\}$.

Weiterhin lockern wir die die Folgekonfigurationen betreffende Vereinbarung etwas auf und erhalten eine Zerlegung von $\mathsf{VALC}_A(\mathcal{M})$ als Schnitt der beiden folgenden Sprachen:

Für $n \geq 2$ sei $\mathsf{VALC}_{A_1}(\mathcal{M})$ die Menge der Wörter w, welche von der Form $w = w_1\$w_2^R\$w_3\$w_4^R \dots w_{2n-1}\$w_{2n}^R$ sind, so dass für alle $i \in \{1, \dots, n\}$ die Folgekonfiguration von w_{2i-1} durch w_{2i} beschrieben wird.

Ebenso sei für $n \geq 2$ ist $\mathsf{VALC}_{A_2}(\mathcal{M})$ die Menge der Wörter w der Form $w = w_1\$w_2^R\$w_3\$w_4^R \dots w_{2n-1}\$w_{2n}^R$ mit den obigen Eigenschaften, so dass für alle $i \in \{1, \dots, n-1\}$ die Folgekonfiguration von w_{2i} durch w_{2i+1} beschrieben wird.

Die Sprachen $\mathsf{INVALC}_{A_1}(\mathcal{M})$ und $\mathsf{INVALC}_{A_2}(\mathcal{M})$ seien ebenso als Komplemente bezüglich $S \cup T \cup \{\$\}$ festgelegt.

Bei manchen der nachfolgenden Beweise wird die angestrebte Konstruktion erheblich vereinfacht, wenn wir in jeder Konfiguration einer akzeptierenden Berechnung schon zu Beginn genau die Bandpositionen beschreiben, die $\mathcal{M}$ betreten wird. Ferner soll auch das neu betretene Feld rechts vom bisher bearbeiteten Bandinhalt in der Kodierung der Konfiguration mit auftreten, sobald es vom Schreib-/Lese-Kopf von $\mathcal{M}$ erreicht wird (und nicht erst, wenn es überschrieben wurde).

Dazu sei für $n \geq 2$ $\mathsf{VALC}_{C'}(\mathcal{M})$ die Menge der Wörter

$$u_1w_1v_1\$u_2w_2v_2\$ \dots u_{2n}w_{2n}v_{2n}\$,$$

in denen neben den obigen Einschränkungen der w_i gilt, dass zu jedem $i \in \{1, \dots, 2n-1\}$ die Konfiguration w_{i+1} auf die Konfiguration w_i folgt und die folgenden Eigenschaften gelten:

- Entweder ist $w_i = t_1 \cdots t_j s t_{j+1} \cdots t_l$ für $0 \leq j < l$ oder es gilt $w_i = t_1 \cdots t_l s_{\sqcup}$.
- Ferner ist $|u_1w_1v_1| = |u_2w_2v_2| = \cdots = |u_{2n}w_{2n}v_{2n}|$, während alle u_i, v_i höchstens aus Leerzeichen bestehen,
- in der letzten Konfiguration keine unbetretenen Bandpositionen mehr vorkommen ($u_{2n} = v_{2n} = \lambda$) und

- sich die absolute Position des betretenen Wortes nicht verändert:
  - $u_{i+1} = u_i$, falls $w_i \notin ST^*$ (also kein Linksschritt auf eine noch nicht betretene Bandposition möglich ist) bzw.
  - $v_{i+1} = v_i$, falls $w_i \notin T^*ST$ (wie oben, nur kein Rechtsschritt) und
  - im Falle der Startkonfiguration $w_1 = s_0\sqcup$ wenigstens $v_2 = v_1$ oder $u_2 = u_1$ ist – je nachdem, wohin der Schreib-/Lese-Kopf **nicht** bewegt wird.

Analog zu Obigem bezeichne die Sprache $\mathsf{INVALC}_{C'}(\mathcal{M})$ das Komplement von $\mathsf{VALC}_{C'}(\mathcal{M})$ bezüglich $S \cup T \cup \{\$\}$.

Viele der wesentlichen Ergebnisse sind schon in [20] zusammengefasst dargestellt, so dass wir ebenfalls auf ausführlichere Beweise hierzu verzichten können:

Lemma 4.2 *Es sei $\mathcal{M}$ eine Turingmaschine. Dann gelten die folgenden Aussagen:*

(1) Wenn $L(\mathcal{M})$ endlich ist, dann auch $\mathsf{VALC}_A(\mathcal{M})$ und $\mathsf{VALC}_{C'}(\mathcal{M})$.

(2) Wenn $L(\mathcal{M})$ endlich ist, dann sind $\mathsf{INVALC}_A(\mathcal{M})$ und $\mathsf{INVALC}_{C'}(\mathcal{M})$ regulär.

(3) Wenn $L(\mathcal{M})$ unendlich ist, dann sind $\mathsf{VALC}_A(\mathcal{M})$ und $\mathsf{VALC}_{C'}(\mathcal{M})$ nicht kontextfrei.

(4) Wenn $L(\mathcal{M})$ unendlich ist, dann sind $\mathsf{INVALC}_A(\mathcal{M})$ und $\mathsf{INVALC}_{C'}(\mathcal{M})$ nicht regulär.

(5) $\mathsf{INVALC}_A(\mathcal{M})$ und $\mathsf{INVALC}_{C'}(\mathcal{M})$ sind linear kontextfreie Sprachen, deren 1-turn-Kellerautomat jeweils effektiv konstruierbar ist.

(6) $\mathsf{VALC}_{A_1}(\mathcal{M})$ und $\mathsf{VALC}_{A_2}(\mathcal{M})$ sind deterministisch kontextfreie Sprachen mit effektiv konstruierbaren, deterministischen Kellerautomaten. Ihr Schnitt ergibt die Sprache $\mathsf{VALC}_A(\mathcal{M})$.

Beweis Die Aussagen (1), (2) und (4) sind unmittelbar einsichtig. In [9] wird die Aussage „$L(\mathcal{M})$ unendlich $\Rightarrow \mathsf{VALC}_A(\mathcal{M})$ nicht kontextfrei“ explizit durch Anwendung eines Pumping Lemmas nachgewiesen. Ähnliches ist auch für $\mathsf{VALC}_{C'}(\mathcal{M})$ möglich, woraus die Korrektheit von (3) folgt.

Zu (5): Ebenfalls in [9] wurde gezeigt, dass $\mathsf{INVALC}_A(\mathcal{M})$ eine linear kontextfreie Sprache ist, deren one-turn-PDA aus $\mathcal{M}$ effektiv konstruiert werden kann.

Ähnlich dazu lässt sich auch nachweisen, dass $\mathsf{INVALC}_{C'}(\mathcal{M})$ ebenfalls effektiv konstruierbar linear kontextfrei ist: Ein 1-turn-PDA muss lediglich

nachweisen, dass die (reguläre) Grundform der Wörter aus $\mathsf{VALC}_{C'}(\mathcal{M})$ verletzt ist bzw. die Stelle raten, an der zwei benachbarte Konfigurationen nicht ineinander übergehen können. Hat die Probe Erfolg und der Fehler wird gefunden, so akzeptiert der Automat, ansonsten bricht er ab (auf einem Wort aus $\mathsf{INVALC}_{C'}(\mathcal{M})$ gibt es also stets eine Möglichkeit, den Fehler zu finden, während es für Wörter aus $\mathsf{VALC}_{C'}(\mathcal{M})$ niemals eine akzeptierende Verarbeitung gibt).

Die unterschiedlichen Ursachen, an denen der 1-turn-Kellerautomat einen Fehler zwischen zwei benachbarten Konfigurationen erkennt, sind z. B. eine Zustandsveränderung, die nicht zur Überführungsfunktion von $\mathcal{M}$ passt oder zwei ungleiche Konfigurationslängen. Alle diese können für eine geratene Stelle gleichzeitig nachgeprüft werden.

Mit der Beobachtung, dass auch das Komplement der (nicht kontextfreien) Sprache $\{w\$w \mid w \in \{a,b\}^*\}$ linear kontextfrei ist (siehe etwa [37]), lässt sich belegen, dass es auch nicht auf das Fehlen der spiegelsymmetrischen Anordnung der Wörter w_i ankommt, womit sich erkennen lässt, dass (5) gilt.

Als Letztes führen wir an, dass $\mathsf{VALC}_{A_1}(\mathcal{M})$ und $\mathsf{VALC}_{A_2}(\mathcal{M})$ bereits in [43] als deterministisch kontextfreie Sprachen, deren deterministischer Kellerautomat sich effektiv aus $\mathcal{M}$ konstruieren lässt, nachgewiesen wurden. Der Rest ist unmittelbar einsichtig, womit sich Aussage (6) ergibt. □

Restartautomaten ohne Sonderzeichen

Nach Lemma 4.2 ist $\mathsf{INVALC}_{C'}(\mathcal{M})$ eine linear kontextfreie und damit auch von (mon-)R(R)WW-Automaten erkennbare Sprache. Wir zeigen nun, dass es jedoch einige Typen von Restart-Automaten gibt, die diese Sprache nur in Spezialfällen erkennen können.

Lemma 4.3 *Es sei $\mathcal{M}$ eine beliebige Turingmaschine. Dann wird die Sprache $\mathsf{INVALC}_{C'}(\mathcal{M})$ von einem RRW-Automaten genau dann erkannt, wenn $L(\mathcal{M})$ endlich ist.*

Beweis Wir wissen, dass $\mathsf{INVALC}_{C'}(\mathcal{M})$ eine reguläre Sprache ist, wenn $L(\mathcal{M})$ endlich ist. In diesem Fall ist offensichtlich, dass $\mathsf{INVALC}_{C'}(\mathcal{M})$ auch von einem RRW-Automaten erkannt wird.

Nun sei $L(\mathcal{M})$ eine unendliche Sprache und wir nehmen zunächst an, dass es einen RRW-Automaten $\mathcal{M}'$ gibt, der $\mathsf{INVALC}_{C'}(\mathcal{M})$ erkennt.

Nach Proposition 2.5 muss $\mathcal{M}'$ auf $\mathsf{INVALC}_{C'}(\mathcal{M})$ fehlererhaltend arbeiten. Das bedeutet, dass ein Wort $w \in \mathsf{VALC}_{C'}(\mathcal{M})$ nicht zu einem Wort w' aus $\mathsf{INVALC}_{C'}(\mathcal{M})$ umgeschrieben werden darf – mit der Ausnahme, dass $\mathcal{M}'$ danach nicht mehr neu startet.

Auf Grund der Unendlichkeit von $L(\mathcal{M})$ gibt es stets eine Zahl l, so dass $\mathcal{M}'$ auf allen $w \in \mathsf{VALC}_{C'}(\mathcal{M})$ mit $|w| > l$ höchstens einmal einen Rewrite-Schritt und niemals einen Restart-Schritt durchführen darf, was sich wie folgt begründet:

- Es gibt nur endlich viele akzeptierende Berechnungen, bei denen die Länge der längsten auftretenden Konfiguration unterhalb der Fenstergröße k von $\mathcal{M}'$ bleibt.
- Also gibt es ein l, so dass jedes Wort $w \in \mathsf{VALC}_{C'}(\mathcal{M})$ mit $|w| > l$ von der Form $w_1\$\cdots\$w_n$ mit $|w_1| = \ldots = |w_n| > k$ ist und
- jede Löschung auf w eine Verkürzung bewirkt, die bedingt, dass das resultierende Wort w' nicht mehr von der Form $w_1\$\cdots\$w_n$ mit $|w_1| = \ldots = |w_n|$ sein kann und damit $w' \in \mathsf{INVALC}_{C'}(\mathcal{M})$ gilt.

Die Sprache, die aus eben diesen Wörtern w gebildet wird, ist damit sicherlich nicht regulär. Bei ihrer Erkennung darf sich $\mathcal{M}'$ jedoch nur wie ein endlicher Automat verhalten, was unserer Annahme der Existenz von $\mathcal{M}'$ widerspricht. □

Mit dieser Aussage ist es uns nun möglich, über Satz 4.1 auf die Existenz eines nichtrekursiven Trade-offs zu schließen.

Satz 4.4 *Der Trade-off zwischen mon-R(R)WW-Automaten (Kellerautomaten) und RRW-Automaten ist nichtrekursiv.*

Beweis Wir wählen S_3 als Menge der Turingmaschinen, $\mathsf{INVALC}_{C'}(\mathcal{M})$ als Sprache $L_{\mathcal{M}}$ und die Eigenschaft P von $L(\mathcal{M})$ als Unendlichkeit. Ferner gibt es nach Lemma 4.2 einen effektiv konstruierbaren Kellerautomaten für $\mathsf{INVALC}_{C'}(\mathcal{M})$, womit es uns möglich ist, S_1 als die Klasse der mon-R(R)WW-Automaten zu wählen. Nach Lemma 4.3 gibt es in der Menge S_2 der RRW-Automaten genau dann einen Beschreiber für $L_{\mathcal{M}}$, wenn P für $L(\mathcal{M})$ nicht gilt. □

Dieses Ergebnis lässt sich mit der vorliegenden Argumentation auch auf die folgenden Klassen erweitern:

Korollar 4.5 *Der Trade-off zwischen Restart-Automaten des Typs X und des Typs Y ist nichtrekursiv für* $X \in \{\mathsf{wmon\text{-}R(R)WW}, \mathsf{RWW}, \mathsf{RRWW}\}$ *und* $Y \in \{\mathsf{RR}, \mathsf{RW}, \mathsf{R}\}$.

Beweis Lemma 4.3 gilt auch für die für Y genannten Möglichkeiten, da diese sowohl alle in der Klasse der RRW-Automaten enthalten sind, als auch

selbst die Klasse der regulären Sprachen enthalten. Analog zum vorigen Beweis ergeben sich daraus die genannten Trade-offs.[13]

Ferner sind die Klassen der RWW- und RRWW-Automatensprachen Obermengen der von monotonen R(R)WW-Automaten erkannten Sprachen, weshalb sich die nachgewiesene Eigenschaft auch auf sie überträgt.[14] □

Im Folgenden wollen wir die Beschreibungseffizienz von RRWW-Automaten gegenüber ihren durch Determinismus bzw. Monotonieeigenschaften eingeschränkten Varianten diskutieren.

Restartautomaten mit Sonderzeichen

Da $\mathsf{VALC}_{C'}(\mathcal{M})$ nach Lemma 4.2 im Allgemeinen keine kontextfreie und damit auch keine von monotonen Restart-Automaten erkennbare Sprache ist, wird sich ihre Betrachtung für Untersuchungen auf nichtrekursive Trade-offs lohnen, wenn wir Folgendes nachweisen:

Lemma 4.6 *Es sei $\mathcal{M}$ eine beliebige Turingmaschine. Dann gibt es einen effektiv konstruierbaren RRWW-Automat, der $\mathsf{VALC}_{C'}(\mathcal{M})$ akzeptiert.*

Beweis Die Idee ist es, dass der RRWW-Automaten zunächst prüft, ob das bearbeitete Wort von seiner Struktur her aus $\mathsf{VALC}_{C'}(\mathcal{M})$ stammen könnte und erst nach einem positiven Befund mit der genaueren Prüfung beginnt. Wesentliches Kriterium hierbei ist, dass das Wort eine Folge von entsprechend getrennten Konfigurationen darstellt (ist das nicht der Fall, so kann sofort verworfen werden).

Auf Wörtern passender Form wird danach eine Abarbeitungsfolge erstellt, die aus genau so vielen Sweeps besteht, wie es Konfigurationen im Wort gibt. Darin wird nun das letzte Zeichen aus jeder Konfiguration gelöscht und jedesmal (nachträglich) geprüft, ob die letzten Stellen der nachfolgenden Konfiguration konsistent dazu waren.

Im Rahmen dieses Beweises verwenden wir den Begriff der Konsistenz (bezüglich der Überführungsfunktion δ von $\mathcal{M}$) wie folgt:
Zwei Wörter $u, v \in T^3 \cup ST^2 \cup TST \cup T^2S$ heißen *konsistent*, wenn sie einer der folgenden Bedingungen genügen:

- $u = x_1x_2x_3 \in T^3$ und $v = y_1y_2y_3 \in (S \cup T)T^2$ mit $y_3 = x_3$,
- $u = x_1x_2x_3 \in ST^2$ und $v = y_1y_2y_3 \in (S \cup T)(S \cup T)T$ mit $y_3 = x_3$,

[13] Solche trivialen Erweiterungen werden wir im Folgenden nicht mehr betrachten.

[14] Man beachte, dass sich die gefundenen Trade-offs im allgemeinen **nicht** auf die Untermengen der weniger ausdruckseffizienten Sprachklasse übertragen lassen, da sie auf Sprachen basieren können, die in den Untermengen nicht mehr vorkommen.

- $u = x_1x_2x_3 \in TST$ und $v = y_1y_2y_3$ mit
  - $y_1 = x_1$, $y_2 = t$ und $y_3 = s$, falls $\delta(x_2, x_3) = (s, t, r)$ gilt,
  - $y_1 = x_1$, $y_2 = s$ und $y_3 = t$, falls $\delta(x_2, x_3) = (s, t, n)$ gilt,
  - $y_1 = s$, $y_2 = x_1$ und $y_3 = t$, falls $\delta(x_2, x_3) = (s, t, l)$ gilt

  oder
- $u = x_1x_2x_3 \in T^2S$ und v beliebig.

Konsistenz gilt also, wenn sich die letzten Zeichen einer Folgekonfiguration im Rahmen der Berechnung von $\mathcal{M}$ aus den letzten Zeichen der vorherigen Konfiguration ergeben könnten (z. B. bleiben die letzten Zeichen gleich, wenn es sich bei ihnen ausschließlich um Bandsymbole handelt). Daher prüft der RRWW-Automat gleichzeitig innerhalb einer festen Umgebung des letzten Zeichens (bei uns drei Zeichen), ob die einzelnen Teilwörter eine gültige Konfigurationen-Folge darstellen.

Ist Letzteres nicht der Fall, verwirft der Automat das Wort, während er im positiven Fall nach derselben Methode so lange ein weiteres Zeichen von jeder der Konfigurationen entfernt, bis er dem Restwort dessen (Un-) Korrektheit ansehen kann (hier: wenn zu Beginn eines neuen, gekoppelten Durchlaufs wenigstens eine der reduzierten Konfigurationen aus weniger als drei Zeichen besteht).

Der oben genannten Idee folgend betrachten wir nun zu einer beliebigen Turingmaschine $\mathcal{M}$ mit Bandalphabet T und Zustandsmenge S den RRWW-Automaten $\mathcal{M}'$ (mit Fenstergröße 4), dessen Meta-Instruktionen sich in vier Kategorien unterteilen lassen:

I Prüfung der Grundstruktur,

II gekoppelte Löschung des jeweils letzten Zeichens jeder Konfiguration,

III Start einer weiteren solchen Abarbeitungsfolge und

IV Prüfung von Wörtern mit zu kurzen bzw. genügend gekürzten Konfigurationen.

Im Folgenden seien $w_i \in (T^*ST^*) \cup T^*$ Konfigurationen bzw. solche, deren Ende bereits entfernt wurde. In den ersten drei Instruktions-Kategorien wird davon ausgegangen, dass alle noch zu bearbeitenden Konfigurationen länger als zwei Zeichen sind, womit stets eine Zerlegung in $w_i = u_iv_i$ mit $|v_i| = 3$ möglich ist. Wenn dies während einer Berechnung nicht der Fall ist, verwirft $\mathcal{M}'$, d. h. es gibt keine entsprechend anwendbare Meta-Instruktion. Ferner sei v_i' die jeweils um das letzte Zeichen verringerte Kopie von v_i, d. h. es gibt ein $x \in S \cup T$, mit $v_i = v_i'x$. Damit besitzt $\mathcal{M}'$ genau die folgenden Meta-Instruktionen:

I

$$\left(\rhd u_1, v_1\$ \to v_1'\#_j, w_2\$Xw_3\$\lhd\right), \text{ wobei}$$

- $u_1v_1, w_2, w_3 \in T^*ST^+$ und $X \in (T^*ST^+\$T^*ST^+\$)^*\, T^*ST^+\$$ sind, d. h. eine gerade Anzahl von – durch \$ getrennten – Konfigurationen von $\mathcal{M}$ vorliegt,
- u_1v_1 eine Startkonfiguration von $\mathcal{M}$ ist, d. h. der Startzustand am ersten Zeichen eines Eingabewortes anliegt und der Rest des Bandes aus Leerzeichen besteht,
- w_3 eine Endkonfiguration ist, d. h. der vorkommende Zustand mit dem auf der aktuellen Kopfposition anliegenden Zeichen ein akzeptierendes Anhalten der Turingmaschine verursacht,
- es mindestens eine Konfiguration gibt, bei der die am weitesten links liegende Bandposition erreicht wird (die also in ST^* liegt),
- es mindestens eine Konfiguration gibt, bei der die am weitesten rechts liegende Bandposition erreicht wird (die also in T^*ST liegt),
- $|u_1v_1| \equiv |w_2| \equiv \cdots \equiv |w_3| \equiv j \pmod 3$ ist, also alle vorkommenden Konfigurationen modulo 3 dieselbe Länge haben (was eine Voraussetzung für die Korrektheit der beabsichtigten Verarbeitungsfolgen darstellt) und
- die letzten drei Bandzeichen von w_2 konsistent zu v_1 sind.

Alle diese Zusatzbedingungen sind regulär, weshalb die genannte Meta-Instruktion formal korrekt ist.

II Für jeweils $j \in \{0, 1, 2\}$ gibt es die folgenden Instruktionen:

$$\Big(\rhd \left((T^*ST^* \cup T^*)\,\#_j\right)^+ u, v\$ \to v'\#_j,$$
$$w\$\left((T^2T^* \cup STT^* \cup T^+ST^+ \cup TT^*S)\$\right)^* \lhd\Big), \text{ wobei}$$

die letzten drei Zeichen von $w \in T^*ST^* \cup T^*$ mit $|w| \geq 3$ konsistent zu v sein müssen, was ebenso für die folgenden Instruktionen erforderlich ist:

$$\Big(\rhd \left((T^*ST^* \cup T^*)\,\#_{(j+2) \bmod 3}\right)^+ u, v\#_j \to v'\#_{(j+2) \bmod 3},$$
$$w\#_j\left((T^2T^* \cup STT^* \cup T^+ST^+ \cup TT^*S)\#_j\right)^* \lhd\Big).$$

Ferner gibt es zu beiden Instruktionssätzen entsprechende Instruktionen für das Verarbeiten der letzten Konfiguration (es erfolgt keine Konsistenzprüfung mehr):

$$\left(\rhd \left((T^*ST^* \cup T^*)\,\#_j\right)^+ u, v\$ \to v'\#_j, \lhd\right)$$

und

$$\left(\rhd\left(\left(T^*ST^* \cup T^*\right)\#_{(j+2) \bmod 3}\right)^+ u, v\#_j \to v'\#_{(j+2) \bmod 3}, \lhd\right).$$

III Ebenso gibt es für jeweils $j \in \{0,1,2\}$:

$$\Big(\rhd u_1, v_1\#_j \to v_1'\#_{(j+2) \bmod 3},$$
$$w_2\#_j \left((T^2T^* \cup STT^* \cup T^+ST^+ \cup TT^*S)\#_j\right)^* \lhd\Big).$$

IV Das Erkennen eines korrekt reduzierten Wortes leisten die drei Instruktionen

$$\left(\rhd\left(\left(TT \cup ST \cup TS\right)\#_j\right)^+ \lhd, \mathsf{Accept}\right),$$

wenn für alle aufeinander folgenden Konfigurationen x_1x_2 und y_1y_2 entsprechend zur Überführungsfunktion δ einer der folgenden Fälle zutrifft:

- $x_1x_2 \in T^2$ und $y_1 = x_1$ und entweder $y_2 = x_2$ oder $y_2 \in S$ gilt,
- $x_1x_2 \in ST$ und
  - $y_1 = t$ und $y_2 = s$, falls $\delta(x_1, x_2) = (s, t, r)$ bzw.
  - $y_1 = s$ und $y_2 = t$, falls $\delta(x_1, x_2) = (s, t, n)$ ist, gilt,
- $x_1x_2 \in TS$, wobei entweder $y_1 = x_1$ und $y_2 \in (T \cup S)$ oder $y_1 \in S$ und $y_2 = x_1$ ist.

Falls die Turingmaschine Wörter, welche genau aus einem Zeichen bestehen, akzeptiert (d. h. wenn es valid computations gibt, die aus Konfigurationen der Länge 2 bestehen), muss dies noch durch die Instruktion

$$\left(\rhd x_1x_2\$\left((ST\$)^2\right)^+ z_1z_2\$\lhd, \mathsf{Accept}\right),$$

mit den folgenden Eigenschaften beschrieben werden:

- $x_1 \in S$ ist der Startzustand und $x_2 \in T$,
- $\delta(z_1, z_2)$ akzeptiert und beendet die Abarbeitung durch $\mathcal{M}$ und
- alle Konfigurationsübergänge sind von der Form $x_1x_2 \vdash y_1y_2$ mit $x_1, y_1 \in S$, $x_2, y_2 \in T$ und es ist $\delta(x_1, x_2) = (y_1, y_2, n)$.

Korrektheit und Vollständigkeit:

Es ist leicht zu sehen, dass auf einem korrekten Wort w ($\in \mathsf{VALC}_{C'}(\mathcal{M})$) stets exakt eine Meta-Instruktion anwendbar ist (sonst stimmen die Markierungen $\#_j$ nicht überein): Zunächst ist genau Instruktion I anwendbar bzw. ein Wort mit zu kurzen Konfigurationen wird direkt über die entsprechende Regel aus IV akzeptiert. Danach wird – durch wiederholtes Anwenden der

entsprechenden Regel aus II – der Reihe nach von jeder weiteren Konfiguration das letzte Zeichen verglichen und entfernt. Am Ende einer solchen Abarbeitungsfolge kann stets nur entweder Regel IV angewandt oder mit Regeln III eine weitere Abarbeitungsfolge gestartet werden. Dies führt eindeutig zu der beabsichtigten Arbeitsweise und dem Akzeptieren von w durch $\mathcal{M}'$.

Betrachten wir nun Wörter $w \notin \mathsf{VALC}_{C'}(\mathcal{M})$: Zu Beginn ist höchstens Regel I anwendbar (oder $\mathcal{M}$ verwirft sofort). Wenn die Berechnung fortschreitet, ist zumindest gesichert, dass ein Wort der Form $w_1\$w_2\$\cdots w_{2n}\$$ mit $w_1, \ldots, w_{2n} \in T^*ST^+$ vorliegt. Ferner kann man davon ausgehen, dass $|w_1| = \ldots = |w_{2n}|$ gilt, denn sonst werden die Regeln aus IV niemals angewandt werden können und $\mathcal{M}'$ akzeptiert nicht. Außerdem ist schon geprüft, dass es sich bei w_1 bzw. w_{2n} wirklich um eine Start- bzw. Endkonfiguration handelt.[15]

Der einzige noch vorkommende Fehler muss also ein unmöglicher Konfigurationsübergang von einem Wort w_i auf ein Wort w_{i+1} sein ($1 \leq i < 2n$). Nach den vorigen Betrachtungen gibt es ein $l \in \mathbb{N}$, so dass $w_i = x_0 \cdots x_l$, $w_{i+1} = y_0 \cdots y_l$ sind und es genau ein $k \in \mathbb{N}_0$, $k < l$, gibt, an dessen Stelle sich innerhalb der ersten Konfiguration der Zustand $x_k \in S$ befindet.

$\mathcal{M}'$ vergleicht nun w_i mit w_{i+1} vom letzten Zeichen aus bis zum ersten. Wenn der in dieser Richtung als erstes auftretende Fehler den Index k' trägt, lassen sich die folgenden Fälle unterscheiden:

- Für $l \geq k' > k+1$ wird der Fehler das Löschen des $(l+1-k')$-ten Zeichens aus dem Wort w_i verhindern. Die entsprechende Meta-Instruktion wird nicht anwendbar sein und $\mathcal{M}'$ verwirft das Gesamtwort.
- Für $k+1 \geq k' \geq k-1 \geq 0$ wird der Fehler das Löschen des $(l+1-k)$-ten Zeichens aus dem Wort w_i verhindern, da $\mathcal{M}'$ nun genau in der Situation ist, wo alle drei Positionen $k-1, \ldots, k+1$ vollständig miteinander verglichen werden.
- Für $k-1 > k' \geq 0$ wird der Fehler wiederum das Löschen des $(l+1-k')$-ten Zeichens aus dem Wort w_i verhindern und
- im Falle, dass $k = 0$, also $k' \in \{0, 1\}$ ist, wird die Anwendung der entsprechenden Regel aus IV verhindert.

Jeder Fehler im Wort wird also einen undefinierten Übergang verursachen und somit ist gezeigt, dass $\mathcal{M}'$ genau $\mathsf{VALC}_{C'}(\mathcal{M})$ erkennt. □

[15] Es wurde ferner geprüft, ob die in den Konfigurationen vorkommenden, äußersten Stellen mindestens einmal vom Kopf der Turingmaschine erreicht wurden, aber dies spielt für unsere weitere Betrachtung keine Rolle.

Korollar 4.7 *Es sei $\mathcal{M}$ eine beliebige Turingmaschine. Dann gibt es einen RRWW(2)-Automaten, der $\mathsf{VALC}_{C'}(\mathcal{M})$ erkennt und effektiv konstruierbar ist.*

Beweis Im erbrachten Beweis wird stets nur ein Zeichen verändert und ein weiteres gelöscht. Beide liegen stets nebeneinander. Die restlichen Auswirkungen der Fensterverkleinerung können jeweils problemlos durch die Verwendung von zusätzlichen Zuständen kompensiert werden.[16] □

Nach der obigen Vorgehensweise wird es nun gelingen, ein analoges Resultat für den Trade-off zwischen RRWW- und monotonen Restart-Automaten nachzuweisen. Wir verzichten zunächst auf die Ermittlung von Trade-offs zu den deterministischen Restart-Automaten, da sich diese unmittelbar aus den Betrachtungen im Verlauf des nächsten Kapitels ergeben werden.

Satz 4.8 *Der Trade-off zwischen RRWW-Automaten und mon-R(R)WW-Automaten ist nichtrekursiv.*

Beweis Wir wählen S_3 als Menge der Turingmaschinen, $\mathsf{VALC}_{C'}(\mathcal{M})$ als Sprache $L_{\mathcal{M}}$ und die Eigenschaft P von $L(\mathcal{M})$ als Unendlichkeit. Ferner haben wir eben gesehen, wie man einen RRWW-Automaten zur Erkennung von $L_{\mathcal{M}}$ konstruiert, womit es uns möglich ist, S_1 als die Klasse der RRWW-Automaten zu wählen. Nach Lemma 4.2 gibt es in der Menge S_2 der mon-R(R)WW-Automaten genau dann einen Beschreiber für $L_{\mathcal{M}}$, wenn P für $L(\mathcal{M})$ nicht gilt (ist $L(\mathcal{M})$ endlich, so ist $\mathsf{VALC}_{C'}(\mathcal{M})$ ebenfalls endlich, jedoch anderenfalls nicht kontextfrei). □

Im Beweis lässt sich sehen, dass die von S_2 gebildete Sprachklasse mindestens die endlichen Sprachen enthalten muss, damit Satz 4.1 anwendbar ist. Es gilt also:

Korollar 4.9 *Der Trade-off zwischen RRWW-Automaten und allen Restart-Automatenklassen, die Subklassen der mon-R(R)WW-Automaten sind, ist nichtrekursiv.*

Ein entsprechendes Ergebnis zur Beschreibungseffizienz von RRWW-Automaten gegenüber ihren deterministischen Varianten wird sich aus den Ergebnissen des nächsten Kapitels folgern lassen.

Wir schließen diesen Abschnitt mit der Erweiterung der Aussage aus Satz 4.8, welche sich ebenfalls über das Erkennen der Sprache $\mathsf{VALC}_{C'}(\mathcal{M})$ –

[16] In den angegebenen Meta-Instruktionen macht sich dies jedoch nicht bemerkbar. Die höhere Fenstergröße diente uns in obigem Beweis nur zur besseren Darstellbarkeit.

diesmal durch einen RWW-Automaten – begründen lässt. Dazu ist es hilfreich, zunächst den Konstruktionsvorgang aus [19], in welchem die Existenz eines RWW-Automaten zur Erkennung der verwandten Gladkij-Sprache $L_{Gl} = \{w\$w^R\$w \mid w \in \{a,b\}^*\}$ bewiesen wird, zu beschreiben.

Die Verarbeitung wird durch zwei separat vorgestellte Restart-Automaten bewältigt – aus diesen lässt sich unter Einführung eines (weiteren) Sonderzeichens ein einzelner RWW-Automat konstruieren, der die Hintereinanderausführung beider Automaten simuliert (genauer gesagt wird die Abarbeitung durch den ersten Automaten zu einem beliebigen Zeitpunkt unterbrochen und kann ab dort nicht mehr fortgesetzt werden). Der erste Automat vergleicht durch eine sukzessive Kodierung der drei Wortteile die Abfolge der einzelnen Zeichen. Dies gelingt jedoch auf Grund der fehlenden Nachlesemöglichkeit nicht vollständig, weshalb der zweite Automat im Wesentlichen eine Längenprüfung der kodierten Teile nachfolgen lassen muss. Die nähere Beschreibung beider Automaten sieht wie folgt aus:

(1) Von der folgenden Beschreibung ausgenommen sind Wörter unterhalb einer gewissen Grundlänge und solche von der falschen Grundform – diese können ohne Löschvorgang erkannt bzw. verworfen werden. Wir können also in unserer Beschreibung vom ausschließlichen Vorhandensein von Wörtern der Form $\{a,b\}^{\geq 4}\$\{a,b\}^{\geq 4}\$\{a,b\}^{\geq 4}$ ausgehen.

Nun werden in jedem der drei vorhandenen Teilwörter Paare von zwei benachbarten Zeichen in jeweils ein Bandsymbol kodiert (sind die Teilwörter von ungerader Länge, so werden jeweils die letzten drei noch nicht kodierten Zeichen entsprechend zusammengefasst). Das Bandsymbol enthält jeweils zusätzlich zur Information über die Zeichen und ihre Positionen einen binären Wert, mit dessen Hilfe der Kodierungsprozess gesteuert wird:

Zwei (Eingabe-)Buchstaben im Teilwort $i \in \{1,2,3\}$ können nur dann kodiert werden, wenn

- sie – vom Anfang des Teilworts aus gesehen – die ersten noch nicht codierten Zeichen der originalen Eingabe sind,
- in allen Teilwörtern $1, \ldots, i-1$ das jeweils zuletzt kodierte Zeichen dieselbe Buchstabenfolge repräsentiert,
- der binäre Wert des zuletzt kodierten Zeichens aller Teilwörter $1, \ldots, i-1$ übereinstimmt und das zuletzt kodierte Zeichen in Teilwort i den komplementären Wert trägt.

Beispiele:

(i) Wir gehen vom Wort *abbaba*\$*abaaba*\$*abaaba* aus und benennen die Positionen der Buchstabenpaare, deren Codierung im Folgenden

diskutiert werden soll, mit

$$\underset{x_1}{a}\underset{x_2}{bbaba}\$\underset{x_3}{abaaba}\$\underset{x_4}{abaaba}.$$

Zuerst kann nur an der Stelle x_1 gearbeitet werden. ab wird in das Zeichen $[a, b, 0]$ umgeschrieben. Danach geschieht dasselbe an der Stelle x_3 mit der spiegelverkehrt passenden Zeichenkette ba (wird zu $[b, a, 0]$ codiert). An der Stelle x_4 – und damit im gesamten dritten Teilwort – kann so lange nichts verändert werden, bis nicht diese beiden ersten Schritte durchgeführt wurden. Ebenso kann wegen des im zweiten Schritt zugeteilten binären Wertes im zweiten Teilwort so lange nichts mehr verändert werden, bis nicht im ersten Wort mindestens eine neue Änderung vorgenommen wurde. Sobald hier jedoch an der Stelle x_2 die Zeichen ba (zu $[b, a, 1]$) umgeschrieben wurden, muss der Automat verwerfen, denn auf Grund des enthaltenen Fehlers existiert keine Möglichkeit mehr, das zweite Teilwort weiter zu verändern.

(ii) Am Wort

$$\underset{x_1}{a}\underset{x_2}{bba}\underset{x_3}{aba}\underset{x_4}{baba}\$\underset{x_5}{ababba}\$\underset{x_6}{abbaba}$$

lässt sich erkennen, dass auch Wörter außerhalb der Gladkij-Sprache vollständig kodiert werden können. Innerhalb der ersten drei Schritte könnte an den Stellen x_1 bis x_3 kodiert werden, so dass das Wort zu

$$[a, b, 0][b, a, 1][a, b, 0]\underset{x_4}{ba}ba\$\underset{x_5}{ababba}\$\underset{x_6}{abbaba}$$

verändert wird. Hiernach ist es ohne Weiteres möglich, mit der Abarbeitung der Stellen x_5, x_6, x_4, usw. fortzufahren, da die unkodierten Stellen aller drei Teilwörter überein stimmen bzw. zum (zweiten) Zeichen $[a, b, 0]$ passen.

In einem Wort, dessen drei Teilwörter gleich lang sind, führt jede Verarbeitung, in der auf dem ersten Wort mehrere Kodierungsschritte hintereinander ausgeführt werden, auf ein Gesamtwort, welches nicht mehr akzeptiert werden kann. Es kann also nur dann vollständig kodiert werden, wenn es zur Gladkij-Sprache gehört. Damit ist es nun möglich, die vollständig kodierten Wörter mit ungleichen Längen im Nachhinein zu verwerfen.

(2) Der zweite Automat hat nun die folgenden Aufgaben:

(i) Prüfung der gleichen Länge aller Teilwörter und

(ii) Test auf Vollständigkeit der Kodierung.

Bei positiven Ergebnissen akzeptiert er das (umgeschriebene) Wort.

Jurdziński et. al. führen nun an, dass die Sprache $\{a^n\$a^n\$a^n \mid n \in \mathbb{N}\}$ wachsend kontextsensitiv ist und Punkt (ii) nur eine reguläre Bedingung darstellt. Da wachsend kontextsensitive Sprachen unter Schnitt mit regulären Sprachen abgeschlossen und in der Klasse der RWW-Sprachen enthalten sind, gibt es einen RWW-Automaten, der das Gewünschte leistet.

Auf Basis dieser Beweisführung und der Konstruktion aus Lemma 4.6 lässt sich ferner skizzieren, warum es auch einen RWW-Automat zur Erkennung einer valid-computations-Sprache gibt. Um aber – wie gehabt – die technische Konstruktion etwas zu entzerren, benötigen wir eine zu Obigem analoge Aussage zur Synchronisation der Längen der einzelnen Teilwörter. Der Nachweis von $L_{el} = \{w_1\$ \cdots w_m\$ \mid w_1, \ldots, w_m \in A^n \wedge m, n \in \mathbb{N}\}$ als wachsend kontextsensitive Sprache (zu beliebigem Alphabet A) reicht hierzu aus.

Um uns eine einfach zu beschreibende Konstruktion zu ermöglichen, betrachten wir jedoch vorübergehend ein anderes Beschreibungssystem, von dem die effektive Konstruierbarkeit eines jeweils äquivalenten RWW-Automaten bekannt ist:

Nach Buntrock wird die Klasse der wachsend kontextsensitiven Sprachen durch die der sog. schrumpfenden Zweikeller-Automaten charakterisiert. Eine relativ knappe Definition hiervon genügt für unsere Zwecke. Der an einer ausführlicheren Beschreibung interessierte Leser sei auf [4] und [5] verwiesen.

Definition 4.10 Ein *schrumpfender Zweikeller-Automat* (sTPDA) ist ein System $\mathcal{M} = \langle S, A, G, \delta, s_0, \bot, F\rangle$, in dem S eine endliche Zustandsmenge, A ein endliches Eingabe- und G (mit $A \subseteq G$) ein endliches Kelleralphabet, $s_0 \in S$ der Startzustand, $\bot \in G$ das initiale Kellerendesymbol, δ eine totale Abbildung von $S \times G \times G$ in die endlichen Teilmengen von $S \times G^* \times G^*$ und $F \subseteq S$ eine Menge von Endzuständen ist und es ferner eine Gewichtsfunktion $f : G^* \cup S \times G^* \to \mathbb{N}_0$ gibt, die die folgenden Eigenschaften hat:

- $f(\lambda) = 0$ und $f(uv) = f(u) + f(v)$ für alle $u, v \in G^* \cup S \times G^*$ und
- $f(sxy) > f(s'x'y')$, falls $(s', x', y') \in \delta(s, x, y)$ ist ($s, s' \in S$, $x, y \in G$ und $x', y' \in G^*$).

Ein sTPDA beginnt seine Verarbeitung im Startzustand mit – bis auf das Kellerendesymbol – leerem linken Keller und mit dem Eingabewort (und dem Kellerendesymbol) gefüllten rechten Keller. Die Verarbeitung endet, falls die Überführungsfunktion nicht anwendbar ist, was in unserem Fall durch einen leeren Kellerinhalt, also die Löschung eines Kellerendesymbols entstehen kann. Das entsprechend verarbeitete Wort liegt genau dann in

der akzeptierten Sprache, wenn der beim Halten anliegende Zustand ein Endzustand ist.

Lemma 4.11 *Es gibt einen effektiv konstruierbaren sTPDA, der für ein beliebiges Alphabet A mit $\$ \notin A$ die Sprache L_{el} erkennt.*

Beweis Trotz seiner zwei Keller kann sich der Automat nicht wie ein linear beschränkter Automat verhalten und solange zyklisch in jedem Wort nur ein Zeichen löschen, bis er eine Entscheidung treffen kann. In diesem Fall wäre nämlich die Existenz einer geeigneten Gewichtsfunktion ausschließbar, da der Automat auf jeweils beliebig langen Zeichenketten nur Zustandsänderungen durchführt und sich somit ein Zyklus von Überführungen ergibt. Diese können somit ihrerseits nicht alle von echt sinkendem Gewicht sein. Daher ist es nötig, in regelmäßigen Abständen ein Zeichen zu löschen, um wieder in einen höher gewichteten Zustand zurück wechseln zu können – zum Beispiel durch die Löschung jedes zweiten Zeichens eines Teilwortes.

Der Zweikeller-Automat wird also den Inhalt des rechten Kellers unter der Löschung jedes zweiten Eingabezeichens in den linken Keller bringen, wobei er darauf achten muss, dass jedes Teilwort modulo zwei denselben Rest liefert. Auf diese Weise ist das zu prüfende Wort genau dann in L_{el} enthalten, wenn es auch das reduzierte Wort am Ende dieses Vorgangs ist.

Derselbe Vorgang wird nun in umgekehrter Richtung wiederholt und die Reduktionsschritte halten so lange an, bis ein Teilwort nur noch aus einem Zeichen besteht und somit nichts mehr gelöscht werden kann. Handelt es sich um das zu Beginn des neuen Durchgangs zuerst betretene Wort, so prüft der Automat, ob alle anderen Teilwörter ebenfalls von der Länge eins sind (da das Wort noch immer von beliebiger Länge sein kann, wird dann auch der letzte Buchstabe gelöscht, um absteigend gewichtete Übergänge zu gewährleisten). Im Erfolgsfall löscht der Automat die Keller und hält im Endzustand. Ansonsten, bzw. falls es sich beim zu kurzen Teilwort nicht um das erste zu Beginn eines neuen Durchgangs handelt, leert der Automat die Keller und bleibt in einem Nicht-Endzustand.

Die Transitionen der (sogar deterministischen) Überführungsfunktion, in denen kein Zeichen gelöscht werden darf (etwa beim Wechsel der Arbeitsrichtung oder dem Überlaufen eines Trennzeichens oder des jeweils nicht zu löschenden Eingabezeichens), erhalten durch absteigend gewichtete Zustände eine absteigende Gesamtgewichtung. Ein Beispiel hierzu wäre

$$\delta(s_x, a, a) = \{(s_y, aa, \lambda)\} \text{ mit } f(s_y) < f(s_x)$$

für eine Linksbewegung der Kellerinhalte.

Da diese Folgen von Zuständen jeweils nur endlich lang sind, reicht es, den Eingabezeichen stets dasselbe Gewicht zu geben, was den Wechsel vom

höchst- zum niedrigstwertigen Zustand übersteigt. Passend zum obigen Beispiel bietet sich hier $f(a) > 1$ an, wenn das zweite a gelöscht wird:

$$\delta(s_y, a, a) = \{(s_x, a, \lambda)\}.$$

In einer exakten Konstruktion hierzu liegt der Wert eines Zeichens etwa bei vier bis sechs. Die Existenz der passenden Gewichtsfunktion ist also beweisbar. □

Damit sind alle nötigen Vorbereitungen zum Beweis der nächsten Aussage getroffen.

Lemma 4.12 *Es sei $\mathcal{M}$ eine beliebige Turingmaschine. Dann gibt es einen effektiv konstruierbaren RWW-Automaten, der $\mathsf{VALC}_{C'}(\mathcal{M})$ akzeptiert.*

Beweis (Idee)

(1) Die Meta-Instruktionen, die den wesentlichen Beitrag zur Erkennung der Gladkij-Sprache leisten, lassen sich ohne Probleme um die folgenden Funktionalitäten erweitern:

- Vergrößerung des Eingabealphabets auf das von $\mathsf{VALC}_{C'}(\mathcal{M})$,
- Prüfung, ob die in den vorhergehenden Teilwörtern zuletzt kodierten (drei bis vier) Zeichen eine konsistente Berechnung darstellen (an Stelle des reinen Vergleichens der jeweils zuletzt kodierten Zeichenpaare),
- Test, ob in jeder Konfiguration genau ein Zustand vorkommt und die erste Konfiguration eine Startkonfiguration von $\mathcal{M}$ ist.

Die Umsetzung hiervon ist jedoch technisch aufwändig und trägt keineswegs zur besseren Verständlichkeit bei. Die zu Grunde liegenden Ideen finden sich bereits im Beweis zu Lemma 4.6. Ob die letzte Konfiguration im Wort eine Endkonfiguration ist bzw. der in ihr angegebene Bereich keine Leerzeichen mehr enthält, kann dann zweckmäßigerweise durch reguläre Bedingungen im zweiten Automatenteil geprüft werden.

(2) Die im angeführten Beweis verwendete Längenprüfung der einzelnen Teilwörter kann von den in Lemma 4.11 beschriebenen sTDPA ausgeführt werden.

Ob das Wort vollständig kodiert wurde, eine gerade Anzahl von wenigstens vier Teilwörtern besitzt und das letzte Teilwort eine Endkonfiguration darstellt, in der keine Leerzeichen mehr vorkommen, lässt sich wiederum einfach durch einen regulären Ausdruck prüfen.

In [5] findet sich ebenfalls eine effektive Konstruktion von schrumpfenden Zweikeller-Automaten für beliebige, wachsend kontextsensitive

Sprachen, so dass wir aus deren Abgeschlossenheit gegenüber Schnitt mit regulären Sprachen auch die effektive Konstruierbarkeit eines einzigen RWW-Automaten für beide Arbeitsschritte und alle angeführten Teilaufgaben folgern können, was die Idee abschließt.

□

Damit sind wir nun im Stande, das Ergebnis aus Satz 4.8 und Korollar 4.9 passend zu erweitern:

Korollar 4.13 *Der Trade-off zwischen R(R)WW-Automaten und allen Restart-Automatenklassen, die Subklassen der mon-R(R)WW-Automaten sind, ist nichtrekursiv.*

Offen bleibt, welche Aussagen es über die relative Beschreibungseffizienz von RRWW- gegenüber RWW-Automaten gibt. Dieses Problem ist eng mit dem bis heute offenen Problem verknüpft, ob die Sprachklassen $\mathcal{L}$(RRWW) und $\mathcal{L}$(RWW) identisch sind (siehe dazu auch [33]).

4.2 Deterministische vs. monotone Restart-Automaten

Wie der Titel angibt, wird nun unser Hauptaugenmerk auf dem Vergleich zwischen den monotonen und deterministischen Restartautomaten liegen. Da die von beiden Automatentypen akzeptierten Sprachklassen bezüglich Inklusion unvergleichbar sind, stellt sich die Frage, ob dann auch jedes der Systeme gegenüber dem anderen jeweils effizientere Beschreibungen liefern kann. Die positive Antwort sei hier schon vorweg genommen: es sind jeweils sogar beliebig effiziente Beschreibungen möglich.

Des Weiteren lassen sich aus den Ergebnissen auch ein paar mit dem ersten Abschnitt verbundene Folgerungen aufstellen, wie wir im Anschluss daran zeigen werden. Unsere Vorgehensweise wird dabei zunächst dieselbe bleiben: Über die Ausnutzung von Nicht-Semi-Entscheidbarkeiten bzw. mit Hilfe von trennenden Sprachen, welche diese Nicht-Semi-Entscheidbarkeiten implizieren, lassen sich auf die mittlerweile bekannte Art Beweise führen.

Solche trennenden Sprachen zu finden, ist jedoch im Allgemeinen relativ schwierig. Dies liegt vor allem in der Problematik, eine Sprache als von bestimmten Restartautomaten nicht erkennbar nachzuweisen. Gerade im Bezug auf Automaten, die über Sonderzeichen verfügen, fehlen uns Mittel wie etwa die Fehlererhaltung, welche zum Beweis von Lemma 4.3 verwendet wurde. Zur Abgrenzung dessen, was von deterministischen Restartautomaten nicht mehr erkennbar ist, begeben wir uns daher in der Hierarchie um etwa anderthalb Schritte höher:

In [4] wird gezeigt, dass jede wachsend kontextsensitive Sprache von einer *nichtdeterministischen Hilfskellermaschine mit Einwegeingabe und logarithmischer Band- und polynomieller Zeitbeschränkung* akzeptiert wird. Einen solchen Automaten werden wir im Folgenden als OWauxPDA bezeichnen, jedoch zur Vermeidung von Missverständnissen stets die erwähnten Beschränkungen mit angeben.

Da die Klasse GCSL durch wmon-R(R)WW-Automaten charakterisiert wird und somit eine Oberklasse von allen von monotonen als auch allen von deterministischen Restartautomaten erkannten Sprachklassen ist, überträgt sich das Nichtenthalten von Sprachen unmittelbar auf die Erkennungsmächtigkeit von monotonen bzw. deterministischen Restartautomaten. Den Nachweis einer solchen Nichterkennbarkeit liefert Brandenburg in seiner Arbeit [2].

Um einen Bezug zu den valid-computations-Sprachen herzustellen, muss die von ihm als von keinem OWauxPDA (auf logarithmisch beschränktem Band in polynomieller Arbeitszeit) erkennbare Sprache

$$\{w_1 c w_2 c w_1^R c w_2^R \mid w_1, w_2 \in \{a, b\}^*\}$$

zunächst passend erweitert werden.

Vorbereitungen

Es seien A und B zwei Alphabete, die jeweils nicht das Zeichen c enthalten und $\Psi \subseteq A^* \times A^* \times B^*$ eine Relation. Damit legen wir die Sprache L_Ψ als

$$L_\Psi = \left\{ w_1 c w_2 c w_1^R c w_2^R c w_3 \mid w_1, w_2 \in A^*, w_3 \in B^*, (w_1, w_2, w_3) \in \Psi \right\}$$

fest.

Anschaulich ist klar, dass ein OWauxPDA, dessen Arbeitsband logarithmisch (in der Länge der Eingabe) beschränkt ist und der in polynomieller Zeit arbeiten soll, den beliebigen Wortteil w_1 nicht vollständig mit w_1^R vergleichen kann, ohne dabei einen wesentlichen Teil der Information über w_2 zu verlieren (bevor der Vergleich mit w_2^R erfolgen kann). Die Informationsmenge, die sich der OWauxPDA während seiner Verarbeitung wahlfrei merken kann, wird im Folgenden *Speicherzustand* genannt und stellt alle Kombinationen von

- angenommenem Zustand,
- möglichem Inhalt des logarithmisch beschränkten Turingbands und
- sichtbarem, oberstem Kellersymbol

dar.

Der Beweis aus [2], dessen Ideen wir im Wesentlichen folgen, schätzt die Zahl der Speicherzustände nach oben ab und ermittelt einen wesentlichen Zustand (später q'_u genannt), über den mehrere verschiedene Wörter durch ähnliche Abarbeitungen akzeptiert werden, so dass sich auf (in Pumping Lemmata) übliche Weise ein Widerspruchsbeweis führen lässt:

Satz 4.14 *Es seien A und B zwei Alphabete, die jeweils nicht das Zeichen c enthalten, $|A| \geq 2$ und $\Psi \subseteq A^* \times A^* \times B^*$ eine Relation, so dass die Relation $\Psi' = \{(w_1, w_2, w_3) \mid (w_1, w_2, w_3) \in \Psi, |w_1| = |w_2| = |w_3|\}$ für unendlich viele $n \in \mathbb{N}$ alle Paare aus $A^* \times A^*$ als erste Komponenten enthält.*
Dann gibt es keinen OWauxPDA mit Zustandsmenge S, Bandalphabet T und Kelleralphabet G, der in polynomieller Zeit und auf logarithmisch beschränktem Band die Sprache L_Ψ erkennt.

Beweis Wir nehmen an, es gäbe einen solchen Automaten $\mathcal{M}$, der in polynomieller Zeit und auf logarithmisch beschränktem Band arbeitet und L_Ψ erkennt. Weiterhin betrachten wir im Folgenden nur Wörter der Form $w = w_1 c w_2 c w_1^R c w_2^R c w_3$ aus L_Ψ, bei denen $|w_1| = |w_2| = |w_3| = n$ ($n \in \mathbb{N}$) ist und für die das oben beschriebene Ψ' alle Paare aus $A^n \times A^n$ als erste Komponenten enthält.

Als erstes schätzen wir nun die Zahl der Speicherzustände, die in akzeptierenden Berechnungen auf Wörtern einer solchen Länge $5n + 4$ vorkommen, ab:

$$\begin{aligned} |Q_n| &\leq |S| \cdot |T|^{\log(5n+4)} \cdot (\log(5n+4) + 1) \cdot |G| \\ &\leq k_1 \cdot 2^{k_2 \cdot \log(5n+4)} \leq k_1 \cdot (5n+4)^{k_3} \leq n^{k_4}, \end{aligned}$$

für passende Konstanten $k_1, k_2, k_3, k_4 > 0$.

Als nächstes suchen wir nun nach Aussagen darüber, welche Speicherzustände an genügend vielen gleichartig verlaufenden Akzeptiervorgängen beteiligt sind:

Allgemein beginnt $\mathcal{M}$ seine Abarbeitung in der Konfiguration (w, q_0, λ) und baut beim Lesen des Präfixes w_1c einen Kellerinhalt z_1 auf:

$$(w_1cw_2cw_1^Rcw_2^Rcw_3, q_0, \lambda) \vdash^* (w_2cw_1^Rcw_2^Rcw_3, q_1, z_1).$$

Danach gibt es stets eine Zerlegung $w_2c = uvc$, so dass während der Verarbeitung des Präfixes u der obere Teil z_2 des Kellerinhaltes $z_1 = z_2z_3$ gelöscht wird und der untere Teil z_3 nicht. Ferner wird z_3 während der Verarbeitung von vc nicht verändert und zusätzlich ein Teilinhalt z_4 auf den Keller geschrieben, so dass sich die folgende Abarbeitung ergibt:

$$\begin{aligned} (w, q_0, \lambda) &\vdash^* (uvcw_1^Rcw_2^Rcw_3, q_1, z_2z_3) \\ &\vdash^* (vcw_1^Rcw_2^Rcw_3, q_u, z_3) \\ &\vdash^* (w_1^Rcw_2^Rcw_3, q_2, z_4z_3) \vdash^* (\lambda, q_f, \lambda). \end{aligned}$$

Die Kellerinhalte z_1, z_2, z_3, z_4 bzw. Wörter u, v können durchaus leer sein, wobei sich sofort sehen lässt, dass dies für z_1 bzw. z_3 nur in einer geringen Anzahl von Fällen zutreffen kann:

Gibt es mehr als $|Q_n|$ Wörter w_1, so dass der Kellerinhalt z_1 nach Verarbeitung des Teilwortes w_1c leer ist, so lassen sich darunter auch zwei finden, bei denen zusätzlich derselbe Speicherzustand anliegt und sich so ein Gesamtwort findet, das akzeptiert wird, obwohl es nicht zu L_Ψ gehört.

Dieses Beispiel dient uns zur Veranschaulichung der Strategie, welche wir in diesem Beweisabschnitt verfolgen müssen: Wir suchen nach einem Speicherzustand, der von genügend vielen Wörtern w_1 erreicht werden muss. Genauer gesagt: es wird die Existenz eines Zustands q'_u gezeigt, der am Ende der Abarbeitung von w_1cu anliegt, wobei es mehr als $|Q_n|$ viele w_1 gibt und zu jedem mindestens ein seiner obigen Beschreibung entsprechendes u.

Ebenso müssen wir aber gleichzeitig die Zusicherung erhalten, dass sich auch die Zahl der (verschiedenen) Endungen v von w_2 nicht so stark verringert, dass $\mathcal{M}$ den entsprechenden Informationsverlust über seine Speicherzustände kompensieren könnte.

Zu diesem Zweck betrachten wir die folgenden Mengen: Für einen beliebigen Speicherzustand $q_1 \in Q_n$ sei

$$\begin{aligned} P_n(q_1) = \{w_1 \mid |w_1| = n \text{ und es gibt } z_2, z_3, uv, w_3 \text{ mit} \\ (w_1 cuvcw_1^R c(uv)^R cw_3, q_0, \lambda) \vdash^* \\ (uvcw_1^R c(uv)^R cw_3, q_1, z_2 z_3) \vdash^* (\lambda, q_f, \lambda)\}. \end{aligned}$$

Ferner sei für ein beliebiges Wort $w_1 \in A^n$ und einen beliebigen Speicherzustand $q_1 \in Q_n$

$$\begin{aligned} W_n(w_1, q_1) = \big\{uv \;\big|\; |uv| = n \text{ und es gibt } z_2, z_3, w_3 \text{ mit} \\ (w_1 cuvcw_1^R c(uv)^R cw_3, q_0, \lambda) \vdash^* \\ (uvcw_1^R c(uv)^R cw_3, q_1, z_2 z_3) \vdash^* (\lambda, q_f, \lambda)\big\}. \end{aligned}$$

Der Faktorisierung von w_2 entsprechend, definieren wir die folgenden, auf W_n basierenden Mengen:

$$\begin{aligned} U_n(w_1, q_1) = \{u \mid \text{es gibt } v, q_u, q_2, z_4, \text{ so dass } uv \in W_n(w_1, q_1) \text{ ist und} \\ (w_1 cuvcw_1^R c(uv)^R cw_3, q_0, \lambda) \vdash^* (uvcw_1^R c(uv)^R cw_3, q_1, z_2 z_3) \vdash^* \\ (vcw_1^R c(uv)^R cw_3, q_u, z_3) \vdash^* (w_1^R c(uv)^R cw_3, q_2, z_4 z_3) \vdash^* (\lambda, q_f, \lambda)\} \end{aligned}$$

und

$$V_n(w_1, q_1) = \{v \mid \text{es gibt ein } u \in U_n(w_1, q_1), \text{ so dass } uv \in W_n(w_1, q_1) \text{ ist}\}.$$

Über diesen Mengen lassen sich die folgenden Abschätzungen begründen:

1. Es gibt einen Speicherzustand q_1, der von wenigstens $2^{\frac{n}{2}}/|Q_n|$ vielen Präfixen akzeptierter Wörter ($w_1 \in P_n(q_1)$) erreicht wird, so dass es zu jedem w_1 ebenfalls wenigstens $2^{\frac{n}{2}}/|Q_n|$ akzeptierbare Fortsetzungen ($w_2 \in W_n(w_1, q_1)$) gibt.

   Wäre das Gegenteil der Fall, so gäbe es zu jedem Speicherzustand weniger als $2^{\frac{n}{2}}/|Q_n|$ Präfixe mit jeweils maximal $|A|^n$ akzeptierbaren Fortsetzungen. Alle restlichen ($|A|^n - 2^{\frac{n}{2}}/|Q_n|$) Wörter hätten dann jeweils weniger als $2^{\frac{n}{2}}/|Q_n|$ akzeptierbare Fortsetzungen, womit sich insgesamt eine Zahl von weniger als

   $$|A|^n \cdot 2^{\frac{n}{2}}/|Q_n| + \left(|A|^n - 2^{\frac{n}{2}}/|Q_n|\right) \cdot 2^{\frac{n}{2}}/|Q_n| \leq 2|A|^{\frac{3n}{2}}/|Q_n|$$

   ergibt, was der Zusicherung von mindestens $|A|^{2n}$ verschiedenen Kombinationen aus w_1 und w_2 in Wörtern der Sprache L_Ψ widerspräche.

   Im Folgenden sei $q_1' \in Q_n$ der Zustand mit den obigen Eigenschaften, dessen Existenz nun gesichert ist. Dazu seien $w_1' \in A^n$ stets Wörter, für die gilt:

   $$|W_n(w_1', q_1')| \geq 2^{\frac{n}{2}}/|Q_n|.$$

2. Unabhängig davon lässt sich für beliebige $q_1 \in Q_n$ und $w_1 \in P_n(q_1)$ folgern, dass es ein z_2 gibt, für das $|U_n(w_1, q_1)| \leq |Q_n| \cdot |z_2|$ ist. Dies begründet sich darüber, dass durch die $u \in U_n(w_1, q_1)$ nur Präfixe des durch die Verarbeitung von $w_1 c$ erzeugten Kellerinhaltes z_1 gelöscht werden. Das längste solche Präfix bezeichnen wir mit z_2'. Da $\mathcal{M}$ nach Löschungen von gleich langen Präfixen z_2 höchstens $|Q_n|$ verschiedene Zustände inne haben darf, ergibt sich $|U_n(w_1, q_1)| \leq |Q_n| \cdot |z_2'|$.

   Wegen der polynomiellen Zeitbeschränkung folgt daraus eine ebenfalls polynomielle Beschränkung der Länge $|z_2'|$ und somit

   $$|U_n(w_1, q_1)| \leq |Q_n| \cdot n^{k_5} \quad \text{für ein entsprechendes } k_5 \in \mathbb{N}.$$

3. Da $W_n(w_1, q_1)$ eine Teilmenge der Konkatenation von $U_n(w_1, q_1)$ und $V_n(w_1, q_1)$ ist, lässt sich $|W_n(w_1, q_1)| \leq |U_n(w_1, q_1)| \cdot |V_n(w_1, q_1)|$ folgern, was für das von uns gewählte Paar (w_1', q_1') bedeutet, dass

   $$|V_n(w_1', q_1')| \geq \frac{|W_n(w_1', q_1')|}{|U_n(w_1', q_1')|} \geq \frac{2^{\frac{n}{2}}}{|Q_n|} \cdot \frac{1}{|Q_n| \cdot n^{k_5}} \text{ ist.}$$

4. Wenn wir $V_n(w_1', q_1')$ nun in Mengen partitionieren, bei denen am Ende der Verarbeitung derselbe Speicherzustand von $\mathcal{M}$ anliegt, können wir nun die angestrebte Aussage über die Existenz eines Zustands q_u, der von genügend vielen Präfixen $w_1 c u$ erreicht wird, so dass es noch genügend viele akzeptierbare Fortsetzungen v gibt, treffen.

   Wir definieren:

   $$\begin{aligned} V_n'(w_1, q_1, q_u) = \{v \mid\ & v \in V_n(w_1, q_1) \text{ und es gibt ein } u, \text{ so dass} \\ & (w_1 c u v c w_1^R c (uv)^R c w_3, q_0, \lambda) \vdash^* (uvcw_1^R c (uv)^R c w_3, q_1, z_2 z_3) \vdash^* \\ & (vcw_1^R c (uv)^R c w_3, q_u, z_3) \vdash^* (\lambda, q_f, \lambda)\}. \end{aligned}$$

   Unsere bisherigen Betrachtungen haben ergeben, dass es zu q_1' wenigstens $2^{\frac{n}{2}}/|Q_n|$ Wörter $w_1' \in P_n(q_1')$ gibt, die wenigstens $2^{\frac{n}{2}}/|Q_n|$ Fortsetzungen $w_2 \in W_n(w_1', q_1')$ haben. Damit gibt es ebenso viele $w_1' \in P_n(q_1')$, für die

   $$|V_n(w_1', q_1')| \geq 2^{\frac{n}{2}} / \left(|Q_n|^2 \cdot n^{k_5}\right)$$

   ist. Damit gibt es ein zu jedem w_1' ein q_u, so dass

   $$|V_n(w_1', q_1', q_u)| \geq 2^{\frac{n}{2}} / \left(|Q_n|^3 \cdot n^{k_5}\right)$$

   gilt.

Da die obigen q_u i. a. unterschiedlich sind, lässt sich nur sagen, dass es wenigstens $2^{\frac{n}{2}}/|Q_n|^2$ Wörter $w_1' \in P_n(q_1')$ gibt, bei denen dasselbe q_u angenommen wird. Dieses bezeichnen wir mit q_u'.

Zusammengefasst gilt:

$$|V_n(w_1', q_1', q_u')| \geq \frac{2^{\frac{n}{2}}}{|Q_n|^3 \cdot n^{k_5}} \geq \frac{2^{\frac{n}{2}}}{n^{3k_4} \cdot n^{k_5}} > |Q_n| \tag{1}$$

und

$$\left| \left\{ w_1' \;\middle|\; |V_n(w_1', q_1', q_u')| \geq \frac{2^{\frac{n}{2}}}{|Q_n|^3 \cdot n^{k_5}} \right\} \right| \geq \frac{2^{\frac{n}{2}}}{|Q_n|^2} \geq \frac{2^{\frac{n}{2}}}{n^{2k_4}} > |Q_n|. \tag{2}$$

Als Letztes bereiten wir nun die Konstruktion des Widerspruchs vor:

Das von uns skizzierte Verhalten von $\mathcal{M}$ auf einem akzeptierbaren Wort $w_1 cuvcw_1^R c(uv)^R cw_3$ kann sich nach der Verarbeitung von vc im Wesentlichen nur auf zwei verschiedene Weisen fortsetzen. Da am Ende der Berechnung ein leerer Keller vorliegen muss, gibt es einen Zeitpunkt, an dem $\mathcal{M}$ zum ersten Mal das erste Zeichen von z_3 erreicht. Eine von beiden Möglichkeiten ist, dass dies schon während der Verarbeitung des Teilwortes $w_1^R c$ erfolgt. Wir zeigen durch die Widerlegung der folgenden Annahme, dass es stets (genügend viele) Fälle gibt, in denen sich der Automat wie eben beschrieben verhält:

Angenommen, es gebe mehr als $|Q_n|$ Wörter w_1 mit Eigenschaft (2), so dass der Kellerinhalt z_3 während der Verarbeitung von $w_1^R c$ nicht betreten wird. Zum Einen lässt sich dann das Restwort $w_2^R cw_3$ in zwei Teile d und e aufteilen, so dass der Kellerinhalt oberhalb von z_3 während der Verarbeitung von d entfernt wird und es gilt:

$$\begin{aligned}(w, q_0, \lambda) &\vdash^* (w_1^R cw_2^R cw_3, q_2, z_4 z_3) \vdash^* (w_2^R cw_3, q_3, z_5 z_3) \\ &\vdash^* (e, q_4, z_3) \vdash^* (\lambda, q_f, \lambda).\end{aligned}$$

Zum anderen gibt es dann zwei Präfixe $\overline{w}_1$ und $\widehat{w}_1$, die denselben Speicherzustand q_4' erreichen, d. h. wir haben die Berechnungen

$$\begin{aligned}(\overline{w}_1 cuvc\overline{w}_1^R cde, q_0, \lambda) &\vdash^* (uvc\overline{w}_1^R cde, q_1', z_2 z_3) \vdash^* (vc\overline{w}_1^R cde, q_u', z_3) \\ &\vdash^* (\overline{w}_1^R cde, q_2, z_4 z_3) \vdash^* (de, q_3, z_5 z_3) \vdash^* (e, q_4', z_3) \vdash^* (\lambda, q_f, \lambda)\end{aligned}$$

und

$$\begin{aligned}(\widehat{w}_1 c\widehat{u}\widehat{v}c\widehat{w}_1^R c\widehat{d}\widehat{e}, q_0, \lambda) &\vdash^* (\widehat{u}\widehat{v}c\widehat{w}_1^R c\widehat{d}\widehat{e}, q_1', \widehat{z}_2\widehat{z}_3) \vdash^* (\widehat{v}c\widehat{w}_1^R c\widehat{d}\widehat{e}, q_u', \widehat{z}_3) \\ &\vdash^* (\widehat{w}_1^R c\widehat{d}\widehat{e}, \widehat{q}_2, \widehat{z}_4\widehat{z}_3) \vdash^* (\widehat{d}\widehat{e}, \widehat{q}_3, \widehat{z}_5\widehat{z}_3) \vdash^* (\widehat{e}, q_4', \widehat{z}_3) \vdash^* (\lambda, q_f, \lambda),\end{aligned}$$

woraus sich folgern ließe, dass auch $\overline{w}_1 cu\widehat{v}c\widehat{w}_1^R c\widehat{de} \notin L_\Psi$ akzeptiert würde:

$$\begin{aligned}(\overline{w}_1 cu\widehat{v}c\widehat{w}_1^R c\widehat{de}, q_0, \lambda) &\vdash^* (u\widehat{v}c\widehat{w}_1^R c\widehat{de}, q_1', z_2z_3) \vdash^* (\widehat{v}c\widehat{w}_1^R c\widehat{de}, q_u', z_3)\\ &\vdash^* (\widehat{w}_1^R c\widehat{de}, \widehat{q}_2, \widehat{z}_4z_3) \vdash^* (\widehat{de}, \widehat{q}_3, \widehat{z}_5z_3) \vdash^* (e, q_4', z_3) \vdash^* (\lambda, q_f, \lambda).\end{aligned}$$

Nach den Ergebnissen des vorigen Abschnitts gibt es somit stets ein Wort w_1', für welches kein $v \in V_n'(w_1', q_1', q_u')$ existiert, so dass der Kellerinhalt erst nach der Verarbeitung von $w_1'^R c$ bis auf z_3 gelöscht wird. Mit anderen Worten: für alle $|V_n'(w_1', q_1', q_u')| > |Q_n|$ Wörter v wird $\mathcal{M}$ schon während der Verarbeitung von $w_1'^R c$ den Keller bis auf z_3 gelöscht haben.

Analog zu Obigem lässt sich dann das Wort $w_1^R c$ in zwei Teile xy aufteilen, so dass der Kellerinhalt z_4 oberhalb von z_3 während der Verarbeitung von x entfernt wird und es gilt:

$$(w, q_0, \lambda) \vdash^* (w_1^R cw_2^R cw_3, q_2, z_4z_3) \vdash^* (ycw_2^R cw_3, q_x, z_3) \vdash^* (\lambda, q_f, \lambda).$$

Nun muss es aber einen Zustand q_x' geben, der während mehr als einer von $v \in V_n'(w_1', q_1', q_u')$ abhängigen Verarbeitung erreicht wird. Wir nennen die beiden beteiligten Teilworte $\overline{v}$ und $\widehat{v}$ und zeigen mit

$$\begin{aligned}(w_1' cuvcxycw_2^R cw_3, q_0, \lambda) &\vdash^* (uvcxycw_2^R cw_3, q_1', z_2z_3)\\ &\vdash^* (vcxycw_2^R cw_3, q_u', z_3) \vdash^* (xycw_2^R cw_3, q_2, z_4z_3)\\ &\vdash^* (ycw_2^R cw_3, q_x', z_3) \vdash^* (\lambda, q_f, \lambda)\end{aligned}$$

und

$$\begin{aligned}(w_1' c\widehat{uv}c\widehat{x}\widehat{y}c\widehat{w}_2^R cw_3, q_0, \lambda) &\vdash^* (\widehat{uv}c\widehat{x}\widehat{y}c\widehat{w}_2^R cw_3, q_1', z_2z_3)\\ &\vdash^* (\widehat{v}c\widehat{x}\widehat{y}c\widehat{w}_2^R cw_3, q_u', z_3)\\ \vdash^* (\widehat{x}\widehat{y}c\widehat{w}_2^R cw_3, \widehat{q}_2, \widehat{z}_4z_3)&\\ &\vdash^* (\widehat{y}c\widehat{w}_2^R cw_3, q_x', z_3) \vdash^* (\lambda, q_f, \lambda),\end{aligned}$$

dass damit auch die folgende Verarbeitung möglich ist:

$$\begin{aligned}(w_1' cu\widehat{v}c\widehat{x}ycw_2^R cw_3, q_0, \lambda) &\vdash^* (\widehat{v}c\widehat{x}ycw_2^R cw_3, q_u', z_3) \vdash^* (\widehat{x}ycw_2^R cw_3, \widehat{q}_2, \widehat{z}_4z_3)\\ &\vdash^* (ycw_2^R cw_3, q_x', z_3) \vdash^* (\lambda, q_f, \lambda).\end{aligned}$$

Doch w ist kein Element aus L_Ψ und damit gibt es in jedem Fall einen Widerspruch. □

Effizienz gegenüber deterministischen Restartautomaten

Aus obiger Aussage lässt sich nun mit Hilfe des Beweisschemas 4.1 zeigen, dass es Sprachen gibt, deren Beschreibung durch monotone Restart-Automaten gegenüber der durch äquivalente deterministische beliebig präzise sein können:

Satz 4.15 *Der Trade-off von monotonen R(R)WW-Automaten (nichtdeterministischen Kellerautomaten) gegenüber deterministischen R(R)WW-Automaten (Church-Rosser Sprachen) ist nichtrekursiv.*

Beweis Es sei $\mathcal{M}$ eine beliebige Turingmaschine und wir wählen $A = \{a, b\}$, B als Alphabet von $\mathsf{VALC}_{C'}(\mathcal{M})$ und Ψ wie folgt: $(w_1, w_2, w_3) \in \Psi$ genau dann, wenn $w_3 \in \mathsf{VALC}_{C'}(\mathcal{M})$ und $|w_1| = |w_2| = |w_3|$.

Die für unsere Zwecke wesentliche Sprache wird nun das Komplement $\overline{L}_\Psi$ von L_Ψ sein. Ein Wort w gehört zu $\overline{L}_\Psi$, wenn entweder

- w nicht von der Form $w_1cw_2cw_3cw_4cw_5$, wobei $w_1, \ldots, w_4 \in A^*$ und $w_5 \in B^*$ sind

oder wenigstens einer der folgenden Fälle vorliegt:

- das Suffix w_5 ist Element von $\mathsf{INVALC}_{C'}(\mathcal{M})$,
- die Längen der Teilwörter $w_1, \ldots, w_5$ stimmen nicht überein,
- $w_3 \neq w_1^R$ oder $w_4 \neq w_2^R$.

Nach Lemma 4.2 gibt es einen effektiv konstruierbaren Kellerautomaten zur Erkennung von $\mathsf{INVALC}_{C'}(\mathcal{M})$. Weiterhin ist bekannt, dass sich das Vorliegen jeder anderen der obigen Eigenschaften durch Akzeptoren von linear kontextfreien Sprachen prüfen lässt. Damit ergibt sich $\overline{L}_\Psi$ als Vereinigung von endlich vielen linear kontextfreien Sprachen und ist damit ebenfalls linear kontextfrei mit effektiv konstruierbarem Akzeptor.

Wenn sich nun herausstellt, dass $\overline{L}_\Psi$ genau dann von deterministischen R(R)WW-Automaten akzeptiert wird, wenn $L(\mathcal{M})$ endlich ist, liegen mit den folgenden Festlegungen alle Voraussetzungen für die Anwendung von Satz 4.1 vor:
Es sei S_3 die Menge aller Turingmaschinen, S_1 die Klasse der von monotonen R(R)WW-Automaten (bzw. von Kellerautomaten) und S_2 die Klasse der von deterministischen R(R)WW-Automaten erkannten Sprachen (Church-Rosser Sprachen). Ferner wählen wir zu $\mathcal{M} \in S_3$ die Sprache $L_\mathcal{M} = \overline{L}_\Psi$ und die Eigenschaft P von $L(\mathcal{M})$ als die Unendlichkeit.

Bleibt also nur noch zu zeigen, dass es genau dann einen deterministischen R(R)WW-Automaten gibt, der $\overline{L}_\Psi$ akzeptiert, wenn $L(\mathcal{M})$ nicht die Eigenschaft P hat, also endlich ist.

Wenn $L(\mathcal{M})$ endlich ist, so ist es auch $\mathsf{VALC}_{C'}(\mathcal{M})$ und damit auch L_Ψ. Daraus folgt, dass $\overline{L}_\Psi$ regulär ist und damit die Existenz eines entsprechenden Automaten.

Ist $L(\mathcal{M})$ jedoch unendlich, dann können wir die Annahme, es gäbe einen akzeptierenden, deterministischen R(R)WW-Automaten wie folgt zum Widerspruch führen: In [31] wurde gezeigt, dass die Klasse der Church-Rosser Sprachen (also die der deterministischen R(R)WW-Automaten) unter Komplementbildung abgeschlossen ist. Das bedeutet, dass es damit auch einen deterministischen R(R)WW-Automaten (und einen OWauxPDA, der in polynomieller Zeit und auf logarithmisch beschränktem Band arbeitet) geben würde, welcher L_Ψ erkennt – dies ist jedoch nach Satz 4.14 nicht möglich, da Ψ nun dessen Voraussetzungen allesamt erfüllt. □

Da wir im vorigen Beweis nur benutzt haben, dass $\overline{L}_\Psi$ eine (effektiv konstruierbare) linear kontextfreie Sprache ist, folgt trivialerweise die umfassendere Aussage:

Korollar 4.16 *Der Trade-off von nichtdeterministischen one-turn-Kellerautomaten gegenüber deterministischen R(R)WW-Automaten (den Church-Rosser Sprachen) ist nichtrekursiv.*

In Anlehnung an die Konstruktion des RRWW-Automaten für die Sprache $\mathsf{VALC}_{C'}(\mathcal{M})$ in Lemma 4.6 lässt sich nun sogar zeigen, dass auch L_Ψ – mit der Wahl $(w_1, w_2, w_3) \in \Psi$ genau dann, wenn $w_3 \in \mathsf{VALC}_{C'}(\mathcal{M})$ und $|w_1| = |w_2| = |w_3|$ – von einem effektiv konstruierbaren RRWW-Automaten erkannt wird. Da die Vorgehensweise identisch zu der im vorigen Kapitel beschriebenen ist, betrachten wir lediglich die Konstruktionsidee und den daraus folgenden Beweisansatz.

Korollar 4.17 *Der Trade-off von RRWW-Automaten gegenüber der Klasse der wachsend-kontextsensitiven Sprachen (wmon-R(R)WW-Automaten) ist nichtrekursiv.*

Beweis Die Prüfung der Grundstruktur des Wortes lässt sich ohne weiteres auf das Vorhandensein der Wörter w_1 bis w_2^R erweitern. Der Automat setzt seine Anfangsmarkierung nur dann, wenn alle diese Wörter untereinander und zum Wort w_3 modulo 3 dieselbe Länge besitzen.

Jeder Verkürzung von w_3 um ein Zeichen (in fortlaufender Konfigurationsreihenfolge) erfolgt nach den entsprechend vier Verkürzungen der Wörter w_1 bis w_2^R. Während dieser Verkürzungen werden ebenfalls die beteiligten Trennzeichen umgeschrieben und die entfernten Zeichen auf Gleichheit getestet.

Es lässt sich also auch für die Sprache L_Ψ ein erkennender RRWW-Automat konstruieren, weshalb man im Beweis zu Satz 4.8 als System S_2 auch das der wachsend kontextsensitiven Sprachen bzw. der schwach monotonen Restart-Automaten verwenden kann. □

Es ist bisher nicht bekannt, ob sich eine analoge Aussage auch über den Trade-off von RWW-Automaten gegenüber GCSL treffen lässt. Wenn sich ein Beweis nach obigem Muster findet, so müsste es darin gelingen, die jeweils vorkommende Zeichenfolge- und Längenprüfung zu synchronisieren. Dieses erscheint jedoch eher unwahrscheinlich. Trotzdem lässt sich zumindest das Folgende sagen:

Korollar 4.18 *Mindestens eine der beiden folgenden Aussagen ist korrekt:*

(1) Der Trade-off von RWW-Automaten gegenüber der Klasse der wachsendkontextsensitiven Sprachen (wmon-R(R)WW-Automaten) ist nichtrekursiv.

(2) Der Trade-off von RRWW-Automaten gegenüber RWW-Automaten ist nichtrekursiv.

Beweis Angenommen, beide obige Aussagen sind falsch. Dann lässt sich jede Sprache L aus dem Schnitt von $\mathcal{L}(\mathsf{RRWW})$ und $\mathcal{L}(\mathsf{RWW})$ durch einen RWW-Automaten mit rekursiv beschränkter Größe darstellen. Genau dann, wenn L zusätzlich in $\mathcal{L}(\mathsf{wmon\text{-}R(R)WW})$ liegt, gibt es aber auch einen wmon-R(R)WW-Automaten mit rekursiv beschränkter Größe, der L erkennt. Da somit alle Sprachen aus dem Schnitt von $\mathcal{L}(\mathsf{RRWW})$ und $\mathcal{L}(\mathsf{wmon\text{-}R(R)WW})$ durch wmon-R(R)WW-Automaten mit rekursiv beschränkter Größe erkannt werden können, ist dies ein Widerspruch zu Korollar 4.17. □

Effizienz von deterministischen Restartautomaten

Darüber hinaus sind wir aber nun – analog zu Kapitel 3.3 – eher daran interessiert, ebenfalls einen Beleg der Ausdrucksmächtigkeit von deterministischen R(R)WW-Automaten gegenüber (one-turn-) Kellerautomaten bzw. monotonen R(R)WW-Automaten zu haben. Dies gelingt mit Hilfe einer passenden Nicht-Semi-Entscheidbarkeit:

Lemma 4.19 *Von einer beliebigen Church-Rosser Sprache L ist nicht semientscheidbar, ob sie nicht kontextfrei ist.*

Beweis Wir gehen zunächst von gegenteiligen Annahme aus. Dann sei $\mathcal{M}$ eine beliebige Turingmaschine und w ein beliebiges Eingabewort. In [28] wurde gezeigt, dass sich aus $\mathcal{M}$ und w ein endliches Church-Rosser Thue System T und ein Wort w' konstruieren lässt, so dass die Kongruenzklasse $[w']_T$ genau dann endlich ist, wenn $\mathcal{M}$ auf w hält.

In demselben Artikel wird die beschriebene Konstruktion nachträglich modifiziert und damit sogar nachgewiesen, dass $[w']_T$ genau dann kontextfrei ist, wenn sie endlich ist.

Nach unserer Annahme wäre aber semi-entscheidbar, ob $[w']_T$ nicht kontextfrei und damit nicht endlich ist. Ebenso wäre nun semi-entscheidbar, ob $\mathcal{M}$ auf w nicht hält, was aber die Entscheidbarkeit des Halteproblems impliziert. □

Mit diesem Nachweis haben wir eine passende Nicht-Semientscheidbarkeit gefunden, welche als Eigenschaft P einen Beweis des folgenden Satzes ermöglicht.

Satz 4.20 *Der Trade-off deterministischer R(R)WW-Automaten (Church-Rosser Sprachen) gegenüber monotonen R(R)WW-Automaten (nichtdeterministischen Kellerautomaten) ist nichtrekursiv.*

Beweis Wie gewohnt, vereinbaren wir die folgenden Festlegungen passend zu Lemma 4.1:

Es sei S_1 die Klasse der deterministischen R(R)WW-Automaten, S_2 die Klasse der monotonen R(R)WW-Automaten und $S_3 = S_1$.

Zu einem Beschreiber $\mathcal{M} \in S_3$ sei $L_{\mathcal{M}}$ die Sprache $L(\mathcal{M})$ und die Eigenschaft P von $L(\mathcal{M})$ ist es, nicht kontextfrei zu sein; also keinen Beschreiber aus S_2 zu besitzen. □

Da im Beweis von 4.19 die Eigenschaften Endlichkeit und Kontextfreiheit der Klasse $[w']_T$ äquivalent sind, eignen sich die gezogenen Schlüsse auch zum Beweis der einiger verwandter Aussagen, wie z. B. über den nichtrekursiven Trade-off von RRWW-Automaten gegenüber monotonen R(R)WW-Automaten oder von deterministischen R(R)WW-Automaten (Church-Rosser Sprachen) gegenüber deterministischen, monotonen R(R)WW-Automaten (deterministischen Kellerautomaten) bzw. 1-turn-Kellerautomaten.

Ersteres wurde jedoch bereits auf dem Weg über Lemma 4.6, welches auch als Basis für weitere Folgerungen (u. a. Korollar 4.17) diente, nachgeweisen. Zweiteres wird sich ohnehin im Rahmen der folgenden Abschnitte (beim Weiterverfolgen der schon in [21] beschriebenen Vorgehensweise) ergeben. Da es zu weit vom eigentlichen Ziel unserer Betrachtungen wegführt, verzichten wir an dieser Stelle auf einen Beweis der verbleibenden Aussage:

Vermutung *Der Trade-off von RWW-Automaten gegenüber 1-turn-Kellerautomaten ist ebenfalls nichtrekursiv.*

Um weitere Ergebnisse an signifikant anderen Stellen zu erhalten, ist es nun jedoch erforderlich, das zweite Nachweiskriterium für nichtrekursive Tradeoffs (aus [20]) zum Einsatz zu bringen.

Satz 4.21 (Kutrib) *Es seien S_1 und S_2 zwei Beschreibungssysteme. Wenn es eine rekursive Funktion $\phi : \mathbb{N} \to \mathbb{N}$ gibt, so dass für beliebige Turingmaschinen $\mathcal{M}$*

(i) ein Beschreiber $\mathcal{D}_1$ in S_1 effektiv konstruierbar ist und

(ii) dass, falls $\mathcal{M}$ auf dem leeren Wort hält und dazu t verschiedene Felder des Eingabebandes verwendet, es zum einen mindestens einen Beschreiber von $L(\mathcal{D}_1)$ in S_2 gibt und zum anderen $\phi(c(\mathcal{D}_2)) \geq t$ für alle Beschreiber $\mathcal{D}_2 \in S_2$ von $L(\mathcal{D}_1)$ gilt,

dann ist der Trade-off zwischen S_1 und S_2 nichtrekursiv.

Der Widerspruchsbeweis hierzu konstruiert auf Basis der Annahme einer rekursiven Beschränkung einen Entscheidungsalgorithmus für das Halteproblem der Maschine $\mathcal{M}$ auf dem leeren Wort:

- Wenn es eine rekursive Funktion f gäbe, die den Größenzuwachs beim Wechsel vom System S_1 nach S_2 beschränkt, dann ließe sich effektiv zu jeder Turingmaschine $\mathcal{M}$ ein Beschreiber $\mathcal{D}_1 \in S_1$ konstruieren (Eigenschaft (i)) und somit die Zahl $f(c(\mathcal{D}_1))$ berechnen.
- Im System S_2 gäbe es dann nur endlich viele Beschreiber $\mathcal{D}'$, deren Größe $f(c(\mathcal{D}_1))$ nicht übersteigt. Da ϕ rekursiv ist, ließe sich auch die folgende Zahl ermitteln:

  $$t' = \max\{\phi(c(\mathcal{D}')) \mid \mathcal{D}' \in S_2, \text{ mit } c(\mathcal{D}') \leq f(c(\mathcal{D}_1))\}.$$

  Das bedeutet, es gäbe höchstens t' verschiedene Bandpositionen, die die Maschine $\mathcal{M}$ im Rahmen einer haltenden Berechnung auf dem leeren Wort annimmt.
- Wenn man nun das Verhalten von $\mathcal{M}$ auf dem leeren Wort und innerhalb des durch t' begrenzten Bereichs simulierte, dann hätte man schon den besagten Entscheidungsalgorithmus geschaffen:

  Entweder verfällt die Maschine $\mathcal{M}$ in eine Endlosschleife oder sie hält innerhalb der vorgegebenen Grenzen an oder sie überläuft diese. In den ersten beiden Fällen ergibt sich unmittelbar die gewünschte Aussage bezüglich des Haltens von $\mathcal{M}$ auf dem leeren Wort.
  Und wenn $\mathcal{M}$ mehr als t' Positionen betritt, kann es in S_2 keinen Beschreiber für $L(\mathcal{D}_1)$ geben und es folgt nach Eigenschaft (ii), dass $\mathcal{M}$ nicht hält.

In [43] findet sich ein passendes Hilfsmittel zur Größenabschätzung von deterministischen Kellerautomaten.

Lemma 4.22 (Valiant) *Es sei $\mathcal{M}$ ein deterministischer Kellerautomat mit Zustandsmenge S und Kelleralphabet T. Wenn das Präfix w ein kürzestes Wort ist, so dass für zwei verschiedene Eingabezeichen c und d beide Wörter wc und wd akzeptiert werden, dann gibt es eine positive Zahl k, so dass gilt: $|S| \cdot |T| \geq (\log |w|)^k$.*

Damit wird es uns nun möglich sein, eine geeignete, rekursive Funktion ϕ zu finden.

Satz 4.23 *Der Trade-off von deterministischen R(R)WW-Automaten (den Church-Rosser Sprachen) gegenüber deterministischen, monotonen R(R)-(W)(W)-Automaten (deterministischen Kellerautomaten) ist nichtrekursiv.*

Beweis Es sei $\mathcal{M}$ eine Turingmaschine mit Zustandsmenge S, Bandalphabet T und Startzustand s_0. Nach [43] (siehe Lemma 4.2) sind $\mathsf{VALC}_{A_1}(\mathcal{M})$ und $\mathsf{VALC}_{A_2}(\mathcal{M})$ deterministisch kontextfreie Sprachen mit jeweils effektiv konstruierbarem Kellerautomaten. Aus letzteren lassen sich ohne weiteres zwei Automaten konstruieren, die $\mathsf{VALC}_{A_1}(\mathcal{M})^R$ bzw. $\mathsf{VALC}_{A_2}(\mathcal{M})^R$ erkennen: dazu müssen nur die Übergänge verändert werden, an denen die Überführungsfunktion von $\mathcal{M}$ beteiligt ist und ferner muss man berücksichtigen, dass nun die Prüfung der Anfangs- bzw. Endkonfiguration von $\mathcal{M}$ am jeweils anderen Ende des Gesamtwortes zu erfolgen hat.

Da man sehen kann, dass sich ein deterministischer Kellerautomat zur Erkennung des Schnitts zwischen einer deterministisch kontextfreien und einer regulären Sprache stets effektiv konstruieren lässt, gibt es zwei deterministische Kellerautomaten für $L_1 = \mathsf{VALC}_{A_1}(\mathcal{M})^R \cap (T^*ST^*\$)^*s_0$ und $L_2 = \mathsf{VALC}_{A_2}(\mathcal{M})^R \cap (T^*ST^*\$)^*s_0$.

Ebenso ist die Familie der deterministisch kontextfreien Sprachen bezüglich markierter Vereinigung effektiv abgeschlossen, so dass sich, für zwei noch nicht vorkommende Zeichen c und d, die Sprache $L = cL_1 \cup dL_2$ durch einen effektiv konstruierbaren Kellerautomaten beschreiben lässt.

In [16] wird gezeigt, wie sich aus einem deterministischen Kellerautomaten (über LR-Grammatiken) ein äquivalenter deterministischer und monotoner R-Automat (und umgekehrt) konstruieren lässt.
Ferner wird in [24] angegeben, dass die (umgebende) Klasse der von deterministischen R(R)WW-Automaten erkannten Sprachen effektiv unter Spiegelung abgeschlossen ist. Es folgt, dass sich ein deterministischer R(R)WW-Automat $\mathcal{D}_1$ zur Erkennung von L^R konstruieren lässt, was Eigenschaft (i) sichert.

Wenn $\mathcal{M}$ nun das leere Wort akzeptiert, enthält der Schnitt der beiden Sprachen L_1^R und L_2^R genau ein Wort w. Damit kann es kein Wort der Länge $2|w|$ geben, das Präfix in beiden Sprachen L_1^R und L_2^R ist (ein solches müsste stets mit dem Wort w beginnen, dürfte jedoch zu der letzten, in w vorkommenden Konfiguration keinerlei Folgekonfiguration haben). Damit gibt es zu $\mathcal{M}$ einen deterministischen Kellerautomaten (oder einen deterministischen, monotonen Restart-Automaten), der L^R erkennt, weil er bereits auf den ersten $2|w|$ Zeichen unterscheiden kann, in welcher der beiden Sprachen L_1^Rc oder L_2^Rd das untersuchte Wort überhaupt liegen kann. Es gibt damit also wenigstens einen Beschreiber $\mathcal{D}_2$ für $L^R = L(\mathcal{D}_1)$.

Mit Lemma 4.22 ergibt sich, dass das Produkt der Zustandszahl und der Anzahl der Kellersymbole für jeden L^R erkennenden, deterministischen Kellerautomaten größer als $(\log |w|)^k$ (für ein positives k) sein muss. Damit lässt sich die rekursive Funktion ϕ, die (ii) erfüllt, leicht bestimmen und Satz 4.21 ist anwendbar. □

Dass auch die Klasse der monotonen R(R)WW-Automaten beliebig effizientere Beschreibungen als die der deterministischen, monotonen Restart-Automaten bietet, lässt sich daran erkennen, dass schon eindeutige, kontextfreie Grammatiken beliebig effizientere Beschreibungen als deterministische Kellerautomaten liefern können, wie bereits in [43] nachgewiesen wurde. Die Klasse der von eindeutigen, kontextfreien Grammatiken erzeugten Sprachen ist jedoch (echt) in der der kontextfreien Sprachen enthalten. Es folgt:

Satz 4.24 (Valiant) *Der Trade-off von eindeutigen, kontextfreien Grammatiken (und damit auch nichtdeterministischen Kellerautomaten bzw. monotonen R(R)WW-Automaten) gegenüber deterministischen Kellerautomaten (deterministischen, monotonen R(R)(W)(W)-Automaten) ist nichtrekursiv.*

Wir sehen also, dass alle diese Ressourcen – im Zusammenspiel mit genügend mächtigen Grundtypen von Restart-Automaten – wesentlich sein können; was sowohl die Ausdrucksmächtigkeit als auch die Beschreibungseffizienz betrifft.

4.3 Weitere nichtrekursive Trade-offs

Legt man die in den beiden vorausgegangenen Abschnitten verwendeten Techniken zu Grunde, so lassen sich manche vergleichbare Nachweise in Bezug auf weniger prominente Automatenklassen auch mit Hilfe von spezifischen Besonderheiten führen. Solche sind etwa passende Nicht-Semi-Entscheidbarkeiten (Beweis nach Schema 1) oder Arbeitsweisen, die eine günstige Abschätzung der Mindestgröße eines erkennenden Automaten (im Sinne von Schema 2) ermöglichen. Die erste Hälfte dieses Abschnitts betrachtet nun solche Klassen, die noch über recht beweisfreundliche Eigenschaften verfügen. Im Unterschied dazu werden wir in der zweiten Hälfte des Abschnitts bestrebt sein, die bisher geführten Beweise auf Automaten, welche ohne Sonderzeichen auskommen müssen, zu übertragen.

Automatenklassen mit speziellen Eigenschaften

Lemma 4.25 *Es sei L eine beliebige, von einem RR-Automaten erkannte Sprache. Dann ist nicht semi-entscheidbar, ob L deterministisch kontextfrei bzw. nicht deterministisch kontextfrei ist.*

Beweis Es sei $\mathcal{M}$ eine Turingmaschine mit Zustandsmenge S und Bandalphabet T und die Sprache $L = cL_1^R \cup L_2^R$ gegeben durch die Sprachen $L_1 = \mathsf{INVALC}_{A_1}(\mathcal{M})$ und $L_2 = \mathsf{INVALC}_{A_2}(\mathcal{M})$ (mit $c \notin L_1 \cup L_2$).

Wir betrachten nun die Spiegelung von L: Zum einen können wir stets einen effektiv konstruierbaren RR-Automaten angeben, der L^R erkennt und zum anderen lässt sich zeigen, dass L^R genau dann deterministisch kontextfrei ist, wenn $\mathcal{M}$ eine endliche Sprache akzeptiert. Letzteres Problem ist – genauso wie sein Gegenteil, die Unendlichkeit einer rekursiv aufzählbaren Sprache – nicht semi-entscheidbar, was den Beweis schließen wird.

Die Begründung der effektiven Konstruierbarkeit von L^R gelingt ähnlich zum Beweis von Satz 4.23: Da $\mathsf{VALC}_{A_1}(\mathcal{M})$ und $\mathsf{VALC}_{A_2}(\mathcal{M})$ deterministisch kontextfreie Sprachen mit effektiv konstruierbarem, deterministischem Kellerautomaten sind, lässt sich sowohl für ihr Komplement, als auch für die markierte Vereinigung der beiden Komplemente effektiv ein DPDA konstruieren. Damit ist nach [16] ein äquivalenter deterministischer und monotoner R-Automat zur Erkennung von L konstruierbar.

In [35] wird – ebenfalls konstruktiv – gezeigt, dass es zu jedem RR-Automaten einen äquivalenten RL-Automaten[17] und umgekehrt gibt. Hieraus lässt sich also leicht ein RL-Automat und damit auch ein RR-Automat zur

[17] Ein Restart-Automat, dem vor und nach dem Rewrite-Schritt auch Fensterbewegungen nach links gestattet sind.

Erkennung von L^R konstruieren (ersterer läuft zuerst ans linke Bandende und arbeitet dann gespiegelt zum Akzeptor von L).

Damit bleibt nur noch zu zeigen, dass L^R genau dann deterministisch kontextfrei ist, wenn die Sprache von $\mathcal{M}$ endlich ist.

Wenn $|L(\mathcal{M})| < \infty$, dann gibt es nur endlich viele Wörter $w \in \mathsf{VALC}_A(\mathcal{M})$ und es sei l die Länge des längsten Wortes. Wir zeigen nun, dass das Komplement von L^R (bezüglich $\Sigma \cup \{c\}$ mit $\Sigma = S \cup T \cup \{\$\}$) und damit auch L^R selbst deterministisch kontextfrei ist. Es gilt:

$$\begin{aligned}\overline{L^R} &= \overline{\mathsf{INVALC}_{A_1}(\mathcal{M}) \cdot c \cup \mathsf{INVALC}_{A_2}(\mathcal{M})} \\ &= \Sigma^* c \Sigma^* c (\Sigma \cup \{c\})^* \cup \Sigma^* c \Sigma^+ \cup \mathsf{VALC}_{A_1}(\mathcal{M}) \cdot c \cup \mathsf{VALC}_{A_2}(\mathcal{M}).\end{aligned}$$

Analog zum Beweis von Satz 4.23 kann ein deterministischer Kellerautomat innerhalb der ersten $2l$ Zeichen jedes Wortes entscheiden, zu welcher der beiden Sprachen $\mathsf{VALC}_{A_1}(\mathcal{M})$ bzw. $\mathsf{VALC}_{A_2}(\mathcal{M})$ das Wort **nicht** gehören kann. Damit gibt es einen Kellerautomaten, der die Vereinigung der nicht auszuschließenden Sprache ($\mathsf{VALC}_{A_1}(\mathcal{M}) \cdot c$ oder $\mathsf{VALC}_{A_2}(\mathcal{M})$) mit den restlichen, regulären Sprachen ($\Sigma^* c \Sigma^* c (\Sigma \cup \{c\})^*$ und $\Sigma^* c \Sigma^+$) erkennen kann.

Geht man – für den zweiten Teil des Nachweises – nun davon aus, dass L^R eine deterministisch kontextfreie Sprache ist, dann gilt dies auch für die Konkatenation mit der endlichen Sprache $\{c, cc\}$. Der Schnitt mit der regulären Sprache $\Sigma^* \cdot cc$ ist ebenso deterministisch kontextfrei und enthält genau die Wörter $w \in (\mathsf{INVALC}_{A_1}(\mathcal{M}) \cup \mathsf{INVALC}_{A_2}(\mathcal{M})) \cdot cc$.

Da die Familie der deterministisch kontextfreien Sprachen bezüglich Rechtsquotient mit einem festen Wort abgeschlossen ist, können wir das Suffix cc von den Wörtern w entfernen und erhalten dann die Aussage, dass sowohl die Sprache $\mathsf{INVALC}_{A_1}(\mathcal{M}) \cup \mathsf{INVALC}_{A_2}(\mathcal{M})$ als auch ihr Komplement $\mathsf{VALC}_A(\mathcal{M})$ deterministisch kontextfrei sind. Nach Lemma 4.2 ist Letzteres jedoch genau dann nicht kontextfrei, wenn $L(\mathcal{M})$ unendlich ist, womit $|L(\mathcal{M})| < \infty$ folgt. □

Nun sind wir in der Lage, Satz 4.1 ein weiteres Mal anwenden zu können.

Satz 4.26 *Der Trade-off von RR-Automaten gegenüber deterministischen, monotonen R(R)(W)(W)-Automaten (deterministischen Kellerautomaten) ist nichtrekursiv.*

Beweis Es sei S_1 die Klasse der RR-Automaten (die effektive Konstruierbarkeit folgt aus den im vorigen Lemma angestellten Überlegungen), S_2 die Klasse der deterministischen, monotonen R(R)(W)(W)-Automaten und $S_3 = S_1$.

Zu einem Beschreiber $\mathcal{M} \in S_3$ sei $L_{\mathcal{M}}$ die Sprache $L(\mathcal{M})$ und die Eigenschaft P von $L(\mathcal{M})$ ist es, nicht deterministisch kontextfrei zu sein; also keinen Beschreiber aus S_2 zu besitzen. □

Wir schließen diesen Abschnitt mit einer Übertragung von Lemma 4.22 und den sich daraus ergebenden Folgerungen bezüglich einiger weiterer, deterministischer Restart-Automatenklassen. Wie im Beweis zu Lemma 4.6 vereinfachen wir uns die Arbeit, indem wir eine leicht angepasste Form der valid computations betrachten.

Für jedes Wort $w_1\$w_2^R\$\cdots w_{2n-1}\$w_{2n}^R \in \mathsf{VALC}_A(\mathcal{M})$ gibt es genau ein Wort $w = u_1w_1v_1\$v_2^Rw_2^Ru_2^R\$\cdots u_{2n-1}w_{2n-1}v_{2n-1}\$w_{2n}^R$, so dass

- $u_1, v_1, \ldots, u_{2n-1}, v_{2n-1} \in \{\sqcup\}^*$,
- $|u_1w_1v_1| = |u_2w_2v_2| = \cdots = |u_{2n-1}w_{2n-1}v_{2n-1}| = |w_{2n}|$,
- für alle $i \in \{1, \ldots, 2(n-1)\}$ die Konfiguration $u_{i+1}w_{i+1}v_{i+1}$ die Folgekonfiguration[18] von $u_iw_iv_i$ und
- w_{2n} die Folgekonfiguration von $u_{2n-1}w_{2n-1}v_{2n-1}$ ist.

Die Menge dieser Wörter bezeichnen wir als $\mathsf{VALC}_{A'}(\mathcal{M})$.

Durch analoge Modifikation erhalten wir aus den Sprachen $\mathsf{VALC}_{A_1}(\mathcal{M})$ und $\mathsf{VALC}_{A_2}(\mathcal{M})$ die Sprachen $\mathsf{VALC}_{A_1'}(\mathcal{M})$ und $\mathsf{VALC}_{A_2'}(\mathcal{M})$, wobei es jedoch zu jeweils einem Wort aus $\mathsf{VALC}_{A_2}(\mathcal{M})$ ($\mathsf{VALC}_{A_1}(\mathcal{M})$) beliebig viele Wörter aus $\mathsf{VALC}_{A_2'}(\mathcal{M})$ ($\mathsf{VALC}_{A_1'}(\mathcal{M})$) gibt, da wir alle bis auf die letzten (die letzten beiden) Konfigurationen mit umgebenden Leerzeichen auffüllen können. Wesentlich dabei ist, dass die benachbarten Teilwörter, die Folgekonfigurationen voneinander sein sollen, damit noch immer automatisch von gleicher Gesamtlänge sind.

Es ergibt sich, analog zu Eigenschaft (6) aus Lemma 4.2, dass $\mathsf{VALC}_{A'}(\mathcal{M})$ der Schnitt von $\mathsf{VALC}_{A_1'}(\mathcal{M})$ und $\mathsf{VALC}_{A_2'}(\mathcal{M})$ ist und dass Letztere beide deterministisch kontextfreie Sprachen mit effektiv konstruierbarem DPDA sind. Wir können nun mit unserem Vorhaben fortfahren.

Lemma 4.27 *Es sei $\mathcal{M}$ eine Turingmaschine mit Zustandsmenge S, Bandalphabet T und Startzustand s_0, die das leere Wort in einer Berechnung mit Endkonfiguration u akzeptiert. Dann gibt es keinen deterministischen R(R)(W)-Automaten, dessen Produkt aus Zustandszahl und möglichen Fensterinhalten kleiner als $|u|$ ist und der die folgende Sprache L erkennt:*

Zu zwei Zeichen $c, d \notin S \cup T \cup \{\$\}$ ist $L = L_1c \cup L_2d$ mit $L_1 = \mathsf{VALC}_{A_1'}(\mathcal{M}) \cap \{\sqcup\}^*s_0\{\sqcup\}^*(\$T^*ST^*)^*$ und $L_2 = \mathsf{VALC}_{A_2'}(\mathcal{M}) \cap \{\sqcup\}^*s_0\{\sqcup\}^*(\$T^*ST^*)^*$.

Beweis Der Schnitt von L_1 und L_2 enthält genau das in $\mathsf{VALC}_{A'}(\mathcal{M})$ vorkommende Wort w, welches die akzeptierende Verarbeitung auf dem leeren

[18] Das Wort Folgekonfiguration beinhaltet die in Abschnitt 4.1 geprägte Sichtweise, dass die absolute Bandposition des Wortes erhalten bleibt.

Wort darstellt. Wir gehen vom Gegenteil der Behauptung aus und zeigen, dass jeglicher Automat mit geringerer Größe beim Erkennen von L Fehler macht:

Da die Länge von w größer oder gleich $4|u| + 3$ sein muss, die Größe des Produktes aus Zustandszahl und Zahl der möglichen Fensterinhalte jedoch nicht größer als $|u|$ ist, können die Worte wc und wd nicht ohne Rewrite-Schritte erkannt werden. Dies begründet sich darauf, dass w stets lang genug ist, so dass man ein (nichtleeres) Teilwort davon pumpen kann. Damit wäre aber das gepumpte Wort mit beiden möglichen Fortsetzungen c und d akzeptierbar und $L_1 \cap L_2$ enthielte mehr als ein Wort.

Wir können also davon ausgehen, dass auf w Rewrite-Schritte durchgeführt werden. Dann aber lässt sich zeigen, dass jeder deterministische Restart-Automat, der keine Sonderzeichen schreiben darf, die Korrektheitserhaltung verletzt, denn jegliches Umschreiben bewirkt eine Verkürzung von mindestens einer, aber höchstens zwei Konfigurationen.

In jeder Verarbeitung, in der das abschließende Zeichen c bzw. d noch nicht erreicht wurde, ist ein Verändern von einer (bis maximal zwei gleichzeitig) der $2n$ Konfigurationen unmöglich: In jedem Fall wird eine der beteiligten Konfigurationen verkürzt und kann dadurch weder Vorgänger- nach Folgekonfiguration von mindestens einem ihrer Nachbarn sein. Die Korrektheitserhaltung von mindestens einem der Wörter wc und wd wäre damit in jedem Fall verletzt.

Wenn also der betreffende Automat bei der Verarbeitung von wc das Zeichen c erreicht und davor nach der obigen Argumentation noch keinen Rewrite-Schritt gemacht hat, so kann er nur noch ein Suffix von uc bearbeiten und muss dieses verkürzen. Die einzige Möglichkeit, die die Korrektheit erhält, ist somit, dass uc zu $u'd$ umgeschrieben wird. Das veränderte Wort $w'd$ liegt nun also ausschließlich in L_2d.

Wegen der deterministischen Arbeitsweise wird dieses Wort weiterhin so lange an der letzten Konfiguration verkürzt (die Endung d wird aus Gründen der Korrektheitserhaltung beibehalten), bis das Schreib-/Lese-Fenster des Automaten auch Teile der vorletzten Konfiguration enthält, wenn das Zeichen d erreicht wird (wir bezeichnen dieses Wort mit $w''d$).

Wenn nun auf der vorletzten Konfiguration von w'' eine Verkürzung stattfindet, so liegt $w'''d$ sicherlich nicht mehr in L_2d. Andererseits kann $w''d$ aber auch nicht zu $w'''c$ umgeschrieben werden, da die Fenstergröße des Automaten echt kleiner als $|u|$ sein muss (es ist $|u| \geq 2$ und die Alphabetgröße des Automaten mindestens drei). Das bewirkt, dass die Länge des umgeschriebenen Restes der letzten Konfiguration aus w''' stets kürzer sein wird, als die Länge der soeben verkürzten, vorletzten Konfiguration, womit sich $w'''c \notin L_1c$ ergibt.

Es wird also spätestens das Verkürzen der vorletzten Konfiguration zu einem Fehler führen, denn spätestens nach $|u| - 2$ Rewrite-Schritten kann die letzte Konfiguration nicht mehr verkürzt werden. Da wir alle anderen Arbeitsweisen schon vorher ausschließen konnten, folgt die Behauptung. □

Nun können wir – analog zum vorigen Kapitel – Satz 4.21 für unsere Zwecke nutzen.

Satz 4.28 *Der Trade-off von deterministischen R(R)WW-Automaten (den Church-Rosser Sprachen) gegenüber deterministischen R-, RW-, RR- und RRW-Automaten ist nichtrekursiv.*

Beweis Es sei $\mathcal{M}$ eine Turingmaschine mit Zustandsmenge S, Bandalphabet T und Startzustand s_0. Nach den zu Lemma 4.2 und Satz 4.23 analogen Überlegungen lässt sich ein deterministischer R(R)WW-Automat $\mathcal{D}_1$ zur Erkennung von $L = L_1 c \cup L_2 d$ mit $L_1 = \mathsf{VALC}_{A_1'}(\mathcal{M}) \cap \{\sqcup\}^* s_0 \{\sqcup\}^* (\$T^*ST^*)^*$ und $L_2 = \mathsf{VALC}_{A_2'}(\mathcal{M}) \cap \{\sqcup\}^* s_0 \{\sqcup\}^* (\$T^*ST^*)^*$ effektiv konstruieren.

Falls $\mathcal{M}$ das leere Wort erkennt, dann gibt es deterministische R(R)(W)-Automaten, die L erkennen. Ihre Größe (Produkt aus Zahl der Zustände und der der möglichen Fensterinhalte) ist jedoch nach Lemma 4.27 nach unten beschränkt durch den (um eins verringerten) Platzbedarf von $\mathcal{M}$ bei der Erkennung des leeren Wortes. Eine geeignete, rekursive Funktion ϕ ist somit bestimmbar. □

Bemerkung 4.29 *Eine vergleichbare Aussage über nichtdeterministische Restart-Automaten lässt sich auf dem obigen Wege leider nicht treffen, denn es stellt sich heraus, dass die bisher betrachtete Sprache L dazu nicht ausreicht: Es gibt schon einen R-Automaten mit einer konstanten Größe, der L erkennt. Wir verzichten auf eine genauere Angabe, da sich hieraus keine nützlichen Folgerungen ergeben.*

Übertragung auf Restartautomaten ohne Sonderzeichen

Die folgende, aus [29] entnommene Idee wird es in diesem Abschnitt ermöglichen, die bisher gewonnenen Ergebnisse auf weniger mächtige Sprachklassen zu übertragen.

Satz 4.30 (Niemann, Otto) *Die Sprache L wird genau dann von einem (deterministischen) R(R)WW-Automaten erkannt, wenn es einen (deterministischen) R(R)W-Automaten $\mathcal{M}_T$ und eine reguläre Sprache R gibt, so dass gilt: $L = L(\mathcal{M}_T) \cap R$.*

Die für uns interessante Hälfte des entsprechenden Beweises hierzu geht vom RWW-Automaten $\mathcal{M}$ aus, welcher die über dem Alphabet A gebildete Sprache L mit Hilfe der Sonderzeichenmenge $T \supseteq A$ verarbeitet und erkennt. Erweitert man das Eingabealphabet nun auf T, so wird der RW-Automat $\mathcal{M}_T$

mit identischer Überführungsfunktion in der Lage sein, die Sprache $L(\mathcal{M}_T)$ zu erkennen. Letztere besteht zum ersten aus allen Wörtern $w \in L$, zum zweiten aus allen, im Verlauf von akzeptierenden Berechnungen auftretenden Reduktionen jedes einzelnen Wortes w und zum dritten aus Wörtern, die von $\mathcal{M}_T$ auf Wörter aus den erstgenannten beiden Mengen reduziert werden können. Alle diejenigen unter ihnen, die nicht schon in L enthalten sind, enthalten stets ein Zeichen aus $T \setminus A$. Damit ergibt sich, dass die Sprache der Wörter aus $L(\mathcal{M}_T)$, welche nur aus Zeichen von A bestehen, wieder gleich L ist. M. a. W.: $L = L(\mathcal{M}_T) \cap A^*$.

Im Nachfolgenden werden solche Sprachen $L(\mathcal{M}_T)$ stets aus den Beweissprachen der vorigen Abschnitte gebildet sein. Die Bennennung innerhalb der Sätze wird daher einheitlich gehandhabt:

- Die der jeweiligen (valid-computations-) Sprache L zu Grunde liegende Turingmaschine sei durchgehend mit $\mathcal{M}$ bezeichnet.
- Die Menge A stehe nicht mehr für das Eingabealphabet von $\mathcal{M}$, sondern für die kleinste, zur Bildung von L nötigen Zeichenmenge.
- Zu jedem, über Sonderzeichen $T \supseteq A$ verfügenden Restart-Automaten $\mathcal{M}'$ für L, gibt es damit einen Automaten $\mathcal{M}'_T$ mit Eingabe- und Bandalphabet T, welcher die Sprache $L(\mathcal{M}'_T)$ mit $L = L(\mathcal{M}'_T) \cap A^*$ erkennt.
- Wenn wir uns nicht auf einen speziellen Automaten $\mathcal{M}'_T$ festlegen, beschreiben wir folglich die Familie $\mathcal{L}$ aller Sprachen $L(\mathcal{M}'_T)$. Für einen vereinfachten Sprachgebrauch identifizieren wir $\mathcal{L}$ stets mit einem beliebigen seiner Elemente.

Korollar 4.31 *Der Trade-off zwischen mon-RRW- bzw. RW-Automaten und det-(mon-)R(R)(W)(W)-Automaten ist nichtrekursiv.*

Beweis Von der Sprache L_Ψ wurde in Satz 4.15 gezeigt, dass ihr Komplement stets eine linear kontextfreie Sprache ist. Wir erinnern daran, dass ein Wort $w \in A^*$ genau dann in $\overline{L}_\Psi$ liegt, wenn entweder

(1) w nicht von der Form $w_1cw_2cw_3cw_4cw_5$ ist (passende, disjunkte Teilalphabete vorausgesetzt)

oder wenigstens einer der folgenden Fälle vorliegt:

(2) $w_3 \neq w_1^R$ oder $w_4 \neq w_2^R$,

(3) die Längen der Teilwörter $w_1, \ldots, w_5$ stimmen nicht überein,

(4) der Suffix w_5 ist Element von $\mathsf{INVALC}_{C'}(\mathcal{M})$.

Zu Erkennung von $\overline{L}_\Psi$ sei nun der spezielle RW-Automat $\mathcal{M}'$ beschrieben, der wie folgt vorgeht:

Zu Beginn (eines neuen Arbeitszyklus) rät $\mathcal{M}'$, welcher Fehler vorliegt bzw. ob schon ein Fehler geraten wurde und eine Fortsetzung der Verarbeitung erfolgen muss. Entscheidet sich der Automat für den ersten Fall, so setzt er an eine – je nach Arbeitsmodus unterschiedliche – Stelle ein Sonderzeichen; im zweiten Fall sucht er danach und verändert es gegebenenfalls. Eine Berechnung kann nur dann erfolgreich enden, wenn sich genau ein Sonderzeichen im Wort findet (oder der Automat im ersten Schritt schon akzeptiert), was eine konsistente Abarbeitungsweise sichert.

Die verschiedenen Arbeitsweisen gestalten sich wie folgt:

(1) Wenn $\mathcal{M}'$ rät, dass eine nicht zu L_Ψ passende Wortform vorliegt, liest er (im ersten Schritt) das komplette Wort und akzeptiert genau dann, wenn es keine Sonderzeichen enthält und wirklich von der falschen Form ist.

(2) Wird geraten, dass das erste Wort kein Spiegelbild des dritten (bzw. das zweite zum vierten) ist, so entfernt $\mathcal{M}'$ das erste Zeichen aus dem zweiten (dritten) Wort und ersetzt das Trennzeichen c zwischen erstem und zweitem (zweitem und dritten) Wort durch das Sonderzeichen $\#_{2,1}$ bzw. ersetzt bei leerem zweiten (dritten) Wort die beiden aufeinanderfolgenden Trennzeichen cc durch $\#_{2,2}$. Nach dem Neustart rät $\mathcal{M}'$ stets, dass er nach Fall 2 fortfahren muss und sucht nach einem der Zeichen $\#_{2,1}$, $\#_{2,2}$ oder $\#_E$:

- Findet er das Zeichen $\#_{2,1}$, so löscht er so lange Zeichen des zweiten (dritten) Wortes, bis es entfernt ist und schreibt $\#_{2,1}$ und das verbleibende Trennzeichen c in $\#_{2,2}$ um.
- Beim Auftreffen auf das Zeichen $\#_{2,2}$ beginnt $\mathcal{M}'$ mit dem Vergleich von w_1 und w_3^R (w_2 und w_4^R), indem er von beiden das jeweils an $\#_{2,2}$ angrenzende Zeichen vergleicht und bei Gleichheit löscht. Bei Ungleichheit wird $\#_{2,2}$ (unter Löschung eines beliebigen Zeichens) in $\#_E$ umgeschrieben.
- Beim Auftreffen auf $\#_E$ prüft $\mathcal{M}'$ das restliche Wort auf die Abwesenheit von weiteren Sonderzeichen und akzeptiert im positiven Fall am Bandende.

(3) Rät $\mathcal{M}'$, dass die Längen zweier Teilwörter w_i und w_j ($i, j \in \{1, \ldots, 5\}$) nicht übereinstimmen, so können nach der eben beschriebenen Weise alle dazischen liegenden Wörter gelöscht werden. Danach ist es – ebenso, wie eben beschrieben – durch paarweises Löschen möglich, die Unstimmigkeit der Längen zu verifizieren und durch das Zeichen $\#_E$ zu vermerken.

Das Akzeptieren verläuft ebenso durch Einlesen des vollständigen, modifizierten Gesamtwortes.

(4) Falls überprüft werden soll, dass das letzte Teilwort w_5 kein Wort aus $\mathsf{VALC}_{C'}(\mathcal{M})$ ist, so kann auch dieses an einer passend geratenen Stelle mit Hilfe einer Sonderzeichenmarkierung erfolgen ($\mathsf{INVALC}_{C'}(\mathcal{M})$ ist ebenso linear kontextfrei, wie die Teilsprachen aus den anderen Fällen).

(5) In allen anderen Fällen – z. B. auch beim unerwarteten Auftreten weiterer Sonderzeichen – verwirft der Automat das Wort.

Dieses Verhalten kann auch von einem mon-RRW-Automaten $\mathcal{M}''$ nachgestellt werden: Die Monotonie in der Verarbeitung durch $\mathcal{M}'$ ist leider nicht gegeben, da der Automat stets vor dem Auftreffen auf ein bereits geschriebenes Sonderzeichen – in der Annahme den ersten Schritt der Berechnung zu tätigen – ein neues setzen kann. Dies lässt sich jedoch dadurch vermeiden, dass $\mathcal{M}''$ stets nachliest und beim Feststellen einer (soeben verursachten) Monotonieverletzung verwirft.

Mit der obigen Beschreibung lässt sich die Sprache $L(\mathcal{M}'_T)$ nun genügend präzise beschreiben: neben den Wörtern $w \in \overline{L}_\Psi$ enthält sie genau die Wörter, an denen von genau einem „Fehler“ die Art und (ungefähre) Position korrekt geraten und mit einem Sonderzeichen markiert wurde. Das Sonderzeichen gibt ferner genügend Information über den Fortschritt der Verifikation, so dass sich folgern lässt, dass die Menge dieser restlichen Wörter – für sich genommen – von einem deterministischen (one-turn-)Kellerautomaten erkennbar ist.

Damit ist es nun möglich, die üblichen Überlegungen anzustellen:
Ist die Sprache $L(\mathcal{M})$ unendlich, so kann $L(\mathcal{M}'_T)$ keine Church-Rosser-Sprache sein (wäre sie es, dann auch $\overline{L}_\Psi = L(\mathcal{M}'_T) \cap A^*$ und somit L_Ψ – siehe Satz 4.14).

Ist $L(\mathcal{M})$ dagegen endlich, so ist L_Ψ es auch, womit $\overline{L}_\Psi$ co-finit – also regulär – ist. Nach obiger Argumentation ist $L(\mathcal{M}'_T)$ – als Vereinigung einer regulären und einer deterministisch kontextfreien Sprache – deterministisch kontextfrei und damit von beliebigen det-(mon-)R(R)(W)(W)-Automaten erkennbar (genau dann, wenn $L(\mathcal{M})$ endlich ist).

Der Rest dieses Beweises verläuft analog zu den bisherigen. □

Es lässt sich somit erkennen, dass auch schon eingeschränkte Versionen von RRW-Automaten mitunter sehr effiziente Beschreibungen liefern können. Das folgende Ergebnis stellt nun das in dieser Hinsicht letzte dar und schließt diesen Abschnitt.

Korollar 4.32 *Der Trade-off von deterministischen RW-Automaten gegenüber deterministischen, monotonen R(R)(W)(W)-Automaten (deterministischen Kellerautomaten) ist nichtrekursiv.*

Beweis Es sei Z die Zustandsmenge und Γ das Bandalphabet der Turingmaschine $\mathcal{M}$ und es gelte $\{c, d, \$\} \cap (Z \cup \Gamma) = \emptyset$. Um einen ähnlichen Beweis wie im Fall von Satz 4.23 finden zu können, ist es wesentlich, eine zu

$$L = \Big(\big(\mathsf{VALC}_{A_1}(\mathcal{M}) \cap z_0(\$\Gamma^* Z \Gamma^*)^* \big) c \Big) \cup \Big(\big(\mathsf{VALC}_{A_2}(\mathcal{M}) \cap z_0(\$\Gamma^* Z \Gamma^*)^* \big) d \Big)$$

verwandte Sprache so zu beschreiben, dass die in Lemma 4.22 geforderte Präfix-Eigenschaft noch gilt. Wir betrachten dazu den folgenden det-RWW-Automaten $\mathcal{M}'$, der wie folgt vorgeht:

(1) Im ersten Verarbeitungszyklus überläuft $\mathcal{M}'$ das Wort, bis sein Fenster das rechte Bandendesymbol erreicht. Ist das Eingabewort von der falschen Form (also nicht aus $z_0(\$\Gamma^* Z \Gamma^*)^*\{c, d\}$), so wird es verworfen.

Andernfalls wird, wenn das letzte Eingabezeichen ein c ist, das letzte Teilwort aus $\Gamma^* Z \Gamma^*$ in mehreren Rewrite-Schritten so lange in Sonderzeichen kodiert, bis das letzte Trennzeichen \$ erreicht ist. Im Detail kann das so geschehen, dass – von rechts nach links – je zwei Zeichen $xy \in (T^2 \cup ST \cup TS)$ zu einem Sonderzeichen $C_{x,y}$ umgeschrieben werden, welches neben der Information über die ursprünglichen Zeichen auch die Information über das letzte Zeichen (c) enthält.

(2) Ist nun das letzte Teilwort (bei ungerader Länge bis auf das erste Zeichen) vollständig kodiert, so beginnt $\mathcal{M}'$, das kodierte Wort über das Trennzeichen \$ mit seinem links stehenden Vorgänger zu vergleichen. Dies kann – ähnlich zum eben geführten Beweis – durch Umschreiben des Trennzeichens (z. B. zu $\#_C$) und gleichzeitiges Löschen erfolgen, bis beide Worte vollständig abgebaut sind (falls sich herausstellt, dass das linke Teilwort keine Vorgängerkonfiguration des rechten, kodierten Teilwortes ist, verwirft der Automat das Gesamtwort).

(3) Nachdem die beiden Teilwörter entfernt wurden, fährt $\mathcal{M}'$ mit der Verarbeitung, wie unter (1) beschrieben, fort. Die Rolle der letzten beiden Zeichen $c\triangleleft$ übernimmt nun das vorletzte Trennzeichen \$ und das umgeschriebene Trennzeichen $\#_C$.

(4) $\mathcal{M}'$ akzeptiert das Wort, wenn sich dieses auf die Form $\#_C(\$\#_C)^+ c$ reduzieren lässt.

(5) Die Beschreibung der Punkte (1) bis (4) gilt sinngemäß auch für Eingabewörter, die auf das Zeichen d enden. Die Kodierung erfolgt jedoch – entsprechend der genannten Absicht – über eine disjunkte Menge von Sonderzeichen.

Auf gewohnte Weise ergibt sich nun aus $\mathcal{M}'$ die vom det-RW-Automaten $\mathcal{M}'_T$ erkannte Sprache $L(\mathcal{M}'_T)$, die neben L auch die während einer akzeptierenden Berechnung entstandenen, sowie weitere, passend reduzierbare Wörter

enthält. Alle diese enthalten jedoch stets mindestens ein Sonderzeichen mit der Information über das letzte Zeichen. Deshalb lassen sich aus ihnen keine Präfixe w entnehmen, so dass wc und wd in $L(\mathcal{M}'_T)$ sind.

Wenn $\mathcal{M}$ nun das leere Wort akzeptiert, ist das Wort

$$w \in \mathsf{VALC}_A(\mathcal{M}) \cap z_0(\$\Gamma^* Z \Gamma^*)^*$$

nach wie vor das einzige (und damit das kürzeste) Wort mit obiger Eigenschaft ($wc, wd \in L(\mathcal{M}'_T)$). Der Beweis lässt sich somit analog zu dem von Satz 4.23 schließen. □

4.4 Zusammenfassung der Trade-offs

Im Zuge des zurückliegenden Kapitels traten manche Fälle auf, in denen eine frühere Aussage als Nebenergebnis einer Verfeinerung erneut bewiesen wurde. Zur Übersicht seien daher die verschiedenen Ergebnisse bezüglich der Beschreibungskomplexität noch zusammengefasst dargestellt. Aus Gründen der Überschaubarkeit sind zunächst in Abbildung 6 nur die wesentlichen Trade-offs verzeichnet:

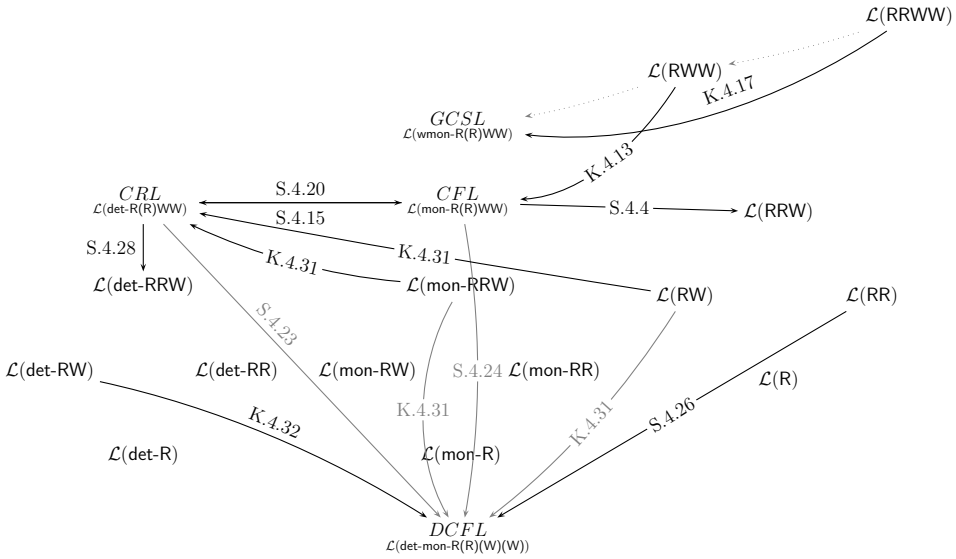

Abbildung 6: Hauptübersicht über nichtrekursive Trade-offs

Die nichtrekursiven Trade-offs einer Klasse $\mathcal{A}$ zur jeweils in der Hierarchie möglichst nahe liegenden Klasse $\mathcal{B}$ sind dabei als schwarze Linien eingezeichnet. Entsprechende Trade-offs derselben Klasse $\mathcal{A}$ bezüglich der möglichst tief in der Hierarchie angesiedelten Klasse $\mathcal{C}$ sind als graue Linien dargestellt. Auch wenn man aus dieser Symbolik nicht schließen darf, dass entsprechende Trade-offs von $\mathcal{A}$ bezüglich jeder zwischen $\mathcal{B}$ und $\mathcal{C}$ liegenden Klasse nachgewiesen wurden[19], so ist dies jedoch oft der Fall. Die entsprechend vollständige Wiedergabe der Ergebnisse in den einzelnen Teilbereichen der Hierarchie findet sich in den nachfolgenden Detailabbildungen wieder.

Zusätzlich zu den Ergebnissen sind die nach Korollar 4.18 vermuteten Trade-offs als gepunktete Linien eingetragen. Nicht eingetragen wurden ferner die trivialen Ergebnisse, wie z. B. der nichtrekursive Trade-off von RWW- gegenüber RRW- oder det-RRW-Automaten, welcher sich unmittelbar aus dem

[19] Zum Beispiel wurde kein Ergebnis über die Beschreibungseffizienz von deterministischen R(R)WW-Automaten (Church-Rosser-Sprachen) gegenüber der Klasse der mon-RRW-Automaten nachgewiesen.

Trade-off der mon-R(R)WW-Automaten gegenüber den RRW- bzw. der det-R(R)WW- gegenüber den det-RRW-Automaten ergibt.

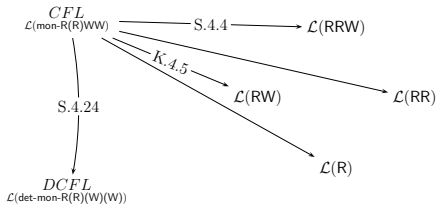

Abbildung 7: nichtrekursive Trade-offs von CFL zu den R(R)(W)-Automaten

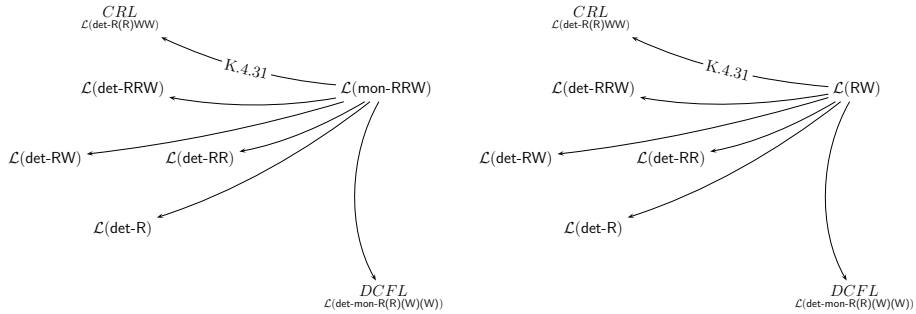

Abbildung 8: nichtrekursive Trade-offs nach Korollar 4.31

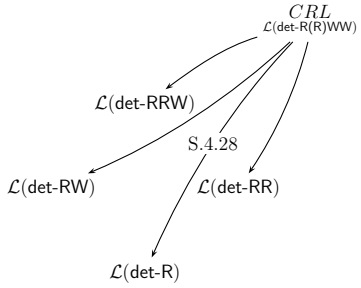

Abbildung 9: nichtrekursive Trade-offs nach Satz 4.28

Offene Fragen

Auf Basis der vorliegenden Ergebnisse sei zum Abschluss eine knappe Auswahl an bisher ungeklärten oder über das Behandelte hinaus möglichen Fragstellungen angegeben:

- Auf welchem Wege ist eine Verbesserung von Korollar 4.18 (siehe Abbildung 6) zu erreichen?
- Liegt die Sprache $\mathsf{VALC}_{C'}(\mathcal{M})$ in GCSL? – Dass sie in $\mathcal{L}(\mathsf{RWW})$ liegt, wurde bewiesen[20] und dass sie auch in CRL liegt, erscheint unwahrscheinlich.
- Auf welchem Weg kommt man zu Ergebnissen bezüglich der relativen Effizienz der Teilklassen von deterministischen bzw. monotonen bzw. Nicht-Sonderzeichen-Automaten untereinander?

[20] Darüber hinaus ist es wahrscheinlich, dass $\mathsf{VALC}_{A'}(\mathcal{M})$ auch von $\mathsf{OWauxPDA}$ in polynomieller Zeit und auf logarithmisch beschränktem Band erkannt werden kann.

4.5 Unentscheidbarkeiten und Nichtminimierbarkeit

In den vorigen Abschnitten sind einige Nicht-Semi-Entscheidbarkeiten zum Nachweis nichtrekursiver Trade-offs verwendet worden. Es wurde auch deutlich, wie eng dies teilweise mit der Erkennbarkeit von valid-computations-Sprachen verknüpft ist. Aus den ermittelten Ergebnissen lassen sich auch in anderer Hinsicht – inspiriert durch Betrachtungen von Malcher in seiner Arbeit [23] – einige zusätzliche Aussagen gewinnen.

Dazu sei – wie stets – $\mathcal{M}$ eine beliebige Turingmaschine mit Zustandsmenge S, Bandalphabet A und $\$ \notin S \cup A$.

In diesem Abschnitt verstehen wir unter einer beliebigen valid-computations-Sprache $L_V(\mathcal{M})$ eine Sprache, die die in Lemma 4.2 beschriebene Eigenschaften (1) und (3) (Nicht-Kontextfreiheit genau dann, wenn die Turingmaschine eine unendliche Sprache erkennt – ansonsten Endlichkeit) besitzt und deren Wörter Kodierungen von akzeptierenden Berechnungen obiger Maschinen sind. Wir nennen $\mathcal{L}_V$ die Familie aller solchen Sprachen.

Die Komplemente der $L_V(\mathcal{M})$ (bezüglich ihrer jeweiligen Alphabete, die im Folgenden mit $\Sigma = S \cup A \cup \{\$\}$ bezeichnet werden), für die wir entsprechend die Eigenschaften (2) und (4) (Regularität genau dann, wenn die Endlichkeit der Turingmaschinensprache vorliegt) fordern, werden in der Familie $\mathcal{L}_I$ zusammengefasst.

Behauptung *Es sei $\mathcal{L}_1$ eine beliebige Sprachklasse, die $\mathcal{L}_V$ enthält und $\mathcal{L}_2$ eine beliebige Sprachklasse, die $\mathcal{L}_I$ enthält. Dann gilt:*

1. *Die Prädikate Endlichkeit, Regularität, Linearität, deterministische Kontextfreiheit und Kontextfreiheit sind nicht semi-entscheidbar für Sprachen in $\mathcal{L}_1$.*
2. *Die zu obigem Punkt komplementären Prädikate Unendlichkeit, ..., Nicht-Kontextfreiheit sind ebenfalls nicht semi-entscheidbar für Sprachen in $\mathcal{L}_1$.*
3. *Ferner ist die Leerheit von Sprachen aus $\mathcal{L}_1$ nicht semi-entscheidbar.*
4. *Die Prädikate Regularität und Nicht-Regularität sind nicht semi-entscheidbar für Sprachen aus $\mathcal{L}_2$.*
5. *Ebenso gilt für beliebige Sprachen aus $\mathcal{L}_2$ mit Alphabet Σ, dass die Äquivalenz zu Σ^* nicht semi-entscheidbar ist.*

Beweis

1. Aus den in Lemma 4.2 ermittelten Eigenschaften folgt, dass $L_V(\mathcal{M})$ jeweils genau dann eine endliche Sprache ist, wenn sie kontextfrei ist

und dies ebenfalls genau dann, wenn die von $\mathcal{M}$ akzeptierte Sprache endlich ist. Wäre die Endlichkeit beliebiger Sprachen aus $\mathcal{L}_1$ semi-entscheidbar, dann wäre es auch die von Turingmaschinen, was die getroffene Aussage beweist.

2. Ebenso kann die Nicht-Semi-Entscheidbarkeit von Unendlichkeit und den weiteren, aufgezählten Prädikaten für Sprachen aus $\mathcal{L}_1$ auf die Nicht-Semi-Entscheidbarkeit der Unendlichkeit von rekursiv aufzählbaren Sprachen zurückgeführt werden.

3. Analog ergibt sich die Nicht-Semi-Entscheidbarkeit der Leerheit von $\mathcal{L}_1$-Sprachen aus dem entsprechenden Ergebnis bezüglich der Leerheit von rekursiv aufzählbaren Sprachen. Über das komplementäre Prädikat kann auf diesem Weg keine Aussage getroffen werden, da die Leerheit von rekursiv aufzählbaren Sprachen co-semi-entscheidbar ist.

4. Nach den vorausgesetzten Eigenschaften ist das Komplement von $L_V(\mathcal{M})$ jeweils genau dann eine reguläre Sprache, wenn $L(\mathcal{M})$ endlich ist. Damit ist die Nicht-Semi-Entscheidbarkeit der Regularität (bzw. ihres Gegenteils) von $\mathcal{L}_2$-Sprachen über die Nicht-Semi-Entscheidbarkeit der (Un-)Endlichkeit von rekursiv aufzählbaren Sprachen beweisbar.

5. Das Komplement von $L_V(\mathcal{M})$ bezüglich des verwendeten Alphabets Σ ist genau dann gleich Σ^*, wenn $L(\mathcal{M})$ leer ist. Damit ist die Verbindung zur Leerheit von rekursiv aufzählbaren Sprachen (siehe Punkt 3) hergestellt.

□

Für die von uns betrachteten Sprachklassen ergibt sich zusammengefasst:

Korollar 4.33 *Es sei L eine Sprache aus $\mathcal{L}$(R(R)WW). Dann ist von L nicht semi-entscheidbar, ob L (un-)endlich, (nicht) linear, (nicht) deterministisch kontextfrei, (nicht) kontextfrei oder leer ist.*

Ist L eine Sprache über Σ und aus $\mathcal{L}$(((w)mon-)R(R)WW). Dann ist von L nicht semi-entscheidbar, ob L (nicht) regulär oder gleich Σ^* ist.

Beweis Nach Lemma 4.12 sind R(R)WW-Automaten in der Lage, zu einer beliebigen Turingmaschine $\mathcal{M}$ die Sprache $\mathsf{VALC}_{C'}(\mathcal{M})$ zu erkennen. Letztere besitzt nach Lemma 4.2 die geforderten Eigenschaften.

Der zweite Teil folgt analog aus dem Erkennen von $\mathsf{INVALC}_A(\mathcal{M})$ bzw. $\mathsf{INVALC}_{C'}(\mathcal{M})$ durch die im zweiten Teil der Aussage genannten Systeme.

□

Auf ähnlichem Wege lässt sich über alle genannten Systeme auch die folgende Aussage treffen:

Satz 4.34 *Es sei S ein beliebiges Beschreibungssystem, dessen Sprachklasse $\mathcal{L}(S)$ die Klasse $\mathcal{L}_V$ bzw. $\mathcal{L}_I$ enthält, so dass es zu jeder Sprache aus $\mathcal{L}_V$ bzw. $\mathcal{L}_I$ mindestens einen effektiv konstruierbaren Beschreiber aus S gibt und c ein sinnvolles Komplexitätsmaß auf S mit den zusätzlichen Eigenschaften, dass es zum einen für die leere Sprache bzw. für Σ^* einen Beschreiber in Größe der minimalen Beschreibungsgröße $c_m = \min\{c(\mathcal{D}) \mid D \in S\}$ gibt und zum anderen von allen Sprachen mit Beschreibern der Größe c_m entscheidbar ist, ob sie gleich der leeren Sprache bzw. Σ^* sind. Dann gibt es keinen Minimierungsalgorithmus für die Elemente von S.*

Beweis Wenn wir vom Gegenteil der Behauptung ausgehen, lässt sich jeweils ein Semi-Entscheidungsalgorithmus für die Leerheit von $L(\mathcal{M})$ konstruieren:

Der Wert von c_m kann als bekannt angesehen werden, da er sich aus der geforderten Aufzählbarkeit aller Beschreiber in aufsteigender Größenreihenfolge ergibt (wenn man einfach die Größe des zuerst aufgezählten Elements ermittelt). Zu einer beliebigen Turingmaschine $\mathcal{M}$ konstruiert der Algorithmus den entsprechenden Beschreiber der Sprache $L_V(\mathcal{M})$ bzw. ihres Komplements. Falls die Sprache $L(\mathcal{M})$ leer ist, so liefert der Minimierungsalgorithmus in beiden Fällen der Betrachtung einen Beschreiber der Größe c_m. Da es nur endlich viele Beschreiber dieser Größe gibt, kann leicht ermittelt werden, ob es sich um einen Beschreiber der leeren Sprache bzw. von Σ^* handelt und somit die Leerheit von rekursiv aufzählbaren Sprachen semientschieden werden. Dies liefert den gewünschten Widerspruch. □

Korollar 4.35 *Für die Restart-Automatenklassen ((w)mon-)R(R)WW gibt es keinen Minimierungsalgorithmus.*

Beweis Wir wählen als Größenmaß das Produkt aus der Zahl der Zustände und der der möglichen Fensterinhalte. Dies ist ein sinnvolles Maß, da die Zahl der verschiedenen Automaten derselben Größe durch die endlich vielen Möglichkeiten, die Überführungsfunktion mit Werten zu belegen, beschränkt ist und sich über eine Kodierung der Überführungsfunktion auch eine Aufzählreihenfolge festlegen lässt.

Die minimale Größe c_m aller Automaten der obigen Typen erhält man bei einer Fenstergröße und Zustandszahl von eins und durch den Verzicht auf Sonderzeichen.[21] Damit entsprechen alle Automaten dieser Größe den in Kapitel 3 beschriebenen R(1)-Automaten. Diese erkennen ausschließlich reguläre Sprachen – somit ist von jedem von ihnen entscheidbar, ob er die leere Sprache bzw. A^* erkennt und ferner gibt es für die letzteren beiden Sprachen jeweils mindestens einen akzeptierenden Automaten der Größe c_m. □

[21] Wie stets sei allen Vergleichen ein festes Eingabealphabet A zu Grunde gelegt.

Literatur

[1] Birget, J.-C. *Intersection and union of regular languages and state complexity*. Information Processing Letters 43 (1992), 185–190.

[2] Brandenburg, F.-J. *On one-way auxiliary pushdown automata*. Proceedings of the 3rd GI-Conference on Theoretical Computer Science (1977), Lecture Notes in Computer Science 48 (1977), 132–144.

[3] Brzozowski, J.A. und Leiss, E. *On equations for regular languages, finite automata, and sequential networks*. Theoretical Computer Science 10, (1980), 19–35.

[4] Buntrock, G. *Wachsend kontextsensitive Sprachen*. Habilitationsschrift, Julius-Maximilians-Universität Würzburg, 1996.

[5] Buntrock, G. und Otto, F. *Growing context-sensitive languages and Church-Rosser languages*. Information and Computation 141 (1998), 1–36.

[6] Chandra, A.K., Kozen, D.C. und Stockmeyer, L.J. *Alternation*. Journal of the ACM 21 (1981), 114–133.

[7] Chrobak, M. *Finite automata and unary languages*. Theoretical Computer Science 47 (1986), 149–158.

[8] Goldstine, J., Kappes, M., Kintala, C. M. R., Leung, H., Malcher, A. und Wotschke, D. *Descriptional complexity of machines with limited ressources*. Journal of Universal Computer Science 8 (2002), 193–234.

[9] Hartmanis, J. *Context-free languages and Turing machine computations*. Proceedings of Symposia in Applied Mathematics 19 (1967), 42–51.

[10] Hartmanis, J. *On the succinctness of different representations of languages*. SIAM Journal on Computing 9 (1980), 114–120.

[11] Holzer, M. und Kutrib, M. *Nondeterministic descriptional complexity of regular languages*. International Journal of Foundations of Computer Science 14 (2003), 1087–1102.

[12] Hromkovič, J. *Communication Complexity and Parallel Computing*. Springer-Verlag, Heidelberg (1997).

[13] Jančar, P., Mráz, F. und Plátek, M. *Characterisation of context-free languages by erasing automata*. Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS'92), Lecture Notes in Computer Science 629 (1992), 305–314.

[14] Jančar, P., Mráz, F., Plátek, M. und Vogel, J. *Restarting Automata*. Proceedings of the 10th International Conference on Fundamentals of Computation Theory (FCT'95), Lecture Notes in Computer Science 965 (1995), 283–292.

[15] Jančar, P., Mráz, F., Plátek, M. und Vogel, J. *On restarting automata with rewriting*. Lecture Notes in Computer Science 1218 (1997), 119–136.

[16] Jančar, P., Mráz, F., Plátek, M. und Vogel, J. *On monotonic automata with restart operation*. Journal of Automata, Languages and Combinatorics 4 (1999), 287–311.

[17] Jirásek, J., Jirásková, G. und Szabari, A. *State Complexity of Concatenation and Complementation of Regular Languages*. Proceedings of the 9th International Conference on Implementation and Application of Automata (CIAA'04), Lecture Notes in Computer Science 3317 (2004), 178–189.

[18] Jirásková, G. *A Note on Minimal Finite Automata*. Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS'01), Lecture Notes in Computer Science 2136 (2001), 421–431.

[19] Jurdziński, T., Loryś, K., Niemann, G. und Otto, F. *Some results on RRW- and RRWW-automata and their relation to the class of growing context-sensitive languages*. Journal of Automata, Languages and Combinatorics 9 (2004), 407–437.

[20] Kutrib, M. *The phenomenon of non-recursive trade-offs*. International Journal of Foundations of Computer Science 16 (2005), 957–973.

[21] Kutrib, M., Holzer, M. und Reimann, J. *Non-recursive trade-offs for deterministic restarting automata* Proceedings of the 7th International Workshop on Descriptional Complexity of Formal Systems (DCFS'05), Rapporto Tecnico 06/05 Dipartimento di Informatica e Comunicazione Milano (2005), 158–169.

[22] Leung, H. *Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata*. SIAM Journal on Computing 27 (1998), 1073–1082.

[23] Malcher, A. *Descriptional complexity of cellular automata and decidability questions*. Journal of Automata, Languages and Combinatorics 7 (2002), 549–560.

[24] McNaughton, R., Narendran, P. und Otto, F. *Church-Rosser Thue systems and formal languages*. Journal of the ACM 35 (1988), 324–344.

[25] Messerschmidt, H., Mráz, F., Otto, F. und Plátek, M. *Correctness preservation and complexity of simple RL-automata*. Proceedings of the 11th International Conference on Implementation and Application of Automata (CIAA'06), Lecture Notes in Computer Science 4094 (2006), 162–172.

[26] Meyer, E. M. und Fischer, M. J. *Economy of description by automata, grammars, and formal systems*. Proceedings of the 12th Annual IEEE Symposium on Switching and Automata Theory (1971), 188–191.

[27] Mráz, F. *Lookahead hierarchies of restarting automata*. Journal of Automata, Languages and Combinatorics 6 (2001), 493–506.

[28] Narendran, P., Ó'Dúnlaing, C. und Rolletschek, H. *Complexity of certain decision problems about congruential languages*. Journal of Computer and System Sciences 30 (1985), 343–358.

[29] Niemann, G. und Otto, F. *Restarting automata and prefix-rewriting systems*. Mathematische Schriften Kassel 18/99 (1999).

[30] Niemann, G. und Otto, F. *Restarting automata, Church-Rosser languages and representations of r.e. languages*. Proceedings of the 4th International Conference on Developments in Language Theory (DLT'99), World Scientific, Singapore (2000), 103–114.

[31] Niemann, G. und Otto, F. *The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages*. Information and Computation 197 (2005), 1–21.

[32] Otto, F. *Restarting automata and their relations to the Chomsky hierarchy*. Proceedings of the 7th International Conference on Developments in Language Theory (DLT'03), Lecture Notes in Computer Science 2710 (2003), 55–74.

[33] Otto, F. *Restarting automata - notes for a course at the 3rd international PhD school in formal languages and applications*. Mathematische Schriften Kassel 6/04 (2004).

[34] Otto, F. *Restarting automata*. Recent Advances in Formal Languages and Applications, Springer Verlag, Berlin (2006), 269–303.

[35] Plátek, M. *Two-way restarting automata and j-monotonicity*. Proceedings of the 28th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'01), Lecture Notes in Computer Science 2234 (2001), 316–325.

[36] Rabin, M. O. und Scott, D. *Finite automata and their decision problems*. IBM Journal of Research and Development 3 (1959), 114–125.

[37] Salomaa, A. *Formal languages*. Academic Press, New York (1973).

[38] Salomaa, K. und Yu, S. *NFA to DFA transfomation for finite languages over arbitrary alphabets*. Journal of Automata, Languages and Combinatorics 2 (1997), 177–186.

[39] Salomaa, K. und Yu, S. *Loop-free alternating finite automata*. Proceedings of the 8th International Conference on Automata and Formal Languages (1996), Publicationes Mathematicae Debrecen Vol. 54 (1999), 979–988.

[40] Salomaa, K. und Yu, S. *Alternating finite automata and star-free languages*. Theoretical Computer Science 234 (2000), 167–176.

[41] Schmidt, E. M. und Szymanski, T. G. *Succinctness of descriptions of unambiguous context-free languages*. SIAM Journal on Computing 6 (1977), 547–553.

[42] Stearns, R. E. *A regularity test for pushdown automata*. Information and Control 11 (1967), 323–340.

[43] Valiant, L. G. *A note on the succinctness of descriptions of deterministic languages*. Information and Control 32 (1976), 139–145.

[44] Yu, S., Zhuang, Q. und Salomaa, K. *The state complexity of some basic operations on regular languages*. Theoretical Computer Science 125 (1994), 315–328.

[45] Yu, S. *State complexity of regular languages*. Journal of Automata, Languages and Combinatorics 6 (2001), 221–234.