Über die algorithmische Komplexität regulärer Sprachen

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt am Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von
Gregor Gramlich
aus Offenbach am Main

Frankfurt 2007
(D 30)

vom Fachbereich Informatik und Mathematik der

Johann Wolfgang Goethe-Universität als Dissertation angenommen.

Dekan: Prof. Dr. Klaus Johannson

Gutachter: Prof. Dr. Georg Schnitger, Prof. Dr. Juraj Hromkovič

Datum der Disputation: 2. Juli 2007

Zusammenfassung

Im Gegensatz zur Minimierung von DFAs ist die exakte Minimierung von NFAs oder regulären Ausdrücken nachweislich schwierig, im allgemeinen Fall PSPACE-schwer. Wir zeigen, dass selbst schwache Approximationen zur Minimierung von NFAs und regulären Ausdrücken wahrscheinlich nicht effizient möglich sind.

Falls als Eingabe ein NFA oder regulärer Ausdruck der Größe n gegeben ist, löst ein Approximationsalgorithmus für das Minimierungsproblem mit Approximationsfaktor $o(n)$ bereits ein PSPACE-vollständiges Problem. Wenn wir uns auf NFAs oder reguläre Ausdrücke über einem unären – also einelementigen – Alphabet beschränken, so ist das Problem der exakten Minimierung NP-vollständig. Wir weisen nach, dass effiziente Approximationen für das unäre Minimierungsproblem mit Approximationsfaktor $n^{1-\delta}$ für jedes $\delta > 0$ nicht möglich sind, sofern $\mathrm{P} \neq \mathrm{NP}$ gilt.

Liegt die Eingabe als DFA mit n Zuständen vor, kann sie exponentiell größer sein als ein äquivalenter NFA oder regulärer Ausdruck. Dennoch bleibt das Minimierungsproblem PSPACE-schwer, wenn die Anzahl der Übergänge oder Zustände in einem äquivalenten NFA oder die Länge eines äquivalenten regulären Ausdrucks zu bestimmen ist. Wir zeigen, dass auch hierfür keine guten Approximationen zu erwarten sind. Unter der Annahme der Existenz von Pseudozufallsfunktionen, die wiederum auf der Annahme basiert, dass Faktorisierung schwierig ist, zeigen wir, dass kein effizienter Algorithmus einen Approximationsfaktor $\frac{n}{\mathrm{poly}(\log n)}$ für die Zahl der Übergänge im NFA oder die Länge des regulären Ausdrucks garantieren kann. Für die Zahl der Zustände im NFA weisen wir nach, dass effiziente Approximationen mit Approximationsfaktor $\frac{\sqrt{n}}{\mathrm{poly}(\log n)}$ ausgeschlossen sind.

Wir betrachten dann Lernprobleme für reguläre Sprachen als Konzeptklasse. Mit den entwickelten Methoden, die auf der Annahme der Existenz von Pseudozufallsfunktionen beruhen, zeigen wir auch, dass es für das Problem des minimalen konsistenten DFAs keine effizienten Approximationen mit Approximationsfaktor $\frac{n}{\mathrm{poly}(\log n)}$ gibt. Für den unären Fall hingegen weisen wir nach, dass es einen effizienten Algorithmus gibt, der einen minimalen konsistenten DFA konstruiert und erhalten somit auch einen effizienten PAC-Algorithmus für unäre reguläre Sprachen, die von DFAs mit n Zuständen akzeptiert werden.

Für unäre Beispielmengen weisen wir außerdem nach, dass es keine effizienten Algorithmen gibt, die minimale konsistente NFAs konstruieren, falls nicht $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$ gilt. Andererseits geben wir einen effizienten Algorithmus an, der zu unären Beispielmengen einen konsistenten NFA mit höchstens $O(\mathrm{opt}^2)$ Zuständen konstruiert, wenn ein minimaler konsistenter NFA opt Zustände hat.

Abschließend betrachten wir das Lernen von DFAs durch Äquivalenzfragen. Für den nicht-unären Fall ist bekannt, dass exponentiell viele Fragen für DFAs mit n Zuständen benötigt werden. Für unäre zyklische DFAs mit primer Zykluslänge und höchstens n Zuständen zeigen wir, dass $\Theta(\frac{n^2}{\ln n})$ Äquivalenzfragen hinreichend und notwendig sind. Erlauben wir größere zyklische DFAs als Hypothesen, kommen wir mit weniger Fragen aus: Um zyklische DFAs mit höchstens n Zuständen durch Äquivalenzfragen mit zyklischen DFAs mit höchstens n^d Zuständen für $d \leq n$ als Hypothesen zu lernen, sind $O(\frac{n^2}{d})$ Fragen hinreichend und $\Omega(\frac{n^2 \cdot \ln d}{d \cdot (\ln n)^2})$ Fragen nötig.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Deterministische und nichtdeterministische endliche Automaten (DFAs und NFAs) sowie reguläre Ausdrücke gehören zu den fundamentalen Berechnungsmodellen für die Beschreibung regulärer Sprachen.

Insbesondere reguläre Ausdrücke werden von den meisten Programmiersprachen unterstützt und unter anderem dazu genutzt, die syntaktische Korrektheit einer Benutzereingabe zu überprüfen oder um nach Mustern in einem Text zu suchen. Intern werden reguläre Ausdrücke meist in NFAs überführt. Die Größe dieses Automaten hat einen direkten Einfluss auf die Laufzeit der Algorithmen, die den Automaten auf zu untersuchende Texte anwenden. Man ist also bemüht, zu einer regulären Sprache möglichst kurze reguläre Ausdrücke oder kleine Automaten zu finden, die diese Sprache beschreiben. Konkret geht es darum, für einen gegebenen DFA, NFA oder regulären Ausdruck einen kleinsten DFA, NFA oder regulären Ausdruck zu bestimmen, der dieselbe Sprache beschreibt.

1.1 Minimierung endlicher Automaten und regulärer Ausdrücke

Das Problem der Minimierung endlicher Automaten wird bereits seit Einführung der regulären Sprachen betrachtet. Die Minimierung eines DFAs ist Inhalt der Grundvorlesungen des Informatik-Studiums. Bereits seit den 1960er Jahren sind Minimierungsverfahren bekannt, die Laufzeit $\Theta(|\Sigma| \cdot n^2)$ bei Eingabe eines DFAs mit n Zuständen über dem Alphabet Σ benötigen. Hopcroft stellt 1971 einen Minimierungsalgorithmus mit Laufzeit $\Theta(|\Sigma| \cdot n \log n)$ vor [18].

Im Gegensatz dazu sind keine effizienten Algorithmen für die Minimierung nichtdeterministischer endlicher Automaten (NFA) und regulärer Ausdrücke bekannt. Anfang der 1970er Jahre erscheinen die ersten Arbeiten, die die Schwierigkeit des Problems erklären.

Meyer und Stockmeyer weisen in ihren Arbeiten [31, 38] die CSL-Vollständigkeit[1] für folgendes Problem nach: Gegeben seien zwei reguläre Ausdrücke R_1 und R_2 über einem Alphabet Σ mit mindestens zwei Buchstaben. Sind die von ihnen beschriebenen Sprachen $L(R_1), L(R_2)$ verschieden? Das Problem bleibt sogar CSL-vollständig, wenn wir $R_2 = \Sigma^*$ setzen.

Mit Hilfe einer Reduktionstechnik, die die Eingabelänge durch angehängte Leerzeichen polynomiell vergrößert [4], zeigen Hunt, Rosenkrantz und Szymanski [21], dass jedes CSL-vollständige Problem auch PSPACE-vollständig ist. PSPACE ist die Klasse der Probleme, die von Turingmaschinen mit polynomiellem Platz akzeptiert werden; siehe auch Abschnitt A.2. Als Konsequenz der PSPACE-Vollständigkeit des Problems $L(R) \neq \Sigma^*$ bei gegebenem regulären Ausdruck R über Σ ergibt sich, dass die Bestimmung eines kürzesten äquivalenten regulären Ausdrucks oder NFAs, ebenfalls PSPACE-schwer ist. Durch ein Kodierungsargument folgt die Schwere des Problems auch bei Einschränkung des Alphabets auf $\{0, 1\}$.

Stockmeyer und Meyer [38] zeigen außerdem, dass das Problem der Nicht-Äquivalenz zweier regulärer Ausdrücke über dem unären Alphabet, also einem Alphabet mit nur einem Buchstaben, NP-vollständig ist. Auch hier ist das eingeschränkte Problem $L(R) \neq \{a\}^*$ weiter NP-vollständig.

Die Komplexität des Problems, zu einem gegebenen DFA einen kleinsten äquivalenten NFA zu bestimmen, wurde erst Anfang der 1990er Jahre von Jiang und Ravikumar [25] als PSPACE-vollständig klassifiziert. Die Einschränkung dieses Problems auf den unären Fall betrachten Jiang, McDowell und Ravikumar [24] und weisen nach, dass Polynomialzeitalgorithmen, die zu einem unären DFA einen kleinsten NFA konstruieren nur existieren, wenn $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$ gilt.

Alle diese Minimierungsprobleme sind also nachweislich schwierig und vermutlich nicht in akzeptabler Rechenzeit lösbar.

> "Even if until now no proof is known that NP-hard problems cannot be solved by means of algorithms that run in polynomial time, there is a strong evidence that such algorithms may not exist and we have to make use of approximation algorithms."
>
> Giorgio Ausiello et al. *Complexity and Approximation* [2]

Wir müssen unsere Erwartungen also senken und uns beispielsweise mit Algorithmen zufrieden geben, die auf vielen aber nicht allen Instanzen in polynomieller Zeit laufen oder nicht die kleinsten NFAs oder regulären Ausdrücke berechnen. In einigen Fällen genügt vielleicht als Lösung ein kleiner NFA oder regulärer Ausdruck, der nicht genau die gegebene Sprache beschreibt, aber auf den meisten Wörtern dasselbe Akzeptanzverhalten aufweist.

[1] CSL ist die Klasse der kontextsensitiven Sprachen, siehe auch Abschnitt A.2.

In dieser Arbeit beschäftigen wir uns mit dem im Zitat von Ausiello et al. beschriebenen Ansatz der effizienten Approximationsalgorithmen, der für viele NP-schwere Optimierungsprobleme erfolgreich angewandt wird. Ein effizienter Approximationsalgorithmus mit Approximationsfaktor $\alpha \geq 1$ garantiert für ein Minimierungsproblem mit optimaler Lösung der Größe opt, dass eine Lösung der Größe höchstens $\alpha \cdot \text{opt}$ in polynomieller Zeit gefunden wird. Wir werden allerdings sehen, dass selbst Approximationen mit sehr schlechten Approximationsfaktoren unter gängigen Komplexitätsannahmen wie P $\neq$ NP oder der Existenz von Pseudozufallsfunktionen nicht effizient möglich sind.

In der Literatur wurde das Problem der effizienten Approximation minimaler NFAs oder regulärer Ausdrücke bis vor kurzem so gut wie nicht behandelt. Jiang und Ravikumar [25] listen 1993 in ihrer Arbeit unter den unbeantworteten Fragen das Problem auf, zu gegebenem k und gegebenem DFA einen äquivalenten NFA zu bestimmen, der höchstens opt^k Zustände hat, wenn ein minimialer äquivalenter NFA opt Zustände hat.

Desweiteren formuliert zum Beispiel 2002 Hromkovič das *Research Problem 6* in [20]: *Zeige untere und obere Schranken für die Polynomialzeit-Approximierbarkeit der Anzahl der Zustände eines minimalen NFAs für eine reguläre Sprache L, die durch einen NFA oder DFA A gegeben ist. Die Komplexität wird gemessen in der Eingabelänge, die die Größe von A ist.*

In einem Tutorial-Vortrag weist Thomas Wilke 2003 [39] darauf hin, dass effiziente Approximationen für die Minimierung von NFAs mit konstanten Approximationsfaktoren ausgeschlossen sind, sofern P $\neq$ PSPACE gilt. Dies folgt daraus, dass es bereits PSPACE-schwer ist, zu entscheiden, ob die von einem NFA A akzeptierte Sprache $L(A) \subseteq \Sigma^*$ gleich Σ^* ist. Hätte man einen Approximationsalgorithmus, der einen äquivalenten NFA mit höchstens $c \cdot \text{opt}$ vielen Zuständen konstruiert, wenn der minimale äquivalente NFA opt Zustände hat, so könnte man zu gegebenem NFA A einen äquivalenten NFA B berechnen, der für den Fall, dass $L(A) = \Sigma^*$ gilt, höchstens c Zustände hat, denn der kleinste NFA hat in diesem Fall genau einen Zustand. Hat B mehr als c Zustände, wissen wir bereits, dass $L(A) \neq \Sigma^*$ gilt. Hat hingegen B höchstens c Zustände, wenden wir auf B die Potenzmengenkonstruktion an und erhalten einen DFA C mit höchstens 2^c Zuständen. Für C können wir effizient entscheiden, ob C verwerfende Zustände hat, die vom Startzustand aus erreichbar sind und wir können wiederum $L(A) \neq \Sigma^*$ effizient entscheiden. Dieses Argument kann übernommen werden, um effiziente Approximationen mit Approximationsfaktor $O(\log n)$ für gegebenen NFA mit n Zuständen auszuschließen.

Dass das Problem der Minimierung andererseits großes Interesse hervorruft, zeigt sich in vielen Arbeiten wie [5, 23, 22, 29], die sich mit Heuristiken beschäftigen, die keine Approximationsgarantien liefern oder im Worst Case exponentielle Laufzeit haben. Diesen Ansätzen gehen wir jedoch nicht nach.

Ein erster Schritt zum Nachweis der Nicht-Approximierbarkeit gelingt

uns 2003 in der Arbeit [13], in der wir die Reduktion im Beweis von Stockmeyer und Meyer [38] für die NP-Vollständigkeit der Nicht-Äquivalenz regulärer Ausdrücke über einem unären Alphabet daraufhin untersuchen, ob sie Lücken schaffend ist. Existiert eine Lücken schaffende Reduktion mit Lücke α von einem NP-vollständigen Problem auf ein Minimierungsproblem, so sind effiziente Approximationen mit Approximationsfaktor α ausgeschlossen, falls P $\neq$ NP gilt; siehe Abschnitt 2.7.1. Die Reduktion aus dem ursprünglichen Beweis schafft tatsächlich eine Lücke, wenn wir einige Einschränkungen auf die Instanz für das 3-SAT Problem anwenden, von dem aus reduziert wird, und die Ergebnisse von Jiang, McDowell und Ravikumar [24] über die minimale Anzahl der Zustände von NFAs, die zyklische unäre Sprachen akzeptieren, nutzen.

Unser Ergebnis zeigt, dass es keinen effizienten Approximationsalgorithmus gibt, der zu einem gegebenen (unären) NFA mit n Zuständen die Anzahl der Zustände eines minimalen äquivalenten NFAs, bestimmt und Approximationsfaktor höchstens $\frac{\sqrt{n}}{\ln n}$ hat, falls P $\neq$ NP gilt.

Wenn wir mehr fordern, nämlich nicht nur dass eine obere Schranke für die Zustandszahl angegeben wird, sondern dass sogar ein äquivalenter NFA mit der besprochenen Zustandszahl konstruiert wird, dann können wir schärfere Schranken angeben. In der Arbeit [15] leiten wir das Korollar ab, dass es für kein $\delta > 0$ möglich ist, zu einem (unären) NFA mit n Zuständen effizient einen äquivalenten NFA mit höchstens opt $\cdot n^{1-\delta}$ Zuständen zu konstruieren, falls ein kleinster äquivalenter NFA opt Zustände hat und P $\neq$ NP gilt. Der Beweis beruht darauf, dass wir – unter der Annahme, dass es einen effizienten konstruktiven Algorithmus für das Minimierungsproblem gibt – den erhaltenen NFA durch mehrfache Anwendung des Algorithmus iterativ verbessern können. Polynomiell viele Iterationen dürfen aber nicht ausreichen, um unter den Approximationsfaktor $\frac{\sqrt{n}}{\ln n}$ zu kommen.

Für allgemeine (nicht-unäre) NFAs zeigen wir in [15] ein noch stärkeres Ergebnis: Es ist nicht möglich, die Anzahl der Zustände eines minimalen NFAs, der äquvialent ist zu einem gegebenen NFA mit n Zuständen, innerhalb eines Approximationsfaktors $o(n)$ effizient zu bestimmen, sofern nicht P = PSPACE gilt. Auch für dieses Ergebnis orientieren wir uns an dem ursprünglichen Beweis von Meyer und Stockmeyer [38] und erreichen durch eine Einschränkung der Instanzen für die Reduktion eine große Approximationslücke. Die genannten Ergebnisse gelten auch, wenn man statt der Zahl der Zustände der NFAs die Zahl der Übergänge der NFAs oder die Länge äquivalenter regulärer Ausdrücke betrachtet.

Auch das Problem, zu einem gegebenen DFA die Anzahl der Zustände eines äquivalenten minimalen NFAs oder die Anzahl der Zeichen eines minimalen regulären Ausdrucks zu bestimmen, ist PSPACE-vollständig. Der Nachweis der PSPACE-Schwere gelang erst 1993 [25]. Das Problem erscheint etwas einfacher als das Minimierungsproblem für gegebene NFAs, da zum

einen DFAs bereits effizient minimiert werden können und somit einfache Eigenschaften wie die Äquivalenz zu Σ^* auch einfach entschieden werden können. Andererseits steht für effiziente Verfahren bei Sprachen mit NFAs der Größe n und DFAs der Größe $\Theta(2^n)$ eine Laufzeit zur Verfügung, die exponentiell in der Zielgröße ist.

In der ursprünglichen Reduktion [25] wird durch die Transformation ein DFA A und eine Zahl k generiert, so dass der kleinste zu A äquivalente NFA entweder k oder $k+1$ Zustände hat. Es ist PSPACE-schwer, diese beiden Fälle zu unterscheiden. Um die Nicht-Approximierbarkeit für die Minimierungsprobleme bei gegebenem DFA zu zeigen, ist deshalb der Ansatz, nach Lücken schaffenden Reduktionen suchen, wenig Erfolg versprechend.

Wir bedienen uns daher einer anderen Methode, die auf der Annahme der Existenz von Pseudozufallsfunktionen basiert. Naor und Reingold [32, 36] zeigen unter der Annahme, dass die Faktorisierung von Blum-Zahlen hinreichend schwierig ist, dass es Pseudozufallsfunktionen mit TC^0-Schaltkreisen[2] gibt. Blum-Zahlen sind Zahlen der Form $p \cdot q$ für Primzahlen p und q, die kongruent zu 3 (mod 4) sind. Basierend auf einer Pseudozufallsfunktion $f : \{0,1\}^m \to \{0,1\}$ definieren wir für ein geeignet gewähltes $k = m^{O(1)}$ die Sprache $L_k(f) = \{x^k | f(x) = 1\}$ der k-malig wiederholten Eingaben, die zu 1 evaluieren. Wir zeigen, dass das Komplement dieser Sprache durch einen regulären Ausdruck oder einen NFA mit nur $m^{O(1)}$ vielen Zeichen beziehungsweise Zuständen beschrieben wird und nutzen dazu aus, dass f einen TC^0-Schaltkreis besitzt. Für die überwiegende Mehrheit zufällig gewählter Funktionen $r : \{0,1\}^m \to \{0,1\}$ werden jedoch reguläre Ausdrücke und NFAs für $\overline{L_k(r)}$ mit $\Omega(2^m)$ Zeichen beziehungsweise Zuständen benötigt.

Die Angabe eines Eingabe-DFAs mit $O(2^m \cdot k)$ Zuständen für $\overline{L_k(h)}$ ist einfach für zufällige und pseudozufällige Funktionen h. Ein Approximationsalgorithmus, der eine gute Approximation für die Größe eines äquivalenten minimalen regulären Ausdrucks oder NFAs liefert, erlaubt somit die Unterscheidung einer Pseudozufallsfunktion von einer Zufallsfunktion. Somit ist eine *effiziente* Approximation nicht möglich, wenn Pseudozufallsfunktionen – wie allgemein erwartet – existieren.

Als Ergebnis erhalten wir, dass es keine effizienten Approximationsalgorithmen gibt, die zu einem gegebenen DFA mit n Zuständen die Anzahl der Zeichen eines minimalen äquivalenten regulären Ausdrucks oder die Anzahl der Übergänge eines minimalen äquivalenten NFAs mit Approximationsfaktor besser als $\frac{n}{\text{poly}(\log n)}$ berechnen, falls die Faktorisierung von Blum-Zahlen nicht effizient möglich ist. Es wird also in aller Wahrscheinlichkeit zu jedem effizienten Approximationsalgorithmus DFAs der Größe n und äquivalente NFAs der Größe $\text{poly}(\log n)$ geben, so dass nur äquivalente NFAs mit $\frac{n}{\text{poly}(\log n)}$ Übergängen berechnet werden. Ebenso wird die effiziente Approximation der Anzahl der Zustände eines minimalen äquivalenten NFAs

[2] siehe Abschnitt A.2

innerhalb eines Faktors $\frac{\sqrt{n}}{\text{poly}(\log n)}$ ausgeschlossen.

1.2 Algorithmisches Lernen regulärer Sprachen

Im zweiten Teil der Arbeit beschäftigen wir uns mit dem algorithmischen Lernen regulärer Sprachen, insbesondere mit dem unären Fall.

Im Modell des passiven algorithmischen Lernens ist ein Konzept anhand von gegebenen positiv oder negativ klassifizierten Beispielen zu lernen, ohne dass der Lernalgorithmus Einfluss auf die Wahl der Beispiele hat. Ein grundlegender Ansatz ist hierfür die Bestimmung von Hypothesen, die konsistent sind mit der Klassifizierung der gegebenen Beispiele. Im Problem des minimalen konsistenten DFAs soll ein Lernalgorithmus die minimale Größe eines DFAs bestimmen, der alle positiven Beispiele akzeptiert und alle negativen Beispiele verwirft.

Das zentrale passive Lernmodell ist das des probably approximately correct oder kurz PAC-Lernens. Hier erhält der Lernalgorithmus eine zufällig gezogene Menge klassifizierter Beispiele und soll eine Hypothese aufstellen, die mit hoher Wahrscheinlichkeit nur einen kleinen Anteil aller Wörter anders klassifiziert als das zu lernende Konzept.

Im aktiven Lernen darf der Lernalgorithmus Anfragen an ein Orakel stellen. Die einfachste Form der Anfrage ist die Elementfrage, bei der der Lernalgorithmus um die Klassifizierung eines Beispiels bittet. Weitere Modelle erlauben Anfragen in Form von Hypothesen an das Orakel zu stellen. Eine der natürlichsten Fragen ist die nach der Äquivalenz von Hypothese und Konzept. Falls Hypothese und Konzept nicht äquivalent sind, gibt das Orakel ein Beispiel zurück, das von der Hypothese falsch klassifiziert wird. Auch andere Anfrageformen wie Disjunktheits-, Ober- oder Teilmengenfragen werden in der Literatur betrachtet.

Das Problem des minimalen konsistenten DFAs wurde bereits 1978 von Gold [10] als NP-vollständig klassifiziert. Auch gute effiziente Approximationen sind unwahrscheinlich, da Pitt und Warmuth 1989 in der Konferenzversion zu [35] zeigen, dass unter der Annahme P $\neq$ NP für jede Konstante k keine effizienten Algorithmen existieren, die zu gegebenen Beispielen konsistente DFAs mit opt^k Zuständen konstruieren, wenn ein minimaler konsistenter DFA opt Zustände hat. In derselben Konferenz zeigen Kearns und Valiant [26] unter der Annahme, dass Einwegfunktionen existieren, dass bei Eingabegröße n kein effizienter Approximationsalgorithmus für das Problem des minimalen konsistenten DFAs mit Approximationsfaktor $\text{opt}^\alpha\, n^\beta$ für beliebiges $\alpha \geq 0$ und $0 \leq \beta < 1$ existiert. Mit unseren auf der Existenz von Pseudozufallsfunktionen basierenden Methoden weisen wir nach, dass effiziente Approximationen mit Approximationsfaktor $\frac{n}{\text{poly}(\log n)}$ ausgeschlossen sind, auch wenn konsistente DFAs der Größe $\text{opt} = (\log n)^{O(1)}$ existieren.

Wenn wir uns auf unäre Beispielmengen beschränken, können wir einen

Algorithmus angeben, der einen minimalen konsistenten DFA in polynomieller Zeit konstruiert, auch wenn die Längen der unären Beispielswörter als Binärkodierung vorliegen. Für das Problem des unären minimalen konsistenten NFAs weisen wir nach, dass keine effizienten Algorithmen existieren, falls nicht NP $\subseteq$ DTIME($n^{O(\log n)}$) gilt, können aber einen effizienten Algorithmus angeben, der einen konsistenten NFA mit $O(\text{opt}^2)$ Zuständen konstruiert, wenn der minimale konsistente NFA opt Zustände hat.

Pitt und Valiant zeigen 1988 in [33], dass eine Konzeptklasse $\mathcal{C}$ nicht effizient PAC-lernbar ist, falls das Konsistenzproblem für $\mathcal{C}$ NP-schwer ist, also die Entscheidung, ob eine Menge klassifizierter Beispiele ein konsistentes Konzept in $\mathcal{C}$ besitzt. Diese Aussage gilt, falls RP $\neq$ NP gilt, das heißt solange nicht alle Probleme in NP durch randomisierte Polynomialzeit-Algorithmen mit einseitigem Fehler entscheidbar sind.

Da das Problem des minimalen konsistenten DFAs NP-vollständig ist, ist die Klasse der Sprachen, die durch DFAs der Größe n akzeptiert werden, nicht effizient durch DFAs der Größe n PAC-lernbar, wenn RP $\neq$ NP gilt. Beschränken wir die Konzeptklasse auf unäre Sprachen mit DFAs der Größe n, so können wir diese im PAC-Modell effizient durch DFAs der Größe n lernen. Für unäre NFAs der Größe n folgern wir, dass diese nicht effizient PAC-lernbar sind, sofern NFAs der Größe n als Hypothesen verwandt werden und es Probleme in NP gibt, die nicht durch randomisierte Algorithmen mit einseitigem Fehler und Laufzeit $n^{O(\log n)}$ lösbar sind. Erlauben wir NFAs mit $O(n^2)$ Zuständen als Hypothesen, so sind unäre NFAs der Größe n effizient PAC-lernbar.

Für das aktive Lernen mit Äquivalenzfragen weist Angluin 1990 in [1] nach, dass es keinen Lernalgorithmus gibt, der einen DFA mit n Zuständen durch polynomiell viele Äquivalenzfragen erlernen kann. Dabei beruht dieses Ergebnis auf keinen Komplexitätsannahmen. Schränken wir die zu lernende Klasse auf zyklische DFAs mit primer Zykluslänge und höchstens n Zuständen ein, so können wir zeigen, dass $\Theta(\frac{n^2}{\ln n})$ Äquivalenzfragen nötig und hinreichend sind. Erlauben wir als Hypothesen zyklische DFAs mit höchstens n^d Zuständen für $d \leq n$, so erhalten wir eine obere Schranke von $O(\frac{n^2}{d})$ und eine untere Schranke von $\Omega(\frac{n^2 \cdot \ln d}{d \cdot (\ln n)^2})$ für die Anzahl der benötigten Äquivalenzfragen.

Kapitel 2

Voraussetzungen und Notationen

Mathematische Grundlagen und Notationen, sowie die Definition einiger Komplexitätsklassen finden sich in Kapitel A.

Wir stellen hier einige der grundlegenden Definitionen vor, die wir im Folgenden regelmäßig benötigen. Wer mit der Theorie der regulären Sprachen vertraut ist, kann Abschnitt 2.1 und Abschnitt 2.2 getrost überspringen.

Viele bekannte Sätze werden hier nicht detailliert behandelt, wir versuchen in erster Linie Intuitionen (insbesondere im Abschnitt 2.6 über unäre reguläre Sprachen) zu vermitteln.

2.1 Formale Sprachen

Die Notationen für formale Sprachen sowie für endliche Automaten und reguläre Ausdrücke orientieren sich am Standardwerk von Hopcroft und Ullman [19].

- Der Begriff Buchstabe, Symbol oder Zeichen wird synonym verwendet und bezeichnet die kleinsten Einheiten, aus denen ein Wort besteht.
- Ein Alphabet ist eine endliche Menge von Buchstaben und wird meist mit Σ bezeichnet.
- Ein Wort über einem Alphabet Σ ist eine endliche Konkatenation, also das Hintereinanderschreiben von Buchstaben aus Σ. Die Länge eines Wortes w ist die Anzahl der Buchstaben von w und wird mit $|w|$ bezeichnet. Für zwei Wörter u und v bezeichnet $u \cdot v$ oder auch uv die Konkatenation der beiden Wörter mit Länge $|uv| = |u| + |v|$.
- Das leere Wort, das keinen Buchstaben enthält, bezeichnen wir mit ε. Es gilt $|\varepsilon| = 0$.
- Eine formale Sprache ist eine Menge von Wörtern über einem Alphabet.
- Sind L_1 und L_2 formale Sprachen, so ist die Konkatenation $L_1 \cdot L_2$ durch die Konkatenationen der enthaltenen Wörter definiert. Es ist also $L_1 \cdot L_2 =$

$\{u \cdot v \mid u \in L_1 \wedge v \in L_2\}$.

- Ist L eine formale Sprache, so ist $L^0 = \{\varepsilon\}$, $L^1 = L$ und $L^{i+1} = L^i \cdot L$ für $i \geq 1$.
- $L^* = \bigcup_{i \geq 0} L^i$ ist der Kleenesche Abschluss einer formalen Sprache L. Desweiteren ist $L^+ = \bigcup_{i \geq 1} L^i$ der positive Abschluss und L^+ unterscheidet sich nur dann von L^*, wenn $\varepsilon \notin L$ gilt.
- Die Menge oder Sprache aller Wörter über dem Alphabet Σ ist also Σ^*. Mit $\Sigma^{\leq n} = \bigcup_{i=0}^{n} \Sigma^i$ bezeichnen wir die Menge aller Wörter der Länge höchstens n.

2.2 Reguläre Ausdrücke und Endliche Automaten

Mit Hilfe endlicher Automaten und regulärer Ausdrücke können die regulären Sprachen beschrieben werden. Die Klasse der regulären Sprachen ist innerhalb der Chomsky-Hierarchie [6] die am stärksten eingeschränkte Sprachklasse und wird von den Typ-3 Grammatiken erzeugt. Natürlich können wir unsere Aussagen über NFAs direkt als Aussagen über rechtslineare Grammatiken umformulieren. Die Zahl der Variablen einer rechtslinearen Grammatik entspricht der Zahl der Zustände im zugehörigen NFA, die Zahl der Produktionen enspricht der Zahl der Übergänge. Auf die Darstellung durch Grammatiken werden wir deshalb in dieser Arbeit nicht weiter eingehen.

2.2.1 Reguläre Ausdrücke

Ein regulärer Ausdruck über einem Alphabet Σ ist rekursiv definiert: Als Basis dienen die Fälle

- $\emptyset$ beschreibt die leere Sprache $\emptyset$,
- ε beschreibt die Sprache $\{\varepsilon\}$, die nur das leere Wort enthält,
- a beschreibt für einen Buchstaben $a \in \Sigma$ die Sprache $\{a\}$.

Einige Autoren erlauben auch den Ausdruck Σ, der die Sprache Σ beschreibt.

Wir verwenden $L(R)$ als Bezeichnung für die Sprache, die vom regulären Ausdruck R beschrieben wird. Sind R_1 und R_2 zwei reguläre Ausdrücke, dann beschreibt $(R_1 + R_2)$ die Vereinigung $L(R_1) \cup L(R_2)$ und $(R_1 \cdot R_2)$ oder auch $(R_1 R_2)$ beschreibt die Konkatenation $L(R_1) \cdot L(R_2)$.

Für einen regulären Ausdruck R beschreibt (R^*) den Kleeneschen Abschluss $(L(R))^*$.

Klammern können weggelassen werden, wir nehmen dann an, dass der Kleenesche Abschluss die höchste Bindung hat, gefolgt von der Konkatenation und schließlich der Vereinigung.

2.2.2 Deterministische endliche Automaten

Ein deterministischer endlicher Automat (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ ist definiert über dem Alphabet Σ und besteht aus der Zustandsmenge Q, der Zustandsüberführungsfunktion $\delta : Q \times \Sigma \to Q$, dem Startzustand $q_0 \in Q$ und der Menge der akzeptierenden Zustände $F \subseteq Q$. Die Überführungsfunktion wird in der gewohnten Weise auf Wörter erweitert: Wir definieren $\delta(q, \varepsilon) = q$ und $\delta(q, wa) = \delta(\delta(q, w), a)$ für $a \in \Sigma$ und $w \in \Sigma^*$. Die von M akzeptierte (oder beschriebene) Sprache ist $L(M) = \{w \in \Sigma^* \mid \delta(q_0, w) \in F\}$.

2.2.3 Nichtdeterministische endliche Automaten

Ein nichtdeterministischer endlicher Automat (NFA) $M = (Q, \Sigma, \delta, q_0, F)$ besteht aus den gleichen Komponenten wie ein DFA, allerdings ist das Ergebnis der Zustandsüberführungsfunktion $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ eine Menge möglicher Folgezustände. $\mathcal{P}(Q)$ bezeichnet die Potenzmenge von Q. Für NFAs erweitern wir die Überführungsfunktion $\delta : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ so, dass sie auf Zustandsmengen und Wörtern operiert. Wir definieren

$$\begin{aligned} \delta(q, \varepsilon) &= \{q\} \\ \delta(P, a) &= \bigcup_{q \in P} \delta(q, a) \\ \delta(P, wa) &= \delta(\delta(P, w), a) \end{aligned}$$

für Zustandsmengen $P \subseteq Q$ und Wörter $w \in \Sigma^*$ sowie Buchstaben $a \in \Sigma$. Die von M akzeptierte Sprache ist $L(M) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$.

Alternativ können auch ε-Übergänge zugelassen werden, also die Übergänge $\delta(q, \varepsilon) = P \subseteq Q$ definiert werden. Es finden sich auch Definitionen von NFAs mit einer Menge $Q_0 \subseteq Q$ von Startzuständen, in diesem Fall ist $L(M) = \{w \mid \delta(Q_0, w) \cap F \neq \emptyset\}$.

Wir beschränken unsere Betrachtungen auf NFAs ohne ε-Übergänge mit einem Startzustand.

2.2.4 Äquivalente reguläre Ausdrücke und endliche Automaten

Wir nennen zwei reguläre Ausdrücke oder endliche Automaten N, N' äquivalent, wenn sie dieselbe Sprache beschreiben, also $L(N) = L(N')$ gilt.

2.2.5 Diagramme für endliche Automaten

Es ist üblich, einen endlichen Automaten durch einen Graphen – das Zustandsübergangsdiagramm – darzustellen. Wir beschreiben die Darstellung für NFAs und verweisen darauf, dass jeder DFA auch als NFA dargestellt werden kann, indem die Funktion δ statt Zuständen einelementige Zustandsmengen als Werte annimmt.

Das Zustandsübergangsdiagramm zu einem NFA $M = (Q, \Sigma, \delta, q_0, F)$ ist ein gerichteter (Multi-)Graph $G = (V, E)$ mit Kanten und Knotenmarkierungen. Als Knotenmenge $V := Q$ übernehmen wir die Zustandsmenge. Wir setzen eine mit $a \in \Sigma$ markierte Kante von q nach p ein, wenn $p \in \delta(q, a)$ gilt. Akzeptierende Zustände $q \in F$ werden durch einen Doppelkreis markiert, der Startzustand q_0 wird durch einen eingehenden Pfeil markiert.

Jede Kante im Multigraphen bezeichnen wir als Übergang des NFAs. Gibt es für Zustände q, p mehrere Übergänge von q nach p, so zeichnen wir in der graphischen Darstellung nur eine Kante, die mit allen Buchstaben der Übergänge markiert ist. Wenn wir von der Anzahl der Übergänge in einem NFA sprechen, zählen wir jedoch alle Kanten des zugrunde liegenden Multigraphen.

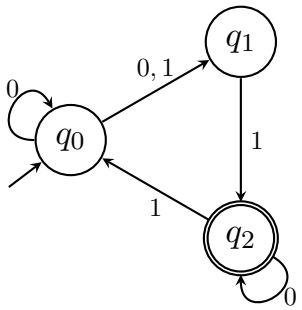


Abbildung 2.1: Zustandsübergangsdiagramm eines NFAs

Abbildung 2.1 stellt das Zustandsübergangsdiagramm eines NFAs N dar. N hat Startzustand q_0, die Menge der akzeptierenden Zustände ist $F = \{q_2\}$ und die Zustandsüberführungsfunktion δ ist wie folgt definiert:

$$\begin{aligned}
\delta(q_0, 0) &= \{q_0, q_1\} \\
\delta(q_0, 1) &= \{q_1\} \\
\delta(q_1, 0) &= \emptyset \\
\delta(q_1, 1) &= \{q_2\} \\
\delta(q_2, 0) &= \{q_2\} \\
\delta(q_2, 1) &= \{q_0\}.
\end{aligned}$$

2.3 Größenmaße für reguläre Ausdrücke und endliche Automaten

Für einen regulären Ausdruck R über dem Alphabet Σ definieren wir die Größe $\text{size}_{\text{re}}(R)$ als die Anzahl der Buchstaben aus Σ, die in R vorkommen. Andere sinnvolle Größenmaße wie die Länge von R, interpretiert als Wort über dem Alphabet $\{\emptyset, \varepsilon, +, \cdot, ^*, (,)\} \cup \Sigma$, oder die Länge von R in

einer Präfix- oder Postfix-Notation beziehungsweise die Anzahl der Knoten in einer Repräsentation als Baum sind bis auf einen konstanten Faktor mit unserer Größendefinition äquivalent, sofern nicht unnötige iterierte Kleenesche Abschlüsse oder Konkatenationen oder Vereinigungen mit ε enthalten sind [23, 8].

Für einen NFA N bezeichnen wir mit $\text{size}_{\text{st}}(N)$ die Zustandszahl und mit $\text{size}_{\text{tr}}(N)$ die Anzahl der Übergänge im Multigraphen des Zustandsübergangsdiagramms. Da wir einen DFA als NFA auffassen können, können wir dieselben Größen-Definitionen auch auf DFAs anwenden. Für einen DFA D über dem Alphabet Σ gilt $\text{size}_{\text{tr}}(D) = |\Sigma| \cdot \text{size}_{\text{st}}(D)$, da aus jedem Zustand genau ein Übergang für jeden Buchstaben aus Σ definiert ist.

2.4 Größenmaße für reguläre Sprachen

Die Größe eines regulären Ausdrucks oder NFAs lässt sich also direkt aus seiner Definition ablesen. Das Problem, das uns hier hauptsächlich beschäftigt, ist die Bestimmung der Größe kleinstmöglicher regulärer Ausdrücke oder NFAs zu einer als regulären Ausdruck, DFA oder NFA gegebenen regulären Sprache. Die genannten Größenmaße sind Maße für die Beschreibungskomplexität der Sprache.

Für eine reguläre Sprache L definieren wir die Größe eines kleinsten regulären Ausdrucks R, der L beschreibt als

$$\text{size}_{\text{re}}(L) := \min\{\text{size}_{\text{re}}(R) \mid R \text{ ist ein regulärer Ausdruck mit } L(R) = L\}.$$

Entsprechend definieren wir die NFA-Ressourcen

$$\begin{aligned} \text{size}_{\text{st}}(L) &:= \min\{\text{size}_{\text{st}}(N) \mid N \text{ ist ein NFA mit } L(N) = L\} \quad \text{und} \\ \text{size}_{\text{tr}}(L) &:= \min\{\text{size}_{\text{tr}}(N) \mid N \text{ ist ein NFA mit } L(N) = L\}. \end{aligned}$$

Fakt 1. *Für eine beliebige reguläre Sprache L gelten die folgenden Beziehungen zwischen den Größenmaßen:*

$$\begin{aligned} \text{size}_{re}(L) &\geq \text{size}_{st}(L) - 1 \\ \text{size}_{tr}(L) &\geq \text{size}_{st}(L) - 1 \end{aligned}$$

Die erste Ungleichung gilt, da sich ein kleinster regulärer Ausdruck für L in einen NFA mit höchstens $\text{size}_{\text{re}}(L)+1$ Zuständen umwandeln lässt [30].

Die zweite Ungleichung ergibt sich aus der Überlegung, dass in einem NFA mit $\text{size}_{\text{tr}}(L)$ Übergängen höchstens $\text{size}_{\text{tr}}(L) + 1$ Zustände vom Startzustand aus durch Übergänge erreichbar sind.

2.5 Minimierungsprobleme für reguläre Sprachen

Die exakte Bestimmung dieser Ressourcen ist im Allgemeinen schwierig. In dieser Arbeit weisen wir nach, dass auch approximative Lösungen schwierig sind.

Das Problem der Bestimmung der Größe eines kleinsten regulären Ausdrucks oder NFAs zu einer (als regulären Ausdruck, DFA oder NFA) gegebenen Sprache wird auch als Minimierungsproblem bezeichnet.

Das zugehörige Entscheidungsproblem wird so formuliert: Gibt es einen regulären Ausdruck, beziehungsweise NFA für L, der Größe höchstens k, wenn L als regulärer Ausdruck, DFA oder NFA spezifiziert ist? Eine Lösung für das Minimierungsproblem, beziehungsweise die Bestimmung von size(L) löst auch das Entscheidungsproblem. Ein effizienter Algorithmus für das Entscheidungsproblem kann andererseits durch Anwendung von binärer Suche auf dem Parameter k das Minimierungsproblem effizient lösen.

Die Bestimmung eines kleinsten regulären Ausdrucks oder NFAs zu einer gegebenen regulären Sprache bezeichnen wir als konstruktive Minimierung. Das Problem der konstruktiven Minimierung ist mindestens so schwierig wie das Minimierungsproblem, da eine optimale konstruktive Lösung die Lösung des Minimierungsproblems liefert. Andererseits ist nicht offensichtlich, wie aus der Lösung des Minimierungsproblems – also der reinen Größenangabe – ein regulärer Ausdruck oder NFA konstruiert werden kann. Das konstruktive Problem könnte also auch deutlich schwieriger sein als das Minimierungsproblem.

2.6 Unäre reguläre Sprachen

Wir nennen eine Sprache, die über einem Alphabet mit nur einem Buchstaben definiert ist, unär. Im Folgenden betrachten wir unäre Sprachen stets über dem unären Alphabet $\Sigma = \{a\}$.

Unäre DFAs, also DFAs, die eine unäre Sprache akzeptieren, haben für jeden Zustand genau einen ausgehenden Übergang. Daraus folgt, dass jeder unäre DFA vom Startzustand aus über einen möglicherweise leeren Pfad von Übergängen in einem Zyklus endet. Die Anzahl der Übergänge im Pfad bezeichnen wir als Pfadlänge, die Anzahl der Übergänge im Zyklus als Zykluslänge.

Für jede reguläre unäre Sprache L gibt es somit eine Konstante c, die Pfadlänge, und eine Konstante d, die Zykluslänge, so dass $\forall i \geq c : a^i \in L \Leftrightarrow a^{i+d} \in L$ gilt.

Für eine reguläre Sprache L kann es verschiedene DFAs geben, aber der minimale DFA, der L akzeptiert, ist bis auf Benennung der Zustände eindeutig bestimmt.

Die Pfadlänge eines beliebigen DFAs für L kann also auch beliebig länger

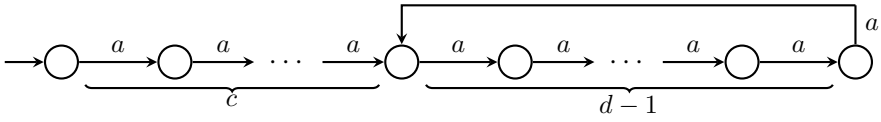


Abbildung 2.2: Ein unärer DFA mit Pfadlänge c und Zykluslänge d

sein. (Man kann sich vorstellen, den Zyklus nach hinten abzurollen.) Die Pfadlänge ist aber stets mindestens so lang wie die des minimalen DFAs. Ebenso ist die Zykluslänge eines DFAs für L mindestens so groß wie die Zykluslänge des minimalen DFAs. Hier gilt zusätzlich, dass jede mögliche Zykluslänge ein ganzzahliges Vielfaches der Zykluslänge des minimalen DFAs ist.

Wir wenden im Folgenden die Begriffe *minimale Pfadlänge* und *minimale Zykluslänge* nicht nur auf DFAs, sondern auch auf reguläre unäre Sprachen an. Für eine Sprache L beziehen wir uns dann auf die Pfadlänge und Zykluslänge des minimalen DFAs für L. Anstelle des Begriffs Zykluslänge verwenden wir auch den Begriff *ultimative Periode*[1] der Sprache. Ist die minimale Pfadlänge von L gleich 0, so nennen wir L *zyklisch* und bezeichnen die ultimative Periode schlicht als *Periode*. Hat eine zyklische Sprache L die Periode d, so bezeichnen wir L auch als d-periodisch.

Für unäre NFAs gibt es eine von Chrobak beschriebene Normalform [7]. Ein NFA für eine unäre Sprache L in Chrobak-Normalform startet mit einem deterministischen Pfad, der dem Pfad eines minimalen DFAs für L bis zum letzten Zustand auf dem Pfad gleicht. Vom letzten Zustand auf dem Pfad verzweigt der NFA dann nichtdeterministisch in mehrere deterministische Zyklen. Eine zyklische Sprache wird durch einen NFA dargestellt, der direkt vom Startzustand aus nichtdeterministisch in die Zyklen verzweigt. (Erlaubt unser NFA-Modell mehrere Startzustände, dann können wir auch den verzweigenden Startzustand weglassen und in jedem Zyklus einen Startzustand definieren.)

Für einen unären NFA N mit $n = \text{size}_{\text{st}}(N)$ Zuständen existiert stets ein äquivalenter NFA N' in Chrobak-Normalform mit höchstens $n^2 + n$ Zuständen im Pfad und die Gesamtzahl der Zustände in allen Zyklen von N' ist höchstens n [7]. Das kleinste gemeinsame Vielfache der einzelnen Zykluslängen eines NFAs in Chrobak-Normalform ist eine ultimative Periode der akzeptierten Sprache. Dies bedeutet sofort, dass der minimale NFA in Chrobak-Normalform für eine unäre Sprache L, deren ultimative Periode eine Primpotenz ist, genau dem minimalen DFA für L gleicht.

Verallgemeinert wird diese Beobachtung durch Jiang, McDowell und Ravikumar [24], die zeigen, dass jeder NFA, der eine zyklische unäre Sprache mit minimaler Periode D mit Primfaktorisierung $D = \prod_{i=1}^{n} p_i^{\alpha_i}$ akzeptiert,

[1] englisch: ultimate period, also letzte / endgültige Periode

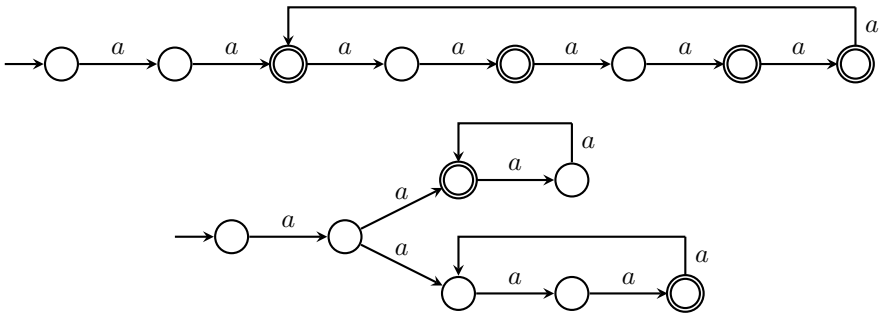


Abbildung 2.3: Ein unärer DFA und ein äquivalenter NFA in Chrobak-Normalform

mindestens $\sum_{i=1}^{n} p_i^{\alpha_i}$ Zustände haben muss.

2.7 Approximationen

Wie in der Einleitung beschrieben, weisen wir nach, dass nicht nur die von uns betrachteten Minimierungsprobleme im Bereich der regulären Sprachen schwierig sind, sondern dass auch gute Approximationen für diese Minimierungsprobleme schwierig sind.

Wir beschränken unsere Betrachtung auf Minimierungsprobleme opt mit Lösungsprädikat Lösung und Zielfunktion size. Wir nennen y eine Lösung für Instanz x, wenn das Prädikat Lösung(x, y) wahr ist. Bei der nicht-konstruktiven Minimierung bestimmen wir für die Eingabe x den optimalen Wert $\text{opt}(x) = \min\{\text{size}(y) \mid \text{Lösung}(x, y)\}$. Bei der konstruktiven Minimierung bestimmen wir eine Lösung y^* mit $\text{size}(y^*) = \text{opt}(x)$. Wir gehen davon aus, dass die Zielfunktion effizient berechenbar ist. Können wir das konstruktive Minimierungsproblem effizient lösen, so können wir auch das nicht-konstruktive Minimierungsproblem effizient lösen. Die Umkehrung gilt im Allgemeinen nicht.

Ein Algorithmus App heißt Approximationsalgorithmus für ein Minimierungsproblem opt mit Approximationsfaktor $\mu \geq 1$, wenn seine Ausgabe $\text{App}(x)$ für jede Eingabe x höchstens um den Faktor μ vom optimalen Wert $\text{opt}(x)$ abweicht, also

$$\text{App}(x) \leq \mu \cdot \text{opt}(x)$$

gilt. Eine Approximation mit Approximationsfaktor 1 bestimmt somit den optimalen Wert.

Da wir nicht nur konstante Approximationsfaktoren betrachten, sondern auch solche, die von der Eingabelänge abhängen, betrachten wir den Approximationsfaktor auch als Funktion $\mu : \mathbb{N} \to \mathbb{R}$ und verlangen dann

$$\text{App}(x) \leq \mu(|x|) \cdot \text{opt}(x).$$

Ein konstruktiver Approximationsalgorithmus App$'$ mit Approximationsfaktor μ berechnet eine Lösung App$'(x)$ mit der Eigenschaft, dass

$$\text{size}(App'(x)) \leq \mu \cdot \text{opt}(x)$$

gilt.

2.7.1 Lücken schaffende Reduktionen

Um Nicht-Approximierbarkeitsergebnisse zu erhalten, nutzen wir das Konzept der Lücken schaffenden Reduktion.

Eine Reduktion von einem Entscheidungsproblem L auf ein Minimierungsproblem opt schafft die Lücke α, wenn ein Wert c sowie eine Transformation T existieren, so dass

$$\begin{array}{rcl} x \in L & \Rightarrow & \text{opt}(T(x)) \leq c \text{ und} \\ x \notin L & \Rightarrow & \text{opt}(T(x)) > \alpha \cdot c \end{array}$$

gilt. Läuft die Transformation T in polynomieller Zeit, sprechen wir von einer Lücken schaffenden Polynomialzeit-Reduktion.

Fakt 2. *Gibt es eine Polynomialzeit-Reduktion von L auf* opt*, die die Lücke α schafft und ist L vollständig für eine Komplexitätsklasse $\mathcal{C}$ mit $P \subset \mathcal{C}$ bezüglich der Polynomialzeitreduktion $\leq_p$, so gibt es keinen effizienten Approximationsalgorithmus für* opt *mit Approximationsfaktor α.*

Beweis. Wir zeigen, wie wir das $\mathcal{C}$-vollständige Problem L mit Hilfe eines effizienten Approximationsalgorithmus App mit Approximationsfaktor α in polynomieller Zeit lösen können. Dies ist aber ein Widerspruch zu P $\neq \mathcal{C}$.

Auf eine Instanz x für L wenden wir die Lücken schaffende Transformation T an und akzeptieren x genau dann, wenn $\text{App}(T(x)) \leq \alpha \cdot c$ gilt. Dieses Vorgehen ist korrekt, weil

$$\begin{array}{rcccl} x \in L & \Rightarrow & \text{opt}(T(x)) \leq c & \Rightarrow & \text{App}(T(x)) \leq \alpha \cdot c \text{ und} \\ x \notin L & \Rightarrow & \text{opt}(T(x)) > \alpha \cdot c & \Rightarrow & \text{App}(T(x)) > \alpha \cdot c \end{array}$$

gilt. Die jeweils letzten Implikationen ergeben sich aus der Definition des Approximationsfaktors, beziehungsweise weil die Approximation nicht kleiner als das Optimum sein kann. □

Kapitel 3

Minimierung regulärer Ausdrücke und NFAs

In diesem Kapitel weisen wir die Nicht-Approximierbarkeit des Minimierungsproblems bei gegebenem NFA oder regulären Ausdruck nach. In Abschnitt 3.1 betrachten wir den unären Fall und in Abschnitt 3.2 betrachten wir Alphabete mit mindestens zwei Buchstaben. Für die Ergebnisse in diesem Kapitel modifizieren wir die aus [38] und [31] bekannten Reduktionen auf die Minimierungsprobleme so, dass diese Lücken schaffen.

3.1 Minimierung unärer regulärer Ausdrücke und NFAs

Der erste Nachweis der Schwere der approximativen Minimierung von NFAs und regulären Ausdrücke zu gegebenen NFAs oder regulären Ausdrücken gelingt uns in [13]. Wir nutzen hier die Vorarbeit von Stockmeyer und Meyer [38], die die NP-Vollständigkeit des Entscheidungsproblems der Nicht-Äquivalenz $L(R) \neq \{a\}^*$ für einen gegebenen regulären Ausdruck oder NFA R über dem Alphabet $\{a\}$ beweisen. Wir zeigen, dass die im folgenden Beweis der NP-Schwere benutzte Reduktion von 3-SAT auf das Problem der Nicht-Äquivalenz Lücken schaffend ist.

Fakt 3. *[38] Es ist NP-schwer, zu entscheiden, ob ein gegebener NFA oder regulärer Ausdruck über dem Alphabet $\{a\}$ nicht die Sprache $\{a\}^*$ beschreibt.*

Beweis. Gegeben sei eine Formel Φ in 3-CNF mit m Klauseln $C_1, \ldots, C_m$ über den Variablen $\{x_1, \ldots, x_n\}$. Wir konstruieren einen NFA N_Φ, der genau dann die Sprache $\{a\}^*$ akzeptiert, wenn Φ nicht erfüllbar ist. Die Konstruktion eines regulären Ausdrucks erfolgt analog.

Mit Hilfe des chinesischen Restsatzes (siehe Fakt 8 in Abschnitt A.3) können wir eine $\{0,1\}$-Belegung der Variablen als natürliche Zahlen definieren: Die Folge $p_1, \ldots, p_n$ sei die Folge der ersten n Primzahlen. Wir definieren

$\mu : \mathbb{N} \to \mathbb{N}^n$ durch

$$\mu(z) = (z \bmod p_1, \ldots, z \bmod p_n)$$

und nennen z die Kodierung einer Belegung, falls $\mu(z) \in \{0,1\}^n$ gilt. Die zugehörige Belegung belegt die Variable x_i mit dem i-ten Eintrag des Vektors $\mu(z)$.

Zu jeder Klausel C_i über den Variablen $x_{i_1}, x_{i_2}, x_{i_3}$ konstruieren wir einen deterministischen zyklischen Teil-Automaten D_i, der genau die Sprache

$$L_i := \left\{ a^k \;\middle|\; \begin{array}{l} \exists \ell \in \{i_1, i_2, i_3\} : \ (k \bmod p_\ell) \geq 2 \text{ oder } \mu(k) \text{ hat} \\ \text{Einträge aus } \{0,1\} \text{ für die Variablen } x_{i_1}, x_{i_2}, x_{i_3}, \\ \text{aber diese Belegung erfüllt die Klausel } C_i \text{ nicht} \end{array} \right\}.$$

akzeptiert. Wir konstruieren den Zyklus D_i der Länge $d_i := p_{i_1} \cdot p_{i_2} \cdot p_{i_3}$ folgendermaßen: D_i besitzt die Zustände $q_{i,j}$ für $0 \leq j < d_i$ und ein Wort a^k erreicht den Zustand $q_{i,k \bmod d_i}$. Nach dem chinesischen Restsatz besteht eine eindeutige Beziehung zwischen dem Zustand $q_{i,k \bmod d_i}$ und den Einträgen von $\mu(k) \in \mathbb{N}^n$ an den Stellen i_1, i_2, i_3. Eingeschränkt auf diese drei Variablen gibt es genau sieben Belegungen, die C_i erfüllen. Wir definieren die zugehörigen sieben Zustände als verwerfend und alle anderen Zustände als akzeptierend.[1]

Den nichtdeterministischen Automaten N_Φ, der die Vereinigung der Teilsprachen L_i akzeptiert, konstruieren wir, indem wir einen neuen Startzustand q_0 einfügen und von diesem nichtdeterministisch unter Lesen von a auf die Zustände $q_{i,1}$ $(1 \leq i \leq m)$ der Teil-Automaten D_i verzweigen. O.B.d.A. komme jede Variable in mindestens einer der Klauseln vor und der Automat akzeptiert die Sprache $L_\Phi :=$

$$\bigcup_{i=1}^{m} L_i = \left\{ a^k \;\middle|\; \begin{array}{l} k \text{ ist keine Kodierung oder es gibt eine Klausel } C_i, \\ \text{so dass die Belegung } \mu(k) \text{ die Klausel } C_i \text{ nicht erfüllt} \end{array} \right\}.$$

Ist Φ nicht erfüllbar, so gilt für jede Belegung $b \in \{0,1\}^n$, dass es eine Klausel gibt, die b nicht erfüllt und somit ist $L(N_\Phi) = \{a\}^*$. Ist Φ andererseits erfüllbar, so gibt es eine Belegung b, die jede Klausel erfüllt. Die zugehörigen Wörter a^k mit $\mu(k) = b$ werden von keinem der Teil-Automaten D_i akzeptiert und somit gilt $L(N_\Phi) \neq \{a\}^*$. □

Da die Sprache $\{a\}^*$ von einem Automaten mit einem Zustand und einem Übergang sowie von einem regulären Ausdruck mit einem a-Symbol beschrieben wird, folgt aus Fakt 3 sofort, dass es NP-schwer ist, zu entscheiden, ob $\text{size}_{\text{st}}(L(R)) > 1$ oder $\text{size}_{\text{tr}}(L(R)) > 1$, beziehungsweise $\text{size}_{\text{re}}(L(R)) > 1$ für einen gegebenen NFA oder regulären Ausdruck R gilt. Es ist somit ausgeschlossen, dass man (unäre) reguläre Ausdrücke oder NFAs effizient minimieren kann, falls P ≠ NP gilt.

[1] Der reguläre Ausdruck für einen solchen Zyklus hat die Form $(A_i)(a^{d_i})^*$, wobei A_i die Menge der $d_i - 7$ Wörter der Länge $< d_i$ beschreibt, die vom Zyklus akzeptiert werden. A_i lässt sich mit Hilfe des Horner-Schemas mit weniger als d_i vielen a-Buchstaben schreiben.

3.1.1 Komplexität der Approximation

Um nachzuweisen, dass auch effiziente Approximationen mit kleinem Approximationsfaktor für das Minimierungsproblem ausgeschlossen sind, zeigen wir den folgenden Satz, dessen Beweis darauf basiert, dass die Reduktion von Stockmeyer und Meyer Lücken schaffend ist.

Satz 1. *Falls $P \neq NP$ gilt, gibt es keinen effizienten Approximationsalgorithmus, der zu einem gegebenen (unären) regulären Ausdruck oder NFA N mit n Buchstaben, beziehungsweise Zuständen oder Übergängen eines der Größenmaße $\text{size}_{re}(L(N)), \text{size}_{st}(L(N))$ oder $\text{size}_{tr}(L(N))$ mit Approximationsfaktor $\frac{\sqrt{n}}{\ln n}$ approximiert.*

Um den Satz zu beweisen, bestimmen wir zunächst eine untere Schranke für die Periode der Sprache L_Φ aus dem Beweis von Stockmeyer und Meyer.

Lemma 1. *Für jede erfüllbare 3-CNF-Formel Φ über n Variablen ist die minimale Periode von L_Φ entweder $D := \prod_{i=2}^{n} p_i$ oder $2D$.*

Beweis. Die Sprache L_Φ ist $2D$-periodisch, da $2D$ das kleinste gemeinsame Vielfache der Zykluslängen von N_Φ ist. Die minimale Periode muss ein Teiler von $2D$ sein. Wir nehmen an, dass L_Φ eine kleinere Periode d als D hat und führen diese Annahme zum Widerspruch. Wenn es eine kleinere Periode als D gibt, dann gibt es ein $i \geq 2$, so dass $d = \frac{D}{p_i}$ eine Periode von L_Φ ist. Es gilt $a^{qp_i+2} \in L_\Phi$ für jedes $q \in \mathbb{N}$, da $qp_i + 2 \equiv 2 \pmod{p_i}$ keine Kodierung ist. Da wir annehmen, dass L_Φ d-periodisch ist, gilt auch $a^{qp_i+rd+2} \in L_\Phi$ für jedes $r \in \mathbb{N}$.

Andererseits gibt es ein Wort a^ℓ, das nicht in L_Φ liegt, da ja Φ erfüllbar ist und somit $L_\Phi \neq \{a\}^*$ gilt. Auch hier führt die Annahme, dass L_Φ d-periodisch ist, zu $a^{\ell+td} \notin L_\Phi$ für $t \in \mathbb{N}$. Wir erhalten den gewünschten Widerspruch, wenn wir $q, r, t \in \mathbb{N}$ finden, so dass $qp_i + rd + 2 = \ell + td$ gilt, da ein Wort mit dieser Länge gemäß der linken Seite der Gleichung in L_Φ liegen muss, während es gemäß der rechten Seite nicht in L_Φ liegen kann.

$$
\begin{array}{lrrcll}
 & \exists q, r, t: & qp_i + 2 + rd & = & \ell + td & \\
\Leftrightarrow & \exists q, r, t: & qp_i & = & \ell - 2 + (t - r)d & \\
\Leftrightarrow & \exists q: & qp_i & \equiv & \ell - 2 & \pmod{d} \\
\Leftrightarrow & \exists q: & q & \equiv & (\ell - 2)p_i^{-1} & \pmod{d}
\end{array}
$$

Die letzte Äquivalenz gilt, da das multiplikative Inverse von $p_i \pmod{d}$ existiert, da $d = \frac{D}{p_i}$ und somit $\text{ggT}(p_i, d) = 1$ gilt. □

Wir möchten nun eine untere Schranke für einen minimalen NFA für L_Φ in Abhängigkeit von der Formellänge herleiten und benötigen eine lineare Beziehung zwischen der Anzahl der Klauseln m und der Anzahl der Variablen n. Wir betrachten hierfür das E3-SAT-E5-Problem. E3-SAT-E5 ist die

Sprache der erfüllbaren booleschen Formeln in 3-CNF, in denen jede Variable in genau fünf Klauseln auftritt und jede Klausel genau drei Literale hat. Das E3-SAT-E5-Problem ist NP-vollständig [9] und die Reduktion aus dem Beweis zu Fakt 3, angewandt auf eine solche Formel Φ, liefert einen NFA oder regulären Ausdruck N_Φ, für den eine effiziente Lösung des Minimierungsproblems eine effiziente Lösung von E3-SAT-E5 impliziert. Wir können eine untere Schranke für die Anzahl der Zustände eines minimalen NFAs für die Sprache L_Φ für den Fall, dass Φ erfüllbar ist, herleiten.

Lemma 2. *Es gibt eine Konstante $c > 0$, so dass für jede Formel $\Phi \in$ E3-SAT-E5 mit m Klauseln jeder NFA, der die Sprache L_Φ akzeptiert, mindestens $cm^2 \ln m$ Zustände hat.*

Beweis. Aus Lemma 1 wissen wir, dass die minimale Periode von L_Φ entweder D oder $2D$ ist, wobei $D = \prod_{i=2}^{n} p_i$ gilt und n die Anzahl der Variablen in Φ ist. Jiang, McDowell und Ravikumar [24] haben gezeigt, dass jeder NFA, der eine zyklische unäre Sprache mit minimaler Periode D mit Faktorisierung $\prod_{i=2}^{n} p_i$ akzeptiert, mindestens $\sum_{i=2}^{n} p_i$ Zustände haben muss. Wir schätzen die i-te Primzahl wie in Abschnitt A.3.1 beschrieben durch $p_i \geq i \ln i$ ab. Jeder NFA für L_Φ muss also mindestens

$$\sum_{i=2}^{n} p_i \geq \sum_{i=1}^{n} i \ln i \geq \int_1^n x \ln x \, dx \geq \frac{n^2}{4} \ln n$$

Zustände haben. Es gilt $5n = 3m$ und somit können wir die untere Schranke auch durch die Anzahl der Klauseln von Φ ausdrücken: Jeder NFA für L_Φ hat mindestens $cm^2 \ln m$ Zustände für eine Konstante c. □

Wir erhalten nach Fakt 1 mit der unteren Schranke für die Zustandszahl auch eine untere Schranke für die Zahl der Übergänge in einem Übergangsminimalen NFA und für die Zahl der Buchstaben in einem minimalen regulären Ausdruck für L_Φ.

Andererseits können wir eine obere Schranke für die Anzahl der Zustände des NFAs N_Φ angeben, der aus der Reduktion entsteht.

Lemma 3. *Für eine Formel Φ in 3-CNF mit m Klauseln und exakt fünf Vorkommen jeder Variable hat der NFA N_Φ höchstens $O(m^4(\ln m)^3)$ Zustände.*

Beweis. Die Anzahl der Zustände in einem Zyklus D_i ist höchstens p_n^3. Für $j \geq 3$ können wir die j-te Primzahl nach oben durch $p_j \leq 2j \ln j$ abschätzen. Wir haben m Klauseln, zu denen jeweils ein Zyklus gehört. Mit Hilfe der linearen Beziehung zwischen n und m können wir die Gesamtzahl der Zustände durch $O(m(m \ln m)^3)$ abschätzen. □

Die gleiche Analyse lässt sich auch für die Anzahl der Übergänge in N_Φ und für die Anzahl der Buchstaben im regulären Ausdruck, der bei der Reduktion zu Fakt 3 entsteht, durchführen. Nun können wir den Beweis zu Satz 1 vollenden.

Beweis zu Satz 1. Wir nehmen an, dass es einen Polynomialzeitalgorithmus App gibt, der zu einem gegebenen NFA oder regulären Ausdruck mit Größe s die Größe eines minimalen äquivalenten NFAs oder regulären Ausdrucks innerhalb des Approximationsfaktors $\frac{\sqrt{s}}{\ln s}$ bestimmt.

Es sei Φ eine Instanz für das E3-SAT-E5-Problem mit m Klauseln. N_Φ sei der NFA aus dem Beweis zu Fakt 3. Falls Φ nicht erfüllbar ist, dann ist die Größe 1 das Optimum und App gibt an, dass die minimale Größe höchstens $\frac{\sqrt{s}}{\ln s}$ ist. Mit $s = O(m^4(\ln m)^3)$ aus Lemma 3 sowie $s \geq m$ ergibt sich

$$\frac{\sqrt{s}}{\ln s} \leq \frac{\sqrt{O(m^4(\ln m)^3)}}{\ln m} = o(m^2 \ln m).$$

Nach Lemma 2 erzeugt andererseits jede erfüllbare Formel Ψ eine zyklische unäre Sprache L_Ψ, für die jeder NFA mindestens Größe $\Omega(m^2 \ln m)$ benötigt. Die von App angegebene Größe ist asymptotisch kleiner und kann somit genutzt werden, um eine erfüllbare Formel zu erkennen. Letztlich können wir mit Hilfe der Reduktion und der Anwendung von App das NP-vollständige Problem E3-SAT-E5 in polynomieller Zeit lösen. □

3.1.2 Komplexität der konstruktiven Approximation

In unserer Arbeit [15] zeigen wir, dass effiziente Approximationen mit noch schwächeren Approximationsfaktoren für das *konstruktive* unäre Minimierungsproblem ausgeschlossen sind. Beim konstruktiven Minimierungsproblem fordern wir vom Approximationsalgorithmus nicht nur, dass *die Größe* eines äquivalenten minimalen regulären Ausdrucks oder NFAs angenähert wird, sondern dass ein äquivalenter approximativ minimaler regulärer Ausdruck oder NFA *konstruiert* wird.

Satz 2. *Falls $P \neq NP$ gilt, dann gilt für jedes $\delta > 0$, dass es keinen effizienten Algorithmus gibt, der zu einem unären regulären Ausdruck oder NFA der Größe n einen äquivalenten regulären Ausdruck oder NFA mit Größe* $\mathrm{opt} \cdot n^{1-\delta}$ *konstruieren kann, wobei* opt *die Größe eines kleinsten äquivalenten regulären Ausdrucks oder NFAs ist.*

Als Größenmaß kann an jeder Stelle entsprechend $\mathrm{size}_{st}(\cdot), \mathrm{size}_{tr}(\cdot)$ *oder* $\mathrm{size}_{re}(\cdot)$ *gewählt werden.*

Beweis. Wir führen den Beweis hier exemplarisch für die konstruktive NFA-Minimierung bezüglich $\mathrm{size}_{\mathrm{st}}(\cdot)$, für die anderen Größenmaße und für reguläre Ausdrücke ist der Beweis entsprechend zu modifizieren.

Wir nehmen an, dass es eine Konstante $\delta > 0$ gibt, so dass ein Polynomialzeitalgorithmus A existiert, der zu einem gegebenen NFA N mit $n = \text{size}_{\text{st}}(N)$ Zuständen einen äquivalenten NFA $A(N)$ mit $\text{size}_{\text{st}}(L(N)) \cdot n^{1-\delta}$ Zuständen berechnet.

Es sei N ein NFA mit $n = \text{size}_{\text{st}}(N)$ Zuständen, der aus der Reduktion im Beweis zu Fakt 3 aus einer Formel für das E3-SAT-E5-Problem entsteht. Die minimale Größe eines äquivalenten NFAs ist entweder $\text{opt} = 1$ oder $\text{opt} > \frac{\sqrt{n}}{\ln n}$. Wir wenden A wiederholt auf seine eigene Ausgabe $A(N)$ an. $A(N)$ ist ein zu N äquivalenter NFA der Größe $\text{size}_{\text{st}}(A(N)) \leq \text{opt} \cdot (\text{size}_{\text{st}}(N))^{1-\delta}$. Für $A(A(N))$ erhalten wir

$$\text{size}_{\text{st}}(A(A(N))) \leq \text{opt} \cdot (\text{size}_{\text{st}}(A(N)))^{1-\delta} \leq \text{opt}^2 \cdot (\text{size}_{\text{st}}(N))^{(1-\delta)^2}.$$

Wiederholen wir diesen Prozess k mal, dann ergibt sich

$$\text{size}_{\text{st}}(A^k(N)) \leq \text{opt}^k \cdot (\text{size}_{\text{st}}(N))^{(1-\delta)^k}.$$

Für $k \geq -\frac{2}{\log(1-\delta)}$ gilt $\text{size}_{\text{st}}(A^k(N)) \leq \text{opt}^k \cdot (\text{size}_{\text{st}}(N))^{\frac{1}{4}}$ und somit folgt $\text{size}_{\text{st}}(A^k(N)) = o(\frac{\sqrt{n}}{\ln n})$, falls $\text{opt} = 1$ gilt, beziehungsweise andernfalls gilt $\text{size}_{\text{st}}(A^k(N)) \geq \text{opt} > \frac{\sqrt{n}}{\ln n}$. □

Damit haben wir die Approximationskomplexität für den unären Fall weitgehend geklärt. Die Frage, ob eine effiziente approximative Bestimmung der minimalen Größe mit besserem Approximationsfaktor als für die konstruktive Minimierung möglich ist, bleibt allerdings offen.

Wir wenden uns nun der Minimierung im nicht-unären Fall zu.

3.2 Minimierung allgemeiner regulärer Ausdrücke und NFAs

In [15] zeigen wir, dass für NFAs und reguläre Ausdrücke N über einem Alphabet mit mindestens zwei Buchstaben effiziente Approximationen mit Approximationsfaktor $o(\text{size}(N))$ sogar unter der schwächeren Annahme $\text{P} \neq \text{PSPACE}$ ausgeschlossen sind.

Auch hier weisen wir nach, dass die Reduktion im ursprünglichen Beweis der PSPACE-Schwere von Meyer und Stockmeyer [31] Lücken schaffend ist, indem wir die Instanz für die Reduktion so einschränken, dass die Mindestgröße der zum Reduktionsergebnis äquivalenten regulären Ausdrücke oder NFAs leicht zu bestimmen ist.

Das folgende Lemma beschreibt die Einschränkungen, die wir später nutzen.

Lemma 4. *Es gibt eine PSPACE-vollständige Sprache, die von einer deterministischen Einband-Turingmaschine M akzeptiert wird, die nur Zellen*

innerhalb des Eingabebereichs besucht und auf Eingaben $x \in L$ mindestens $2^{|x|}$ Schritte läuft. Außerdem hat M nur einen akzeptierenden Zustand und macht in jedem Rechenschritt eine Kopfbewegung.

Beweis. Wir nehmen an, L' sei eine PSPACE-vollständige Sprache, die von einer deterministischen Turingmaschine M' mit polynomieller Platzschranke $f(n) = n^{O(1)}$ akzeptiert wird. Dass es nur einen akzeptierenden Zustand gibt und jeder Rechenschritt auch den Kopf bewegt, können wir mit Standardmethoden garantieren und wir nehmen an, dass dies bereits für M' gilt. Ebenso können wir annehmen, dass M' auf einem nach links beschränkten Band arbeitet.

Mit Hilfe einer Technik, die schon Book [4] anwendet, reduzieren wir L' auf eine Sprache L, die von einer Turingmaschine M mit der geforderten Platzschranke und der Mindest-Laufzeit akzeptiert wird.

Eine Eingabe w für M' wird in polynomieller Zeit transformiert in die Eingabe $x := \triangleright w \mathbb{B}^{f(|w|)-|w|} \triangleleft$. Hierbei sind $\mathbb{B}, \triangleright, \triangleleft$ neue Symbole, die als Blanksymbole und Begrenzer interpretiert werden. M soll also die Sprache $\{\triangleright w \mathbb{B}^{f(|w|)-|w|} \triangleleft \mid w \in L'\}$ akzeptieren. Hierzu prüft die Turingmaschine M zunächst, ob die Eingabe mit $\triangleright$ beginnt und mit $\triangleleft$ abgeschlossen ist und ob die Anzahl der $\mathbb{B}$-Zeichen stimmt. Danach fährt sie wieder zum Anfang der Eingabe und simuliert die Berechnung von M' auf w, wobei $\mathbb{B}$ als das Blank-Symbol interpretiert wird. M besucht nur Zellen auf dem Eingabebereich, da für M' die Platzbeschränkung $f(n)$ gilt.

Wenn die Simulation von M' abgeschlossen ist und der akzeptierende Zustand erreicht wird, müssen wir nur sicherstellen, dass M mindestens $2^{|x|}$ viele Schritte läuft. Dazu zählen wir am Ende der Berechnung einen binären Zähler geeignet hoch. □

Die im Lemma beschriebene Einschränkung auf Turingmaschinen, die nur auf dem Eingabebereich arbeiten, erleichtert uns auch den Beweis des Satzes in [31], den wir hier als Fakt 4 wiedergeben.

Fakt 4. *[31] Es ist PSPACE-schwer, zu entscheiden, ob ein gegebener NFA oder regulärer Ausdruck über einem Alphabet Σ mit $|\Sigma| \geq 2$ nicht die Sprache Σ^* beschreibt.*

Beweis. $M = (Q_M, \Sigma_M, \Gamma_M, \delta, q_0, \{q_f\})$ sei eine wie in Lemma 4 beschriebene Turingmaschine mit akzeptierendem Zustand q_f, die die PSPACE-vollständige Sprache $L(M)$ akzeptiert. Unsere Transformation konstruiert zu einer Eingabe $w = w_1 \dots w_n$ einen regulären Ausdruck R_w mit der Eigenschaft $w \in L(M) \Leftrightarrow L(R_w) \neq \Sigma^*$. Genauer: R_w beschreibt alle Wörter, die *keine* Konkatenation aufeinander folgender legaler Konfigurationen von M sind, die mit $q_0 w$ beginnen und zum akzeptierenden Zustand q_f führen.

Das Alphabet[2] sei $\Sigma = (Q_M \times \Gamma_M) \cup \Gamma_M \cup \{\#\}$, dies erlaubt es uns,

[2]Durch eine entsprechende Kodierung ist es möglich, das Alphabet $\{0, 1\}$ zu nutzen wobei die Eingabe um einen konstanten Faktor verlängert wird.

Folgen von Konfigurationen der Turingmaschine, die durch # getrennt sind, zu beschreiben. Eine legale Konfiguration ist ein Wort der Länge $n = |w|$ aus $\Gamma_M^* \cdot (Q_M \times \Gamma_M) \cdot \Gamma_M^*$ und beschreibt den Bandinhalt, die Position des Kopfes sowie den Zustand. Das Zeichen $[q, a] \in Q_M \times \Gamma_M$ steht hierbei für die Zelle mit Inhalt a, auf der der Kopf steht, während sich M im Zustand q befindet. R_w definieren wir als die Vereinigung der regulären Ausdrücke R_1, R_2, R_3 und R_4.

- R_1 beschreibt alle Wörter, die nicht mit $\#[q_0, w_1]w_2 \dots w_n\#$ beginnen, also

$$R_1 = N_\# \quad + \quad \#(N_{[q_0,w_1]} + [q_0, w_1](N_{w_2} + w_2(N_{w_3} + w_3(\dots \\ \dots (N_{w_n} + w_n N_\#) \dots))))$$

  mit $N_a = \varepsilon + (\Sigma \setminus a)\Sigma^*$. Wir benutzen hierbei abkürzend

$$\Sigma = \sum_{\sigma \in \Sigma} \sigma \text{ und } (\Sigma \setminus a) = \sum_{\sigma \in \Sigma \setminus \{a\}} \sigma.$$

- R_2 beschreibt alle Wörter, die nicht $[q_f, \gamma]$ für beliebige Buchstaben $\gamma \in \Gamma_M$ enthalten.

$$R_2 = \left(\sum_{a \in \Sigma \setminus (\{q_f\} \times \Gamma_M)} a \right)^* .$$

- $R_3 = \Sigma^*(\Sigma \setminus \#)$ beschreibt alle Wörter, die nicht mit # enden.
- R_4 beschreibt jede illegale Änderung des Bandinhalts zwischen zwei aufeinanderfolgenden Konfigurationen: In einer legalen Folge y von Konfigurationen ist für jedes Tripel $y_{i-1}y_iy_{i+1} \in \Sigma^3$ das neue mittlere Symbol y_{i+n+1} eindeutig durch $y_{i-1}y_iy_{i+1}$ bestimmt. Für eine illegale Folge x von Konfigurationen gibt es also entweder eine Position i mit $x_{i+n+1} \neq x_i$, falls der Kopf sich nicht über x_i befindet oder andernfalls ist x_{i+n+1} nicht das Zeichen, das gemäß der Überführungsfunktion δ auf das Band geschrieben werden muss.
  Wir decken zunächst den Fall ab, dass sich der Kopf nicht in der Nähe befindet: Generiere für alle $a_1, a_2, a_3 \in \Gamma_M \cup \{\#\}$ jedes Wort x, welches nicht a_2 an der entsprechenden mittleren Position der Nachfolgekonfiguration hat, durch den regulären Ausdruck

$$\Sigma^* \cdot a_1a_2a_3 \cdot \Sigma^{n-1} \cdot (\Sigma \setminus a_2) \cdot \Sigma^*.$$

  Wenn sich der Kopf an einer der drei Stellen befindet, dann generieren wir für alle $a_1, a_2 \in \Gamma_M$ und jedes $[q, a] \in Q_M \times \Gamma_M$ mit $\delta(q, a) = (q', b, \rightarrow)$ mit $q' \in Q_M$ und $b \in \Gamma_M$ alle illegalen Folgen durch

$$\Sigma^* \cdot [q, a]a_1a_2 \cdot \Sigma^{n-1} \cdot (\Sigma \setminus [q', a_1]) \cdot \Sigma^* \quad +$$

$$\Sigma^* \cdot a_1[q, a]a_2 \cdot \Sigma^{n-1} \cdot (\Sigma \setminus b) \cdot \Sigma^* \quad +$$

$$\Sigma^* \cdot a_1 a_2[q,a] \cdot \Sigma^{n-1} \cdot (\Sigma \setminus a_2) \cdot \Sigma^*.$$

Behandle die Linksbewegungen $[q,a] \in Q_M \times \Gamma_M$ mit $\delta(q,a) = (q', b, \leftarrow)$ entsprechend. R_4 ist schließlich die Vereinigung aller eben beschriebenen regulären Ausdrücke.

Die Konstruktion von R_w ist in polynomieller Zeit bezüglich $|w|$ machbar. Es gilt $L(R_w) \neq \Sigma^* \Leftrightarrow w \in L(M)$. Ein Algorithmus, der zu gegebenem regulären Ausdruck R_w entscheidet, ob die beschriebene Sprache Σ^* ist, löst ein PSPACE-schweres Problem. □

Wenn wir die Größe des im Beweis konstruierten regulären Ausdrucks R_w und die Mindestgröße von regulären Ausdrücken für die von R_w beschriebenen Sprache analysieren, erkennen wir, dass die Transformation eine Lücke schafft.

Satz 3. *Falls $P \neq PSPACE$ gilt, dann gibt es keinen effizienten Approximationsalgorithmus, der zu einem regulären Ausdruck oder NFA der Größe n über einem Alphabet Σ mit $|\Sigma| \geq 2$ die Größe eines minimalen äquivalenten regulären Ausdrucks oder NFAs mit Approximationsfaktor $o(n)$ bestimmt.*

Als Größenmaß kann an jeder Stelle entsprechend $\text{size}_{st}(\cdot), \text{size}_{tr}(\cdot)$ *oder* $\text{size}_{re}(\cdot)$ *gewählt werden.*

Beweis. Wir führen den Beweis zu Fakt 4 fort und bestimmen zunächst die Länge des regulären Ausdrucks R_w. Die Länge des Ausdrucks R_4 dominiert die Gesamtlänge. Die vier Teilausdrücke von R_4 haben jeweils die Länge $|\Sigma| + 3 + (n-1) \cdot |\Sigma| + (|\Sigma| - 1) + |\Sigma|$ und müssen für alle Kombinationen der drei Zeichen aufgeschrieben werden. Als obere Schranke für die Gesamtlänge von R_4 können wir $|\Sigma|^3 \cdot 4 \cdot (|\Sigma| \cdot (|w| + 2) + 3) = O(|w|)$ angeben.

Wir bezeichnen die Länge von R_w mit $m := \text{size}_{\text{re}}(R_w) = O(|w|)$. Im Anhang in Kapitel B findet sich die Beschreibung eines äquivalenten NFAs mit $O(|w|)$ Übergängen und Zuständen. Die Existenz des NFAs mit $O(|w|)$ Zuständen folgt bereits aus der Länge von R_w. Für die Übergänge gilt im Allgemeinen jedoch nur die obere Schranke $O(n \cdot 2^{\log^* n})$ für $\text{size}_{\text{tr}}(L(R))$ für reguläre Ausdrücke R der Länge n über konstant großem Alphabet [37].

Falls M die Eingabe w nicht akzeptiert, dann gibt es keine legale Folge von Konfigurationen, die q_f beinhaltet und es gilt $L(R_w) = \Sigma^*$. Falls jedoch w akzeptiert wird, dann gibt es eine legale Folge y von Konfigurationen, die zu q_f führt. Das Wort y ist das kürzeste Wort, das nicht von R_w erzeugt wird und umfasst mindestens $2^{|w|}$ Konfigurationen, da wir davon ausgehen, dass die Turingmaschine diese Mindestlaufzeit hat.

Um eine Sprache $L \subset \Sigma^*$ mit $L = \Sigma^{\leq k-1} L'$ für eine Sprache $L' \subset \Sigma^*$ zu akzeptieren, benötigt der minimale DFA mindestens k Zustände. Da wir mit Hilfe der Potenzmengenkonstruktion aus jedem NFA mit n Zuständen einen DFA mit höchstens 2^n Zuständen konstruieren können, hat ein minimaler NFA für L also mindestens $\log k$ Zustände. Nach Fakt 1 gilt diese untere

Schranke auch für die Anzahl der Übergänge in NFAs für L und für die Länge regulärer Ausdrücke für L. Angewandt auf die Sprache $L(R_w)$ für Eingaben $w \in L(M)$ folgt, dass jeder NFA oder reguläre Ausdruck für $L(R_w)$ mindestens die Größe $|w|$ hat.

Andererseits kann für $w \notin L(M)$ die Sprache $L(R_w) = \Sigma^*$ von einem NFA mit einem Zustand beziehungsweise $|\Sigma|$ Übergängen und von einem regulären Ausdruck der Größe $|\Sigma|$ beschrieben werden.

Ein Approximationsalgorithmus App mit Approximationsfaktor $o(n) = o(|w|)$ gibt für $w \notin L(M)$ also an, dass die Größe eines minimalen NFAs oder regulären Ausdrucks für $L(R_w)$ durch $o(|w|)$ beschränkt ist, während für den Fall $w \in L(M)$ nur eine Größe mindestens $|w|$ in Frage kommt. App erlaubt uns also die Trennung der beiden Fälle und löst ein PSPACE-schweres Problem. Ist App effizient, widerspricht dies der Annahme P $\neq$ PSPACE. □

Kapitel 4

Minimierung bei gegebenem DFA

Wir weisen jetzt die Nicht-Approximierbarkeit der Größe eines äquivalenten NFAs oder regulären Ausdrucks zu einem gegebenen DFA nach. Da die Eingabe hier exponentiell größer sein kann, als bei NFAs oder regulären Ausdrücken als Eingabe, steht für effiziente Algorithmen zwar deutlich mehr Rechenzeit zur Verfügung, das Problem bleibt aber weiterhin schwierig. Für unsere Nicht-Approximierbarkeitsergebnisse müssen wir allerdings auf stärkere als die in Kapitel 4 genutzten Annahmen P $\neq$ NP oder P $\neq$ PSPACE zurückgreifen.

4.1 Pseudozufallsfunktionen

Für die Untersuchung der Approximierbarkeit minimaler NFAs und regulärer Ausdrücke, die zu einem gegebenem DFA äquivalent sind, nutzen wir kyptografische Annahmen, nämlich die Existenz von Pseudozufallsfunktionen. Pseudozufallsfunktionen wurden von Goldreich et al. [11] eingeführt. Wir orientieren uns an der Definition eines Pseudozufallsensembles und beschränken unsere Betrachtung auf boolesche Funktionen.

Definition 1. *Es sei B_n die Menge aller booleschen n-Bit Funktionen $g : \{0,1\}^n \to \{0,1\}$.*

Ein Funktionenensemble ist eine Folge $f = (f_n)_n$, in der jedes Folgenglied $f_n \subseteq B_n$ eine Menge von Funktionen $\{f_n^s | s \in S_n\}$ für eine Menge von Schlüsseln S_n ist. Wir sagen, dass das Ensemble f effizient berechenbar ist, wenn aus der Angabe der Größe n, eines Schlüssels $s \in S_n$ und der Eingabe $x \in \{0,1\}^n$ der Funktionswert $f_n^s(x)$ effizient – gemessen in n – bestimmt werden kann. Effizient kann in diesem Zusammenhang verschiedene Bedeutungen haben. Für unsere Ergebnisse betrachten wir sowohl Funktionenensembles, die durch (nicht-uniforme) TC^0-Schaltkreise berechnet wer-

den können [32], als auch Funktionenensembles, die in (nicht-uniformem) LogSPACE berechnet werden können. Abschnitt A.2 erklärt die Komplexitätsklassen und den Begriff der Uniformität.

Für unsere Zwecke reicht folgende Definition eines starken Pseudozufallsensembles:

Definition 2. *Sei $f_n = (f_n^s)_{s \in S_n}$ ein Funktionenensemble mit Funktionen $f_n^s \in B_n$ für einen Schlüssel $s \in S_n$ und sei $(r_n^i)_{i \in \{1,\ldots,2^{2^n}\}}$ das Ensemble aller booleschen n-Bit Funktionen, das Zufallsensemble.*

Ein randomisierter Algorithmus A heißt Unterscheider, wenn A in jedem Schritt einen Funktionswert $h_n(x)$ einer unbekannten Funktion h_n für ein Argument x erfragen kann, und am Ende seiner Berechnung eine 0/1*-Ausgabe erzeugt. Wir nennen $|\operatorname{prob}[A(f_n) = 1] - \operatorname{prob}[A(r_n) = 1]|$ den Vorteil, den A beim Unterscheiden von f_n und r_n erreicht. Hierbei sind die Wahrscheinlichkeiten* prob *definiert über die zufälligen Wahlen, die A vornimmt, sowie über die gleichverteilte Wahl der Schlüssel s beziehungsweise i aus der jeweiligen Schlüsselmenge.*

Wir sagen, dass $f = (f_n)_n$ ein starkes Pseudozufallsensemble mit Parameter ε ist, falls für jeden Unterscheider A

$$|\operatorname{prob}[A(f_n) = 1] - \operatorname{prob}[A(r_n) = 1]| < \frac{1}{3},$$

gilt, wenn A in Zeit $2^{O(n^\varepsilon)}$ läuft.

Die übliche Definition für starke Pseudozufallsensembles verlangt sogar, dass der Vorteil für jedes c und genügend großes n kleiner als n^{-c} ist. Offensichtlich impliziert die Existenz von starken Pseudozufallsensembles im üblichen Sinne die Existenz unserer starken Pseudozufallsensembles.

Es wird allgemein angenommen, dass es ein $0 < \varepsilon < 1$ gibt, so dass die Faktorisierung von (zufällig gewählten) n-Bit Blum-Zahlen erwartete Laufzeit $2^{\Omega(n^\varepsilon)}$ benötigt. Naor und Reingold [32] konstruieren ein Funktionensemble, das durch nicht-uniforme TC^0-Schaltkreise berechnet wird und ein starkes Pseudozufallsensemble mit Parameter ε ist, falls Faktorisierung hinreichend schwierig ist: Sie zeigen, dass es einen randomisierten Algorithmus gibt, der n-Bit Blum-Zahlen in Zeit $2^{O(n^\varepsilon)}$ mit Wahrscheinlichkeit $> \frac{1}{n^c}$ für ein $c > 0$ faktorisieren kann, wenn ein Unterscheider mit Laufzeit $2^{O(n^\varepsilon)}$ zur Verfügung steht, der die Pseudozufallsfunktionen aus [32] und Zufallsfunktionen mit Vorteil $> \frac{1}{n^{c'}}$ für ein $c' > 0$ unterscheidet.

4.2 Pseudozufallsfunktionen und Nicht-Approximierbarkeit

Wir nutzen die Annahme der Existenz von Pseudozufallsfunktionen, um gute effiziente Approximationen für das Minimierungsproblem bei gegebenem DFA auszuschließen. Ziel ist die Darstellung von booleschen Funktionen

durch reguläre Ausdrücke oder NFAs, wobei Pseudozufallsfunktionen kurze reguläre Ausdrücke und kleine NFAs haben, während Zufallsfunktionen mit hoher Wahrscheinlichkeit lange reguläre Ausdrücke und große NFAs benötigen.

Wir definieren zu einer Funktion h eine reguläre Sprache $L(h)$, die durch einen DFA einfach zu beschreiben ist und führen ein Funktional $G = (G_n)_n$ mit $G_n : B_n \to \mathbb{N}$ ein, welches die Komplexität einer booleschen Funktion h misst. $G_n(h)$ definieren wir als die minimale Größe $\text{size}_{\text{st}}(L(h)), \text{size}_{\text{tr}}(L(h))$ beziehungsweise $\text{size}_{\text{re}}(L(h))$ von Automaten oder regulären Ausdrücken für $L(h)$.

Wir bestimmen Schranken $t_1(\cdot)$ und $t_2(\cdot)$, so dass jede Funktion f_n^s aus einem Pseudozufallsensemble f_n eine niedrige Komplexität $G_n(f_n^s) \leq t_1(n)$ hat, während für eine zufällig gewählte Funktion r_n^i die Komplexität $G_n(r_n^i)$ mit hoher Wahrscheinlichkeit mindestens $t_2(n)$ ist. Da nach Definition eines starken Pseudozufallsensembles kein Algorithmus die Unterscheidung der beiden Fälle in Zeit $2^{O(n^\varepsilon)}$ mit hoher Wahrscheinlichkeit vornehmen kann, kann auch kein Algorithmus einen kleineren Approximationsfaktor als $\frac{t_2(n)}{t_1(n)}$ erreichen. Die Schranke $t_1(n)$ liegt bei allen unseren Anwendungen im polynomiellen Bereich, während die Schranke $t_2(n)$ jeweils im exponentiellen Bereich liegt.

Um einen DFA für die Sprache $L(h)$ aufzubauen, benötigen wir allerdings den Zugriff auf alle Funktionswerte von h, was für n-stellige Funktionen und $\varepsilon < 1$ nicht in Zeit $2^{O(n^\varepsilon)}$ möglich ist. Wir passen unser Vorgehen deshalb noch an, indem wir den Orakelzugriff durch einen Zugriff auf alle Funktionswerte einer Restriktion ersetzen und in die Analyse des Approximationsfaktors den Parameter ε miteinbeziehen.

Ein erster Kandidat für die Sprache $L(h)$ ist die Sprache der Eingaben mit Funktionswert 1, also $L(h) = \{x \mid h(x) = 1\}$. Allerdings sind reguläre Ausdrücke und NFAs zu schwach, um TC^0 oder LogSPACE-Funktionen in dieser Weise nachzubilden. Wir orientieren uns an dem Ansatz von Pitt und Warmuth [34] und wiederholen Eingaben. Für eine Funktion $h \in B_n$ betrachten wir das Komplement der Sprache

$$L_p(h) := \{x^p \mid x \in \{0,1\}^n \wedge h(x) = 1\} \subseteq \{0,1\}^{n \cdot p}$$

für eine geeignete Anzahl von Wiederholungen p.

Wir zeigen in Lemma 8, dass für diese Sprachen stets DFAs mit $\Theta(n \cdot p \cdot 2^n)$ Zuständen existieren. Diese DFAs dienen als Eingabe-DFAs für die Zustands- oder Übergangsminimierung äquivalenter NFAs beziehungsweise für die Längenminimierung äquivalenter regulärer Ausdrücke.

4.3 Separierende Funktionale und randomisierte Approximationsalgorithmen

Wir definieren in diesem Abschnitt zunächst abstrakt, wann ein Funktional G eine Funktionenklasse $\mathcal{C}$ von der Klasse der zufälligen Funktionen separiert und leiten ein allgemeines Nicht-Approximierbarkeitsergebnis für ein solches Funktional G, für den Fall her, dass $\mathcal{C}$ ein starkes Pseudozufallsensemble enthält.

Definition 3. *Wir definieren die Kompression $k_m : B_n \to B_m$ für $n > m$ angewandt auf eine Funktion $f \in B_n$ als Restriktion von f mit $(k_m(f))(x) = f(0^{n-m}x)$ für $x \in \{0,1\}^m$.*

Wir sagen, dass das Funktional $G = (G_n)_n$ mit $G_n : B_n \to \mathbb{N}$ eine Funktionenklasse $\mathcal{C}$ von der Klasse des Zufallsensembles B_n für monoton wachsende Schranken $t_1(\cdot)$ und $t_2(\cdot)$ separiert, wenn $G_n(f) < t_1(n)$ für jede Funktion $f \in \mathcal{C} \cap B_n$ gilt, während $G_n(h_n) > t_2(n)$ für fast alle Funktionen in B_n gilt, also

$$\lim_{n\to\infty} \frac{|\{h_n \in B_n | G_n(h_n) \leq t_2(n)\}|}{|B_n|} = 0$$

folgt. Außerdem verlangen wir, dass $G_m(k_m(f)) \leq t_1(n)$ für jede Funktion $f \in \mathcal{C} \cap B_n$ und jedes $m < n$ gilt.

Im Folgenden verwenden wir r_n, um eine zufällige gleichverteilt gewählte Funktion aus B_n zu bezeichnen, f_n, um eine n-stellige Funktion aus einem Pseudozufallsensemble mit zufällig gewähltem Schlüssel zu bezeichnen und h_n als eine konkrete n-stellige Funktion, für die der Wert $G_n(h_n)$ approximativ zu bestimmen ist, beziehungsweise für die entschieden werden soll, ob sie pseudozufällig oder zufällig gewählt wurde.

Da wir randomisierte Unterscheider betrachten, führen wir auch den Begriff des randomisierten Approximationsalgorithmus, der Fehler machen darf, ein. Wenn wir effiziente randomisierte Approximationsalgorithmen mit kleinem Approximationsfaktor für ein Optimierungsproblem ausschließen können, sind natürlich auch effiziente deterministische Approximationen mit kleinem Approximationsfaktor ausgeschlossen.

Definition 4. *Wir nennen einen randomisierten Algorithmus $App : X \to \mathbb{N}$ einen Approximationsalgorithmus mit Approximationsfaktor $\mu(\cdot)$ und Überschätzungsfehler ϵ_+ und Unterschätzungsfehler ϵ_- für ein Minimierungsproblem* opt*, falls*

$$\begin{aligned} \epsilon_+ &\geq \sup_{x\in X} \text{prob}[App(x) > \mu(|x|) \cdot \text{opt}(x)] \quad \textit{und} \\ \epsilon_- &\geq \sup_{x\in X} \text{prob}[App(x) < \text{opt}(x)] \end{aligned}$$

gilt. Die Wahrscheinlichkeiten werden hierbei über die zufälligen Wahlen des randomisierten Algorithmus gemessen.

Wir geben nun ein allgemeines Lemma für die Nicht-Approximierbarkeit von Funktionalen an, die Pseudozufallsensembles von Zufallsensembles separieren. Wir gehen bei der Eingabe für die Approximationsalgorithmen davon aus, dass wir alle Funktionswerte der zu untersuchenden Funktion in einer Wertetabelle zur Verfügung gestellt bekommen. Dies ermöglicht uns später, einen DFA für die Funktion zu spezifizieren. Um wieder den Bezug zum Orakelzugriff des Unterscheiders auf eine Funktion h_n herzustellen, arbeitet der Approximationsalgorithmus auf der Wertetabelle der Kompression $k_m(h_n)$. Gelingt eine gute Approximation auf der Wertetabelle der Kompression, die ja weniger Information über h_n zur Verfügung stellt als durch beliebige Orakelzugriffe auf h_n, dann kann diese Approximation auch genutzt werden, um Pseudozufallsensembles von Zufallsensembles mit signifikantem Vorteil zu unterscheiden.

Lemma 5. *Das Funktional G separiere $\mathcal{C}$ von zufällig gewählten Funktionen für Schranken t_1 und t_2. Wir nehmen an, dass $\mathcal{C}$ ein Pseudozufallsensemble mit Parameter ε enthält. App sei ein randomisierter Algorithmus, der den Wert $G_m(h_m)$ für eine gegebene Wertetabelle von $h_m \in B_m$ der Größe 2^m approximiere.*

Dann gilt für jedes $l \geq 1$: Falls App in Zeit $2^{O(m^l)}$ läuft und einen Approximationsfaktor $\mu(2^m) < \frac{t_2(m)}{t_1(m^{l/\varepsilon})}$ hat, dann muss App die Fehler $\epsilon_+ + \epsilon_- \geq \frac{2}{3}$ haben.

Die Länge der Wertetabelle ist 2^m. Diese Eingabelänge wird als Argument für den Approximationsfaktor verwendet.

Beweis. Nach Annahme enthält $\mathcal{C}$ ein starkes Pseudozufallsensemble mit Parameter ε. Wir führen einen Widerspruchsbeweis und nehmen an, dass App bei gegebener Wertetabelle von h_m für ein $l \geq 1$ den Wert $G_m(h_m)$ mit Laufzeit $2^{O(m^l)}$ und Approximationsfaktor $1 \leq \mu(2^m) < \frac{t_2(m)}{t_1(m^{l/\varepsilon})}$ sowie Fehlern $\epsilon_+ + \epsilon_- < \frac{2}{3}$ approximiert. Wir konstruieren einen Unterscheider A, der App als Unterprogramm nutzt und n-Bit Funktionen aus $\mathcal{C}$ von zufällig gewählten n-Bit Funktionen mit Vorteil mindestens $\frac{1}{3}$ unterscheidet.

Wir setzen $m = \lceil n^{\varepsilon/l} \rceil$. A hat Orakelzugriff auf die Eingabefunktion $h_n \in B_n$ und erzeugt die Wertetabelle für die Einschränkung $k_m(h_n)$. A lässt daraufhin App auf $k_m(h_n)$ laufen und akzeptiert, also $A(h_n) = 1$, wenn $\mathrm{App}(k_m(h_n)) \leq t_2(m)$ gilt. Andernfalls verwirft A, also $A(h_n) = 0$. Der Vorteil ist

$$\begin{aligned} & |\operatorname{prob}[A(f_n) = 1] - \operatorname{prob}[A(r_n) = 1]| \\ = \; & |\operatorname{prob}[\mathrm{App}(k_m(f_n)) \leq t_2(m)] - \operatorname{prob}[\mathrm{App}(k_m(r_n)) \leq t_2(m)]|, \end{aligned}$$

wobei die Wahrscheinlichkeiten sowohl über die Zufallsentscheidungen von App als auch über die zufällige Wahl der Schlüssel $s \in S_n$ für $f_n = f_n^s$,

beziehungsweise die gleichverteilt zufällige Wahl der Zufallsfunktion $r_n \in B_n$ definiert ist.

Da G die Klasse $\mathcal{C}$ von der Klasse der zufälligen Funktionen separiert, gilt $G_m(k_m(f_n)) \leq t_1(n)$ für $f_n \in \mathcal{C}$. Nach Annahme gilt für den Approximationsfaktor $\mu(2^m) \cdot t_1(m^{l/\varepsilon}) < t_2(m)$. Weil $m = \lceil n^{\varepsilon/l} \rceil$ gilt und t_1 gemäß Definition 3 monoton wächst, folgt $\mu(2^m) \cdot t_1(n) < t_2(m)$. Wir erhalten

$$\begin{aligned}
& \text{prob}[\text{App}(k_m(f_n)) \leq t_2(m)] \\
\geq\ & \text{prob}[\text{App}(k_m(f_n)) \leq \mu(2^m) \cdot t_1(n)] \\
=\ & 1 - \text{prob}[\text{App}(k_m(f_n)) > \mu(2^m) \cdot t_1(n)] \\
\geq\ & 1 - \text{prob}[\text{App}(k_m(f_n)) > \mu(2^m) \cdot G_m(k_m(f_n))] \\
\geq\ & 1 - \epsilon_+,
\end{aligned}$$

wobei die letzte Ungleichung aus der Definition des Überschätzungsfehlers in Definition 4 folgt. Wir möchten nun $\text{prob}[\text{App}(k_m(r_n)) \leq t_2(m)]$ für gleichverteilt gewählte Funktionen $r_n \in B_n$ abschätzen und bemerken zunächst, dass das gemeinsame Eintreffen der Ereignisse $[G_m(k_m(r_n)) > t_2(m)]$ und $[\text{App}(k_m(r_n)) \geq G_m(k_m(r_n))]$ das Ereignis $[\text{App}(k_m(r_n)) > t_2(m)]$ nach sich zieht. Es folgt

$$\begin{aligned}
& \text{prob}[G_m(k_m(r_n)) > t_2(m) \ \wedge\ \text{App}(k_m(r_n)) \geq G_m(k_m(r_n))] \\
\leq\ & \text{prob}[\text{App}(k_m(r_n)) > t_2(m)] \\
\Rightarrow\ & \text{prob}[\text{App}(k_m(r_n)) \leq t_2(m)] \\
\leq\ & \text{prob}[G_m(k_m(r_n)) \leq t_2(m)] \ +\ \text{prob}[\text{App}(k_m(r_n)) < G_m(k_m(r_n))] \\
\leq\ & \text{prob}[G_m(k_m(r_n)) \leq t_2(m)] \ +\ \epsilon_-.
\end{aligned}$$

Auch hier folgt die letzte Ungleichung aus der Definition des Unterschätzungsfehlers in Definition 4. Da die Kompression einer gleichverteilt gewählten Funktion r_n aus B_n zu einer gleichverteilt gewählten Funktion r_m aus B_m führt, ergibt sich

$$\begin{aligned}
& \text{prob}[\text{App}(k_m(r_n)) \leq t_2(m)] \\
\leq\ & \text{prob}[G_m(k_m(r_n)) \leq t_2(m)] + \epsilon_- \\
=\ & \frac{|\{r_m | G_m(r_m) \leq t_2(m)\}|}{|B_m|} + \epsilon_- \\
=\ & \epsilon_- + o(1).
\end{aligned}$$

Für den Vorteil folgt

$$\begin{aligned}
& |\,\text{prob}[A(f_n) = 1] - \text{prob}[A(r_n) = 1]| \\
=\ & |\,\text{prob}[\text{App}(k_m(f_n)) \leq t_2(m)] \ -\ \text{prob}[\text{App}(k_m(r_n)) \leq t_2(m)]| \\
\geq\ & 1 - \epsilon_+ - \epsilon_- - o(1) \\
>\ & \frac{1}{3}
\end{aligned}$$

für genügend große Werte von m. Der Unterscheider A hat Laufzeit $O(2^m) + 2^{O(m^l)} = 2^{O(n^\varepsilon)}$, was aber der Annahme widerspricht, dass $\mathcal{C}$ ein starkes Pseudozufallsensemble mit Parameter ε enthält. □

4.4 Reguläre Ausdrücke und Formeln logarithmischer Tiefe

Wir arbeiten später mit einem von Naor und Reingold in [32] beschriebenen Pseudozufallsensemble $\mathcal{C}_1$, dessen Funktionen sehr einfache Formeln besitzen.

Definition 5. *Eine Formel mit Eingabe $x = x_1x_2\cdots x_n \in \{0,1\}^n$ ist ein binärer Baum mit $\wedge$ und $\vee$ Gattern als inneren Knoten. Die Blätter sind mit Literalen aus $\{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$ beschriftet. Für eine Formel $\mathbf{f}$ sei $\ell(\mathbf{f})$ die Länge der Formel, also die Anzahl der Blätter von $\mathbf{f}$. Die Tiefe der Formel ist die Tiefe des Baumes.*

Das Pseudozufallsensemble von Naor und Reingold liegt in nicht-uniformem $TC^0 \subseteq NC^1$. Ein NC^1-Schaltkreis lässt sich in eine Formel verwandeln, indem man Gatter, deren Ausgaben mehrfach als Eingaben anderer Gatter verwendet werden, dupliziert. Für das Funktionenensemble $\mathcal{C}_1 \subset NC^1$ gibt es eine Konstante c, so dass jede Funktion $f_m \in \mathcal{C}_1 \cap B_m$ eine Formel $\mathbf{f}$ der Tiefe höchstens $c \cdot \log m$ hat. Die Länge $\ell(\mathbf{f})$ einer solchen Formel ist höchstens $p_1(m) := m^c$.

Wir konstruieren aus einer Formel $\mathbf{f}$ für $f_m \in \mathcal{C}_1 \cap B_m$ der Länge $\ell(\mathbf{f}) \leq p_1(m)$ einen regulären Ausdruck für das Komplement $\overline{L_{p_1(m)}(f_m)}$ der Sprache $L_{p_1(m)}(f_m) = \{x^{p_1(m)} \mid f_m(x) = 1\}$. Als Komplexitätsmaß $G_m^{(1)}(h_m)$ wählen wir $\mathrm{size}_{\mathrm{re}}(\overline{L_{p_1(m)}(h_m)})$, also die Länge eines kürzesten regulären Ausdrucks für $\overline{L_{p_1(m)}(h_m)}$. Wir zeigen in Lemma 6, dass wir für Funktionen $f_m \in \mathcal{C}_1 \cap B_m$ reguläre Ausdrücke der Länge $O(m^{2c+1})$ erhalten, während wir in Lemma 7 zeigen, dass fast alle Funktionen $r_m \in B_m$ reguläre Ausdrücke der Länge $\Omega(2^m)$ benötigen.

Wir geben zunächst die grundlegende Konstruktion eines regulären Ausdrucks $R(\mathbf{f})$ an, für den $L(R(\mathbf{f})) \cap W = L_{\ell(\mathbf{f})}(f)$ gilt, wobei $\mathbf{f}$ eine Formel für f ist und $W = \{w \mid \exists x \in \{0,1\}^m \wedge w \in \{x\}^*\}$ die Menge aller Wiederholungen für Wörter aus $\{0,1\}^m$ ist.

Definition 6. *Es sei $\mathbf{f}$ eine Formel für eine Funktion $f : \{0,1\}^m \to \{0,1\}$. Wir definieren den regulären Ausdruck $R(\mathbf{f})$ folgendermaßen rekursiv:*

- *Für $\mathbf{f} = x_i$, setze $R(\mathbf{f}) := (0+1)^{i-1}\ 1\ (0+1)^{m-i}$.*
- *Für $\mathbf{f} = \overline{x_i}$, setze $R(\mathbf{f}) := (0+1)^{i-1}\ 0\ (0+1)^{m-i}$.*
- *Für $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$, setze $R(\mathbf{f}) := R(\mathbf{f}_1) \cdot R(\mathbf{f}_2)$.*
- *Für $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$, setze $R(\mathbf{f}) := R(\mathbf{f}_1) \cdot (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \cdot R(\mathbf{f}_2)$.*

Für die Projektion auf eine Variable x_i erzeugen wir also alle Wörter, die an der i-ten Stelle eine 1 haben. Das sind genau die Eingaben x, für die $f(x) = 1$ gilt. Das Vorgehen für $\overline{x_i}$ ist das gleiche, nur dass wir hier die 0 an der i-ten Stelle erzwingen. Für die Konjunktion beschreibt der reguläre Ausdruck alle Wörter, die in der ersten Hälfte Eingaben sind, die unter f_1 zu 1 evaluieren, während in der zweiten Hälfte die 1-Eingaben von f_2 erzeugt werden. Erzwingt man durch den Schnitt mit W, dass der erste und zweite Teil gleich sind, erhalten wir die wiederholten Eingaben, die für beide Funktionen den Funktionswert 1 haben. Für die Disjunktion erzwingen wir in der ersten Hälfte eine 1-Eingabe für f_1 und beschreiben beliebige Wörter für die zweite Hälfte oder wir erzwingen in der zweiten Hälfte die 1-Eingabe für f_2 und beschreiben beliebige Wörter in der ersten Hälfte. Nach dem Schnitt mit W erhalten wir die wiederholten Eingaben, die unter f_1 oder unter f_2 zu 1 evaluieren.

Lemma 6. *Es sei $W = \{w \mid \exists x \in \{0,1\}^m \wedge w \in \{x\}^*\}$ die Sprache der beliebig oft wiederholten Eingaben der Länge m. Es gilt:*
(a) $L(R(\mathbf{f})) \cap W = \{x^{\ell(\mathbf{f})} \mid f(x) = 1\} = L_{\ell(\mathbf{f})}(f)$.
(b) Falls $\mathbf{f}$ *eine Formel der Tiefe höchstens k ist, hat der reguläre Ausdruck* $R(\mathbf{f})$ *höchstens die Länge* $2 \cdot 4^k m$.
(c) Für eine Formel $\mathbf{f}$ *der Tiefe k gibt es einen regulären Ausdruck* $\overline{R_\mathbf{f}}$ *der Länge $O(4^k m)$, der das Komplement von $L_{\ell(\mathbf{f})}(f)$ beschreibt.*
(d) Insbesondere hat die Sprache $\overline{L_{p_1(m)}(f_m)}$ *reguläre Ausdrücke der Länge höchstens* $t_1^{(1)}(m) = \Theta(m^{2c+1})$ *für jede Funktion* $f_m \in \mathcal{C}_1 \cap B_m$.

Beweis. Teil (a) zeigen wir per Induktion über die Struktur der Formel $\mathbf{f}$: Für die Länge der beschriebenen Wörter gilt offensichtlich $L(R(\mathbf{f})) \subseteq \{0,1\}^{m \cdot \ell(\mathbf{f})}$. Falls $\mathbf{f} = x_i$, dann ist $R(\mathbf{f}) = (0+1)^{i-1}\ 1\ (0+1)^{m-i}$ und somit

$$L(R(\mathbf{f})) \cap W = \{x | x \in \{0,1\}^m \wedge x_i = 1\} = \{x | f(x) = 1\}.$$

Der Fall $\mathbf{f} = \overline{x_i}$ folgt analog. Falls $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$, dann gilt $R(\mathbf{f}) = R(\mathbf{f}_1) \cdot R(\mathbf{f}_2)$ und somit

$$\begin{aligned}
L(R(\mathbf{f})) \cap W &= (L(R(\mathbf{f}_1)) \cdot L(R(\mathbf{f}_2))) \cap W \\
&= ((L(R(\mathbf{f}_1)) \cap W) \cdot (L(R(\mathbf{f}_2)) \cap W)) \cap W \\
&= \Big(\{x^{\ell(\mathbf{f}_1)} | f_1(x) = 1\} \cdot \{x^{\ell(\mathbf{f}_2)} | f_2(x) = 1\}\Big) \cap W \\
&= \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_1(x) = 1 \wedge f_2(x) = 1\} \\
&= \{x^{\ell(\mathbf{f})} | f(x) = 1\}.
\end{aligned}$$

Falls $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$, dann gilt $R(\mathbf{f}) = R(\mathbf{f}_1) \cdot (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \cdot R(\mathbf{f}_2)$

und somit

$$\begin{aligned} L(R(\mathbf{f})) \cap W &= (L(R(\mathbf{f}_1)) \cdot \{0,1\}^{m\cdot\ell(\mathbf{f}_2)} \cup \{0,1\}^{m\cdot\ell(\mathbf{f}_1)} \cdot L(R(\mathbf{f}_2))) \cap W \\ &= ((L(R(\mathbf{f}_1)) \cdot \{0,1\}^{m\cdot\ell(\mathbf{f}_2)}) \cap W) \\ &\quad \cup ((\{0,1\}^{m\cdot\ell(\mathbf{f}_1)} \cdot L(R(\mathbf{f}_2))) \cap W) \\ &= \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_1(x) = 1\} \cup \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_2(x) = 1\} \\ &= \{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} | f_1(x) = 1 \vee f_2(x) = 1\} \\ &= \{x^{\ell(\mathbf{f})} | f(x) = 1\}. \end{aligned}$$

(b) Wir bezeichnen die maximale Länge eines regulären Ausdrucks $R(\mathbf{f})$ für eine Formel $\mathbf{f}$ der Tiefe k mit $s(k)$ und zeigen induktiv, dass $s(k) \leq 2 \cdot 4^k m$ gilt. Für $k = 0$ ergibt sich $s(0) = 2m - 1 \leq 2m = 2 \cdot 4^k m$.

Für Formeln $\mathbf{f}_1$ und $\mathbf{f}_2$ der Tiefe höchstens k hat der reguläre Ausdruck $R(\mathbf{f}_1 \wedge \mathbf{f}_2)$ die Länge höchstens $2s(k) \leq 2 \cdot 2 \cdot 4^k m = 4^{k+1} m$, und der reguläre Ausdruck $R(\mathbf{f}_1 \vee \mathbf{f}_2)$ hat die Länge höchstens

$$\begin{aligned} 2 \cdot s(k) + 2m(\ell(\mathbf{f}_1) + \ell(\mathbf{f}_2)) &\leq 2 \cdot 2 \cdot 4^k m + 2m \cdot (2^k + 2^k) \\ &\leq 2m \cdot (2^{2k+1} + 2^{k+1}) \\ &\leq 2m \cdot (2^{k+1} \cdot (2^k + 1)) \\ &\leq 2m \cdot (2^{k+1})^2 \leq 2 \cdot 4^{k+1} m. \end{aligned}$$

(c) Wir benötigen einen kurzen regulären Ausdruck für $\overline{L_{\ell(\mathbf{f})}(f)}$ und bemerken zunächst, dass die Tiefe von $\mathbf{f}$ nicht wächst, wenn wir $\mathbf{f}$ mit De-Morgan komplementieren. Somit hat $L(R(\overline{\mathbf{f}}))$ einen regulären Ausdruck der Länge höchstens $2 \cdot 4^k m$. Wenn wir annehmen, dass $\overline{L(R(\mathbf{f}))} \cap \{0,1\}^{m\cdot\ell(\mathbf{f})} = L(R(\overline{\mathbf{f}}))$ gilt, was wir als nächstes per Induktion zeigen, dann erhalten wir mit Teil (a)

$$\begin{aligned} \overline{L_{\ell(\mathbf{f})}(f)} &= \overline{L(R(\mathbf{f})) \cap W} \\ &= \overline{L(R(\mathbf{f}))} \cup \overline{W} \\ &= L(R(\overline{\mathbf{f}})) \cup \{x \in \{0,1\}^* : |x| \neq m \cdot \ell(\mathbf{f})\} \cup \overline{W}, \end{aligned}$$

da $L(R(\mathbf{f})) \subseteq \{0,1\}^{m\cdot\ell(\mathbf{f})}$ gilt. Wir können für $\overline{W}$ mit dem regulären Ausdruck

$$\big((0+1)^*\, 1\, (0+1)^{m-1}\, 0\, (0+1)^*\big) \;+\; \big((0+1)^*\, 0\, (0+1)^{m-1}\, 1\, (0+1)^*\big)$$

prüfen, ob die Eingabe nicht aus Wiederholungen besteht und decken alle Wörter falscher Länge mit $(0+1+\varepsilon)^{m\cdot\ell(\mathbf{f})-1} \;+\; (0+1)^{m\cdot\ell(\mathbf{f})+1}(0+1)^*$ ab.

Um $\overline{L(R(\mathbf{f}))} \cap \{0,1\}^{m\cdot\ell(\mathbf{f})} = L(R(\overline{\mathbf{f}}))$ zu beweisen, bleibt zu zeigen, dass für jedes Wort $w \in \{0,1\}^{m\cdot\ell(\mathbf{f})}$ entweder $w \in L(R(\mathbf{f}))$ oder $w \in L(R(\overline{\mathbf{f}}))$ gilt. Dies ist nicht offensichtlich, da w ein Wort sein kann, das nicht aus Wiederholungen besteht.

Falls $\mathbf{f} = x_i$, dann gilt $R(\mathbf{f}) = (0+1)^{i-1}1(0+1)^{m-i}$. Andererseits folgt $\overline{\mathbf{f}} = \overline{x_i}$ und somit $R(\overline{\mathbf{f}}) = (0+1)^{i-1}0(0+1)^{m-i}$. Wir nehmen nun an, dass $w_1 \dots w_m \in \{0,1\}^m$ gilt und somit offensichtlich $w \in L(R(\mathbf{f})) \Leftrightarrow w \notin L(R(\overline{\mathbf{f}}))$, da w_i entweder 0 oder 1 ist. Der Fall $\mathbf{f} = \overline{x_i}$ folgt analog.

Die Annahme gelte für $\mathbf{f}_1$ und $\mathbf{f}_2$ und es sei $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$. Somit gilt

$$R(\mathbf{f}) = R(\mathbf{f}_1) \cdot (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \cdot R(\mathbf{f}_2)$$

und

$$R(\overline{\mathbf{f}}) = R(\overline{\mathbf{f}_1} \wedge \overline{\mathbf{f}_2}) = R(\overline{\mathbf{f}_1}) \cdot R(\overline{\mathbf{f}_2}).$$

Es sei $w \in \{0,1\}^{m \cdot \ell(\mathbf{f})}$, dann folgt

$$\begin{aligned}
w \in L(R(\mathbf{f})) \quad &\Leftrightarrow \quad w_1 \dots w_{m \cdot \ell(\mathbf{f}_1)} \in L(R(\mathbf{f}_1)) \\
&\qquad\qquad \vee\ w_{m \cdot \ell(\mathbf{f}_1)+1} \dots w_{m \cdot \ell(\mathbf{f})} \in L(R(\mathbf{f}_2)) \\
&\Leftrightarrow \quad w_1 \dots w_{m \cdot \ell(\mathbf{f}_1)} \notin L(R(\overline{\mathbf{f}_1})) \\
&\qquad\qquad \vee\ w_{m \cdot \ell(\mathbf{f}_1)+1} \dots w_{m \cdot \ell(\mathbf{f})} \notin L(R(\overline{\mathbf{f}_2})) \\
&\Leftrightarrow \quad w_1 \dots w_{m \cdot \ell(\mathbf{f}_1)} \cdot w_{m \cdot \ell(\mathbf{f}_1)+1} \dots w_{m \cdot \ell(\mathbf{f})} \\
&\qquad\qquad \notin L(R(\overline{\mathbf{f}_1})) \cdot L(R(\overline{\mathbf{f}_2})).
\end{aligned}$$

Die letzte Äquivalenz folgt, da nach Konstruktion der regulären Ausdrücke $R(\overline{\mathbf{f}_1})$ und $R(\overline{\mathbf{f}_2})$ für die Länge der erzeugten Wörter $L(R(\overline{\mathbf{f}_i})) \subseteq \{0,1\}^{m \cdot \ell(\overline{\mathbf{f}_i})}$ gilt. Wir führen die Kette der Äquivalenzen fort und erhalten das gewünschte Ergebnis, dass für ein Wort $w \in \{0,1\}^{m \cdot \ell(\mathbf{f})}$ entweder $w \in L(R(\mathbf{f}))$ oder $w \in L(R(\overline{\mathbf{f}}))$ gilt:

$$\begin{aligned}
&\Leftrightarrow \quad w \notin L(R(\overline{\mathbf{f}_1}) \cdot R(\overline{\mathbf{f}_2})) \\
&\Leftrightarrow \quad w \notin L(R(\overline{\mathbf{f}})).
\end{aligned}$$

Der Beweis für $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$ ergibt sich analog.

(d) Da alle Funktionen in $\mathcal{C}_1 \cap B_m$ Formeltiefe höchstens $c \cdot \log m$ haben, können wir auch annehmen, dass alle diese Formeln exakt die Tiefe $c \cdot \log m$ und Länge $p_1(m) = m^c$ haben. Mit Teil (a) stimmt $L_{p_1(m)}(f_m)$ mit $L(R(\mathbf{f})) \cap W$ überein und mit Teil (c) hat $\overline{L_{p_1(m)}(f_m)}$ reguläre Ausdrücke der Länge $O(4^{c \log m} m) = O(m^{2c+1})$. □

Wir wählen als $\mathcal{C}_1 \subset NC^1$ das von Naor und Reingold [32] vorgestellte Funktionenensemble, das ein starkes Pseudozufallsensemble mit Parameter ε ist, wenn die Faktorisierung von Blum-Zahlen hinreichend schwierig ist. Es gibt eine Konstante c, so dass alle Funktionen in $\mathcal{C}_1 \cap B_m$ Formeln der Tiefe $c \cdot \log m$ und Formeln der Länge $p_1(m) = m^c$ haben.

Wir wissen nach Lemma 6, dass für Funktionen $f_m \in \mathcal{C}_1 \subset NC^1$ reguläre Ausdrücke für $\overline{L_{p_1(m)}(f_m)}$ der Länge höchstens $t_1^{(1)}(m) = \Theta(m^{2c+1})$ existieren. Im folgenden Lemma zeigen wir, dass fast alle m-Bit Funktionen reguläre Ausdrücke der Länge mindestens $\Omega(2^m)$ benötigen.

Lemma 7. *Die Zahl der Sprachen, die von regulären Ausdrücken der Länge höchstens* $t_2^{(1)}(m) = \frac{2^m}{40}$ *beschrieben werden, ist beschränkt durch* $\sqrt{2^{2^m}} = o(|B_m|)$.

Beweis. Wir wählen eine sehr grobe Abschätzung mit Hilfe der Länge eines regulären Ausdrucks R in umgekehrt polnischer Notation. Die rpn-Länge eines regulären Ausdrucks R über dem Alphabet Σ ist die Anzahl der Symbole aus $\Sigma \cup \{+, \cdot, ^*, \varepsilon, \emptyset\}$, die wir benötigen, wenn R in umgekehrt polnischer Notation aufgeschrieben wird. Gilt für die Länge $\text{size}_{\text{re}}(R) \leq t$, so hat R rpn-Länge höchstens $6t$ [23].

An jeder Position im regulären Ausdruck kann eines der sieben Symbole $0, 1, +, \cdot, ^*, \varepsilon, \emptyset$ stehen. Wir sind nur an einer oberen Schranke für die Anzahl der verschiedenen regulären Ausdrücke interessiert und ignorieren, dass durch beliebige Symbol-Wahl auch ungültige Audrücke entstehen. Es gibt also höchstens $\sum_{j \leq 6t} 7^j \leq 7^{7t} \leq 2^{20t}$ verschiedene reguläre Ausdrücke mit rpn-Länge höchstens $6t$. Die Behauptung folgt, da $2^{20t_2^{(1)}(m)} = 2^{20\frac{2^m}{40}} = 2^{2^{m-1}}$ gilt. □

Das Funktional $G^{(1)}$ misst die Länge kürzester regulärer Ausdrücke. Um zu zeigen, dass $G^{(1)}$ das Pseudozufallsensemble $\mathcal{C}_1$ vom Zufallsensemble separiert, müssen wir ebenfalls nachweisen, dass die Sprache $\overline{L_{p_1(m)}(k_m(f_n))}$ für die Einschränkung $k_m(f_n)$ keine längeren regulären Ausdrücke benötigt als die Sprache $\overline{L_{p_1(n)}(f_n)}$. Es gilt $G_m^{(1)}(k_m(f_n)) \leq t_1^{(1)}(n)$ für Funktionen $f_n \in \mathcal{C}_1 \cap B_n$, da eine Formel $\mathbf{f}$ für f_n in eine Formel für $k_m(f_n)$ derselben Tiefe umgewandelt werden kann: Bei der Restriktion werden die Blätter der Variablen x_i für $1 \leq i \leq n - m$ in der Formel auf konstante Werte gesetzt. Blätter müssen aber nach Definition 5 mit Literalen beschriftet sein. Wir betrachten das Gatter des Elternknotens der konstanten Blätter und ersetzen den entsprechenden Teilbaum. In Abbildung 4.1 werden die einzelnen Ersetzungen dargestellt. Wenn der Wert des Gatters eine Konstante ist (zum Beispiel $0 \wedge x_i$), so ersetzen wir den Teilbaum durch $x_1 \wedge \overline{x_1}$, beziehungsweise $x_1 \vee \overline{x_1}$. Falls der Wert des Gatters eine Variable ist (zum Beispiel $1 \wedge x_i$), so setzen wir beide Blätter des Teilbaums auf diese Variable.

Wir wissen nach Lemma 6 und Lemma 7, dass $G^{(1)}$ das Ensemble $\mathcal{C}_1$ vom Zufallsensemble für Schranken $t_1^{(1)}(m) = \Theta(m^{2c+1})$ und $t_2^{(1)}(m) = \frac{2^m}{40}$ separiert. Wir können also Lemma 5 anwenden, um zu zeigen, dass für eine gegebene Funktion h keine effizienten Approximationsalgorithmen mit kleinem Approximationsfaktor existieren, die die Länge eines kürzesten regulären Ausdrucks für $\overline{L_{p_1(m)}(h)}$ approximieren. Wir müssen aber die Eingabe für den Approximationsalgorithmus als Wertetabelle angeben. Unser Ziel ist jedoch ein Ergebnis für die Minimierung von regulären Ausdrücken bei gegebenem DFA. Wir zeigen, dass wir zu einer gegebenen Wertetabelle für h einen nicht zu großen DFA für $\overline{L_{p_1(m)}(h)}$ erzeugen können.

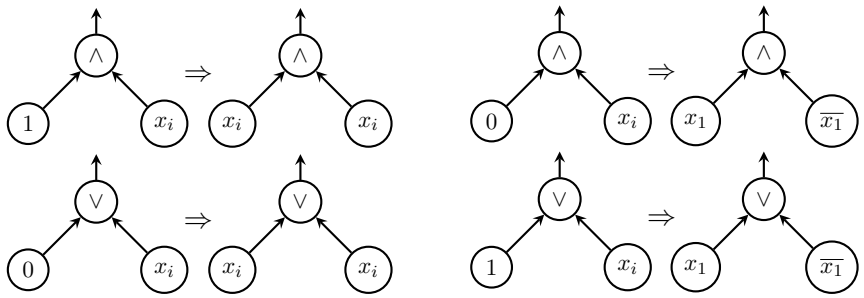


Abbildung 4.1: Ersetzen von konstanten Blättern in der Formel der Restriktion

Lemma 8. *Es sei $h \in B_m$ und p eine Funktion mit $p(m) = 2^{O(m)}$. Dann gibt es einen DFA $D_p(h)$, der $\overline{L_{p(m)}(h)} = \overline{\{x^{p(m)} \mid x \in \{0,1\}^m \wedge h(x) = 1\}}$ akzeptiert, $\Theta(m \cdot 2^m \cdot p(m))$ Zustände hat und in Zeit $\mathrm{poly}(2^m)$ für ein Polynom $\mathrm{poly}(\cdot)$ konstruiert werden kann.*

Beweis. Der DFA $D_p(h)$ besteht aus einem binären Baum der Tiefe m, dessen Wurzel der Startzustand ist. Ein Blatt des Baums, das von einem Wort x mit $h(x) = 0$ erreicht wird, erhält für jedes Bit einen Übergang zu einem akzeptierenden Trap-Zustand. Ein Blatt, das von einem Wort x mit $h(x) = 1$ erreicht wird, ist Startpunkt eines Pfades der Länge $m \cdot (p(m) - 1)$, der nur von der Eingabe $x^{p(m)-1}$ verfolgt werden kann. Der Pfad endet mit einem verwerfenden Zustand. Ein falscher Buchstabe auf dem Pfad sowie jede längere Eingabe führt zum akzeptierenden Trap-Zustand. Jeder Zustand außer den beschriebenen Endpunkten der Pfade ist akzeptierend. (Siehe Abbildung 4.2.) Der DFA hat $\Theta(m \cdot 2^m \cdot p(m))$ Zustände. □

Da nach Lemma 5 Laufzeit $2^{O(m^l)}$ nicht ausreicht, um gute Approximationen zu liefern, erhalten wir den folgenden Satz.

Satz 4. *Es existiere eine Konstante c und ein starkes Pseudozufallsensemble $\mathcal{C}_1$ mit Parameter ε, so dass jede Funktion $f \in \mathcal{C}_1 \cap B_m$ eine Formel der Tiefe $c \cdot \log m$ habe.*

App sei ein randomisierter Algorithmus, der die Länge eines kürzesten äquivalenten regulären Ausdrucks für einen gegebenen DFA mit k Zuständen approximiere. Dann gibt es ein Polynom poly*, so dass für jedes $l \geq 1$ gilt: Falls App in Zeit $2^{O((\log k)^l)}$ läuft und einen Approximationsfaktor $\mu(k) < \frac{k}{\mathrm{poly}((\log k)^{l/\varepsilon})}$ hat, dann muss App die Fehler $\epsilon_+ + \epsilon_- \geq \frac{2}{3}$ haben.*

Beweis. Wir betrachten einen randomisierten Algorithmus App mit Laufzeit $2^{O((\log k)^l)}$ und Fehlern $\epsilon_+ + \epsilon_- < \frac{2}{3}$, der die Länge eines kürzesten äquivalenten regulären Ausdrucks für einen gegebenen DFA mit k Zuständen approximiert. Wir zeigen, wie wir mit Hilfe von App eine Approximation

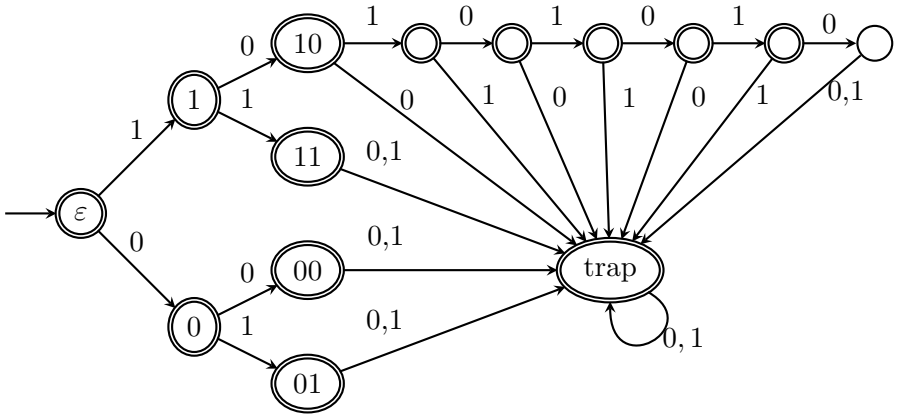
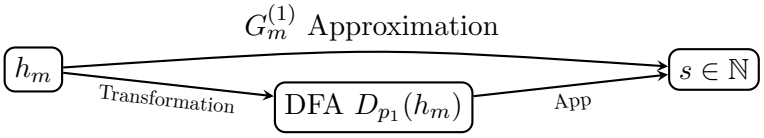


Abbildung 4.2: Ein DFA für $\overline{L_4(x_1 \wedge \neg x_2)}$

für die Länge eines regulären Ausdrucks für die Sprache $\overline{L_{p_1(m)}(h_m)}$ zu einer gegebenen Funktion h_m erhalten. Nach Lemma 5 ist aber eine gute Approximation für $G_m^{(1)}(h_m)$ nicht erreichbar und App kann nur schwach approximieren.

Wir erinnern daran, dass alle Pseudozufallsfunktionen $f_m \in \mathcal{C}_1 \cap B_m$ Formeln der Tiefe $c \cdot \log m$ und der Länge $p_1(m) = m^c$ haben. Desweiteren hat die Sprache $\overline{L_{p_1(m)}(f_m)}$ nach Lemma 6 (d) einen regulären Ausdruck der Länge $t_1^{(1)}(m) = \Theta(m^{2c+1})$. Andererseits existieren für fast alle Funktionen $r_m \in B_m$ nach Lemma 7 für die Sprache $\overline{L_{p_1(m)}(r_m)}$ nur reguläre Ausdrücke mit Mindestlänge $t_2^{(1)}(m) = \frac{2^m}{40}$.

Nach Lemma 5 sind Approximationen mit besserem Approximationsfaktor als $\frac{t_2(m)}{t_1(m^{l/\varepsilon})}$ nicht möglich. Allerdings approximieren wir nicht direkt den Wert des Funktionals $G_m^{(1)}(h_m)$, sondern transformieren zunächst mit Lemma 8 die zu untersuchende Funktion h_m in einen DFA $D_{p_1}(h_m)$ als Eingabe für App.



Für eine beliebige Funktion $h_m \in B_m$ hat die Sprache $\overline{L_{p_1(m)}(h_m)}$ nach Lemma 8 einen DFA $D_{p_1}(h_m)$ mit $k = \Theta(2^m \cdot m^{c+1})$ Zuständen und der DFA ist in Zeit $2^{O(m)}$ konstruierbar. Als nächstes wenden wir App auf $D_{p_1}(h_m)$ an und erhalten eine Approximation von $G_m^{(1)}(h_m)$ mit Approximationsfaktor

$\mu(k)$. Wir wissen, dass

$$\mu(k) \geq \frac{t_2(m)}{t_1(m^{l/\varepsilon})} = \Omega\left(\frac{2^m}{m^{(2c+1)\cdot(l/\varepsilon)}}\right)$$

gelten muss. Nun erweitern wir den Bruch, um den Parameter m durch die Zustandszahl $k = \Theta(2^m \cdot m^{c+1})$ ersetzen zu können, wodurch wir

$$\frac{2^m}{m^{(2c+1)\cdot(l/\varepsilon)}} = \frac{2^m \cdot m^{c+1}}{m^{c+1} \cdot m^{(2c+1)\cdot(l/\varepsilon)}} = \Theta\left(\frac{k}{m^{c+1} \cdot m^{(2c+1)\cdot(l/\varepsilon)}}\right)$$

erhalten und haben somit für ein geeignet gewähltes Polynom „poly“ und genügend große Zustandszahl k

$$\mu(k) \geq \frac{k}{\mathrm{poly}((\log k)^{l/\varepsilon})}$$

nachgewiesen. □

Bemerkung 1. *Wir werden das Vorgehen im Beweis von Satz 4 noch mehrfach anwenden. Die einzigen Unterschiede, die sich ergeben, sind die verschiedenen Werte der Schranken t_1 und t_2. Es sind jeweils effiziente Approximationen mit Approximationsfaktor $\frac{t_2(m)}{t_1(m^{l/\varepsilon})}$ ausgeschlossen.*

Für m-stellige Pseudozufallsfunktionen in $\mathcal{C}_1$ gibt es also stets DFAs mit $k = \Theta(m \cdot 2^m \cdot p_1(m))$ Zuständen und kürzesten äquivalenten regulären Ausdrücken der Länge $\mathrm{poly}(\log k)$, so dass ein effizienter Approximationsalgorithmus nur reguläre Ausdrücke der Länge $\frac{k}{\mathrm{poly}(\log k)}$ bestimmen kann. Jiang und Ravikumar [25] stellen die Frage, ob sich zu gegebenem DFA D ein äquivalenter NFA mit höchstens $\mathrm{size}_{\mathrm{st}}(L(D))^c$ Zuständen effizient finden lässt. Diese Frage, bezogen auf die Länge regulärer Ausdrücke, können wir jetzt verneinen unter der Annahme der Existenz von Pseudozufallsensembles in NC^1. Auch in Bezug auf die Anzahl der Zustände oder Übergänge von NFAs hat die Frage eine negative Antwort modulo kryptographischer Annahmen.

4.5 Approximation der Anzahl der Zustände und Übergänge von NFAs

In diesem Abschnitt übertragen wir unsere Erkenntnisse über die Nicht-Approximierbarkeit von regulären Ausdrücken auf NFAs. Wir definieren die Funktionale $G^{(2)}$ und $G^{(3)}$, wobei $G^{(2)}_m(h_m) = \mathrm{size}_{\mathrm{st}}(\overline{L_{p_1(m)}(h_m)})$ die minimale Anzahl von Zuständen und $G^{(3)}_m(h_m) = \mathrm{size}_{\mathrm{tr}}(\overline{L_{p_1(m)}(h_m)})$ die minimale Anzahl von Übergängen ist, die benötigt wird, um $\overline{L_{p_1(m)}(h_m)}$ durch einen NFA zu akzeptieren. Wir verwenden weiterhin $p_1(m) = m^c$ aus dem

vorherigen Abschnitt als eine obere Schranke für die Länge kurzer Formeln für Funktionen in $\mathcal{C}_1 \cap B_m$ und erhalten obere Schranken $t_1^{(2)}$ für die Zahl der Zustände und $t_1^{(3)}$ für die Zahl der Übergänge aus der oberen Schranke $t_1^{(1)} = \Theta(m^{2c+1})$ für die Länge eines kürzesten regulären Ausdrucks für $\overline{L_{p_1(m)}(h_m)}$.

Wir setzen $t_1^{(2)} = t_1^{(1)} + 1$ und $t_1^{(3)} = \left(2 \cdot t_1^{(1)} + 2\right)^2$, womit wir ausnutzen, dass nach Fakt 1 $\text{size}_{\text{st}}(L) \leq \text{size}_{\text{re}}(L) + 1$ gilt und $\text{size}_{\text{tr}}(L)$ höchstens quadratisch in $(|\Sigma| \cdot \text{size}_{\text{st}}(L))$ ist. Somit haben alle Funktionen in $\mathcal{C}_1$ NFAs mit höchstens $t_1^{(2)}$ Zuständen beziehungsweise $t_1^{(3)}$ Übergängen. Andererseits werden für fast alle Funktionen $h_m \in B_m$ NFAs mit mindestens $t_2^{(2)}(m) = 2^{\frac{m}{2}-1}$ Zuständen, beziehungsweise $t_2^{(3)}(m) = \frac{2^m}{20m}$ Übergängen benötigt, um $\overline{L_{p_1(m)}(h_m)}$ zu akzeptieren:

Lemma 9. *(a) Die Anzahl der Sprachen, die von NFAs mit höchstens $t_2^{(2)}(m) = 2^{\frac{m}{2}-1}$ Zuständen akzeptiert wird, ist durch $\sqrt{2^{m+2^m}} = o(|B_m|)$ beschränkt.*

(b) Die Zahl der Sprachen, die von NFAs mit höchstens $t_2^{(3)}(m) = \frac{2^m}{20m}$ Übergängen akzeptiert wird, ist höchstens $\sqrt{2^{2^m}} = o(|B_m|)$.

Beweis. (a) $N(k)$ sei die Zahl der verschiedenen Sprachen über einem Alphabet mit zwei Buchstaben, die von NFAs mit höchstens k Zuständen akzeptiert werden. Dann gilt nach Domaratzki, Kisman und Shallit [8]: $N(k) \leq 2k \cdot 2^{2 \cdot k^2}$ und es folgt

$$\begin{aligned} N(t_2^{(2)}(m)) &\leq \frac{2}{2} \cdot 2^{\frac{m}{2}} \cdot 2^{2 \cdot \left(2^{\frac{m}{2}}/2\right)^2} = 2^{\frac{m}{2}} \cdot 2^{\frac{2}{4} \cdot \left(2^{(m/2)}\right)^2} \\ &= 2^{\frac{m}{2}} \cdot 2^{\frac{2^m}{2}} = \sqrt{2^{m+2^m}}. \end{aligned}$$

(b) Wir zeigen im Folgenden für $k \geq 4$ die sehr grobe Abschätzung $M(k) = k^{10k}$ als obere Schranke für die Anzahl der Sprachen über einem Alphabet mit zwei Buchstaben, die von NFAs mit höchstens k Übergängen akzeptiert werden. Diese Abschätzung genügt uns, um die Behauptung zu zeigen. Für $t_2^{(3)}(m) = \frac{2^m}{20m}$ folgt

$$M(t_2^{(3)}(m)) = \left(\frac{2^m}{20m}\right)^{10\frac{2^m}{20m}} \leq 2^{10m \cdot \frac{2^m}{20m}} = \sqrt{2^{2^m}}.$$

Die Schranke $M(k)$ ergibt sich aus folgenden Überlegungen: Für jeden NFA N mit s Zuständen und k Übergängen gibt es einen äquivalenten NFA N' mit $s+1$ Zuständen und höchstens $2k$ Übergängen, der genau einen akzeptierenden Zustand hat. Wir erhalten N', indem wir einfach einen akzeptierenden Zustand f hinzufügen und alle anderen Zustände nicht-akzeptierend machen, aber für jeden Übergang, der in N zu einem akzeptierenden Zustand

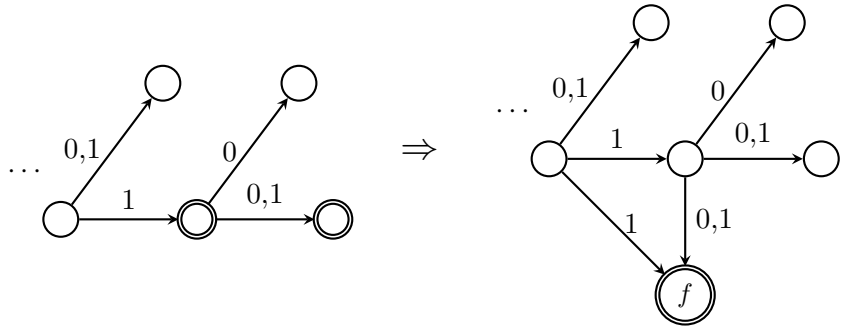


Abbildung 4.3: Konstruktion eines NFAs mit nur einem akzeptierenden Zustand

führt, einen Übergang nach f hinzufügen. Alle anderen Übergänge behalten wir bei. (Siehe Abbildung 4.3.)

Es gibt höchstens $\binom{(s+1)^2}{2k}^2 \cdot s^2 \leq s^{8k+2}$ verschiedene Sprachen über dem Alphabet $\{0,1\}$, die von NFAs mit s Zuständen und k Übergängen akzeptiert werden, da dies eine obere Schranke für die Zahl der Möglichkeiten ist, $2k$ Übergänge für jeden der Buchstaben des Alphabets $\{0,1\}$ zu setzen bei gleichzeitiger Wahl eines Startzustands und eines akzeptierenden Zustands.

Wir wollen die Schranke $M(k)$ nur in Abhängigkeit der Zahl k der Übergänge ausdrücken und schätzen gemäß Fakt 1 $s \leq k+1$ ab. Es gibt deshalb höchstens $(k+1)^{8k+2}$ verschiedene Sprachen, was für $k \geq 4$ durch $M(k) = k^{10k}$ beschränkt ist. □

Wir erinnern uns an Bemerkung 1 und nutzen das Vorgehen mit den Schranken $t_1^{(2)} = \Theta(m^{2c+1})$ und $t_2^{(2)} = 2^{\frac{m}{2}-1}$ für die Zustandsminimierung, beziehungsweise $t_1^{(3)} = \Theta(m^{4c+2})$ und $t_2^{(3)} = \frac{2^m}{20m}$ für die Übergangsminimierung.

Korollar 1. *Es existiere eine Konstante c und ein starkes Pseudozufallsensemble $\mathcal{C}_1$ mit Parameter ε, so dass jede Funktion $f \in \mathcal{C}_1 \cap B_m$ eine Formel der Tiefe $c \cdot \log m$ habe. App sei ein randomisierter Algorithmus, der für einen gegebenen DFA D mit k Zuständen die Größe* $\text{size}_{st}(L(D))$*, beziehungsweise* $\text{size}_{tr}(L(D))$ *approximiere.*

Dann gibt es ein Polynom poly*, so dass für jedes $l \geq 1$ gilt: Falls App in Zeit $2^{O((\log k)^l)}$ läuft und einen Approximationsfaktor $\mu(k) < \frac{\sqrt{k}}{\text{poly}((\log k)^{l/\varepsilon})}$, beziehungsweise $\mu(k) < \frac{k}{\text{poly}((\log k)^{l/\varepsilon})}$ hat, dann muss App die Fehler $\epsilon_+ + \epsilon_- \geq \frac{2}{3}$ haben.*

Bessere Schranken für die Nicht-Approximierbarkeit der Zustandszahl sind mit unserem kryptographischen Ansatz nicht zu erwarten, da jede Funktion $h \in B_m$ immer einen NFA für $\overline{L_p(h)}$ mit $O(2^{\frac{m}{2}} + p \cdot m)$ Zuständen hat, wie wir gleich sehen werden.

Wir konstruieren zunächst einen NFA $N_{\overline{h}}$ mit ε-Übergängen und weniger als $4 \cdot 2^{\frac{m}{2}}$ Zuständen, der $L_1(\overline{h})$ akzeptiert. Unser NFA besteht aus einem Zustand q_w und p_w für jedes Wort $w \in \{0,1\}^*$ mit $|w| \leq \frac{m}{2}$. Der Startzustand ist q_ε und p_ε ist der einzige akzeptierende Zustand. Es gibt Übergänge $\delta(q_w, \sigma) = \{q_{w\sigma}\}$ für $\sigma \in \{0,1\}$ und $|w| < \frac{m}{2}$. Wir bauen also zunächst einen binären Baum auf. Für $|w| = \frac{m}{2}$, fügen wir ε-Übergänge $\delta(q_w, \varepsilon) = \{p_x \mid x \in \{0,1\}^{\frac{m}{2}} \wedge h(wx) = 0\}$ zu den Blättern p_x eines zweiten binären Baums ein, die den Suffixen x entsprechen, die zum Funktionswert 0 führen.

Die restlichen Übergänge sind $\delta(p_{\sigma w}, \sigma) = \{p_w\}$ für $\sigma \in \{0,1\}$ und $|w| < \frac{m}{2}$. Es gibt keine Übergänge $\delta(p_{\sigma w}, \sigma')$ für $\sigma \neq \sigma'$. (Siehe Abbildung 4.4.)

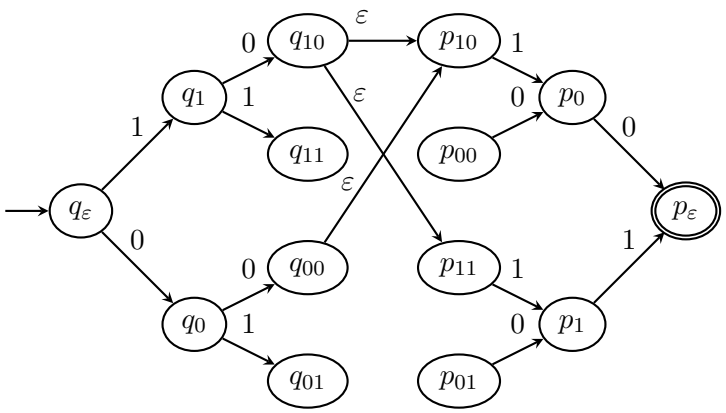


Abbildung 4.4: Der NFA $N_{\overline{h}}$ für $L_1(\overline{h})$ mit $h(x) = x_2 \vee \overline{x_3} \vee (\overline{x_1} \wedge x_4)$

Ein Wort y, welches von $N_{\overline{h}}$ akzeptiert wird, muss die Länge m haben und den Funktionswert $h(y) = 0$, andererseits gibt es für jede Eingabe y mit $h(y) = 0$ einen Pfad vom Startzustand zum akzeptierenden Zustand.

Wir verwandeln $N_{\overline{h}}$ in einen Automaten für $(L_1(\overline{h}))^+$, indem wir einen ε-Übergang vom akzeptierenden Zustand zum Startzustand einsetzen.

Wir bauen außerdem wie in Abbildung 4.5 dargestellt einen NFA mit $2(m+1)$ Zuständen auf, der jedes Wort akzeptiert, das keine Wiederholung eines Wortes der Länge m ist.

Zusätzlich konstruieren wir einen NFA mit $p \cdot m + 2$ Zuständen, der alle Wörter akzeptiert, die nicht die Länge $p \cdot m$ haben. Die Vereinigung der drei beschriebenen NFAs akzeptiert $\overline{L_p(h)}$ mit $O(2^{\frac{m}{2}} + p \cdot m)$ Zuständen. Das Entfernen der ε-Übergänge können wir nach der bekannten Methode [19] vornehmen und erhöhen dabei die Zahl der Zustände nicht.

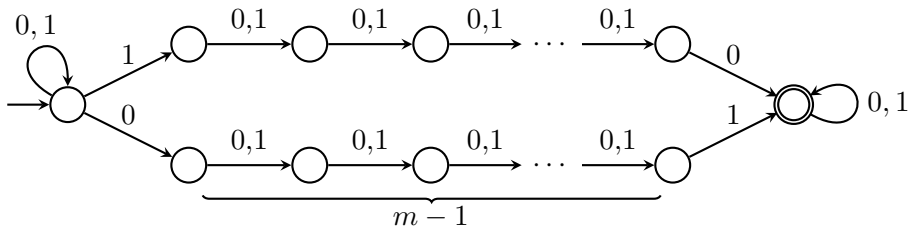


Abbildung 4.5: Ein NFA für
$\big((0+1)^*\ 1\ (0+1)^{m-1}\ 0\ (0+1)^*\big)\ +\ \big((0+1)^*\ 0\ (0+1)^{m-1}\ 1\ (0+1)^*\big)$

4.6 NFAs und Zweiweg-DFAs polynomieller Größe

Anstelle der Annahme der Existenz starker Pseudozufallsensembles in nicht-uniformem NC^1, die in Korollar 1 verwendet wird, können wir für die Nicht-Approximierbarkeit der NFA-Größe bei gegebenem DFA auch eine schwächere Annahme nutzen, nämlich die der Existenz starker Pseudozufallsensembles in nicht-uniformem LogSPACE.

Wir charakterisieren nicht-uniformen LogSPACE durch Zweiweg-DFAs mit polynomiell vielen Zuständen. Dies ist möglich, da eine Turingmaschine mit $|Q|$ Zuständen und Arbeitsbandalphabet Γ, die bei Eingaben der Länge m höchstens $c \cdot \log m$ Zellen auf dem Arbeitsband besucht, durch einen Zweiweg-DFA mit höchstens $|Q| \cdot |\Gamma|^{c \cdot \log m}$ vielen Zuständen akzeptiert werden kann. Ein Zustand des DFAs gibt dabei den Inhalt des Arbeitsbandes sowie den Zustand der Turingmaschine wieder.

Auch hier gehen wir wieder den Weg von Pitt und Warmuth [34] und zeigen, dass jeder Zweiweg-DFA effizient von einem NFA simuliert werden kann, wenn wir die Eingabe genügend oft wiederholen.

Lemma 10. *Es seien m und k natürliche Zahlen. Dann gibt es zu jedem Zweiweg-DFA A_m mit höchstens m^k Zuständen einen NFA N_m mit $O(m^{k+1})$ Zuständen und Übergängen, der das Komplement von*

$$L_{m^{k+1}}(A_m) := \{x^{m^{k+1}} \mid x \in \{0,1\}^m \wedge A_m \text{ akzeptiert } x\}$$

akzeptiert.

Beweis. Wir gehen davon aus, dass A_m als Zweiweg-DFA stets hält. Dann kann A_m höchstens $p(m) = m \cdot m^k$ Schritte auf Eingaben $x \in \{0,1\}^m$ machen, da keine Zelle zweimal im selben Zustand besucht werden kann. Wie in [34] gezeigt, kann A_m auf Eingabe $x \in \{0,1\}^m$ durch einen Einweg-DFA D_m mit $p(m)$ Zuständen, der auf der Eingabe $x^{p(m)}$ arbeitet, simuliert werden.

Der NFA N_m entscheidet sich zunächst nichtdeterministisch, ob er D_m mit negiertem Akzeptanzverhalten laufen lässt, oder ob er überprüft, dass die Eingabe die falsche Länge hat oder nicht aus Wiederholungen besteht. N_m hat $O(p(m))$ Zustände beziehungsweise Übergänge. □

Wir können nun die Aussage von Korollar 1 unter der schwächeren Hypothese beweisen, dass starke Pseudozufallsfunktionen in B_m existieren, die von Zweiweg-DFAs mit höchstens m^k Zuständen für ein festes k akzeptiert werden.

Um Lemma 5 anwenden zu können, wählen wir als Klasse $\mathcal{C}_2$ ein starkes Pseudozufallsensemble, dessen Funktionen in $\mathcal{C}_2 \cap B_m$ von Zweiweg-DFAs mit höchstens m^k Zuständen berechnet werden. Das Funktional $G_m(h_m)$ misst nun für eine Funktion $h_m \in B_m$ die Zahl der Zustände oder Übergänge, die ein NFA für die Sprache $\overline{L_{p_2(m)}(h_m)}$ benötigt, wobei wir als Zahl der Wiederholungen $p_2(m) := m^{k+1}$ wählen.

Gemäß Lemma 10 gilt $G_m(f_m) = O(p_2(m))$ für Funktionen $f_m \in \mathcal{C}_2 \cap B_m$, und wir können die Schranke $t_1(m) = \Theta(p_2(m))$ entsprechend setzen. Die Schranke t_2 liegt weiter bei den in Lemma 9 beschriebenen exponentiell großen Werten.

Um zu zeigen, dass das Funktional G die Klasse $\mathcal{C}_2$ vom Zufallsensemble separiert, weisen wir noch nach, dass $G_m(k_m(f_n)) \leq t_1(n)$ für $m \leq n$ gilt. Wir wandeln einen Zweiweg-DFA A_n für $L_1(f_n) = \{x | x \in \{0,1\}^n \wedge f(x) = 1\}$ mit Zustandsmenge Q in einen Zweiweg-DFA A'_m für $L_1(k_m(f_n)) = \{y | y \in \{0,1\}^m \wedge f_n(0^{n-m}y) = 1\}$ mit derselben Zustandsmenge Q um. Der neue Startzustand q'_0 ist der Zustand, in dem A_n die Position $n - m + 1$ zum ersten Mal erreicht, wenn an den ersten $n-m$ Stellen der Eingabe nur Nullen stehen. Die Übergänge für A'_m sind dieselben wie bei A_n, außer wenn A'_m die linke Begrenzung des Eingabebereichs $\triangleright$ liest. Wir definieren $\delta(q, \triangleright) := (q', \rightarrow)$, wenn A_n in Position $n-m$ der Eingabe $0^{n-m}y$ startend zum ersten Mal die Position $n - m + 1$ im Zustand q' erreicht.

Somit erhalten wir im Wesentlichen die gleichen Nicht-Approximierbarkeitsergebnisse wie in Korollar 1 unter der schwächeren Annahme der Existenz eines starken Pseudozufallsensembles in nicht-uniformem LogSPACE.

Kapitel 5

Algorithmisches Lernen regulärer Sprachen

In diesem Kapitel erläutern wir Definitionen und Grundlagen des algorithmischen Lernens. In Kapitel 6 und Kapitel 7, stellen wir ab Abschnitt 6.2 Ergebnisse aus der Arbeit [14] vor. Die Arbeit entstand als Erweiterung der Diplomarbeit [17] von Ralf Herrmann.

In Kapitel 6 gehen wir auf die Aspekte des passiven Lernens ein, bei dem der Lernalgorithmus keinen Einfluss auf die angebotenen Beispiele nehmen kann. Mit unseren Methoden aus Kapitel 4 untersuchen wir die Nicht-Approximierbarkeit des Problems des kleinsten konsistenten DFAs. Ein DFA ist konsistent mit gegebenen positiven und negativen Beispielen, wenn er alle positiven Beispiele akzeptiert und alle negativen Beispiele verwirft. Die Frage nach konsistenten Hypothesen zu gegebenen klassifizierten Beispielen ist grundlegend für das algorithmische Lernen. So existieren effiziente Lernalgorithmen für das in Abschnitt 5.3 beschriebene PAC-Lernen, bei dem zu zufällig gewählten Beispielen eine möglichst gute Hypothese konstruiert wird, nur, wenn effiziente randomisierte Algorithmen für das Konsistenzproblem existieren.

Durch die Einschränkung auf unäre gegenüber allgemeinen regulären Sprachen erzielen wir positive Ergebnisse für das Problem des minimalen konsistenten DFAs und effizientes PAC-Lernen. Wir zeigen aber auch, dass effiziente Algorithmen für die Bestimmung minimaler unärer konsistenter NFAs ausgeschlossen sind, falls nicht NP $\subseteq$ DTIME($n^{O(\log n)}$) gilt.

In Abschnitt 6.4 bestimmen wir die VC-Dimension der unären regulären Sprachen, die von DFAs mit höchstens n Zuständen akzeptiert werden, als $n + \log n \pm \Theta(\log \log n)$ und erhalten in Abschnitt 6.4 das Ergebnis, dass unäre DFAs effizient PAC-lernbar sind.

In Kapitel 7 beschäftigen wir uns mit dem aktiven Lernen durch Äquivalenzfragen, bei dem der Lernalgorithmus einem Orakel eine Hypothese zur Begutachtung vorlegt und das Orakel mit einem Gegenbeispiel antwor-

tet, falls die Hypothese nicht mit dem zu lernenden Konzept äquivalent ist. Auch für das Lernen mit Äquivalenzfragen, spielen konsistente Hypothesen ein wichtige Rolle, da der Lernalgorithmus nur dann neue Informationen über das zu lernende Konzept erzwingen kann, wenn seine Hypothese mit den bisher erhaltenen Beispielen konsistent ist.

Für das Lernen unärer zyklischer DFAs mit primer Zykluslänge und höchstens n Zuständen durch Äquivalenzfragen erhalten wir als obere und untere Schranke $\Theta\left(\frac{n^2}{\ln n}\right)$ für die Anzahl der benötigten Gegenbeispiele. Erlauben wir als Hypothesen zyklische DFAs mit höchstens n^d Zuständen für $d \leq n$, so erhalten wir als obere Schranke $O\left(\frac{n^2}{d}\right)$ und als untere Schranke $\Omega\left(\frac{n^2 \cdot \ln d}{d \cdot (\ln n)^2}\right)$ für die Zahl der Gegenbeispiele.

5.1 Beispiele, Konzepte und Hypothesen

Ein grundlegendes Problem, mit dem sich das algorithmische Lernen beschäftigt, ist das Erlernen eines Konzeptes aus klassifizierten Beispielen. Das Universum, aus dem die Beispiele stammen, heißt Beispielraum und ein Konzept c ist eine Menge von Beispielen. Wir sprechen von einem positiven oder positiv klassifizierten Beispiel x, wenn $x \in c$ gilt und von einem negativen oder negativ klassifizierten Beispiel x, wenn $x \notin c$ gilt. Die folgende Definition führt die Begriffe der Konzeptklasse und der Hypothesenklasse ein.

Definition 7. *Eine Konzeptklasse $\mathcal{C}$ über einem Beispielraum X ist eine Menge von Konzepten $c \subseteq X$, es gilt also $\mathcal{C} \subseteq \mathcal{P}(X)$.*

Eine Hypothese $h \subseteq X$ ist ein Konzept. Eine Hypothesenklasse $\mathcal{H} \subseteq \mathcal{P}(X)$ ist eine Menge von Hypothesen.

In unserem Zusammenhang ist ein zu lernendes Konzept eine reguläre Sprache und der Beispielraum die Menge aller Wörter aus Σ^*. Eine Hypothese wird durch einen endlichen Automaten dargestellt.

Eine parametrisierte Konzeptklasse $(\mathcal{C}_n)_n$ ist eine Folge von Konzeptklassen. Wir parametrisieren die Konzeptklassen und Hypothesenklassen geeignet, indem wir zum Beispiel fordern, dass die Konzepte durch endliche Automaten mit höchstens n Zuständen ausgedrückt werden können.

5.2 Das Konsistenzproblem

Als Konsistenzproblem bezeichnen wir die Frage, ob ein konsistentes Konzept aus einer Konzeptklasse $\mathcal{C}$ für eine gegebene Menge klassifizierter Beispiele existiert.

Betrachten wir die parametrisierte Konzeptklasse der regulären Sprachen, die von DFAs mit n Zuständen akzeptiert werden, so stimmt das Konsistenzproblem mit dem Entscheidungsproblem überein, ob ein mit den Beispielen konsistenter DFA mit n Zuständen existiert. Das Optimierungsproblem der Größenbestimmung eines kleinsten konsistenten DFAs ist genau so schwer wie das Entscheidungsproblem. Das Konstruktionsproblem, bei dem ein minimaler konsistenter DFA ausgegeben werden soll, ist mindestens so schwer.

5.3 PAC-Lernen

Im Modell des *probably approximately correct* oder PAC-Lernens erhält ein Lernalgorithmus eine Menge von klassifizierten Beispielen, die zufällig gemäß einer Verteilung $\mathcal{D}$ aus dem Beispielraum gezogen werden. Der Algorithmus stellt eine Hypothese auf, die mit hoher Wahrscheinlichkeit nur einen kleinen Anteil von Beispielen – gemessen an der Verteilung $\mathcal{D}$ – falsch klassifiziert. Die Größe dieses Anteils bezeichnen wir als den Fehler der Hypothese.

Definition 8. *Für ein Konzept $c \in \mathcal{C}$, eine Hypothese $h_c \in \mathcal{H}$ und eine Verteilung $\mathcal{D}$ auf dem Beispielraum ist*

$$\text{fehler}_{\mathcal{D}}(c, h_c) := \sum_{x \in c \oplus h_c} \mathcal{D}(x),$$

wobei $c \oplus h_c = (c \cup h_c) \setminus (c \cap h_c)$ die symmetrische Differenz von c und h_c beschreibt und $\mathcal{D}(x)$ die Wahrscheinlichkeit ist, dass x unter der Verteilung $\mathcal{D}$ gezogen wird.

Wir können nun festlegen, wann ein Lernalgorithmus ein PAC-Algorithmus ist und führen den Fehlerparameter ϵ und den Vertrauensparameter δ ein.

Definition 9. *Ein Lernalgorithmus ist genau dann ein PAC-Algorithmus, der die parametrisierte Konzeptklasse $(\mathcal{C}_n)_n$ durch Hypothesen in $(\mathcal{H}_n)_n$ lernt, wenn für alle $\epsilon > 0$ und $\delta > 0$, für alle n, alle Konzepte $c \in \mathcal{C}_n$ und für alle Verteilungen $\mathcal{D}$*

$$\text{prob}_{\mathcal{D}}[\text{fehler}_{\mathcal{D}}(c, h_c) \leq \epsilon] \geq 1 - \delta$$

gilt, wobei $h_c \in \mathcal{H}_n$ die vom Algorithmus bestimmte Hypothese ist. Der Algorithmus erhält als Eingabe die Parameter ϵ und δ sowie den Parameter n, der die zu lernende Konzeptklasse $\mathcal{C}_n$ spezifiziert. Der Algorithmus bestimmt selbst, wie viele Beispiele anzufordern sind.

Ein PAC-Algorithmus, der polynomiell in n, $1/\epsilon$ und $1/\delta$ viele Beispiele anfordert und polynomielle Laufzeit bezüglich n, $1/\epsilon$ und $1/\delta$ hat, wird effizient genannt.

Existiert zu einer parametrisierten Konzeptklasse $(\mathcal{C}_n)_n$ ein effizienter PAC-Algorithmus, der $(\mathcal{C}_n)_n$ durch Hypothesen in $(\mathcal{H}_n)_n$ lernt, so sagen wir, dass $(\mathcal{C}_n)_n$ effizient durch $(\mathcal{H}_n)_n$ PAC-lernbar ist.

Existiert ein effizienter PAC-Algorithmus für eine parametrisierte Konzeptklasse, so lässt sich das Konsistenzproblem durch einen randomisierten Algorithmus lösen.

Fakt 5. *[33] Ist die parametrisierte Konzeptklasse $(\mathcal{C}_n)_n$ effizient durch $(\mathcal{H}_n)_n$ PAC-lernbar, dann gibt es einen effizienten randomisierten Algorithmus mit einseitigem Fehler, der mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ entscheidet, ob es zu gegebenem n und einer Menge klassifizierter Beispiele eine konsistente Hypothese aus $\mathcal{H}_n$ gibt.*

Beweis. Der effiziente PAC-Algorithmus, der $(\mathcal{C}_n)_n$ durch $(\mathcal{H}_n)_n$ lernt, sei A. Es sei POS die Menge der positiven Beispiele und NEG die Menge der negativen Beispiele. Wir setzen den Vertrauensparameter $\delta = \frac{1}{2}$ und den Fehlerparameter $\epsilon = \frac{1}{|\mathrm{POS}|+|\mathrm{NEG}|+1}$. Wir übergeben A die Parameter ϵ, δ und n und erzeugen die von A geforderte Anzahl an Beispielen, indem wir die einzelnen Beispiele gleichverteilt aus $\mathrm{POS} \cup \mathrm{NEG}$ ziehen. Jedes Beispiel erhält somit die Wahrscheinlichkeit $\frac{1}{|\mathrm{POS}|+|\mathrm{NEG}|} > \epsilon$. Der PAC-Algorithmus erzeugt mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ eine Hypothese $h \in \mathcal{H}_n$, für die der Fehler höchstens ϵ ist. Aufgrund unserer Wahl $\epsilon = \frac{1}{|\mathrm{POS}|+|\mathrm{NEG}|+1}$ kann der Fehler aber nur dann höchstens ϵ sein, wenn kein Beispiel durch h falsch klassifiziert wird, also wenn h konsistent ist. □

Wir leiten das folgende Korollar ab.

Korollar 2. *Hat die parametrisierte Konzeptklasse $(\mathcal{H}_n)_n$ ein NP-schweres Konsistenzproblem und ist $\mathcal{C}_n \subseteq \mathcal{H}_n$, dann gibt es keinen effizienten PAC-Algorithmus, der $(\mathcal{C}_n)_n$ durch Hypothesen in $(\mathcal{H}_n)_n$ lernt, falls $RP \neq NP$ gilt.*

5.4 VC-Dimension

In unserem Kontext betrachten wir zum Beispiel die Konzeptklasse aller regulären Sprachen, die von einem endlichen Automaten mit n Zuständen akzeptiert werden. Fixiert man das Alphabet, so haben wir es mit einer endlichen Konzeptklasse zu tun.

Für eine endliche Beispielmenge $S \subseteq X$ gibt es genau $2^{|S|}$ viele verschiedene Teilmengen $s \subseteq S$. Um also alle möglichen Klassifizierungen auf den Beispielen aus S zu erzeugen, benötigen wir eine Konzeptklasse mit mindestens $2^{|S|}$ verschiedenen Konzepten. Die Größe einer Beispielmenge, für die jede Klassifizierung durch $\mathcal{C}$ erzeugt werden kann, ist also durch $\log |\mathcal{C}|$ nach oben beschränkt.

Die Größe von Beispielmengen, für die jede Klassifizierung erzeugt werden kann, wird durch die nach Vapnik und Chervonenkis benannte VC-Dimension gemessen und lässt auch eine Anwendung auf Konzeptklassen mit unendlich vielen Konzepten zu.

Definition 10. *Wir sagen, dass eine endliche Menge $S \subseteq X$ durch die Konzeptklasse $\mathcal{C}$ zerschmettert wird, falls es für jede Teilmenge $s \subseteq S$ ein Konzept $c \in \mathcal{C}$ gibt, so dass $s = c \cap S$ gilt. Die VC-Dimension von $\mathcal{C}$, die wir mit $\mathrm{VC}(\mathcal{C})$ bezeichnen, ist die Kardinalität der größten Menge S, die durch $\mathcal{C}$ zerschmettert wird.*

Wie beschrieben, kann eine endliche Konzeptklasse $\mathcal{C}$ nur Mengen S mit $|S| \leq \log |\mathcal{C}|$ zerschmettern. Es gilt für endliche Konzeptklassen also stets $\mathrm{VC}(\mathcal{C}) \leq \log |\mathcal{C}|$.

Die VC-Dimension liefert außerdem eine obere Schranke für die Anzahl der Beispiele, die aus einem Algorithmus, der eine konsistente Hypothese für gegebene Beispiele ausgibt, einen PAC-Algorithmus macht [3, 27].

Fakt 6. *$\mathcal{C}$ sei eine Konzeptklasse und $\mathcal{H}$ sei eine Hypothesenklasse mit VC-Dimension d. A sei ein beliebiger Algorithmus, der als Eingabe eine Menge S von m klassifizierten Beispielen für ein Konzept in $\mathcal{C}$ erhält und als Ausgabe eine Hypothese $h \in \mathcal{H}$ erzeugt, die konsistent ist mit den Klassifikationen für S. Dann gibt es eine Konstante $c_0 > 0$, so dass A ein PAC-Algorithmus für $\mathcal{C}$ mit Hypothesen aus $\mathcal{H}$ ist, wenn A m zufällig aus dem Beispielraum gewählte Beispiele erhält, wobei*

$$m \geq c_0 \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d}{\epsilon} \log \frac{1}{\epsilon} \right)$$

gilt.

Wenn die VC-Dimension einer parametrisierten Konzeptklasse $(\mathcal{H}_n)_n$ polynomiell in n ist, und ein effizienter Algorithmus für das Konsistenzproblem mit Hypothesen aus $(\mathcal{H}_n)_n$ existiert, dann sind Konzeptklassen $\mathcal{C}_n \subseteq \mathcal{H}_n$ effizient durch $\mathcal{H}_n$ PAC-lernbar.

5.5 Lernen mit Äquivalenzfragen

Im Modell des Lernens mit Äquivalenzfragen stellt der Lernalgorithmus einzelne Hypothesen als Fragen an ein Orakel. Das Orakel überprüft, ob die angebotene Hypothese h mit dem zu lernenden Konzept c übereinstimmt und gibt im Falle einer Übereinstimmung die Korrektheit der Hypothese bekannt. Sind h und c verschieden, so gibt das Orakel ein klassifiziertes Gegenbeispiel aus $h \oplus c$ zurück.

Der Lernalgorithmus entwickelt eine neue Hypothese und stellt diese wieder als Frage. Der Vorgang wiederholt sich so lange, bis das Orakel die

Korrektheit einer Hypothese bestätigt. Wir gehen davon aus, dass ein Lernalgorithmus stets Hypothesen erstellt, die mit den bisherigen Beispielen konsistent sind. Tut er dies nicht, kann das Orakel mit einem schon genannten Gegenbeispiel antworten.

Eine Kenngröße für das Lernen mit Äquivalenzfragen ist die Anzahl der Fragen und somit die Anzahl der Gegenbeispiele, die ein Lernalgorithmus benötigt, um beliebige Konzepte aus einer Konzeptklasse lernen zu können.

Kapitel 6

Das Konsistenzproblem für endliche Automaten

Beim Problem des minimalen konsistenten DFAs sind als Eingabe die Mengen $\text{POS}, \text{NEG} \subseteq \Sigma^*$ mit $\text{POS} \cap \text{NEG} = \emptyset$ gegeben. Das Ziel ist es, die Größe eines minimalen DFAs D zu bestimmen, für den $\text{POS} \subseteq L(D)$ und $\text{NEG} \cap L(D) = \emptyset$ gilt.

In Abschnitt 6.1 wenden wir unsere Methoden aus Kapitel 4 an, um die Nicht-Approximierbarkeit der Größe eines minimalen konsistenten DFAs im Fall $|\Sigma| \geq 2$ zu bestimmen. In Abschnitt 6.2 und Abschnitt 6.3 betrachten wir das Konsistenzproblem für unäre Beispielmengen und geben in Abschnitt 6.4 die Konsequenzen für die PAC-Lernbarkeit an.

6.1 Minimale konsistente DFAs

Wir nehmen wie in Abschnitt 4.6 beschrieben an, dass die Klasse $\mathcal{C}_2$ der Funktionen, die auf Eingaben der Länge m von Zweiweg-DFAs mit höchstens m^k Zuständen berechnet werden, ein starkes Pseudozufallsensemble enthält. Wir wiederholen wiederum die Eingaben $p_2(m) = m^{k+1}$ mal und definieren $G_m^{(4)}(h_m)$ als die Anzahl der Zustände eines kleinsten DFAs, der jedes Beispiel aus der Menge $\text{POS} = \{x^{p_2(m)} \mid h_m(x) = 1\}$ akzeptiert und jedes Beispiel aus $\text{NEG} = \{x^{p_2(m)} \mid h_m(x) = 0\}$ verwirft.

Für jede Pseudozufallsfunktion $f_m \in \mathcal{C}_2 \cap B_m$ gilt $G_m^{(4)}(f_m) \leq t_1^{(4)}(m) := p_2(m)$, da der Zweiweg-DFA A_m für $L_1(f_m) = \{x \in \{0,1\}^m \mid f_m(x) = 1\}$ mit m^k Zuständen durch einen Einweg-DFA D_m mit $p_2(m) = m^{k+1}$ Zuständen simuliert werden kann, sofern die Eingabe $x \in \{0,1\}^m$ des Zweiweg-DFAs als wiederholte Eingabe $x^{p_2(m)}$ vorliegt [34]. Der DFA D_m akzeptiert also eine Sprache $L(D_m)$ mit $L(D_m) \cap \{x^{p_2(m)} \mid x \in \{0,1\}^m\} = L_{p_2(m)}(f_m)$ und ist somit konsistent mit POS und NEG. Also haben alle Pseudozufallsfunktionen $f_m \in \mathcal{C}_2 \cap B_m$ einen Funktionalwert von höchstens $p_2(m) = m^{k+1}$, während fast alle Funktionen einen exponentiell großen Funktionalwert $G^{(4)}(\cdot)$ haben.

Lemma 11. *$G_m^{(4)}(h_m) \leq t_2^{(4)}(m) := \frac{2^m}{6m}$ gilt für höchstens $\sqrt{2^{2^m}} = o(|B_m|)$ Funktionen in B_m.*

Beweis. $K(s)$ sei die Anzahl der verschiedenen Sprachen, die von DFAs mit höchstens s Zuständen über einem Alphabet mit zwei Buchstaben akzeptiert werden. Dann gilt $K(s) \leq s^{3s}$ [8] und somit folgt

$$K(t_2^{(4)}(m)) \leq \left(\frac{2^m}{6m}\right)^{3\frac{2^m}{6m}} \leq 2^{3m\frac{2^m}{6m}} = \sqrt{2^{2^m}}.$$

Die Behauptung folgt, da ein DFA nicht konsistent mit den Beispielmengen zweier verschiedener Funktionen aus B_m sein kann. □

Dass $G_m(k_m(f_n)) \leq t_1^{(4)}(n)$ für jede Funktion $f_n \in \mathcal{C}_2 \cap B_n$ und jedes $m < n$ gilt, folgt wiederum aus der Überlegung, dass wir einen Zweiweg-DFA A_n für $f_n \in \mathcal{C}_2 \cap B_n$ in einen Zweiweg-DFA A'_m für $k_m(f_n)$ umwandeln können, ohne die Anzahl der Zustände zu erhöhen. Somit separiert $G_m^{(4)}$ die Klasse $\mathcal{C}_2$ vom Zufallsensemble für die Schranken $t_1^{(4)}, t_2^{(4)}$ und wir können wieder Bemerkung 1 anwenden. Diesmal erzeugen wir aus der gegebenen Funktion h_m allerdings keinen DFA für $L_p(h_m)$, sondern die Beispielmengen $\text{POS} = \{x^{p_2(m)} | h_m(x) = 1\}$ und $\text{NEG} = \{x^{p_2(m)} | h_m(x) = 0\}$. Wir erhalten den folgenden Satz.

Satz 5. *Es existiere eine Konstante k und ein starkes Pseudozufallsensemble $\mathcal{C}_2$ mit Parameter ε, so dass jede Funktion $f \in \mathcal{C}_2 \cap B_m$ einen Zweiweg-DFA für $\{x \in \{0,1\}^m \mid f(x) = 1\}$ mit m^k Zuständen habe.*

App sei ein randomisierter Algorithmus, der für gegebene Beispielmengen POS und NEG die Anzahl der Zustände eines minimalen konsistenten DFAs approximiere. $N = \sum_{x \in POS \cup NEG} |x|$ bezeichne die Länge der Eingabe. Dann gibt es ein Polynom poly*, so dass für jedes $l \geq 1$ gilt: Falls App in Zeit $2^{O((\log N)^l)}$ läuft und einen Approximationsfaktor $\mu(N) < \frac{N}{\text{poly}((\log N)^{l/\varepsilon})}$ hat, dann muss App die Fehler $\epsilon_+ + \epsilon_- \geq \frac{2}{3}$ haben.*

Wir folgern somit, dass es Instanzen für das Problem des minimalen konsistenten DFAs gibt, mit Instanzgröße N und konsistenten DFAs optimaler Größe $\text{opt} = \text{poly}(\log N)$, so dass effiziente Approximationsalgorithmen nur konsistente DFAs der Größe mindestens $\frac{N}{\text{poly}(\log N)} \geq 2^{\text{opt}^{\frac{1}{l}}} \cdot N^{\beta}$ garantieren können, für $\beta < 1$ und genügend großes l. Dieses Ergebnis ist stärker als das entsprechende Resultat von Kearns und Valiant [26], die Nicht-Approximierbarkeit mit Approximationsfaktor mindestens $opt^{\alpha} \cdot N^{\beta}$ für jedes konstante α zeigen. Unser stärkeres Ergebnis führen wir auf die stärkere Annahme zurück, dass Pseudozufallsensembles existieren, während Kearns und Valiant nur von der Existenz von Einwegfunktionen ausgehen.

6.2 Unäre minimale konsistente DFAs

Wir betrachten nun das Problem des minimalen konsistenten DFAs für den unären Fall. Es seien wieder $\mathrm{POS}, \mathrm{NEG} \subseteq \{a\}^*$ die disjunkten Mengen der positiven und negativen Beispiele. Gesucht ist die Größe eines minimalen DFAs D, so dass $\mathrm{POS} \subseteq L(D)$ und $\mathrm{NEG} \cap L(D) = \emptyset$ gilt.

Im nicht-unären Fall ist das Konsistenzproblem NP-vollständig [10] und auch effiziente Approximationsalgorithmen mit vernünftigen Approximationsfaktoren sind unwahrscheinlich (vergleiche Satz 5 und [35, 26]). Im Gegensatz dazu zeigen wir, dass wir einen minimalen konsistenten DFA zu unären Beispielmengen effizient finden können, auch wenn die Eingabe kompakt dargestellt wird.

Wir nehmen an, dass ein Beispiel $w = a^n$ nicht in ausgeschriebener Form $aa \dots a$ vorliegt, sondern, dass die Länge binär kodiert wird und somit etwa $\log |w|$ Bits als Eingabelänge in Anspruch nimmt. Die gesamte Eingabelänge für das Konsistenzproblem ist also ungefähr $\ell(\mathrm{POS}, \mathrm{NEG}) = \sum_{w \in \mathrm{POS} \cup \mathrm{NEG}} \log |w|$ und ist mindestens so groß wie die Anzahl aller Beispiele $|\mathrm{POS}| + |\mathrm{NEG}|$.

Für jedes genügend lange Wort w, das den Zyklus des DFAs erreicht, entscheidet allein die Restklasse modulo der Zykluslänge darüber, ob w zu akzeptieren oder zu verwerfen ist. Die folgende Definition gruppiert Wörter einer Menge S gemäß ihrer Restklassenzugehörigkeit und wir bestimmen mit λ die Länge des längsten Worts in der Menge.

Definition 11. *Für eine endliche Menge $S \subseteq \{a\}^*$ sei $S_{r,z} = \{w : w \in S \wedge |w| \equiv r \pmod z\}$ die Restklassen-Teilmenge von S für den Rest $0 \leq r < z$ und es sei $\lambda(S) = max\{|w| : w \in S\}$, beziehungsweise $\lambda(\emptyset) = -1$.*

Für eine festgelegte Zykluslänge z ist es einfach, die Größe eines minimalen konsistenten DFAs mit Zykluslänge z zu bestimmen.

Lemma 12. *[17] Die Größe eines minimalen DFAs mit fester vorgegebener Zykluslänge $z \in \mathbb{N}$, der konsistent ist mit den Beispielmengen $POS, NEG \subseteq \{a\}^*$, ist*

$$s_z(POS, NEG) := z + 1 + \max\left\{\min\{\lambda(POS_{r,z}), \lambda(NEG_{r,z})\} : 0 \leq r < z\right\}.$$

Beweis. Wir zeigen zunächst, dass für jedes feste $0 \leq r < z$ ein minimaler DFA mit Zykluslänge z, der konsistent ist mit $\mathrm{POS}_{r,z}$ und $\mathrm{NEG}_{r,z}$ genau $1 + \min\{\lambda(\mathrm{POS}_{r,z}), \lambda(\mathrm{NEG}_{r,z})\}$ Zustände in seinem Pfad und z Zustände in seinem Zyklus benötigt.

Falls eine der Mengen $\mathrm{POS}_{r,z}$, $\mathrm{NEG}_{r,z}$ leer ist, besteht der minimale mit ($\mathrm{POS}_{r,z}$, $\mathrm{NEG}_{r,z}$) konsistente DFA mit Zykluslänge z nur aus dem Zyklus und hat einen leeren Pfad.

Wir gehen also davon aus, dass beide Mengen Beispiele enthalten und nehmen ohne Beschränkung der Allgemeinheit an, dass $0 \leq \lambda(\mathrm{POS}_{r,z}) <$

$\lambda(\mathrm{NEG}_{r,z})$ gilt und führen einen Widerspruchsbeweis unter der Annahme, dass der Pfad weniger als $1+\lambda(\mathrm{POS}_{r,z})$ Zustände hat. Die längsten Beispiele $w_{\mathrm{POS}} \in \mathrm{POS}_{r,z}$ und $w_{\mathrm{NEG}} \in \mathrm{NEG}_{r,z}$ erreichen somit beide den Zyklus und enden im selben Zustand. Also werden entweder beide – w_{POS} und w_{NEG} – akzeptiert oder beide werden verworfen und der DFA ist somit nicht konsistent mit den Beispielmengen.

Andererseits sind $1+\lambda(\mathrm{POS}_{r,z})$ auch ausreichend, da kein positives Beispiel den Zyklus erreicht und wegen der Disjunktheit von POS und NEG die akzeptierenden und verwerfenden Zustände auf dem Pfad konsistent gewählt werden können.

Ein DFA, der konsistent mit POS und NEG ist, muss für alle Restklassen konsistent sein und benötigt somit als Pfadlänge die längste, die für einen Rest r berechnet wird. Außerdem benötigen wir nicht mehr als das beschriebene Maximum als Pfadlänge, da die Restklassen-Teilmengen paarweise disjunkt sind und zu paarweise disjunkten Mengen von erreichbaren Zuständen in einem DFA mit $1+\max\{\min\{\lambda(\mathrm{POS}_{r,z}), \lambda(\mathrm{NEG}_{r,z})\} : 0 \leq r < z\}$ Zuständen im Pfad und z Zuständen im Zyklus führen. □

Wir folgern, dass die Größe eines minimalen konsistenten DFAs einfach beschrieben werden kann und zeigen, dass die Größe nur polynomiell in der Eingabelänge $\ell(\mathrm{POS}, \mathrm{NEG})$ ist.

Lemma 13. *Wir bezeichnen die Größe eines minimalen DFAs, der konsistent mit (POS, NEG) ist, als $s(POS, NEG)$. Dann gilt $s(POS, NEG) = \min\{s_z(POS, NEG) : z \in \mathbb{N}\}$ und $s(POS, NEG) < (\ell(POS, NEG))^3$ gilt für hinreichend lange Eingaben.*

Beweis. Dass die Behauptung $s(\mathrm{POS}, \mathrm{NEG}) = \min\{s_z(\mathrm{POS}, \mathrm{NEG}) : z \in \mathbb{N}\}$ gilt, ist offensichtlich, da wir die entsprechende Zykluslänge wählen können. Um die Behauptung $s(\mathrm{POS}, \mathrm{NEG}) < (\ell(\mathrm{POS}, \mathrm{NEG}))^3$ zu beweisen, setzen wir zunächst $n := s(\mathrm{POS}, \mathrm{NEG})$ sowie $\ell := \ell(\mathrm{POS}, \mathrm{NEG})$ und beobachten, dass für jede Zykluslänge $z < n$ ein konsistenter DFA einen nicht-leeren Pfad haben muss. Andernfalls bestünde der DFA nur aus den $z < n$ Zuständen des Zyklus, was im Widerspruch zur Definition von n steht. Für jedes $z < n$ gibt es also ein Paar positiver und negativer Beispiele derselben Länge modulo z, da der minimale konsistente DFA sicherstellen muss, dass nicht beide Beispiele durch ihre Länge den Zyklus erreichen. Es gilt also

$$\forall z < n \; \exists (a^x, a^y) \in \mathrm{POS} \times \mathrm{NEG} : x \equiv y \pmod{z}.$$

Für nicht-triviale Beispielmengen können wir voraussetzen, dass $|\mathrm{POS}| + |\mathrm{NEG}| \leq \ell$ gilt und somit die Anzahl der Paare $|\mathrm{POS} \times \mathrm{NEG}|$ durch $\ell^2/4$ beschränkt ist. Um einen Widerspruchsbeweis zu führen, nehmen wir nun an, dass $n \geq \ell^3$ gilt. Somit gibt es für jedes $z < \ell^3 \leq n$ ein Paar $(a^x, a^y) \in \mathrm{POS} \times \mathrm{NEG}$, so dass $x \equiv y \pmod{z}$ gilt. Wir beschränken uns für unsere Abschätzungen auf prime Zykluslängen $z < \ell^3$ und setzen $q = \pi(\ell^3 -$

1) $\approx \frac{\ell^3}{3 \ln \ell}$ als die Anzahl dieser Primzahlen. Da es höchstens $\ell^2/4$ Paare in POS $\times$ NEG gibt, folgt aus dem Schubfachprinzip, dass es ein Paar $(a^x, a^y) \in \text{POS} \times \text{NEG}$ geben muss, so dass $4q/\ell^2$ verschiedene Primzahlen $z_1, \ldots, z_{4q/\ell^2}$ mit $x \equiv y \pmod{z_i}$ für jedes $1 \leq i \leq 4q/\ell^2$ existieren. Aufgrund der Teilerfremdheit gilt nach Fakt 8 in Abschnitt A.3 $x \equiv y \pmod{\prod z_i}$ und aus $x \neq y$ folgt $|x - y| \geq \prod z_i > (4q/\ell^2)!$.

Daraus folgt, dass es mindestens ein Beispiel $a^x \in \text{POS} \cup \text{NEG}$ mit $\log x > \log((4q/\ell^2)!)$ geben muss. Wir wenden Stirlings Approximation auf $4q/\ell^2 \approx \frac{4\ell}{3 \ln \ell}$ an und erhalten $\log x > \ell$ für hinreichend große Werte von ℓ.

Dies widerspricht aber der Definition von $\ell = \sum_{w \in \text{POS} \cup \text{NEG}} \log |w|$. Eine Größe $n \geq \ell^3$ für einen minimalen konsistenten DFA ist somit unmöglich. □

In [17] wird der Algorithmus 1 vorgestellt, der das Ergebnis aus Lemma 12 implementiert und zu gegebenen unären Beispielmengen einen minimalen konsistenten DFA berechnet. Die Laufzeit von Algorithmus 1 wird in [17] in Abhängigkeit von der Anzahl der Beispiele und der Größe des konstruierten DFAs analysiert, es bleibt allerdings offen, wie die DFA-Größe abgeschätzt werden kann. Lemma 13 erlaubt es uns nun nachzuweisen, dass der Algorithmus polynomielle Laufzeit in der Eingabelänge hat.

Algorithmus 1 Unärer minimaler konsistenter DFA

Eingabe: Beispielmengen $\text{POS}, \text{NEG} \subseteq \{a\}^*$.
$z := 1$;
$n := \infty$;
while $n > z$ **do**
  if $s_z(\text{POS}, \text{NEG}) < n$ **then**
    $n := s_z(\text{POS}, \text{NEG})$;
    $z' := z$;
  end if
  $z := z + 1$;
end while
Ausgabe: DFA mit Zykluslänge z' und Pfadlänge $n - z'$, der konsistent mit den Beispielen aus POS und NEG ist.

Satz 6. *Es sei $\ell = \ell(POS, NEG)$ die Länge der Eingabe für die Mengen $POS, NEG \subseteq \{a\}^*$. Dann berechnet Algorithmus 1 einen minimalen konsistenten DFA mit n Zuständen für POS, NEG in der Zeit $O(n \cdot \ell) = O(\ell^4)$.*

Beweis. Der Algorithmus berechnet die korrekte Größe, da er für jede Zykluslänge $z = 1, 2, \ldots$ die korrekte Automatengröße berechnet, bis ein DFA mit $n \leq z$ Zuständen gefunden wird. Größere Zykluslängen müssen nicht betrachtet werden, da sie zu größeren DFAs als dem bereits gefundenen führen.

Offensichtlich benötigen wir n Iterationen der While-Schleife. Die Zeit für jeden Schleifendurchlauf wird durch die Zeit dominiert, die nötig ist, $s_z(\text{POS}, \text{NEG})$ zu bestimmen. Die Berechnung von $s_z(\text{POS}, \text{NEG})$ gelingt nach Lemma 12 in $O(\ell)$ Schritten, indem wir die Länge des längsten Beispiels für jede Restklasse speichern. □

6.3 Unäre minimale konsistente NFAs

Als nächstes betrachten wir das Problem des unären minimalen konsistenten NFAs: Gegeben sind die Beispielmengen POS, NEG sowie einen Schwellenwert k, gibt es einen NFA mit k Zuständen, der alle Beispiele aus POS akzeptiert und alle Beispiele aus NEG verwirft?

Das folgende Lemma zeigt, dass für eine unäre Sprache L mit minimaler Periode d die Angabe *aller* klassifizierten Beispiele aus $\{a\}^{<2d^2+d} := \bigcup_{i<2d^2+d}\{a^i\}$ garantiert, dass ein minimaler konsistenter NFA genau L akzeptiert.

Wir wissen, dass die Bestimmung der Größe eines minimalen NFAs, der zu einem gegebenen unären zyklischen DFA äquivalent ist, nur dann effizient möglich ist, falls NP $\subseteq$ DTIME$(n^{O(\log n)})$ gilt [24]. Zusammen mit dem Ergebnis aus Lemma 14, dass ein minimaler konsistenter NFA bei hinreichender Anzahl an Beispielen aus einer zyklischen Sprache L sogar genau L akzeptiert und somit äquivalent zu einem zyklischen DFA für L ist, folgt die Schwere des Problems der exakten Bestimmung eines unären minimalen konsistenten NFAs.

Lemma 14. *$L \subseteq \{a\}^*$ sei eine zyklische Sprache mit minimaler Periode d. Falls M ein minimaler NFA ist, der konsistent mit $POS = L \cap \{a\}^{<2d^2+d}$ und $NEG = \{a\}^{<2d^2+d} \setminus POS$ ist, dann gilt $L(M) = L$.*

Für den Beweis des Lemmas benötigen wir noch ein zahlentheoretisches Ergebnis.

Fakt 7. *Für $a, b \in \mathbb{N} \setminus \{0\}$ ist die Menge $\{ma + nb \mid m \in \mathbb{N}, n \in \mathbb{Z}\}$ gleich der Menge $\{m \cdot \text{ggT}(a, b) \mid m \in \mathbb{Z}\}$.*

Beweis. Wir zeigen, dass für jedes x gilt: Es gibt genau dann $m \in \mathbb{N}, n \in \mathbb{Z}$ mit $x = ma + nb$, wenn x ein Vielfaches von $\text{ggT}(a, b)$ ist.

„⇒“: Wir können den $\text{ggT}(a, b)$ ausklammern und erhalten

$$x = \text{ggT}(a, b) \cdot \left(m\frac{a}{\text{ggT}(a, b)} + n\frac{b}{\text{ggT}(a, b)} \right).$$

Somit ist x ein ganzzahliges Vielfaches von $\text{ggT}(a, b)$.

„⇐“: Nach dem erweiterten Euklidischen Algorithmus gibt es $m' \in \mathbb{N}, n' \in \mathbb{Z}$, so dass $\text{ggT}(a, b) = m'a + n'b$ gilt. Durch Multipliktion mit $\frac{x}{\text{ggT}(a,b)}$ erhalten die gewünschte Darstellung, falls x ein Vielfaches von $\text{ggT}(a, b)$ ist. □

Nun führen wir den Beweis zu Lemma 14 aus.

Beweis zu Lemma 14. A sei ein minimaler NFA mit $L(A) = L$ und $|Q_A|$ Zuständen. Offensichtlich ist A dann konsistent mit (POS, NEG) und es gilt $|Q_A| \leq d$. Um einen Widerspruchsbeweis zu führen, nehmen wir also an, dass es einen NFA M mit $|Q_M| < |Q_A|$ Zuständen gibt, der konsistent ist mit (POS, NEG), aber nicht die Sprache L akzeptiert.

Wir können M in einen äquivalenten NFA M' in Chrobak-Normalform mit höchstens $|Q_M|^2 + |Q_M| < d^2 + d$ Zuständen im Pfad umwandeln. Der letzte Zustand des Pfades hat Übergänge zu disjunkten deterministischen Zyklen mit Zykluslängen $c_1, \ldots, c_m$. Die Sprache $L(M')$ hat eine ultimative Periode $c := \text{kgV}(c_1, \ldots, c_m)$ und es gilt $\sum_{i=1}^{m} c_i \leq |Q_M| < d$. Für jedes $j \geq d^2 + d$ gilt also $a^j \in L(M') \Leftrightarrow a^{j+c} \in L(M')$.

Wir zeigen zuerst, dass $c < d$ nicht gelten kann, also die ultimative Periode von $L(M')$ mindestens so groß ist, wie die Periode von L. Da L zyklisch mit *minimaler* Periode d ist, gibt es für jede potenziell kleinere Periode c ein Zeugenpaar, dass c als Periode ausschließt:

$$\forall c < d \; \exists i : \; a^i \in L \Leftrightarrow a^{i+c} \notin L.$$

Aus dieser Aussage alleine können wir noch nicht schließen, dass M' nicht konsistent mit den gegebenen Beispielen ist, aber wir können die Wortlänge $i + c$ auf den durch die Beispielmengen abgedeckten Bereich eingrenzen. Die Aussage gilt insbesondere für ein i' mit $d^2 + d \leq i' < d^2 + 2d$: Es gelte $a^i \in L \Leftrightarrow a^{i+c} \notin L$, dann gilt für jedes $x \in \mathbb{Z}$ mit $i + xd \geq 0 : a^{i+xd} \in L \Leftrightarrow a^{i+xd+c} \notin L$. Offensichtlich können wir $i' = i + xd$ aus dem Intervall $[d^2 + d, d^2 + 2d - 1]$ wählen. Somit kann M' nicht konsistent mit (POS, NEG) sein, falls $c < d$ gilt.

Falls $c = d$ gilt, dann folgt $L = L(M') = L(M)$ im Widerspruch zu unserer Annahme, dass $L(M) \neq L$ gilt.

Wir nehmen schließlich an, dass $c > d$ gilt. Wir zeigen im nächsten Absatz, wie wir M' durch einen kleineren mit (POS, NEG) konsistenten NFA M'' ersetzen können. Dieser ist im Allgemeinen nicht äquivalent zu M' und wir zeigen, dass M'' eine zyklische Sprache mit Periode d akzeptiert. Deshalb gibt es einen konsistenten NFA in Chrobak-Normalform, dessen Anfangspfad nur den Startzustand enthält und der im Widerspruch zur Annahme weniger Zustände als M hat.

Für jede Zykluslänge c_i der Zyklen von M' gilt $c_i \leq |Q_M| < d$. Da $c = \text{kgV}(c_1, \ldots, c_m) > d$ gilt, muss es insbesondere in M' einen Zyklus C_k mit Länge $c_k < d$ geben, so dass c_k kein Teiler der Periode d ist. Wir können annehmen, dass es mindestens einen akzeptierenden Zustand in C_k gibt. Andernfalls entfernen wir C_k komplett. Da C_k einen akzeptierenden Zustand hat, gibt es auch eine Wortlänge i mit $d^2 + d \leq i < d^2 + d + c_k$, so dass für jedes $x \in \mathbb{N} : a^{i+xc_k} \in L(M')$ gilt. Da L d-periodisch ist und M'

mit (POS, NEG) konsistent ist, folgt, dass jedes Beispiel der Form a^{i+xc_k+yd}, dass mindestens die Länge $d^2 + d$ hat, in der Sprache liegen muss:

$$\forall x \in \mathbb{N}, y \in \mathbb{Z} : \; d^2 + d \leq i + xc_k + yd < 2d^2 + d \;\Rightarrow\; a^{i+xc_k+yd} \in L.$$

Aus Fakt 7 folgt für jedes $x \in \mathbb{N}$ mit $i + x \cdot \mathrm{ggT}(c_k, d) < 2d^2 + d$, dass $a^{i+x\,\gcd(c_k,d)} \in L$ gilt. Für jedes Wort a^i, das einen akzeptierenden Zustand in C_k erreicht, werden also die Wörter a^j mit $j \equiv i \pmod{\mathrm{ggT}(c_k, d)}$ und $d^2+d \leq j < 2d^2+d$ ebenfalls von M' akzeptiert. Wir können C_k durch einen Zyklus der Länge $\mathrm{ggT}(c_k, d)$ ersetzen und der neue NFA bleibt konsistent mit (POS, NEG). Durch dieses Verfahren können wir jeden Zyklus, dessen Länge nicht d teilt durch einen Zyklus mit einer Länge, die d teilt, ersetzen und der NFA M'' bleibt konsistent mit (POS, NEG). Da M'' die ultimative Periode d hat und sich auf dem Pfad d-periodisch verhält, können wir den Pfad ganz entfernen. Dies widerspricht der Annahme, dass M ein minimaler konsistenter NFA ist, da wir einen NFA mit $1 + \sum_{k=1}^{m} \mathrm{ggT}(c_k, d) < \sum_{k=1}^{m} c_k \leq |Q_M|$ Zuständen sowie disjunkten Zyklen und einem Startzustand konstruieren können, der mit (POS, NEG) konsistent ist und L akzeptiert. □

Wir erhalten nun das Resultat über die Schwere des Problems des unären minimalen konsistenten NFAs. Mit MinNFA-UCL (minimum NFA for a unary cyclic language [24]) bezeichnen wir folgendes Problem: Gegeben sei ein unärer zyklischer DFA M und eine Zahl $k \in \mathbb{N}$ in Binärnotation. Gibt es einen NFA mit höchstens k Zuständen, der $L(M)$ akzeptiert? Es ist bekannt, dass MinNFA-UCL nur dann in P liegt, wenn $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$ gilt.

Satz 7. *(a) Es gibt eine Polynomialzeitreduktion von MinNFA-UCL auf das Problem des unären minimalen konsistenten NFAs.*

(b) Das Problem des unären minimalen konsistenten NFAs liegt in NP, aber nicht in P, sofern nicht $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{O(\log n)})$ *gilt.*

Beweis. (a) Zu einem gegebenen unären zyklischen DFA M mit d Zuständen konstruieren wir $\mathrm{POS} = L(M) \cap \{a\}^{<2d^2+d}$ und $\mathrm{NEG} = \{a\}^{<2d^2+d} \setminus \mathrm{POS}$ in polynomieller Zeit. Gemäß Lemma 14 gibt es einen NFA mit k Zuständen, der genau dann konsistent mit (POS, NEG) ist, wenn es einen NFA mit k Zuständen gibt, der $L(M)$ akzeptiert.

(b) Das Problem liegt in NP, da wir einen unären NFA A mit höchsten k Zuständen raten und effizient überprüfen können, ob N konsistent mit (POS, NEG) ist. Jiang, McDowell und Ravikumar [24] zeigen eine Reduktion vom Problem der Knotenüberdeckung auf MinNFA-UCL, die in Zeit $O(n^{O(\log n)})$ läuft. Wir führen diese Reduktion mit Hilfe von Teil (a) auf das Konsistenzproblem für unäre NFAs fort und erhalten das gewünschte Ergebnis. □

Da eine exakte effiziente Lösung für das Problem des unären minimalen konsistenten NFAs nicht zu erwarten ist, müssen wir uns mit einer approximativen Lösung zufrieden geben. Aus Abschnitt 6.1 wissen wir, dass für das

allgemeine (nicht-unäre) Problem des minimalen konsistenten DFAs auch gute effiziente approximative Lösungen nicht zu erwarten sind. Die Situation sieht aber für das unäre Problem des minimalen konsistenten NFAs bedeutend besser aus. Mit Algorithmus 2 auf Seite 67 haben wir ein Verfahren, das einen konsistenten NFA zu gegebenen Beispielmengen effizient konstruiert. Dieser NFA hat höchstens $O(\text{opt}^2)$ Zustände, wenn ein minimaler konsistenter NFA opt Zustände hat.

Für $k = 1, 2, \ldots$ konstruiert Algorithmus 2 einen NFA in Chrobak-Normalform mit $k(k+1)$ Zuständen im Pfad und k Zyklen mit allen möglichen Zykluslängen $1, \ldots, k$. Der Algorithmus markiert jeden Zustand, der nicht von einem Beispiel aus NEG erreicht wird, als akzeptierend und prüft danach, ob jedes Beispiel aus POS von einem akzeptierenden Zustand überdeckt wird. Ist dies der Fall, gibt der Algorithmus den Automaten M mit $O(k^2)$ Zuständen aus, andernfalls startet die nächste Iteration mit $k := k+1$. M ist konsistent mit (POS, NEG), da der Konstruktionsprozess sicherstellt, dass kein Beispiel aus NEG akzeptiert wird und dass jedes Beispiel aus POS einen akzeptierenden Zustand erreicht. Wir zeigen, dass kein Automat mit weniger als k Zuständen konsistent mit (POS, NEG) ist.

Satz 8. *Es sei* opt *die Zahl der Zustände eines minimalen NFAs, der konsistent mit $POS, NEG \subseteq \{a\}^*$ ist und $\ell = \ell(POS, NEG)$ sei die Eingabelänge. Algorithmus 2 berechnet dann in Zeit $O(\text{opt}^3 \cdot (|POS| + |NEG|)) = O(\ell^{10})$ einen NFA M mit $O(\text{opt}^2)$ Zuständen, der mit (POS, NEG) konsistent ist.*

Beweis. Wir zeigen, dass der Algorithmus nach $k \leq \text{opt}$ Iterationen terminiert. Somit ist die Zahl der Zustände von M offensichtlich durch $O(\text{opt}^2)$ und die Laufzeit durch $O(\text{opt}^3 \cdot (|\text{POS}| + |\text{NEG}|))$ beschränkt, da wir für jeden Zustand q und jedes Beispiel w prüfen, ob w den Zustand q erreicht. Aus Lemma 13 folgt $O(\text{opt}^3 \cdot (|\text{POS}| + |\text{NEG}|)) = O(\ell^{10})$.

Sei M_{opt} ein minimaler mit (POS, NEG) konsistenter NFA mit opt Zuständen. Zu M_{opt} existiert ein NFA M' in Chrobak Normalform mit höchstens $\text{opt}(\text{opt}+1)$ Zuständen im Pfad und Zyklen mit Längen aus $\{1, \ldots, k\}$. In Iteration $k = \text{opt}$ konstruieren wir einen NFA M mit $\text{opt}(\text{opt}+1)$ Zuständen im Pfad und allen Zykluslängen aus $\{1, \ldots, k\}$. Das Akzeptanzverhalten des Automats M' kann somit durch M nachgebildet werden. Spätestens in Iteration $k = \text{opt}$ erhalten wir also einen Automaten, der konsistent mit (POS, NEG) ist. □

6.4 Die VC-Dimension und PAC-Algorithmen für unäre reguläre Sprachen

Die Vapnik Chervonenkis Dimension (VC-Dimension) führt zu einer oberen Schranke für die Anzahl der Beispiele, die von einem erfolgreichen PAC-Lernalgorithmus benötigt werden.

Für die Klasse der unären Sprachen, die von DFAs mit höchstens n Zuständen akzeptiert werden, können wir die VC-Dimension fast exakt bestimmen. Der Nachweis der unteren Schranke verwendet DFAs mit primer Zykluslänge, so dass die Anzahl $\pi(n)$ der Primzahlen der Größe höchstens n in die Schranke eingeht. Der folgende Satz wurde bereits in [17] vorgestellt.

Satz 9. *[17] Es sei $\mathcal{L}_n \subset \mathcal{P}(\{a\}^*)$ die Konzeptklasse der unären regulären Sprachen, die von DFAs mit höchstens n Zuständen akzeptiert werden. Dann gilt für $n \in \mathbb{N}$*

$$n - 1 + \lfloor \log(\pi(n) + 1) \rfloor \leq \mathrm{VC}(\mathcal{L}_n) \leq n + \log(n).$$

Beweis. Ohne Beschränkung der Allgemeinheit werde jede Sprache $L \in \mathcal{L}_n$ von einem DFA mit genau n Zuständen $q_0, \ldots q_{n-1}$ akzeptiert. Bei DFAs mit weniger als n Zuständen verlängern wir einfach den Pfad, indem wir den Zyklus entsprechend weit „abrollen“. Es gibt höchstens $n \cdot 2^n$ verschiedene Sprachen in $\mathcal{L}_n$, da es n verschiedene Pfadlängen gibt und 2^n Möglichkeiten, die akzeptierenden Zustände zu setzen. Die obere Schranke $n + \log n$ folgt aus dem in Abschnitt 5.4 beschriebenen Fakt, dass $\mathrm{VC}(\mathcal{C}) \leq \log(|\mathcal{C}|)$ für endliche Konzeptklassen $\mathcal{C}$ gilt.

Für $n \geq 5$ geben wir eine Menge $S = S_1 \cup S_2$ mit $n - 1 + \lfloor \log(\pi(n) + 1) \rfloor$ Elementen an, die durch $\mathcal{L}_n$ zerschmettert wird. Wir zeigen also, dass es für jede Teilmenge $s \subseteq S$ ein Konzept $L \in \mathcal{L}_n$ gibt, welches durch einen DFA mit n Zuständen dargestellt wird, so dass $s = L \cap S$ gilt. Wir setzen einfach $S_1 = \{a^0, \ldots, a^{n-3}\}$. Für S_2 wählen wir $u = \lfloor 1 + \log(\pi(n) + 1) \rfloor$ Wörter $w_1, \ldots, w_u$, die wir mit Hilfe des chinesischen Restsatzes, Fakt 8 in Abschnitt A.3, konstruieren.

Die Teilmengen $s \subseteq S_2$ stellen wir durch ihre Inzidenzvektoren aus $\{0,1\}^u$ dar. Wir wählen $r := 2^{u-1} - 1$ Inzidenzvektoren $s_1, \ldots, s_r$ so aus, dass sich alle nichtleeren Teilmengen von S_2 darstellen lassen, die nicht w_1 enthalten. Die Menge der Vektoren ist somit $(\{0\} \times \{0,1\}^{u-1}) \setminus \{0\}^u$ und hat die Mächtigkeit $r = 2^{u-1} - 1 = 2^{\lfloor \log(\pi(n)+1) \rfloor} - 1 \leq \pi(n)$. Wir nutzen die ersten r Primzahlen $p_1, \ldots, p_r$ als mögliche Zykluslängen der unären DFAs mit n Zuständen und definieren eine Kodierung $\mu : \mathbb{N} \to \mathbb{N}^r$ durch

$$\mu(k) = (k - n + 2 \bmod p_1, \ldots, k - n + 2 \bmod p_r).$$

Für einen Wert $k \geq n$, dessen Kodierung nur Komponenten aus $\{0,1\}$ hat, also $\mu(k) \in \{0,1\}^r$, erreicht das Wort a^k wie in Abbildung 6.1 dargestellt entweder den letzten oder vorletzten Zustand q_{n-1} oder q_{n-2} eines unären DFAs mit n Zuständen und Zykluslänge p_i für ein beliebiges $1 \leq i \leq r$.

Die Inzidenzvektoren $s_1, \ldots, s_r \in (\{0\} \times \{0,1\}^{u-1}) \setminus \{0\}^u$ seien als Spalten einer Matrix $(m_{j,i})_{1 \leq j \leq u, 1 \leq i \leq r}$ angeordnet. Die u paarweise verschiedenen Zeilen der Matrix $(m_{j,i})$ fassen wir jetzt als Kodierung der Wörter w_j für

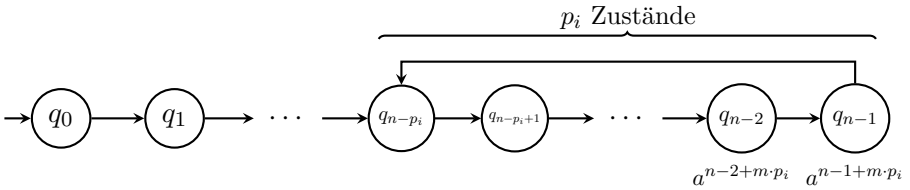


Abbildung 6.1: Unärer DFA mit n Zuständen und Zykluslänge p_i

$1 \leq j \leq u$ auf, die die Menge $S_2 = \{w_1, \ldots, w_u\}$ bilden. Für $1 \leq j \leq u$ setzen wir also $w_j = a^k$ mit

$$n \leq k < n + \prod_i p_i \quad \wedge \quad \mu(k) = (m_{j,1}, m_{j,2}, \ldots, m_{j,r}).$$

Da jeder der Vektoren $s_1, \ldots, s_r$ einen 0-Eintrag an erster Stelle hat, ist das Wort $w_1 = a^k$ mit $\mu(k) = 0^r$.

w_j	$\|w_j\| - 3 \bmod 2$	$\|w_j\| - 3 \bmod 3$	$\|w_j\| - 3 \bmod 5$
a^{33}	0	0	0
a^{19}	0	1	1
a^{24}	1	0	1

Tabelle 6.1: Die Matrix zur Bestimmung der Menge S_2 für $n = 5$

Um eine beliebige Teilmenge $\emptyset \neq s \subset S_2$ zu zerschmettern, betrachten wir den Inzidenzvektor $v(s)$ von s. Falls $w_1 \notin s$ gilt, dann kommt $v(s)$ als Spaltenvektor s_{i^*} für ein $1 \leq i^* \leq r$ vor. Andernfalls gibt es ein i^*, so dass für die negierte Version $v(S_2 \setminus s) = s_{i^*}$ gilt. Wir wählen M als unären DFA mit n Zuständen und Zykluslänge p_{i^*}. Für $w_1 \notin s$ wählen wir q_{n-1} akzeptierend und q_{n-2} verwerfend, beziehungsweise für $w_1 \in s$ wählen wir q_{n-1} verwerfend und q_{n-2} akzeptierend. Somit gilt $s = L(M) \cap S_2$ als direkte Konsequenz der Konstruktion von S_2. Um $\emptyset$ zu akzeptieren, setzen wir alle Zustände verwerfend und für $s = S_2$ setzen wir q_{n-1} und q_{n-2} akzeptierend.

Für beliebige Teilmengen $s \subseteq S = S_1 \cup S_2$ stellen wir für den DFA M mit n Zuständen zunächst durch die Wahl der Zykluslänge und des Akzeptanzverhaltens von q_{n-1} und q_{n-2} wie eben beschrieben das richtige Verhalten für Wörter aus S_2 sicher. Außerdem setzen wir das Akzeptanzverhalten der Zustände $q_0, \ldots, q_{n-3}$ so, dass M auch die Wörter aus $S_1 = \{a^0, \ldots, a^{n-3}\}$ richtig klassifiziert. Somit gilt $s = L(M) \cap S$.

Die Größe von $S = S_1 \cup S_2$ ist $n-2+u = n-1+\lfloor \log(\pi(n)+1) \rfloor$ und ist eine untere Schranke für $\mathrm{VC}(\mathcal{L}_n)$. Für $n < 5$ wird $S = \{0, 1, \ldots, n-1\}$ bereits durch das Setzen des Akzeptanzverhaltens eines DFAs mit n Zuständen und beliebiger Zykluslänge zerschmettert. □

Da $\pi(n) \approx n/\ln n$ gilt, unterscheiden sich die untere und die obere Schranke für die VC-Dimension nur um einen additiven Term der Größenordnung $O(\log \ln n)$.

Abbildung 6.2 zeigt eine Familie von DFAs mit 5 Zuständen und primen Zykluslängen $2, 3, 5$, die $S = \{a^0, a^1, a^2, a^{19}, a^{24}, a^{33}\}$ zerschmettert. Die Abbildung zeigt auch, welche Zustände die Wörter erreichen. Die zugehörigen Kodierungen sind $\mu(33) = (0, 0, 0),\ \mu(19) = (0, 1, 1),\ \mu(24) = (1, 0, 1)$.

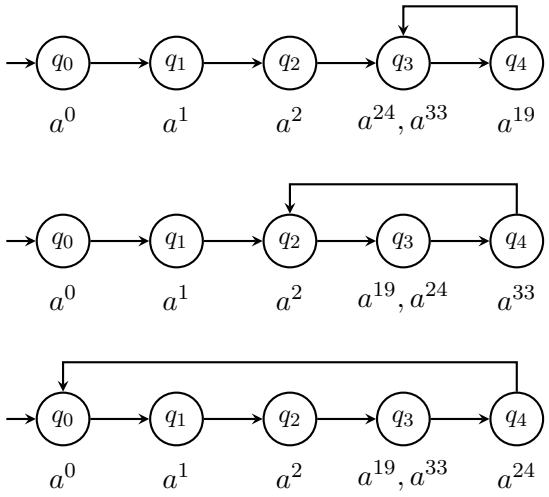


Abbildung 6.2: Eine Familie von DFAs mit 5 Zuständen, die $S = \{a^0, a^1, a^2, a^{19}, a^{24}, a^{33}\}$ zerschmettert

Die VC-Dimension liefert gemäß Fakt 6 eine obere Schranke für die Anzahl der Beispiele, die benötigt werden, um einen Algorithmus für das Konsistenzproblem als PAC-Algorithmus zu verwenden. Wir erhalten das folgende Korollar

Korollar 3. *$\mathcal{L}_n$ sei die Klasse aller unären regulären Sprachen, die von DFAs mit höchstens n Zuständen akzeptiert werden. Dann ist $(\mathcal{L}_n)_n$ mit $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n+\log n}{\epsilon}\log\frac{1}{\epsilon})$ Beispielen effizient PAC-lernbar.*

Beweis. Algorithmus 1 berechnet effizient eine minimale konsistente Hypothese für gegebenen Beispiele. Die VC-Dimension von $\mathcal{L}_n$ ist höchstens $n + \log n$. Somit ist Algorithmus 1 gemäß Fakt 6 ein effizienter PAC-Algorithmus, wenn er $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n+\log n}{\epsilon}\log\frac{1}{\epsilon})$ Beispiele anfordert. □

Betrachten wir die parametrisierte Konzeptklasse $\mathcal{N}_n$ der unären regulären Sprachen, die von NFAs mit höchstens n Zuständen akzeptiert werden, dann ergibt sich aus der Schwere des Problems des unären minimalen konsistenten NFAs, dass vermutlich keine effizienten PAC-Algorithmen existieren, die $\mathcal{N}_n$ lernen, wenn nur Hypothesen aus $\mathcal{N}_n$ verwendet werden.

Allerdings ist effizientes PAC-Lernen möglich, wenn als Hypothesen NFAs mit $O(n^2)$ Zuständen zugelassen sind. Mit QRP bezeichnen wir die Klasse aller Sprachen, die von randomisierten Turingmaschinen mit einseitigem Fehler in Quasipolynomialzeit $n^{O(\log n)}$ akzeptiert werden können.

Korollar 4. *$\mathcal{N}_n$ sei die Klasse der unären regulären Sprachen, die von NFAs mit höchstens n Zuständen akzeptiert werden. Dann ist $(\mathcal{N}_n)_n$ nicht effizient PAC-lernbar, wenn Hypothesen aus $\mathcal{N}_n$ für Konzepte aus $\mathcal{N}_n$ benutzt werden und NP $\not\subseteq$ QRP gilt. Andererseits ist $(\mathcal{N}_n)_n$ effizient PAC-lernbar, wenn Hypothesen aus $\mathcal{N}_{O(n^2)}$ für Konzepte aus $\mathcal{N}_n$ verwandt werden und $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n^2\log n}{\epsilon}\log\frac{1}{\epsilon})$ Beispiele angefordert werden.*

Beweis. Aus Satz 7 folgt, dass das Konsistenzproblem nicht zu QRP gehört, falls nicht NP $\subseteq$ QRP gilt. Wenn das Problem der minimalen konsistenten Hypothese nicht in QRP liegt, kann gemäß Fakt 5 auch $(\mathcal{N}_n)_n$ nicht in Quasipolynomialzeit PAC-lernbar sein, wenn Hypothesen aus $\mathcal{N}_n$ für Konzepte aus $\mathcal{N}_n$ verwendet werden.

Das positive Ergebnis erhalten wir wie in Korollar 3 durch die effiziente Bestimmung eines approximativ minimalen konsistenten NFAs mittels Algorithmus 2 und der Abschätzung $\mathrm{VC}(\mathcal{N}_{O(n^2)}) \leq \log|\mathcal{N}_{O(n^2)}| = O(n^2\log n)$ für hinreichend großes n: Die Anzahl der Sprachen, die durch unäre NFAs mit k Zuständen akzeptiert werden, ist höchstens k^k für genügend großes k [8]. □

Algorithmus 2 Approximation eines minimalen konsistenten NFAs

1: Eingabe: Beispielmengen $\text{POS}, \text{NEG} \subseteq \{a\}^*$.
2: $k := 0$;
3: **repeat**
4: $k := k + 1$;
5: Konsistent[M]:= **true**;
6: /*Konstruiere den NFA M in Chrobak-Normalform wie folgt.*/
7: Es sei p ein Pfad mit $k(k+1)$ der konsistent mit (POS, NEG) ist für Beispiele der Länge $< k(k+1)$.
8: **for** $i := 1$ to k **do**
9: C_i sei ein Zyklus mit i Zuständen, der durch einen Übergang vom letzten Zustand des Pfades p erreicht wird.
10: Initialisiere jeden Zustand von C_i als akzeptierend.
11: **end for**
12: **for all** $a^j \in \text{NEG} \wedge j \geq k(k+1)$ **do**
13: **for** $i := 1$ to k **do**
14: Definiere den Zustand in C_i, der durch a^j erreicht wird als verwerfend.
15: **end for**
16: **end for**
17: /*Zu diesem Zeitpunkt akzeptiert der NFA so viele Wörter der Länge $\geq k(k+1)$ wie möglich, ohne der Klassifizierung durch NEG zu widersprechen. Wir prüfen nun, ob auch jedes Wort in POS von irgendeinem akzeptierenden Zustand in einem der Zyklen abgedeckt wird.*/
18: **for all** $a^j \in P \wedge j \geq k(k+1)$ **do**
19: Abgedeckt[j]:=**false**;
20: **for** $i := 1$ to k **do**
21: **if** Der Zustand in C_i, der von a^j erreicht wird, ist akzeptierend **then**
22: Abgedeckt[j]:=**true**;
23: **end if**
24: **end for**
25: **if** $\neg$ Abgedeckt[j] **then**
26: /*Ist ein einziges Beispiel aus POS nicht abgedeckt, so ist M nicht konsistent.*/
27: Konsistent[M]:=**false**;
28: **end if**
29: **end for**
30: **until** Konsistent[M]
31: Gib NFA M mit $k(k+1) + k(k+1)/2$ Zuständen aus, der konsistent mit (POS, NEG) ist.

Kapitel 7

Lernen unärer zyklischer DFAs mit Äquivalenzfragen

Wir beschäftigen uns nun mit dem Modell des Lernens mit Hilfe von Äquivalenzfragen. In diesem Modell stellt der Lernende eine Hypothese aus der Hypothesenklasse auf und bittet um Bewertung. Ein Orakel bestätigt entweder die Korrektheit der Hypothese oder gibt ein Gegenbeispiel aus der symmetrischen Differenz.

Für den Fall der unären zyklischen DFAs zeigen wir in Satz 10 aus [17], dass $\Theta(\frac{n^2}{\ln n})$ Gegenbeispiele nötig und hinreichend sind, wenn wir Konzeptklasse und Hypothesenklasse auf Zyklen mit $p \leq n$ Zuständen für eine Primzahl p einschränken. Satz 11 zeigt, dass die Zahl der benötigten Äquivalenzfragen abnimmt, wenn wir größere DFAs mit nicht-primen Zykluslängen als Hypothesen zulassen. Welchen Einfluss eine solche Vergrößerung der Hypothesenklasse hat, ließ Herrmann in [17] noch offen.

Satz 10. *[17] Es sei $\mathcal{C}_n = \mathcal{H}_n$ die Menge der unären zyklischen DFAs mit $p \leq n$ Zuständen für eine Primzahl p.*

(a) Es gibt einen Lernalgorithmus, der DFAs aus $\mathcal{H}_n$ als Hypothesen erzeugt und höchstens $O(\frac{n^2}{\ln n})$ Gegenbeispiele benötigt, um ein Konzept $L(C)$ für $C \in \mathcal{C}_n$ zu lernen.

(b) Für jeden Lernalgorithmus, der DFAs aus $\mathcal{H}_n$ als Hypothesen erzeugt, gibt es ein Orakel mit dazugehörigem Konzept $L(C)$ für $C \in \mathcal{C}_n$, so dass der Algorithmus mindestens $\Omega(\frac{n^2}{\ln n})$ Gegenbeispiele benötigt, um $L(C)$ zu lernen.

Beweis. (a) Unser Algorithmus geht wie folgt vor. Für jede prime Zykluslänge $p \leq n$ erzeugt der Algorithmus zyklische DFAs mit p Zuständen und benötigt höchstens p Gegenbeispiele, um zu erkennen, dass kein zyklischer DFA mit p Zuständen das Konzept $L(C)$ akzeptieren kann: Der Algorithmus erzeugt zunächst einen DFA für die leere Sprache, indem kein Zustand als akzeptierend markiert wird. Wenn das Orakel ein positives Gegenbeispiel $a^x \in L(C)$ zurückgibt, passt der Algorithmus die Hypothese so an, dass alle

a^y mit $y \equiv x \pmod{p}$ akzeptiert werden. Er fügt also den entsprechenden Zustand zu der Menge der akzeptierenden Zustände hinzu und legt die neue Hypothese wiederum dem Orakel vor. Solange das Orakel nur mit positiven Gegenbeispielen $a^{x'} \in L(C)$ antwortet, fügt der Algorithmus akzeptierende Zustände hinzu. Erhält der Algorithmus sein erstes negatives Gegenbeispiel $a^{\overline{x}} \notin L(C)$ schließt er daraus, dass p nicht die richtige Zykluslänge ist, da der akzeptierende Zustand, in dem $a^{\overline{x}}$ landet, durch ein positives Gegenbeispiel $a^x \in L(C)$ mit $x \equiv \overline{x} \pmod{p}$ erzwungen wurde. Der Algorithmus wechselt zur nächsten Primzahl und wiederholt sein Vorgehen bis das Orakel die Korrektheit der Hypothese bescheinigt.

Die Gesamtzahl der Gegenbeispiele ist somit durch $\sum_{p \leq n, p \text{ prim}} p$ beschränkt. Für die i-te Primzahl p_i gilt gemäß Abschnitt A.3.1 $i \ln i \leq p_i \leq 2i \ln i$ für $i \geq 3$ und die Anzahl der Primzahlen, über die wir summieren ist $\pi(n) \approx n / \ln n$. Wir erhalten als obere Schranke für die Zahl der Gegenbeispiele

$$\sum_{i=1}^{\pi(n)} p_i \leq 2 \sum_{i=1}^{\pi(n)} i \ln i \approx 2 \int_1^{\frac{n}{\ln n}} x \ln x \, \mathrm{d}x = O\left(\frac{n^2}{\ln n}\right).$$

(b) Wir beschreiben eine Orakelstrategie, um einen beliebigen Lernalgorithmus zu zwingen, $\Omega(\frac{n^2}{\ln n})$ Äquivalenzfragen zu stellen. Das Orakel speichert für jede Primzahl $p \leq n$ eine Liste der Restklassen $\pmod{p}$, die bestätigt wurden. Wir nennen eine Restklasse mit Repräsentanten y bestätigt für eine Primzahl p, falls das Orakel ein Gegenbeispiel a^x mit $x \equiv y \pmod{p}$ vorgelegt hat. Beachte, dass ein Gegenbeispiel für jede Primzahl eine Restklasse bestätigt. Wir nennen eine Primzahl vollständig bestätigt, wenn jede ihrer Restklassen bestätigt ist. Wir gehen davon aus, dass das Orakel von vornherein die Beispiele $a^0 \notin L(C)$ und $a \in L(C)$ angibt und somit die Restklassen mit Repräsentanten 0 und 1 für alle Primzahlen bestätigt.

Das Orakel antwortet generell mit alten Gegenbeispielen, falls es eine Hypothese beurteilen soll, die zu den bisherigen Gegenbeispielen inkonsistent ist. Wir können also annehmen, dass das Orakel nur konsistente Hypothesen zur Begutachtung erhält.

Für eine Hypothese mit primer Zykluslänge p^* kann das Orakel mit Hilfe des chinesischen Restsatzes ein Gegenbeispiel wie folgt konstruieren, dass nur eine neue Restklasse $\pmod{p^*}$ bestätigt, aber für alle anderen Primzahlen nur die bereits bestätigten Reste 0 und 1 liefert. Für einen zyklischen DFA D mit Zykluslänge p^*, die noch nicht vollständig bestätigt wurde, wählt das Orakel einen noch nicht bestätigten Rest $r \pmod{p^*}$. Falls $a^r \in L(D)$ gilt, dann antwortet das Orakel mit $a^{r'} \notin L(C)$, wobei $r' \equiv r \pmod{p^*}$ und $r' \equiv 0 \pmod{p}$ für jede Primzahl $p \neq p^*$, $p \leq n$ gilt. Falls $a^r \notin L(D)$ gilt, dann antwortet das Orakel entsprechend mit $a^{r'} \in L(C)$, wobei $r' \equiv r \pmod{p^*}$ und $r' \equiv 1 \pmod{p}$ für jede Primzahl $p \neq p^*$ gilt.

Ist p^* bereits vollständig bestätigt und D konsistent mit den bisherigen Gegenbeispielen, dann gibt das Orakel das Gegenbeispiel $a^r \in L(C)$ mit

$r \equiv 0 \pmod{p^*}$ und $r \equiv 1 \pmod{p}$ für jede Primzahl $p \neq p^*$. Somit kann kein DFA mit Zykluslänge p^* mit den gegebenen Gegenbeispielen konsistent sein.

Das Orakel bestätigt schließlich die Korrektheit der Hypothese $L(D)$, wenn jede Primzahl vollständig bestätigt ist und der zu beurteilende DFA D konsistent mit den bisherigen Beispielen ist. (Ein konsistenter DFA D kann so gewählt werden, dass er als Zykluslänge p^* die letzte vollständig bestätigte Primzahl hat und die akzeptierenden Zustände gemäß den Gegenbeispielen der Bestätigungen für die Primzahl p^* gesetzt sind.) Der Lernalgorithmus benötigt somit stets

$$\sum_{i=1}^{\pi(n)} (p_i - 1) \geq \int_1^{\frac{n}{\ln n}} x \ln x \, \mathrm{d}x = \Omega\left(\frac{n^2}{\ln n}\right)$$

Gegenbeispiele. □

Die Situation ändert sich, wenn wir Hypothesen mit nicht-primen Zykluslängen, die größer als n sind, zulassen. Insbesondere benötigen wir nur $O(n)$ Gegenbeispiele, wenn wir zyklische DFAs mit $n!$ Zuständen zulassen. Für die obere Schranke der Zahl der Gegenbeispiele erlauben wir als Konzeptklasse Sprachen, die von zyklischen DFAs mit beliebiger Zykluslänge $\leq n$ akzeptiert werden, während wir uns für die untere Schranke auf Orakel beschränken, die nur prime Zykluslängen für die Konzepte verwenden.

Satz 11. *Es sei $\mathcal{C}_n$ die Menge der unären zyklischen DFAs mit höchstens n Zuständen und $\mathcal{C}_n^* \subset \mathcal{C}_n$ enthalte nur DFAs mit primer Zykluslänge. $\mathcal{H}_{n,d}$ sei die Menge der unären zyklischen DFAs mit höchstens n^d Zuständen für $d \leq n$.*

(a) Es gibt einen Lernalgorithmus, der DFAs aus $\mathcal{H}_{n,d}$ als Hypothesen erzeugt und höchstens $O(\frac{n^2}{d})$ Gegenbeispiele benötigt, um ein Konzept $L(C)$ für $C \in \mathcal{C}_n$ zu lernen.

(b) Für jeden Lernalgorithmus, der DFAs aus $\mathcal{H}_{n,d}$ als Hypothesen erzeugt, gibt es ein Orakel mit dazugehörigem Konzept $L(C)$ für $C \in \mathcal{C}_n^*$, so dass der Algorithmus mindestens $\Omega(\frac{n^2 \cdot \ln d}{d \cdot (\ln n)^2})$ Gegenbeispiele benötigt, um $L(C)$ zu lernen.

Beweis. (a) Unser Algorithmus teilt die Faktoren der Fakultät $n!$ in $\lceil \frac{n}{d} \rceil$ Blöcke, die bis auf den letzten Block jeweils d Faktoren haben. Konkret setzen wir $n_i = \frac{(d \cdot i)!}{(d \cdot (i-1))!}$ für $i < \lceil \frac{n}{d} \rceil$ und $n_{\lceil \frac{n}{d} \rceil} = \frac{n!}{(d \cdot (\lceil \frac{n}{d} \rceil - 1))!}$. Für jeden Block gilt offensichtlich $n_i \leq n^d$.[1] Der Einfachheit halber nennen wir $d \cdot (i-1) + 1, \ldots, d \cdot i$ die Faktoren von n_i, beziehungsweise $d \cdot (\lceil \frac{n}{d} - 1 \rceil) + 1, \ldots, n$ die Faktoren von $n_{\lceil \frac{n}{d} \rceil}$.

[1] Wir können die Faktoren besser auf die Blöcke verteilen, um etwas weniger Blöcke mit ausgeglichener Größe zu erhalten, aber das ändert nichts am asymptotischen Verhalten.

Der Algorithmus betrachtet jeden Block n_i für $i = 1, \ldots, \lceil \frac{n}{d} \rceil$ einzeln und konstruiert so lange Hypothesen der Zykluslänge n_i, bis er jeden Faktor von n_i als Zykluslängen des Konzepts ausschließen kann. Wir zeigen, wie der Algorithmus alle Faktoren von n_i als potenzielle Zykluslänge des Konzepts C mit $k_i + d$ Gegenbeispielen ausschließen kann, wobei k_i der größte Faktor von n_i ist.

Für jeden Faktor z von n_i speichert der Algorithmus für jede Restklasse modulo z, ob der entsprechende Zustand in einem zyklischen DFA mit z Zuständen akzeptierend sein soll oder unbestimmt ist. Wenn der Lernalgorithmus ein positives Gegenbeispiel a^x erhält, dann markiert sich der Algorithmus die Restklasse $x \pmod z$ als akzeptierend für alle Faktoren z von n_i. Erhält der Algorithmus ein negatives Gegenbeispiel $a^x \notin L(C)$, dann kann er bereits die Faktoren z als Zykluslängen ausschließen, für die $x \pmod z$ als akzeptierend markiert wurde.

Der erste DFA, den der Lernalgorithmus als Hypothese konstruiert, akzeptiert die leere Sprache. Von da ab konstruiert der Algorithmus für $i = 1, \ldots, \lceil \frac{n}{d} \rceil$ zyklische DFAs mit n_i Zuständen als Hypothesen. Ein Zustand für eine Restklasse $x' \pmod{n_i}$ in diesem DFA, der von einem Wort $a^{x'}$ erreicht wird, ist genau dann akzeptierend, wenn es einen noch nicht als Zykluslänge ausgeschlossenen Faktor z von n_i gibt, für den die Restklasse $x' \pmod z$ als akzeptierend markiert ist.

Um die Anzahl der Gegenbeispiele abzuschätzen, die benötigt werden, um alle Faktoren von n_i als Zykluslängen auszuschließen, betrachten wir die Konsequenzen, die jedes Gegenbeispiel nach sich zieht: Erhält der Lernalgorithmus ein positives Gegenbeispiel, wird genau eine Restklasse für jeden Faktor als akzeptierend markiert. Jedes negative Gegenbeispiel führt zum Ausschluss eines Faktors als potenzielle Zykluslänge. Für Block n_i benötigt der Algorithmus höchstens $k_i + d$ Gegenbeispiele, wobei k_i der größte Faktor von n_i ist und somit die Anzahl der Restklassen des größten Faktors, während d eine obere Schranke für die Anzahl der Faktoren von n_i ist. Der Algorithmus benötigt also insgesamt höchstens

$$\sum_{i=1}^{\lceil \frac{n}{d} \rceil} (k_i + d) \leq \lceil \frac{n}{d} \rceil \cdot (n + d) = O(\frac{n^2}{d})$$

Gegenbeispiele.

(b) Wie im Beweis von Satz 10 (b) speichert das Orakel ein Liste der bestätigten Restklassen für jede Primzahl $p \leq n$. Die Bestätigung ist dabei genau so definiert wie in diesem Beweis.

Wir nehmen zunächst an, dass der Lernalgorithmus nur DFAs mit Zykluslängen konstruiert, die sich als Produkt von s Primzahlen ergeben. Wir zeigen später, wie wir s und d miteinander in Verbindung bringen und wie wir mit zusammengesetzten Zykluslängen mit Primpotenzen als Faktoren umgehen.

Das Orakel bestätigt 0 und 1 für jede Primzahl. Danach bestätigt das Orakel höchstens s neue Restklassen für jeden DFA, den es als Hypothese erhält.

D sei nun ein zyklischer DFA, der mit den vorangegangenen Beispielen konsistent ist und Zykluslänge z hat, die das Produkt der Faktoren $p_{i_1} \cdots p_{i_s}$ ist. $F \subseteq \{i_j | j = 1, \ldots, s\}$ sei die Menge der Indizes der vollständig bestätigten Primzahlen und $N = \{i_j | j = 1, \ldots, s\} \setminus F$ sei die Menge der Indizes der Primteiler von z, die nicht bestätigte Restklassen haben – F kann auch leer sein. Das Orakel wählt eine noch nicht bestätigte Restklasse $r_{i_j} \pmod{p_{i_j}}$ für jeden Index $i_j \in N$ und berechnet ein x mit $x \equiv r_{i_j} \pmod{p_{i_j}}$ für jedes $i_j \in N$ und $x \equiv 0 \pmod{p_{i_j}}$ für jedes $i_j \in F$.[2]

Falls $a^x \in L(D)$ gilt, dann berechnet das Orakel ein x', so dass $x' \equiv x \pmod{z}$ und $x' \equiv 0 \pmod{p}$ für jede Primzahl $p \leq n$ gilt, die z nicht teilt. Das Orakel gibt das Gegenbeispiel $a^{x'} \notin L(C)$. Entsprechend berechnet das Orakel für $a^x \notin L(D)$ ein x', so dass $x' \equiv x \pmod{z}$ und $x' \equiv 1 \pmod{p}$ für jede Primzahl $p \leq n$ gilt, die z nicht teilt, und antwortet mit dem Gegenbeispiel $a^{x'} \in L(C)$.

Wiederum bestätigt das Orakel die Korrektheit einer Hypothese, die durch einen DFA D dargestellt wird, erst dann, wenn alle Primzahlen vollständig bestätigt sind und wenn $D \in \mathcal{H}$ konsistent mit den gegebenen Beispielen ist. Auch hier ist es stets möglich, einen solchen konsistenten DFA zu konstruieren, da seine Zykluslänge als eine der Primzahlen p gewählt werden kann, die erst im letzten Schritt vollständig bestätigt wurden, und die akzeptierenden Zustände entsprechend der bestätigenden Beispiele $\pmod{p}$ gewählt werden können.

Um die Anzahl der benötigten Gegenbeispiele abzuschätzen, beobachten wir zunächst, dass die Zahl der nicht bestätigten Restklassen zu Beginn $\Omega(\frac{n^2}{\ln n})$ ist und dass das Orakel für jeden zu beurteilenden DFA höchstens s Restklassen bestätigt. Insgesamt werden also $\Omega(\frac{n^2}{s \cdot \ln n})$ Gegenbeispiele benötigt.

Wir schätzen nun die Zahl s der verschiedenen Primfaktoren für Zahlen der Größe höchstens n^d ab. Das Produkt $\prod_{i=1}^{s} p_i$ ist die kleinste Zahl mit s verschiedenen Primfaktoren. Wir können diese Zahl nach unten abschätzen durch

$$\prod_{i=1}^{s} p_i \geq \prod_{i=1}^{s} i \ln i \geq \prod_{i=\frac{s}{2}}^{s} \frac{s}{2} \ln \frac{s}{2} \geq \left(\frac{s}{2}\right)^{\frac{s}{2}}.$$

Wir zeigen, dass daraus $s \leq \frac{2d \ln n}{\ln d}$ folgt, falls der größte zulässige DFA aus n^d Zuständen besteht. Um dies zu beweisen, nehmen wir an, dass $s > \frac{2d \ln n}{\ln d}$

[2]Die Wahl der Restklasse für die vollständig bestätigten Primzahlen spielt eigentlich keine Rolle, der Einfachheit halber setzen wir sie auf 0.

gilt. Dann folgt

$$\frac{s}{2}\ln\frac{s}{2} > \frac{d\ln n}{\ln d}\cdot\ln\left(\frac{d\ln n}{\ln d}\right) \geq \frac{d\ln n}{\ln d}\cdot\left(\ln d + \ln\left(\frac{\ln n}{\ln d}\right)\right) \geq d\ln n$$

für $d \leq n$ und somit $\prod_{i=1}^{s} p_i \geq \left(\frac{s}{2}\right)^{\frac{s}{2}} = e^{\frac{s}{2}\ln\frac{s}{2}} > n^d$, was der oberen Schranke für die Zykluslänge widerspricht. Die Anzahl der Gegenbeispiele ist somit mindestens $\Omega\left(\frac{n^2}{s\cdot\ln n}\right) \geq \Omega\left(\frac{n^2\ln d}{d(\ln n)^2}\right)$.

Wenn der Lernalgorithmus einen DFA als Hypothese konstruiert, dessen Zykluslänge durch eine Primpotenz p^α für $1 < \alpha \in \mathbb{N}$ teilbar ist, können wir diesen DFA einfach so behandeln, als ob nur der Primfaktor p^1 vorläge. □

Kapitel 8

Offene Fragen

Unsere Nicht-Approximierbarkeitsergebnisse bei gegebenem DFA basieren auf der Annahme der Existenz von Pseudozufallsfunktionen in NC^1 beziehungsweise LogSPACE. Welche Approximationsfaktoren lassen sich unter einer schwächeren Annahme wie P $\neq$ NP ausschließen? Gruber und Holzer [16] bereiten eine Arbeit vor, die zeigt, dass unter der Annahme P $\neq$ NP für jedes $\varepsilon > 0$ effiziente Approximationen mit Approximationsfaktor $n^{\frac{1}{2}-\varepsilon}$ für die Bestimmung der Anzahl der Zustände eines minimalen äquivalenten NFAs zu gegebenem DFA mit n Zuständen ausgeschlossen sind. Für die Minimierung der Übergänge schließen sie unter derselben Annahme $n^{\frac{1}{3}-\varepsilon}$ aus.

Für die Nicht-Approximierbarkeit bei gegebenem DFA mit n Zuständen konnten wir in Korollar 1 nur einen Approximationsfaktor $\frac{\sqrt{n}}{\text{poly}(\log n)}$ ausschließen, wenn wir äquivalente NFAs mit wenig Zuständen fordern. Wir haben im Anschluss an Korollar 1 gezeigt, dass NFAs, die m-stellige boolesche Funktionen berechnen, mit $O(\sqrt{2^m})$ Zuständen auskommen. Gibt es andere Ansätze, durch die effiziente Approximationsalgorithmen zu größeren Approximationsfaktoren als $\sqrt{n}$ gezwungen werden?

Welche Komplexität hat die approximative Minimierung, wenn wir NFAs mit beschränktem Nichtdeterminismus oder eindeutige NFAs als Eingabe der Approximation betrachten? Ebenso stellt sich die Frage nach der Approximationskomplexität bei eingeschränktem Nichtdeterminismus für die Ausgabe. In [25] weisen Jiang und Ravikumar nach, dass die Bestimmung der Anzahl der Zustände eines minimalen eindeutigen NFAs zu einem gegebenen eindeutigen NFA NP-vollständig ist. Malcher zeigt in [28], dass auch die Minimierung von endlichen Automaten mit DFA Struktur, bei denen nichtdeterministisch einer von konstant vielen Startzuständen gewählt werden darf, ebenso wie die Minimierung von NFAs, die bei jeder akzeptierenden Berechnung nur eine konstante Anzahl von nichtdeterministischen Entscheidungen treffen, NP-vollständig ist.

In Satz 7 zeigen wir, dass das Problem des minimalen unären konsisten-

ten NFAs unter der Annahme NP $\not\subseteq$ DTIME($n^{O(\log n)}$) nicht effizient lösbar ist, allerdings können mit Algorithmus 2 konsistente NFAs mit $O(\text{opt}^2)$ Zuständen effizient gefunden werden. Welche untere Schranken kann man für die Approximationskomplexität dieses Problems angeben?

Beim Lernen unärer regulärer Sprachen mit Äquivalenzfragen fällt auf, dass das Orakel für Satz 10 (b) Gegenbeispiele mit Länge $2^{\Theta(n)}$ für das Lernen zyklischer DFAs mit höchstens n Zuständen und primer Zykluslänge verwendet. Da es sich um unäre Gegenbeispiele handelt, können diese mit $\Theta(n)$ Bits dargestellt werden und die Auswerung des Akzeptanzverhaltens eines zyklischen DFAs auf dem Beispiel ist durch die modulo-Rechnung effizient möglich. Dennoch stellt sich beispielsweise die Frage, ob es zu jeder zyklischen unären Sprache ein „gutmütiges“ Orakel gibt, welches nur Gegenbeispiele polynomieller Länge ausgibt und falls dies der Fall ist, gibt es dann Lernalgorithmen, die weniger Gegenbeispiele benötigen als in Satz 10 angegeben?

Anhang A

Mathematische Grundlagen, Notationen und Definitionen

A.1 Notationen

$\mathbb{N}$	Die natürlichen Zahlen $\mathbb{N} = \{0, 1, 2, \ldots\}$.
$\mathbb{Z}$	Die ganzen Zahlen $\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$.
$A \subseteq B$	A ist Teilmenge von B.
$A \subset B$	A ist echte Teilmenge von B.
$\mathcal{P}(B)$	Die Potenzmenge von B: $\mathcal{P}(B) = \{A \mid A \subseteq B\}$.
$\mathrm{ggT}(m, n)$	Größter gemeinsamer Teiler von m und n.
$\mathrm{kgV}(m, n)$	Kleinstes gemeinsames Vielfaches von m und n.
$\log n$	Der Logarithmus zur Basis 2 von n.
$\ln n$	Der natürliche Logarithmus von n.
$\log^{(i)} n$	Der i-fach angewandte Logarithmus von n: $\log^{(0)} n = n$ und $\log^{(i)} n = \log(\log^{(i-1)} n)$.
$\log^* n$	Der iterierte Logarithmus zur Basis 2 von n: $\log^* n = \min\{i \geq 0 \mid \log^{(i)} n \leq 1\}$.
$\pi(n)$	Die Anzahl der Primzahlen $\leq n$, siehe Abschnitt A.3.1.

A.1.1 Asymptotische Notationen

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

$$
\begin{aligned}
f(n) = O(g(n)) &\iff \exists c > 0, n_0 \in \mathbb{N},\ \forall n \geq n_0 : f(n) \leq c \cdot g(n), \\
f(n) = \Omega(g(n)) &\iff g(n) = O(f(n)), \\
f(n) = \Theta(g(n)) &\iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n)), \\
f(n) = o(g(n)) &\iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0, \\
f(n) = \omega(g(n)) &\iff g(n) = o(f(n)).
\end{aligned}
$$

A.2 Komplexitätsklassen

TC^0	Die Klasse der Entscheidungsprobleme, die von Schaltkreisen polynomieller Größe und konstanter Tiefe mit Negationsgattern sowie Mehrheitsgattern mit unbeschränktem Fan-In akzeptiert werden. Ein Mehrheitsgatter gibt 1 zurück, falls mindestens die Hälfte seiner Eingaben 1 ist und 0 sonst.
NC^1	Die Klasse der Entscheidungsprobleme, die von Schaltkreisen polynomieller Größe und Tiefe $O(\log n)$ mit Negationsgattern sowie $\wedge$ und $\vee$ Gattern mit Fan-In 2 akzeptiert werden. NC^1 enthält TC^0.
DSPACE($f(n)$)	Die Klasse der Entscheidungsprobleme, die von deterministischen Turingmaschinen mit Eingabeband und Arbeitsband mit Platz $O(f(n))$ auf dem Arbeitsband akzeptiert werden.
NSPACE($f(n)$)	Die Klasse der Entscheidungsprobleme, die von nichtdeterministischen Turingmaschinen mit Eingabeband und Arbeitsband mit Platz $O(f(n))$ auf dem Arbeitsband akzeptiert werden.
DTIME($f(n)$)	Die Klasse der Entscheidungsprobleme, die von deterministischen Turingmaschinen in Zeit $O(f(n))$ akzeptiert werden.
NTIME($f(n)$)	Die Klasse der Entscheidungsprobleme, die von nichtdeterministischen Turingmaschinen in Zeit $O(f(n))$ akzeptiert werden.
LogSPACE	Die Klasse DSPACE($\log n$). LogSPACE enthält NC^1.
CSL	Die Klasse der kontextsensitiven Sprachen (Entscheidungsprobleme). Sie stimmt mit NSPACE(n) überein.
PSPACE	Die Klasse der Entscheidungsprobleme, die von Turingmaschinen mit polynomiellem Platz akzeptiert werden. Es gilt PSPACE = DSPACE($n^{O(1)}$) = NSPACE($n^{O(1)}$).
P	Die Klasse der Entscheidungsprobleme, die von deterministischen Turingmaschinen in polynomieller Zeit akzeptiert werden, also P = DTIME($n^{O(1)}$). Wir bezeichnen die Probleme in P als effizient lösbar.
NP	Die Klasse der Entscheidungsprobleme, die von nichtdeterministischen Turingmaschinen in polynomieller Zeit akzeptiert werden, also NP = NTIME($n^{O(1)}$).
RP	Die Klasse der Entscheidungsprobleme, die von randomisierten Turingmaschinen mit einseitigem Fehler in polynomieller Zeit akzeptiert werden.

Um die Schwierigkeit von Entscheidungsproblemen miteinander vergleichen zu können, führen wir den Begriff der Reduktion ein. Ein Problem L_1 ist nicht wesentlich schwieriger als ein Problem L_2, wenn sich L_1 mit Hilfe eines Algorithmus für L_2 lösen lässt.

Definition 12. *Wir sagen, dass ein Entscheidungsproblem L_1 polynomiell auf L_2 reduzierbar ist und schreiben $L_1 \leq_p L_2$, wenn es eine deterministische Turingmaschine T mit Ausgabe $T(\cdot)$ gibt, die in polynomieller Zeit eine Eingabe-Instanz x für L_1 in eine Eingabe-Instanz $T(x)$ für L_2 transformiert, so dass $x \in L_1 \Leftrightarrow T(x) \in L_2$ gilt.*

Wir nennen ein Entscheidungsproblem L schwer für eine Komplexitätsklasse $\mathcal{C}$, falls $\forall L' \in \mathcal{C} : L' \leq_p L$ gilt. Wir nennen L vollständig für $\mathcal{C}$, falls L schwer für $\mathcal{C}$ ist und $L \in \mathcal{C}$ gilt.

Die Pseudozufallsfunktionen, deren Existenz wir in Kapitel 4 voraussetzen, werden durch nicht-uniforme Schaltkreisfamilien oder nicht-uniformen LogSPACE berechnet. Nicht-uniformen LogSPACE definieren wir hier durch Zweiweg-DFAs polynomieller Größe.

Definition 13. *Eine Schaltkreisfamilie $\mathcal{S}$, besitzt für jede Eingabelänge n einen Schaltkreis S_n. $\mathcal{S}$ heißt uniform, wenn es eine Turingmaschine gibt, die für jede Eingabelänge n die Beschreibung von S_n ausgibt. Die Turingmaschine muss dabei mit Speicherplatz $O(\log s_n)$ auskommen, wobei s_n die Anzahl der Gatter von S_n ist.*

Bei einer nicht-uniformen Schaltkreisfamilie kann für jede Eingabelänge ein eigener Schaltkreis angegeben werden, ohne eine allgemeine Konstruktionsvorschrift zu beachten.

A.3 Grundlagen aus der diskreten Mathematik

Wir benötigen einige Grundlagen aus der diskreten Mathematik, die insbesondere bei den unären regulären Sprachen Anwendung finden.

A.3.1 Größenabschätzungen für Primzahlen

Für die Anzahl der Primzahlen $\leq n$ gilt der Primzahlsatz

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1.$$

Wir verwenden als Abschätzung $\pi(n) \approx n/\ln n$. Als Konsequenz ergibt sich für die Größe der i-ten Primzahl p_i

$$\lim_{i\to\infty} \frac{p_i}{i \cdot \ln i} = 1.$$

Für jedes $i \geq 1$ gilt $p_i \geq i \ln i$ und als grobe obere Schranke genügt uns, dass $p_i \leq 2i \ln i$ für $i \geq 3$ gilt [12].

A.3.2 Modulo Rechnung und Chinesischer Restsatz

Es gelte $k, n, m \in \mathbb{N}$, mit $m \geq 2$. Notationen:

$n \bmod m$	bezeichnet den ganzzahligen Rest der Division von n durch m: $n \bmod m = n - \lfloor \frac{n}{m} \rfloor \cdot m$.
$k \equiv n \pmod m$	$\Leftrightarrow (k \bmod m) = (n \bmod m)$, wir nennen k kongruent zu n (modulo m).

Der chinesische Restsatz erlaubt es, eine Zahl als die Folge ihrer Reste modulo verschiedener primer Moduln zu repräsentieren. Eine Version des Satzes lautet:

Fakt 8 (Chinesischer Restsatz). *Es seien $m_1, m_2, \ldots, m_k$ paarweise teilerfremde natürliche Zahlen und $M = \prod_{i=1}^{k} m_i$. Ferner seien $a_1, a_2, \ldots, a_k$ natürliche Zahlen. Dann bildet die Menge aller x mit*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\ldots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

genau eine Restklasse modulo M.

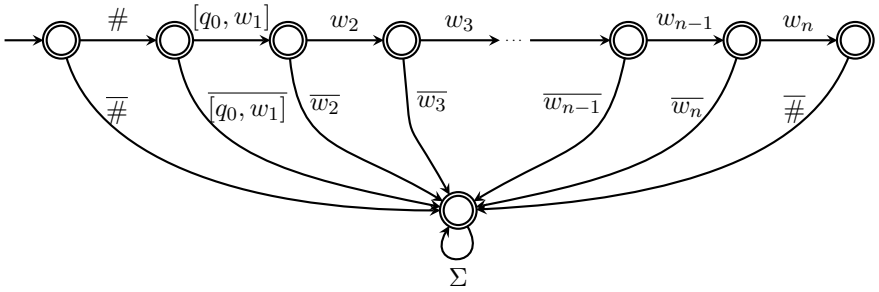
Anhang B

NFA-Konstruktionen

Wir konstruieren hier einen NFA N_w, der äquivalent zum regulären Ausdruck R_w aus dem Beweis zu Fakt 4 auf Seite 25 ist. N_w akzeptiert alle Wörter, die keine legale Folge von Konfigurationen für eine akzeptierende Berechnung einer Turingmaschine auf Eingabe w kodieren.

Der Automat wird aus Teilautomaten aufgebaut, die anschließend zu einem Automaten zusammengeführt werden, der die Vereinigung der Teilsprachen akzeptiert. Hierbei wird ein neuer Startzustand eingefügt. Die Anzahl der Zustände erhöht sich also um 1 gegenüber der Gesamtzahl der Zustände der Teilautomaten. Betrachten wir die Anzahl der Übergänge, so sehen wir, dass bei der Konstruktion des NFAs für die Vereinigung für jeden Übergang aus den Startzuständen der Teilautomaten ein neuer Übergang eingefügt wird und sich somit die Gesamtzahl der Übergänge höchstens verdoppelt.

- Wir akzeptieren jedes Wort, das nicht mit $\#[q_0, w_1]w_2 \dots w_n\#$ beginnt.

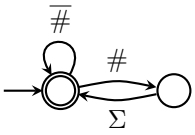


  Ein Übergang, der mit Σ beschriftet ist, beschreibt Übergänge für jedes Zeichen aus Σ. Ein Übergang, der mit $\overline{a}$ beschriftet ist, beschreibt Übergänge für jedes Zeichen aus $\Sigma \setminus \{a\}$. Der Automat verwirft nur, wenn im Zustand ganz rechts ein # gelesen wird.

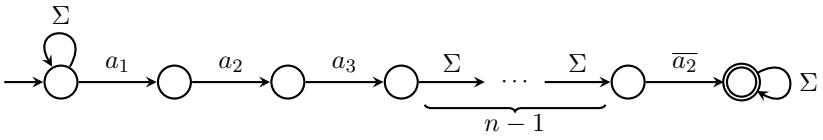
- Wir akzeptieren jedes Wort, welches nicht $[q_f, \gamma]$ für irgendein $\gamma \in \Gamma_M$ enthält.

Hier ist der Übergang $\overline{[q_f, \cdot]}$ als ein Übergang für jedes Symbol aus $\Sigma \setminus (\{q_f\} \times \Gamma_M)$ zu lesen.

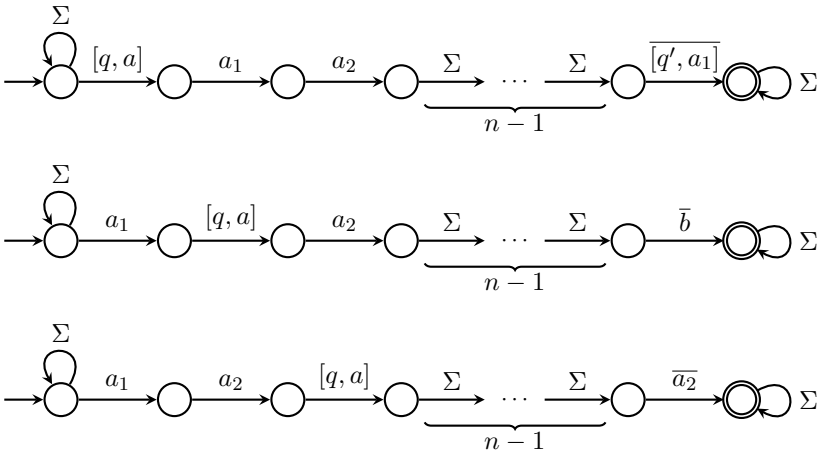
- Akzeptiere jedes Wort, welches nicht mit # endet.



- Schließlich prüfen wir, ob zwei aufeinander folgende Konfigurationen mit der Überführungsfunktion δ konsistent sind. Befindet sich der Kopf der Turingmaschine nicht in der Nähe, darf sich keine Änderung des Bandinhalts ergeben. Wir akzeptieren illegale Folgen, indem wir für alle $a_1, a_2, a_3 \in \Gamma_M \cup \{\#\}$ prüfen, ob das mittlere Symbol a_2 verändert wird.



Befindet sich der Kopf in der Nähe, so ergeben sich Änderungen gemäß der Überführungsfunktion δ. Wir akzeptieren „falsche aufeinander folgende“ Konfigurationen. Für alle $a_1, a_2 \in \Gamma_M$ und $[q, a] \in Q_M \times \Gamma_M$ mit $\delta(q, a) = (q', b, \rightarrow)$ für ein $q' \in Q_M$ und $b \in \Gamma_M$ konstruiere die drei Teilautomaten:



Für Linksbewegungen $\delta(q, a) = (q', b, \leftarrow)$ werden die Automaten entsprechend modifiziert.

Insgesamt hat der Automat N_w, der aus der Vereinigung der Teilautomaten entsteht höchstens $2 \cdot |w| \cdot |\Sigma|^3 = O(|w|)$ Zustände und höchstens $4 \cdot |w| \cdot |\Sigma|^4 = O(|w|)$ Übergänge.

Literaturverzeichnis

[1] D. Angluin. Negative Results for Equivalence Queries. *Mach. Learn.*, 5(2) 121–150, 1990.

[2] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4) 929–965, 1989.

[4] R. V. Book. On Languages Accepted in Polynomial Time. *SIAM J. Comput.*, 1(4) 281–287, 1972.

[5] J.-M. Champarnaud, F. Coulon. NFA reduction algorithms by means of regular inequalities. *Theor. Comput. Sci.*, 327(3) 241–253, 2004.

[6] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3) 113–124, 1956.

[7] M. Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(2) 149–158, 1986.

[8] M. Domaratzki, D. Kisman, J. Shallit. On the Number of Distinct Languages Accepted by Finite Automata with n States. *Journal of Automata, Languages and Combinatorics*, 7(4) 469–486, 2002.

[9] M. R. Garey, D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences).* W. H. Freeman, 1979.

[10] E. M. Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3) 302–320, 1978.

[11] O. Goldreich, S. Goldwasser, S. Micali. How to construct random functions. *J. ACM*, 33(4) 792–807, 1986.

[12] R. L. Graham, D. E. Knuth, O. Patashnik. *Concrete mathematics: a foundation for computer science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[13] G. Gramlich. Probabilistic and Nondeterministic Unary Automata. In B. Rovan, P. Vojtás (Herausgeber), *MFCS*, Band 2747 von *Lecture Notes in Computer Science*, 460–469. Springer-Verlag, 2003.

[14] G. Gramlich, R. Herrmann. Learning Unary Automata. In C. Mereghetti, B. Palano, G. Pighizzini, D. Wotschke (Herausgeber), *DCFS05*, 122–133. 2005.

[15] G. Gramlich, G. Schnitger. Minimizing NFA's and Regular Expressions. In V. Diekert, B. Durand (Herausgeber), *STACS05*, Band 3404 von *Lecture Notes in Computer Science*, 399–411. Springer-Verlag, 2005.

[16] H. Gruber, M. Holzer. Persönliche Kommunikation, 2006.

[17] R. Herrmann. *Lernen regulärer unärer Sprachen*. Diplomarbeit, Johann Wolfgang Goethe–Universität, Institut für Informatik, 2004.

[18] J. Hopcroft. An $n \log n$ Algorithm for Minimizing States in a Finite Automaton. *Theory of Machines and Computations*, 189–196, 1971.

[19] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[20] J. Hromkovič. Descriptional Complexity of Finite Automata: Concepts and Open Problems. *Journal of Automata, Languages and Combinatorics*, 7(4) 519–531, 2002.

[21] H. B. Hunt III, D. J. Rosenkrantz, T. G. Szymanski. On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages. *J. Comput. Syst. Sci.*, 12(2) 222–268, 1976.

[22] L. Ilie, G. Navarro, S. Yu. On NFA Reductions. In J. Karhumäki, H. A. Maurer, G. Paun, G. Rozenberg (Herausgeber), *Theory Is Forever*, Band 3113 von *Lecture Notes in Computer Science*, 112–124. Springer-Verlag, 2004.

[23] L. Ilie, S. Yu. Follow automata. *Inf. Comput.*, 186(1) 140–162, 2003.

[24] T. Jiang, E. McDowell, B. Ravikumar. The Structure and Complexity of Minimal NFA's over a Unary Alphabet. *Int. J. Found. Comput. Sci.*, 2(2) 163–182, 1991.

[25] T. Jiang, B. Ravikumar. Minimal NFA Problems are Hard. *SIAM J. Comput.*, 22(6) 1117–1141, 1993.

[26] M. J. Kearns, L. G. Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *J. ACM*, 41(1) 67–95, 1994.

[27] M. J. Kearns, U. V. Vazirani. *An introduction to computational learning theory.* MIT Press, Cambridge, MA, 1994.

[28] A. Malcher. Minimizing finite automata is computationally hard. *Theor. Comput. Sci.*, 327(3) 375–390, 2004.

[29] O. Matz, A. Potthoff. Computing small nondeterministic finite automata. In *Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 74–88. Dpt. of CS., Univ. of Aarhus, 1995.

[30] R. McNaughton, H. Yamada. Regular Expressions and State Graphs for Automata. *IEEE Transactions on Electronic Computers*, 9 39–47, 1960.

[31] A. R. Meyer, L. J. Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In *Proc. 13th Ann. IEEE Symp. on Switching and Automata Theory*, 125–129. 1972.

[32] M. Naor, O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2) 231–262, 2004.

[33] L. Pitt, L. G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4) 965–984, 1988.

[34] L. Pitt, M. K. Warmuth. Prediction-Preserving Reducibility. *J. Comput. Syst. Sci.*, 41(3) 430–467, 1990.

[35] L. Pitt, M. K. Warmuth. The Minimum Consistent DFA Problem Cannot be Approximated within any Polynomial. *J. ACM*, 40(1) 95–142, 1993.

[36] O. Reingold. *Pseudo-Random Synthesizers, Functions and Permutations.* Dissertation, Weizmann Institute of Science, Department of Applied Mathematics and Computer Science, 1998.

[37] G. Schnitger. Regular Expressions and NFAs Without ε-Transitions. In B. Durand, W. Thomas (Herausgeber), *STACS*, Band 3884 von *Lecture Notes in Computer Science*, 432–443. Springer, 2006.

[38] L. J. Stockmeyer, A. R. Meyer. Word Problems Requiring Exponential Time: Preliminary Report. In *Proc. of the 5th Annual ACM Symposium on Theory of Computing*, 1–9. 1973.

[39] T. Wilke. Minimizing Automata on Infinite Words. http://www.ti.informatik.uni-kiel.de/~wilke/Vortraege/Minimizing_Automata.pdf, 2003.

Lebenslauf

Persönliche Daten

Name: Gregor Gramlich
Familienstand: unverheiratet, keine Kinder
Geburtsdatum: 19.09.1973
Nationalität: deutsch

Schulausbildung

1980 - 1984 Grundschule: Beethovenschule in Offenbach am Main
1984 - 1986 Förderstufe: Edith-Stein-Schule in Offenbach am Main
1986 - 1993 Gymnasium: Leibnizschule in Offenbach am Main

- Abschluss: Abitur 1993
- Note: 1,3
- Leistungskurse: Mathematik und Physik
- Fremdsprachen: Englisch (9 Jahre), Latein (6 Jahre), Französisch (2 Jahre)

Studium

April 1994 - Juli 2002
Studium im Diplomstudiengang der Informatik an der Johann Wolfgang Goethe-Universität in Frankfurt am Main

Vordiplom

Abschluss: Vordiplom (12.09.1996)
Gesamtnote: gut
Einzelprüfungen:

- Praktische Informatik: gut
- Theoretische Informatik: gut
- Mathematik A: gut
- Mathematik B: sehr gut
- Nebenfach Linguistik: gut

Späterer Wechsel des Nebenfachs zur Mathematik, Diplomvorprüfung Mathematik: sehr gut

Hauptstudium

Gesamtnote: mit Auszeichnung
Prüfungen:
- Praktische/Technische Informatik: 1,0
  Datenbanken (1 & 2)
- Theoretische Informatik: 1,0
  Beschreibungskomplexität (1 & 2)
- Vertiefungsfach: 1,0
  Effiziente Algorithmen und Kommunikationskomplexität
- Nebenfach: 1,0
  Höhere Stochastik und Analysis IV

Diplomarbeit „Unäre stochastische Automaten“: 1,0

Derzeitige Tätigkeit

Seit September 2002 als Wissenschaftlicher Mitarbeiter angestellt am Lehrstuhl *Theoretische Informatik* bei Prof. Dr. G. Schnitger im Institut für Informatik der Johann Wolfgang Goethe-Universität Frankfurt am Main.

Erfahrungen in der Lehre

Okt. 1997 - Jul. 1998: Tutorentätigkeit – Betreuung von Übungsgruppen zu den Veranstaltungen
- Theoretische Informatik 1; WS 97/98
- Theoretische Informatik 2; SS 98

seit September 2002: Organisation und Betreuung von Übungsbetrieben als Wissenschaftlicher Mitarbeiter. Die Veranstaltungen:
- Approximationsalgorithmen; WS 02/03
- Effiziente Algorithmen; SS 03
- Internet Algorithmen; WS 03/04
- Komplexitätstheorie; SS 04
- Theoretische Informatik 2; SS 05
- Theoretische Informatik 1; WS 05/06
- Theoretische Informatik 2; SS 06
- Algorithmentheorie; WS 06/07
- Datenstrukturen; SS 07

Parallel dazu: Betreuungen im Rahmen von Seminaren, Proseminaren und Diplomarbeiten.

Publikationen

- Probabilistic and Nondeterministic Unary Automata – Mathematical Foundations of Computer Science (MFCS), August 2003
- mit Georg Schnitger: Minimizing NFA's and Regular Expressions – Symposium on Theoretical Aspects of Computer Science (STACS), Februar 2005
- mit Ralf Herrmann: Learning Unary Automata – Descriptional Complexity of Formal Systems (DCFS), Juni/Juli 2005
- mit Georg Schnitger: Minimizing NFA's and Regular Expressions – Journal of Computer and System Sciences Vol. 73(6), 2007

Vorträge

26.08.2003 - Probabilistic and Nondeterministic Unary Automata (MFCS, Bratislava, Slowakei)
12.01.2005 - Minimierung von NFAs und Regulären Ausdrücken (AG-Treffen)
25.02.2005 - Minimizing NFA's and Regular Expressions (STACS, Stuttgart)
01.06.2005 - Lernen unärer Automaten (AG-Treffen)
30.06.2005 - Learning Unary Automata (DCFS, Como, Italien)
18.01.2007 - Über die algorithmische Komplexität regulärer Sprachen (Doktorandenkollquium des Instituts für Informatik)
2004-2007 Mehrere Vorstellungen der Informatik-Studiengänge anlässlich der Infotage und des Tags der Naturwissenschaften

Erfahrungen in der Selbstverwaltung

Studienberatung
Berufungskommissionen
Arbeitskreis Naturwissenschaften