

Lernen regulärer unärer Sprachen

Ralf Herrmann

Diplomarbeit

Betreut von Prof. Dr. Georg Schnitger

28. April 2004

Fachbereich Biologie und Informatik
Johann Wolfgang Goethe-Universität
Frankfurt am Main

Erklärung

Hiermit erkläre ich, die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel eingesetzt zu haben.

Frankfurt am Main, den 28. April 2004

Ralf Herrmann

Zusammenfassung

Wir untersuchen das algorithmische Lernen regulärer unärer Sprachen. Dabei heißt eine reguläre Sprache genau dann unär, wenn sie über einer Alphabet mit nur einem Buchstaben erklärt ist. Zuerst werden die bekannten Lernmodelle des PAC-Lernens und des Lernens mit Äquivalenz- und Elementfragen und wichtige bekannte Ergebnisse, insbesondere zum Lernen regulärer Sprachen, vorgestellt.

Für das für das PAC-Lernen wichtige Konsistenzproblem der regulären unären Sprachen wird ein Algorithmus angegeben, der in Zeit $O(n \cdot |S|)$ den mit der Beispielmengung S minimalen konsistenten DFA berechnet, wobei n die Zahl der Zustände des minimalen DFA für die zu lernende Sprache L ist. Außerdem werden eine obere und eine untere Schranke für die VC-Dimension der regulären unären Sprachen angegeben. Sei L_n die Menge der von einem DFA mit n Zuständen akzeptierten regulären unären Sprachen. Dann gilt

$$VC(L_n) \geq n - 1 + \lfloor \log\left(\frac{n}{\ln(n)} + 1\right) \rfloor$$

und

$$VC(L_n) \leq n + \log(n)$$

für $n \geq 5$. Weiter wird nachgewiesen, dass die regulären unären Sprachen, im Gegensatz zu den regulären Sprachen, mit Äquivalenz- oder Elementfragen alleine lernbar sind. Für das Lernen mit Äquivalenzfragen wird eine obere Schranke von $O(n^2)$ Fragen gezeigt, wobei n die Zahl der Zustände des minimalen DFA der zu lernenden Sprache L ist. Für das Lernen der Konzeptklasse L_n durch die Hypothesenklasse der unären deterministischen endlichen Automaten mit primitiver Zykluslänge wird eine untere Schranke von

$$\Omega\left(\frac{n^2}{\ln(n)}\right)$$

Fragen bestimmt. Für das Lernen mit Elementfragen wird eine obere Schranke von $2n - 1$ Fragen bei bekanntem n gegeben. Wird dem Lernalgorithmus zusätzlich gestattet eine abgeschwächte Äquivalenzfrage zu stellen, eine Frage, die ihm beantwortet ob seine Hypothese äquivalent zu der zu lernenden Sprache L ist, aber kein Gegenbeispiel liefert, dann ergibt sich für das Lernen der regulären unären Sprachen eine obere Schranke von $2n$ Elementfragen und n abgeschwächten Äquivalenzfragen.

Inhaltsverzeichnis

1	Einleitung	4
2	Grundlagen	6
2.1	Zahlentheoretische Grundlagen	6
2.2	Reguläre Sprachen und deterministische endliche Automaten	7
2.3	Notationen	8
3	Lernmodelle	10
3.1	Lernen aus Beispielen	10
3.2	PAC-Lernen	11
3.2.1	Obere Schranke der Beispielskomplexität	13
3.2.2	VC-Dimension und Beispielskomplexität	16
3.2.3	Effiziente PAC-Algorithmen	19
3.2.4	Kryptographie und Lernkomplexität	24
3.3	Lernen regulärer Sprachen mit Element- und Äquivalenzfragen	32
3.3.1	Lernen mit Äquivalenz- oder Elementfragen	34
3.3.2	Angluins Algorithmus	34
4	Reguläre unäre Sprachen	41
4.1	Klassifizierung der unären deterministischen endlichen Automaten	41
4.2	Konsistenzproblem der Klasse $M(n, z)$	42
4.3	VC-Dimension und Beispielskomplexität	46
5	Konsistente Hypothesen für reguläre unäre Sprachen	49
5.1	Obere Schranke der Zustandszahl des minimalen konsistenten DFA	49
5.2	Berechnung des minimalen konsistenten DFA	51
6	Aktives Lernen regulärer unärer Sprachen	55
6.1	Lernen mit Äquivalenzfragen	55
6.1.1	Obere Schranke für die Zahl der Äquivalenzfragen	56
6.1.2	Untere Schranke für die Zahl der Äquivalenzfragen	59
6.2	Lernen mit Elementfragen	62
6.2.1	Obere Schranke für die Zahl der Elementfragen	63
7	Ausblick	67
7.1	VC-Dimension und Beispielskomplexität	67
7.2	Konsistente Hypothesen für reguläre unäre Sprachen	67
7.3	Aktives Lernen regulärer unärer Sprachen	68
7.4	Aktives Lernen anderer Sprachklassen	70

Kapitel 1

Einleitung

In dieser Arbeit wird das algorithmische Lernen der regulären unären Sprachen untersucht. Eine reguläre Sprache heißt genau dann unär, wenn sie über einem Alphabet mit nur einem Buchstaben erklärt ist. Die Einschränkung auf reguläre unäre Sprachen ergab sich dabei aus der Motivation der Arbeit: eine Teilklasse von Sprachen zu finden, die mit Äquivalenzfragen lernbar ist.

Das Deutsche Universalwörterbuch [Du96] definiert das *Lernen*, unter anderem, als “Kenntnisse, Fertigkeiten erwerben” und “im Laufe der Zeit (durch Erfahrungen, Einsichten), zu einer bestimmten Einstellung, einem bestimmten Verhalten gelangen”. Die Umsetzung dieses Verfahrens in Algorithmen wird in der Theorie des algorithmischen Lernens behandelt. Dabei wird das Lernen vom bloßen Sammeln von Informationen abgegrenzt. Stattdessen erfordert jede neue Erfahrung ein Überdenken des bisherigen Verhaltens. Ob ein Satz grammatikalisch richtig ist, wird ein Mensch sicherlich nicht dadurch lernen, dass er sich alle möglichen grammatikalisch richtigen Sätze einprägt. Eher wird er aus einigen Beispielsätzen eine Grammatik ableiten, und mit ihr neue Sätze formulieren. Sind diese falsch, gewinnt er, dank der Verbesserung eines Dritten, neue Erfahrungen, anhand derer er seine Vorstellung der Grammatik zu korrigieren, zu verfeinern vermag. Diesen Ansatz verfolgen auch die betrachteten Lernalgorithmen.

Um die Qualität und die Komplexität eines Lernalgorithmus abzuschätzen, muss das Lernen mathematisch modelliert werden. In Kapitel 3 werden hierzu zwei der bekanntesten Lernmodelle eingeführt: das PAC-Lernen und das Lernen mit Äquivalenz- und Elementfragen. Beiden Modellen gemein ist, dass der Lernalgorithmus auf klassifizierte Beispiele, das sind Beispiele von denen er weiß, ob sie gemäß der zu lernenden Regel korrekt sind, zurückgreifen darf. Im PAC-Modell kann der Lernalgorithmus nur zufällig gezogene Beispiele anfordern, um eine mit diesen Beispielen konsistente Vorstellung der Regel zu gewinnen. Da er in diesem Modell nicht aktiv nach der Klassifizierung eines bestimmten Beispiels fragen darf, muss er genügend Beispiele anfordern, damit seine Vorstellung der Regel annähernd korrekt ist. Das Lernen mit Äquivalenz- und Elementfragen weist eine größere Lernkraft als das PAC-Lernen auf. In diesem Modell darf der Lernalgorithmus eine Elementfrage, um die Klassifizierung eines bestimmten Beispiels zu erhalten, oder eine Äquivalenzfrage, um sich von der Richtigkeit seiner Vorstellung der Regel zu überzeugen oder ein Gegenbeispiel zu erhalten, stellen. Es wird gezeigt, dass im PAC-Modell reguläre Sprachen nicht effizient lernbar sind, mit Äquivalenz- und Elementfragen dagegen schon.

In Kapitel 4 werden die Grundlagen für die Untersuchung der regulären unären Sprachen erarbeitet. Dazu wird, über den Graphen der Übergangsfunktion, eine Klassifizierung der deterministischen endlichen Automaten mit unärem Eingabealphabet vorgenommen. Außerdem wird ein erstes Ergebnis vorgestellt: eine obere und eine untere Schranke für die VC-Dimension der regulären unären Sprachen.

Das Konsistenzproblem der regulären unären Sprachen wird in Kapitel 5 behandelt. Hier wird ein Algorithmus vorgestellt, der den zu einer Menge klassifizierter Beispiele minimalen konsistenten DFA effizient berechnet.

In Kapitel 6 wird das Lernen mit Äquivalenzfragen und das Lernen mit Elementfragen untersucht. Es wird eine obere und eine untere Schranke für die Zahl der Äquivalenzfragen gezeigt. Beim Lernen mit Elementfragen wird eine obere Schranke für die Zahl der Elementfragen unter der Voraussetzung, dass die Größe des minimalen DFA der zu lernenden Sprache bekannt ist, gezeigt. Das Modell wird dann um eine abgeschwächte Äquivalenzfrage, eine Äquivalenzfrage die als Antwort nur “ja” oder “nein” aber kein Gegenbeispiel liefert, erweitert. Es wird nachgewiesen, dass die regulären unären Sprachen mit Element- und abgeschwächten Äquivalenzfragen lernbar sind und eine obere Schranke für die Zahl der Fragen angegeben.

Wir beschließen die Arbeit in Kapitel 7 mit einem Ausblick auf offene Fragen, deren weitere Erforschung lohnenswert erscheint. Dazu werden einige Lösungsansätze angeführt, die möglicherweise bei der Beantwortung dieser Fragen helfen.

Kapitel 2

Grundlagen

Die grundlegenden allgemein bekannten mathematischen Begriffe sollen in dieser Arbeit nicht exakt definiert, sondern nur ansatzweise eingeführt werden, um Notationen zu verdeutlichen und auf bekannte Ergebnisse verweisen zu können.

2.1 Zahlentheoretische Grundlagen

Die Definitionen und Sätze aus der Zahlentheorie wurden vor allem [RU95] und [Den03] entnommen.

Mit \mathbb{Z} wird die Menge der ganzen Zahlen bezeichnet. Soweit nicht anders angemerkt sind alle Zahlen in dieser Arbeit positive ganze Zahlen. Eine Zahl $d \in \mathbb{Z}$ heißt ein Teiler der Zahl $a \in \mathbb{Z}$, geschrieben als $d \mid a$, wenn es eine Zahl $b \in \mathbb{Z}$ gibt, so dass $b \cdot d = a$ gilt. Eine ganze Zahl $p > 1$ wird prim oder Primzahl genannt, wenn 1 und p die einzigen positiven Teiler von p sind. Mit $\text{ggT}(a, b)$ wird der größte gemeinsame Teiler zweier Zahlen $a, b \in \mathbb{Z}$ bezeichnet. Gilt $\text{ggT}(a, b) = 1$, so heißen a und b relativ prim.

Zwei Zahlen $a, b \in \mathbb{Z}$, die bei der Division durch m den selben Rest ergeben, also mit $m \mid (a - b)$, werden nach GAUSS kongruent modulo m genannt. Geschrieben wird dies als $a \equiv b \pmod{m}$. Die Kongruenz ist eine Äquivalenzrelation deren Äquivalenzklassen mit $a \pmod{m} = \{b \mid b \in \mathbb{Z} \wedge b \equiv a \pmod{m}\} = \{a + i \cdot m \mid i \in \mathbb{Z}\}$ bezeichnet werden. Diese Äquivalenzklassen heißen aus naheliegenderem Grund auch Restklassen modulo m . Falls es aus dem Zusammenhang klar ersichtlich ist, etwa wenn direkt damit gerechnet wird, bezeichnet $a \pmod{m}$ den kleinsten positiven Repräsentanten der Restklasse. In einer Formel wird dieser Repräsentant zur Verdeutlichung durch Klammersetzung abgegrenzt: $c + (a + b \pmod{m})$. Die Menge aller Restklassen $\mathbb{Z}_m = \{0 \pmod{m}, 1 \pmod{m}, \dots, m - 1 \pmod{m}\}$ wird durch die Definition der Addition als $(a \pmod{m}) + (b \pmod{m}) = a + b \pmod{m}$ und der Multiplikation als $(a \pmod{m}) \cdot (b \pmod{m}) = a \cdot b \pmod{m}$ zum Restklassenring modulo m . Der Restklassenring $(\mathbb{Z}_m, +, \cdot)$ ist genau dann ein Körper, wenn m prim ist; allgemein ist er nur ein kommutativer Ring. Falls es aus dem Zusammenhang ersichtlich ist, steht \mathbb{Z}_m auch für den Ring $(\mathbb{Z}_m, +, \cdot)$. Mit \mathbb{Z}_m^* bezeichnen wir entsprechend die Menge der primen Restklassen modulo m und den primen Restklassenring $(\mathbb{Z}_m^*, +, \cdot)$.

Satz 2.1.1 (Satz von Euklid)

Es gibt unendlich viele Primzahlen.

Beweis: EUKLID beweist dies in Buch IX der Elemente (270 v. Chr.). Eine stärkere Aussage die auch ein Konstruktionsverfahren für immer neue Primzahlen liefert wird in [RU95] gezeigt. \square

Definition 2.1.2

$\pi(n)$ bezeichnet die Anzahl aller Primzahlen zwischen 1 und n , also

$$\pi(n) = \sum_{\substack{p \leq n \\ p \text{ prim}}} 1.$$

Satz 2.1.3 (Großer Primzahlsatz)

Es gilt

$$\pi(n) \approx \frac{n}{\ln(n)}.$$

Beweis: Der große Primzahlsatz geht auf GAUSS zurück. Er wird in [RU95] gezeigt. □

Satz 2.1.4 (Chinesischer Restsatz)

Seien m_1, m_2, \dots, m_k paarweise teilerfremde natürliche Zahlen, also mit $\text{ggT}(m_i, m_j) = 1$ für alle $i, j, i \neq j$. Seien a_1, a_2, \dots, a_k ganze Zahlen. Dann existiert eine Lösung des Systems simultaner Kongruenzen

$$x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2}, \dots, x \equiv a_k \pmod{m_k}$$

und sie ist modulo $m = m_1 \cdot m_2 \cdot \dots \cdot m_k$ eindeutig.

Beweis: [RU95]. □

2.2 Reguläre Sprachen und deterministische endliche Automaten

Die Notationen zu regulären Sprachen und deterministischen endlichen Automaten orientieren sich an [HU88] und [Har78].

Definition 2.2.1 (Alphabet, Wort)

Ein Alphabet Σ ist eine endliche Menge von Buchstaben. Ein Wort w über dem Alphabet Σ ist eine endliche Konkatenation von Buchstaben aus Σ . Das leere Wort, das Wort das aus keinem Buchstaben besteht, wird mit ϵ bezeichnet.

Wie üblich steht $|w|$ für die Länge eines Wortes w , das heißt für die Anzahl der Buchstaben von w . Außerdem wird

$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k \quad \Sigma^+ = \bigcup_{k=1}^{\infty} \Sigma^k \quad \Sigma^{\leq n} = \bigcup_{k=0}^n \Sigma^k$$

geschrieben, wobei Σ^n für die Menge aller Worte mit Länge n steht.

Definition 2.2.2 (Deterministischer endlicher Automat)

Ein deterministischer endlicher Automaten, kurz DFA, wird definiert als $M = (Q, \Sigma, \delta, q_0, F)$. Q ist eine endliche Menge von Zuständen, Σ ein endliches Eingabealphabet, δ die Übergangsfunktion, $q_0 \in Q$ der Startzustand und $F \subseteq Q$ die Menge der akzeptierenden Zustände. Die Übergangsfunktion δ bildet $Q \times \Sigma$ auf Q ab, das heißt $\delta(q, a)$ ist ein Zustand für jeden Zustand $q \in Q$ und jedes Eingabesymbol $a \in \Sigma$.

Ein Zustand $q \in F$ wird akzeptierend und ein Zustand $q \notin F$ verwerfend genannt. Die Funktion δ wird rekursiv auf ganze Eingabeworte erweitert und die von einem DFA M akzeptierte Sprache dann mit $L(M) = \{w \mid \delta(q_0, w) \in F \wedge w \in \Sigma^*\}$ bezeichnet. Die Klasse der von deterministischen endlichen Automaten akzeptieren Sprachen ist die Menge der regulären Sprachen.

Definition 2.2.3

Zwei deterministische endliche Automaten M und M' heißen genau dann äquivalent, wenn $L(M) = L(M')$.

Satz 2.2.4

Das Äquivalenzproblem, die Frage ob zwei DFA M und M' äquivalent sind, ist entscheidbar.

Beweis: [HU88]. □

Satz 2.2.5 (Satz von Nerode)

Die folgenden drei Aussagen sind äquivalent:

- i. Die Menge $L \subseteq \Sigma^*$ wird von einem endlichen Automaten akzeptiert.
- ii. L ist die Vereinigung von einigen der Äquivalenzklassen einer rechts-invarianten Äquivalenzrelation mit endlichem Index.
- iii. Sei die Äquivalenzrelation R_L wie folgt definiert: $xR_Ly \iff \forall z \in \Sigma^* : xz \in L \iff yz \in L$. Dann ist R_L von endlichem Index.

Beweis: [HU88]. □

Die wichtige Konsequenz des Satzes von Nerode ist, dass es für jede reguläre Menge einen bis auf Isomorphien eindeutigen DFA mit einer minimalen Anzahl von Zuständen gibt. Dabei entspricht die Zustandsmenge des minimalen DFA den Äquivalenzklassen der Nerode-Relation R_L .

Definition 2.2.6 (Äquivalente Zustände)

Sei M ein deterministischer endlicher Automat. Seien q und q' zwei Zustände von M . Die Klasse der bezüglich M äquivalenten Zustände wird definiert als

$$q \equiv q' \iff \forall z \in \Sigma^* : \delta(q, z) \in F \iff \delta(q', z) \in F.$$

Gilt $q \equiv q'$ nennen wir q und q' äquivalent, ansonsten unterscheidbar.

Äquivalente Zustände werden, wie in [HU88] gezeigt, bei der Minimierung deterministischer endlicher Automaten zu einem Zustand zusammengefasst. Daraus ergibt sich, dass bei einem minimalen deterministischen endlichen Automaten alle Zustände voneinander unterscheidbar sind.

2.3 Notationen

- \mathbb{N} Menge der natürlichen Zahlen
- \mathbb{Z} Menge der ganzen Zahlen
- \mathbb{R} Menge der reellen Zahlen
- \subseteq Teilmenge
- \subset Echte Teilmenge
- \oplus Symmetrische Differenz zweier Mengen X, Y ($X \oplus Y = (X \cup Y) \setminus (X \cap Y)$)
- \ln Natürlicher Logarithmus

- \log Logarithmus zur Basis 2
- 2^X Potenzmenge der Menge X (die Menge aller Teilmengen von X)
- $|X|$ Kardinalität der Menge X
- $|w|$ Länge des Wortes w
- $f(n) = O(g(n))$ $f(n)$ wächst asymptotisch nicht schneller als $g(n)$
- $f(n) = \Omega(g(n))$ $f(n)$ wächst asymptotisch mindestens so schnell wie $g(n)$
- $f(n) = \text{poly}(n)$ f wächst asymptotisch polynomiell in n
- $a \mid b$ a teilt b
- $\text{ggT}(a, b)$ Größter gemeinsamer Teiler von a und b
- $\text{kgV}(a, b)$ Kleinstes gemeinsames Vielfaches von a und b
- $\text{prob}_D[x]$ Wahrscheinlichkeit für das Eintreffen des Ereignisses x unter der Verteilung D

Kapitel 3

Lernmodelle

In diesem Kapitel werden zwei Lernmodelle für eine besondere Form des algorithmischen Lernens vorgestellt: das *Lernen aus klassifizierten Beispielen*.¹ Hierbei erhält der Lernende von einem Orakel positiv oder negativ klassifizierte Beispiele. Anhand dieser Beispiele versucht der Lernende nun zu ergründen, nach welchen Regeln das Orakel die Beispiele klassifizierte. Griffen wir das in der Einleitung angesprochene Lernen einer Fremdsprache nochmals auf, wären die Beispiele korrekte oder inkorrekte Sätze, aus denen der Lernende die Grammatik der Sprache, also deren Regeln, ableitet.

Das Lernen aus Beispielen findet, obwohl die Theorie mehr negative als positive Ergebnisse aufweist, bereits heute vielfache praktische Anwendung: beispielsweise beim Ausfiltern unerwünschter E-Mail, sogenannter SPAM-Mail, oder der Schriftprobenerkennung. Dies sei aber nur als Anmerkung angeführt, denn auf die Frage, ob und wie praxisnah die Untersuchungen und Betrachtungen in dieser Arbeit sind, wird nicht näher eingegangen.

3.1 Lernen aus Beispielen

Um das Lernen mathematisch vernünftig untersuchen zu können, müssen die Lernmodelle, insbesondere die Frage des erfolgreichen Lernens, präzise definiert werden. Darauf aufbauend können dann die Lernalgorithmen hinsichtlich ihrer Korrektheit und ihrer Effizienz bewertet werden.

Definition 3.1.1 (Konzept, Konzeptklasse, Hypothese, Hypothesenklasse)

Gegeben sei eine nicht notwendigerweise endliche Menge Σ . Die Menge Σ^* wird als Universum, ein Element $w \in \Sigma^*$ als Beispiel bezeichnet. Ein Konzept c über Σ ist eine Teilmenge von Σ^* . Eine Konzeptklasse \mathcal{C} ist eine Menge von Konzepten. Eine Hypothese h über Σ ist ein Konzept über Σ . Eine Hypothesenklasse \mathcal{H} ist eine Menge von Hypothesen.

Definition 3.1.2 (Klassifiziertes Beispiel)

Gegeben sei ein Konzept $c \in \mathcal{C}$. Ein Beispiel $w \in \Sigma^*$ wird ein gemäß c klassifiziertes Beispiel genannt, wenn bekannt ist, ob $w \in c$ gilt. Falls der Zusammenhang ersichtlich ist, wird auch kurz von einem klassifizierten Beispiel oder noch kürzer von einem Beispiel gesprochen. Falls $w \in c$ wird w ein positives Beispiel, ansonsten ein negatives Beispiel genannt.

Der Einfachheit halber übernehmen wir noch zwei Begriffe aus der Terminologie der formalen Sprachen und sprechen künftig davon, dass ein Konzept c ein Beispiel w akzeptiert (verwirft), wenn $w \in c$ ($w \notin c$).

¹In der Literatur wird zuweilen für das passive Lernen aus klassifizierten Beispielen auch der Begriff *Lernen durch Beobachtung* gebraucht.

Im Allgemeinen wird in dieser Arbeit die folgende Lernsituation betrachtet: Ein Lernalgorithmus versucht, ein *unbekanntes* Konzept aus einer *bekannt* Konzeptklasse zu lernen. Durch Befragung des Orakels² erhält er gemäß dem zu lernenden Konzept als positiv oder negativ klassifizierte Beispiele. Als Ergebnis gibt der Lernalgorithmus eine Hypothese aus einer, nicht unbedingt mit der Konzeptklasse identischen, Hypothesenklasse aus.

Ein zentraler Aspekt dieser Lernsituation ist die Art und Weise, wie die klassifizierten Beispiele erzeugt werden. Hiernach werden zwei Modelle unterschieden: passives Lernen, bei dem der Lernende keinen Einfluss auf die Wahl der Beispiele hat, und aktives Lernen, bei dem der Lernende gezielte Fragen stellen darf. Ein anderer zentraler Aspekt ist, ob die exakte Identifikation des gesuchten Konzeptes, die in der Praxis nicht unbedingt möglich oder notwendig ist, gefordert wird. Weitere Aspekte führen letztlich zu einer Unzahl verschiedener Lernmodelle für die vorab benannte Lernsituation. Zwei der wichtigsten Lernmodelle sollen vorgestellt werden.

In dem wohl bekanntesten passiven Lernmodell, dem *PAC-Lernen*, erhält der Lernende vom Orakel klassifizierte Beispiele, die zufällig auf einer unbekanntem Wahrscheinlichkeitsverteilung erzeugt wurden. Es wird vom Lernenden keine exakte Identifikation gefordert, sondern nur, dass seine Hypothese für die gegebene Verteilung mit einer hohen Wahrscheinlichkeit approximativ richtig ist. Daraus ergeben sich die Fragen, wieviele Beispiele ein Lernalgorithmus anfordern muss, damit die Wahrscheinlichkeit für ein erfolgreiches Lernen hoch ist, und in welchen Fällen effizientes Lernen möglich oder ausgeschlossen ist. Es wird das enttäuschende Ergebnis gezeigt werden, dass etwa die Konzeptklasse der regulären Mengen im PAC-Modell (unter gewissen kryptographischen Annahmen) nicht effizient gelernt werden kann.

Beim *Lernen mit Element- und Äquivalenzfragen*, einem aktiven Lernmodell, wird dem Lernenden gestattet, zwei Arten von Fragen zu stellen. Eine Elementfrage, um ein bestimmtes Beispiel zu klassifizieren, oder eine Äquivalenzfrage “ist die augenblickliche Hypothese richtig”, die das Orakel entweder mit “ja” oder einem Gegenbeispiel, einem Beispiel, das durch die Hypothese anders als durch das Orakel klassifiziert wird, beantwortet. In diesem Modell wird die exakte Identifikation gefordert. Trotz dieser stärkeren Anforderung wird gezeigt werden, dass sogar die Konzeptklasse der regulären Sprachen mit der Hypothesenklasse der deterministischen endlichen Automaten, im Gegensatz zum PAC-Modell, effizient erlernt werden kann.

Auch wenn das aktive Lernmodell eine größere Lernkraft aufweist, ist das passive Lernmodell in der Praxis wichtiger, da das aktive Modell höhere Anforderungen an zumeist menschliche Experten bei der Beantwortung der Fragen stellt [An93].

3.2 PAC-Lernen

Das *probably approximately correct distribution-free learning*, kurz PAC-Lernen³, geht auf VALIANT [Val84] zurück. Es ist ein passives Lernmodell, der Lernalgorithmus hat also keinen Einfluss auf die Wahl der Beispiele. Das genaue Modell ist wie folgt:

Gegeben seien eine unbekannte Verteilung D , die Konzeptklasse \mathcal{C} und die Hypothesenklasse \mathcal{H} . Die Konzeptklasse \mathcal{C} sei bekannt. Der Lernalgorithmus versucht zu einem unbekanntem Konzept $c \in \mathcal{C}$ eine Hypothese $h \in \mathcal{H}$ aufzustellen, die mit einer hohen Wahrscheinlichkeit nur einen kleinen Fehler aufweist. Dazu darf der Lernalgorithmus klassifizierte Beispiele anfordern, die gemäß der Verteilung D zufällig erzeugt wurden.

²Das Orakel wird nicht formal definiert. Es entspricht einem Lehrer, der seinem Schüler erlaubt ganz bestimmte Fragen zu stellen und diese auch immer korrekt beantwortet.

³In der Literatur, wird das PAC-Lernen auch als *distribution-free learning* [KV89] oder *uniformly learnable* [BEHW89] bezeichnet.

Einen Fehler macht die Hypothese, wenn sie ein Beispiel anders als das Orakel klassifiziert: falsch sind alle Beispiele aus der symmetrischen Differenz $c \oplus h = (c \cup h) \setminus (c \cap h)$. Der Fehler sollte dabei aber nicht unter der Gleichverteilung, sondern unter der Verteilung gemessen werden, unter der auch die Beispiele erzeugt wurden, da sonst auch "gute" Hypothesen als stark fehlerhaft erscheinen können. Außerdem soll der Algorithmus nicht für eine "schlechte" Verteilung bestraft werden.

Definition 3.2.1 (Fehler der Hypothese)

Gegeben seien ein Konzept $c \in \mathcal{C}$, eine Hypothese $h \in \mathcal{H}$ und eine Verteilung D auf dem Universum. Der Fehler der Hypothese wird definiert als

$$\text{fehler}_D(c, h) = \sum_{w \in c \oplus h} D(w)$$

Beispiel 3.2.2

Gegeben sei die Konzeptklasse $\text{BOOLEAN}_n = \{c \mid c \subseteq \{0, 1\}^n\}$ für ein $n > 0$. Sei die Verteilung D , unter der die Beispiele erzeugt werden, als

$$D(w) = \begin{cases} \frac{1}{2^{n-1}} & \text{falls } w \text{ mit } 1 \text{ beginnt} \\ 0 & \text{sonst} \end{cases}$$

definiert. Angenommen, das zu lernende Konzept c enthält nur Beispiele die mit 0 beginnen. Dann werden nur negative Beispiele erzeugt. Eine intuitiv als "gut" empfundene Hypothese h ist also die leere Menge. Doch wird ihr Fehler unter der Gleichverteilung D_u bewertet, ergibt sich, da $c \oplus h = c$, $\text{fehler}_{D_u}(c, h) = |c| \cdot \frac{1}{2^n}$, also im schlimmsten Fall ein Fehler von $\frac{1}{2}$, statt einem Fehler von 0.

Selbst eine "gute" Verteilung kann, wenn auch mit geringer Wahrscheinlichkeit, "schlechte" Beispielmengen liefern. Daher darf nicht verlangt werden, dass der Lernalgorithmus immer eine gute Hypothese liefert. Stattdessen wird erlaubt, dass auf einem Teil der möglichen Beispielmengen eine schlechte Hypothese abgegeben wird. Zur Formalisierung dieses Zieles werden der Vertrauensparameter δ und der Fehlerparameter ϵ eingeführt, so dass $\text{fehler}_D(c, h) \leq \epsilon$ mit einer Wahrscheinlichkeit von $1 - \delta$ gefordert wird.

Definition 3.2.3 (PAC-Algorithmus)

Gegeben seien die Konzeptklasse \mathcal{C} und die Hypothesenklasse \mathcal{H} . Ein Lernalgorithmus A wird genau dann ein PAC-Algorithmus für \mathcal{C} durch \mathcal{H} genannt, wenn für alle Fehlerparameter $\epsilon \in (0, 1]$, für alle Vertrauensparameter $\delta \in (0, 1]$, für alle Konzepte $c \in \mathcal{C}$ und für alle Verteilungen D gilt, dass A eine Hypothese $h \in \mathcal{H}$ mit

$$\text{prob}_D \left[\text{fehler}_D(c, h) \leq \epsilon \right] \geq 1 - \delta$$

ausgibt. Existiert ein PAC-Algorithmus für \mathcal{C} durch \mathcal{H} , wird die Konzeptklasse \mathcal{C} PAC-lernbar durch \mathcal{H} genannt; falls $\mathcal{H} = \mathcal{C}$ wird der Zusatz "durch \mathcal{H} " auch fallengelassen.

Die Grundstruktur eines Lernalgorithmus im PAC-Modell ist mit Algorithmus 3.2.1 auf Seite 13 angegeben. Damit Algorithmus 3.2.1 aber zu einem PAC-Algorithmus wird, muss er sich, nach Definition 3.2.3, der (unbekannten) Verteilung, dem Fehlerparameter ϵ und dem Vertrauensparameter δ anpassen. Im Allgemeinen gilt, dass eine größere Beispielmenge mit höherer Zuverlässigkeit zu einer Hypothese mit geringem Fehler führt. Damit der Lernalgorithmus eine Hypothese mit einem Fehler von höchstens ϵ und einer Wahrscheinlichkeit von mindestens $1 - \delta$ ausgibt, muss er also eine genügend große Zahl an Beispielen anfordern. Die minimale Anzahl an Beispielen, die ein PAC-Algorithmus anfordern muss, um \mathcal{C} zu lernen, wird die Beispielskomplexität der Konzeptklasse \mathcal{C} genannt.

Eingabe: Vertrauensparameter δ , Fehlerparameter ϵ

Ausgabe: Hypothese $h \in \mathcal{H}$

- 1: $s \leftarrow s(\epsilon, \delta)$ /* Bestimme die Anzahl der anzufordernden Beispiele */
- 2: **for** $i = 1, 2, \dots, s$ **do**
- 3: Fordere ein klassifiziertes Beispiel $w_i \in \Sigma^*$ an
- 4: /* Das Beispiel w_i wird vom Orakel mit “ja” klassifiziert, wenn es */
- 5: /* zum gesuchten Konzept gehört, ansonsten mit “nein” */
- 6: **end for**
- 7: Bestimme eine Hypothese $h \in \mathcal{H}$

Algorithmus 3.2.1: Lernalgorithmus für \mathcal{C}

Definition 3.2.4 (Beispielskomplexität einer Konzeptklasse)

Sei A ein PAC-Algorithmus für die Konzeptklasse \mathcal{C} . Für $\epsilon \in (0, 1]$, $\delta \in (0, 1]$ sei $s_A(\epsilon, \delta)$ die Anzahl der von A angeforderten Beispiele für Fehlerparameter ϵ und Vertrauensparameter δ . Die Beispielskomplexität der Konzeptklasse \mathcal{C} ist definiert durch

$$\text{beispiel}_{\mathcal{C}}(\epsilon, \delta) = \min\{s_A(\epsilon, \delta) \mid A \text{ ist ein PAC-Algorithmus für } \mathcal{C}\}$$

Algorithmus 3.2.1 weist allerdings noch einen Makel auf. Da er seine Hypothese auch unabhängig von den Beispielen aufstellen kann, ist, selbst wenn er genügend Beispiele anfordert, nicht gesichert, dass seine Hypothese “gut” ist. Daher muss noch eine weitere Bedingung an den Lernalgorithmus gestellt werden: Er darf nur eine konsistente Hypothese aufstellen, das heißt eine Hypothese, die die Beispiele genau so klassifiziert, wie sie vom Orakel klassifiziert wurden. Diese Anforderung erscheint “natürlich”, da bei “vernünftigen” Konzeptklassen trivialerweise zu jeder inkonsistenten Hypothese h eine konsistente Hypothese h' mit geringerem Fehler gefunden werden kann.

Definition 3.2.5 (Konsistente Hypothese, Konsistenzproblem)

Sei $c \in \mathcal{C}$ das zu lernende Konzept. Eine Hypothese $h \in \mathcal{H}$ wird genau dann konsistent mit den klassifizierten Beispielen w_1, w_2, \dots, w_s genannt, wenn

$$c \text{ akzeptiert } w_i \iff h \text{ akzeptiert } w_i$$

für $i = 1, 2, \dots, s$ gilt. Als Konsistenzproblem wird das Problem der Konstruktion einer konsistenten Hypothese bezeichnet.

Bisher wurde noch nicht gefordert, dass ein PAC-Algorithmus im Allgemeinen berechenbar sein muss. Effiziente PAC-Algorithmen werden erst in Abschnitt 3.2.3 eingeführt. In diesem Abschnitt wird sich auch ein starker Zusammenhang zwischen dem Konsistenzproblem und der Existenz eines effizienten PAC-Algorithmus zeigen.

3.2.1 Obere Schranke der Beispielskomplexität

Es soll nun eine obere Schranke für die Beispielskomplexität endlicher Konzeptklassen gezeigt werden.

Satz 3.2.6 (Anzahl anzufordernder Beispiele)

Sei \mathcal{C} eine endliche Konzeptklasse. Lernalgorithmus 3.2.1 wird zu einem PAC-Algorithmus für \mathcal{C} mit Fehlerparameter $\epsilon \in (0, 1]$ und Vertrauensparameter $\delta \in (0, 1]$ wenn er mindestens

$$\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{\ln(|\mathcal{C}|)}{\epsilon}$$

Beispiele anfordert und eine mit der Beispielmenge konsistente Hypothese $h \in \mathcal{C}$ ausgibt.

Beweis: Wir folgen dem Beweis aus [Schn97] und [BEHW89]. Sei h die von Algorithmus 3.2.1 ausgegebene Hypothese. Gemäß Voraussetzung ist sie konsistent. Sei weiter $s = s(\epsilon, \delta)$ und $S = \{w_1, w_2, \dots, w_s\}$ die Menge der klassifizierten Beispiele. Damit der Lernalgorithmus ein PAC-Algorithmus wird, muss für jedes Konzept $c \in \mathcal{C}$ und jede Wahrscheinlichkeitsverteilung D

$$\text{prob}_D[\text{fehler}_D(c, h) \leq \epsilon] \geq 1 - \delta$$

gelten. Seien das gesuchte Konzept $c \in \mathcal{C}$ und die Verteilung D gegeben. Angenommen, wir wählen eine beliebige Hypothese $h' \in \mathcal{C}$ mit $\text{fehler}_D(h', c) > \epsilon$, dann ist die Wahrscheinlichkeit, dass ein zufällig gewähltes Beispiel $w_i \in S$ konsistent mit h' ist, durch $1 - \epsilon$ nach oben beschränkt. Also ergibt sich für h'

$$\text{prob}[h' \text{ konsistent mit allen } w_i \in S] \leq (1 - \epsilon)^s.$$

Die Wahrscheinlichkeit, dass die von Algorithmus 3.2.1 gewählte Hypothese h einen zu großen Fehler aufweist, ist daher

$$\text{prob}[\text{fehler}_D(c, h) > \epsilon] \leq \sum_{h' \in \mathcal{C}} \text{prob}[h' \text{ konsistent mit allen } w_i \in S] \leq |\mathcal{C}| \cdot (1 - \epsilon)^s.$$

Wird $s \geq \frac{1}{\epsilon} \ln(\frac{1}{\delta}) + \frac{\ln(|\mathcal{C}|)}{\epsilon}$ gewählt, dann gilt $|\mathcal{C}| \cdot (1 - \epsilon)^s \leq \delta$. Also stellt Algorithmus 3.2.1 eine konsistente Hypothese auf, die mit einer Wahrscheinlichkeit von $1 - \delta$ höchstens einen Fehler von ϵ aufweist. \square

Satz 3.2.6 gilt sogar für alle Lernalgorithmen. Das heißt, jeder Lernalgorithmus der entsprechend viele Beispiele anfordert und eine konsistente Hypothese ausgibt ist ein PAC-Algorithmus [BEHW89]. Außerdem ergibt sich aus Satz 3.2.6 auch eine erste obere Schranke für die Beispielskomplexität endlicher Konzeptklassen.

Satz 3.2.7 (Obere Schranke der Beispielskomplexität endlicher Konzeptklassen)

Sei \mathcal{C} eine endliche Konzeptklasse über Σ . Es gilt

$$\text{beispiel}_{\mathcal{C}}(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{\ln(|\mathcal{C}|)}{\epsilon}\right).$$

Beweis: Folgt direkt aus Satz 3.2.6. \square

Es werden nun einige konkrete Konzeptklassen⁴ eingeführt, um die Ergebnisse dieses Abschnittes zu verdeutlichen. So kann mit Satz 3.2.6 sofort eine obere Schranke der Beispielskomplexität dieser Klassen nachgewiesen werden.

Definition 3.2.8 (Konzeptklassen für aussagenlogische Formeln)

Für eine aussagenlogische Formel α über den Literalen $\{x_1, \neg x_1, \dots, x_n, \neg x_n\}$ definieren wir die Menge der erfüllenden Wahrheitsbelegungen $\text{wahr}(\alpha)$ als

$$\text{wahr}(\alpha) = \{x \mid x \in \{0, 1\}^n \wedge \alpha(x) = 1\}.$$

Darauf aufbauend definieren wir die folgenden Konzeptklassen:

⁴Es erfolgt hier eine willkürliche Einschränkung auf Konzeptklassen, die vornehmlich auf aussagenlogischen Formeln beruhen. Es gibt natürlich viel mehr Konzeptklassen, wie Rechtecke, Schaltkreise, symmetrische Funktionen, Schwellwertfunktionen oder Entscheidungsbäume [Schn97], [BEHW89], [EHKV89].

- i. Die Konzeptklasse $k\text{-CNF}_n$ besitzt für jede aussagenlogische Formel α in konjunktiver Normalform, mit höchstens k Literalen pro Klausel, das Konzept $\text{wahr}(\alpha)$.
- ii. Die Konzeptklasse $k\text{-DNF}_n$ ist die zur Konzeptklasse $k\text{-CNF}_n$ analoge Konzeptklasse für Formeln in disjunktiver Normalform, wobei maximal k Literale in jedem Term auftreten.
- iii. Die Konzeptklasse $k\text{-Klausel-CNF}_n$ besitzt für jede aussagenlogische Formel α in konjunktiver Normalform mit höchstens k Klauseln das Konzept $\text{wahr}(\alpha)$.
- iv. Die Konzeptklasse $k\text{-Term-DNF}_n$ ist die zur Konzeptklasse $k\text{-Klausel-CNF}$ analoge Konzeptklasse für Formeln in disjunktiver Normalform mit höchstens k Termen.
- v. Die Konzeptklasse CNF_n besitzt für jede Formel in konjunktiver Normalform das Konzept $\text{wahr}(\alpha)$.
- vi. Die Konzeptklasse DNF_n ist die zur Konzeptklasse CNF_n analoge Konzeptklasse für Formeln in disjunktiver Normalform.

Definition 3.2.9 (Weitere Konzeptklassen)

Wir definieren die folgenden Konzeptklassen:

- i. Die Konzeptklasse BOOLEAN_n besteht aus allen Teilmengen von $\{0, 1\}^n$.
- ii. Die Konzeptklasse $\text{DFA}_{n,\Sigma}$ besitzt für jeden deterministischen endlichen Automaten M mit höchstens n Zuständen und dem Eingabealphabet Σ die Menge $L(M) \cap \Sigma^{\leq n}$ als Konzept.

Satz 3.2.10 (Obere Schranken für die Beispielskomplexität konkreter Klassen)

Es gilt:

- i. $\text{beispiel}_{\mathcal{C}}(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n^k}{\epsilon}\right)$ für $k\text{-CNF}_n$ und $k\text{-DNF}_n$ mit festem k .
- ii. $\text{beispiel}_{\mathcal{C}}(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n}{\epsilon}\right)$ für $k\text{-Klausel-CNF}_n$ und $k\text{-Term-DNF}_n$ mit festem k .
- iii. $\text{beispiel}_{\mathcal{C}}(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{2^n}{\epsilon}\right)$ für CNF_n , DNF_n und BOOLEAN_n .
- iv. $\text{beispiel}_{\mathcal{C}}(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n \cdot |\Sigma| \cdot \log_2(n)}{\epsilon}\right)$ für $\mathcal{C} = \text{DFA}_{n,\Sigma}$.

Beweis: Der Beweis folgt [Schn97]. Aufgrund Satz 3.2.6 reicht es, die Kardinalität der Konzeptklassen zu berechnen.

- i. $|k\text{-CNF}_n| = |k\text{-DNF}_n| = O(2^{n^k})$ für festes k
- ii. $|k\text{-Klausel-CNF}_n| = |k\text{-Term-DNF}_n| \leq 2^{2nk}$ für festes k
- iii. $|\text{CNF}_n| = |\text{DNF}_n| = |\text{BOOLEAN}_n| = 2^{2^n}$
- iv. $|\text{DFA}_{n,\Sigma}| \leq n^{|\Sigma| \cdot n} \cdot 2^n$

□

Die Ergebnisse aus Satz 3.2.10 (i) bis (iii) sind optimal in dem Sinne, dass sie der unteren Schranke in Satz 3.2.17 entsprechen.

3.2.2 VC-Dimension und Beispielskomplexität

In Satz 3.2.6 wurde bereits gezeigt, dass eine endliche Konzeptklasse \mathcal{C} durch jeden Lernalgorithmus, der

$$s(\epsilon, \delta) \geq \frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{\ln(|\mathcal{C}|)}{\epsilon}$$

Beispiele anfordert und eine konsistente Hypothese ausgibt, mit einem Fehler von höchstens ϵ und einer Zuverlässigkeit von $1 - \delta$ gelernt wird. Diese Konstruktion ist nicht direkt auf unendliche Konzeptklassen anwendbar. Daher werden in diesem Abschnitt Bedingungen für die Lernbarkeit (im PAC-Sinne) einer Konzeptklasse, endlich oder unendlich, ausgearbeitet. Dazu wird zuerst die Vapnik-Chervonenkis-Dimension, kurz VC-Dimension, einer Konzeptklasse eingeführt.

In Satz 3.2.15 wird gezeigt, dass eine Konzeptklasse \mathcal{C} genau dann PAC-lernbar ist, wenn die VC-Dimension von \mathcal{C} endlich ist. Außerdem wird gezeigt, dass es bei endlicher VC-Dimension von \mathcal{C} einen PAC-Algorithmus gibt, der \mathcal{C} mit einem Fehler von höchstens ϵ und einer Zuverlässigkeit von $1 - \delta$ lernt und dazu nicht mehr als

$$s(\epsilon, \delta) \geq \max\left(\frac{4}{\epsilon} \ln\left(\frac{2}{\delta}\right), \frac{8 \cdot \text{VC}(\mathcal{C})}{\epsilon} \ln\left(\frac{13}{\epsilon}\right)\right)$$

Beispiele anfordern muss.

Definition 3.2.11 (Vapnik-Chervonenkis-Dimension)

Sei \mathcal{C} eine Konzeptklasse über Σ . Sei $S \subseteq \Sigma^*$. Wir sagen, dass \mathcal{C} die Menge S zertrümmert, falls jede Teilmenge von S ein Konzept ist, das heißt, falls

$$2^S = \{S \cap c \mid c \in \mathcal{C}\}$$

gilt. Die VC-Dimension von \mathcal{C} , geschrieben als $\text{VC}(\mathcal{C})$, ist die Kardinalität der größten Menge $S \subseteq \Sigma^*$, die von \mathcal{C} zertrümmert wird. Falls es kein Maximum der Kardinalität von S gibt, ist $\text{VC}(\mathcal{C})$ unendlich.

Die VC-Dimension der Konzeptklasse \mathcal{C} ist, informell und grob gesprochen, die Zahl der voneinander unabhängigen Entscheidungen, die die Konzeptklasse \mathcal{C} treffen kann, so dass erst bei Kenntniss aller Entscheidungen nur ein einziges konsistentes Konzept übrigbleibt. Somit würde man intuitiv für die Konzeptklasse BOOLEAN_n eine VC-Dimension von 2^n erwarten, da es genau $|\{0, 1\}^n| = 2^n$ Worte gibt und die Konzeptklasse für jedes Wort $w \in \{0, 1\}^n$ entscheiden kann, ob sie es akzeptieren oder verwerfen will.

Beispiel 3.2.12

Es gilt $\text{VC}(\text{BOOLEAN}_n) = 2^n$. Einerseits zertrümmert BOOLEAN_n die Menge $S = \{0, 1\}^n$, da BOOLEAN_n für jede Teilmenge von S ein Konzept besitzt. Andererseits ist S auch die größte von BOOLEAN_n zertrümmerte Menge, da eine Menge S' mit $|S'| \geq 2^n$ wenigstens ein Element enthalten muss, das in BOOLEAN_n nicht enthalten ist.

Satz 3.2.13 (VC-Dimension einiger konkreter Konzeptklassen)

Es gilt:

- i. $\text{VC}(\mathcal{C}) = \Theta(n^k)$ für $k\text{-CNF}_n$ und $k\text{-DNF}_n$ mit festem k .
- ii. $\text{VC}(\mathcal{C}) = \Theta(n)$ für $k\text{-Klausel-CNF}_n$ und $k\text{-Term-DNF}_n$ mit festem k .
- iii. $\text{VC}(\mathcal{C}) = 2^n$ für CNF_n , DNF_n und BOOLEAN_n .

Beweis: EHRENFUCHT, HAUSSLER, KEARNS und VALIANT zeigen (i) und (ii) in [EHKV89]. In Beispiel 3.2.12 wurde bereits angesprochen, dass $VC(\text{BOOLEAN}_n) = 2^n$ gilt. Wegen $\text{CNF}_n = \text{DNF}_n = \text{BOOLEAN}_n$ folgt (iii). \square

Einer der wichtigsten Sätze für das PAC-Lernen, Satz 3.2.15, stellt einen Zusammenhang zwischen der VC-Dimension einer Konzeptklasse und der Beispielskomplexität der Konzeptklasse her.

Definition 3.2.14 (Triviale Konzeptklassen)

Sei X das Universum. Eine Konzeptklasse \mathcal{C} mit $\mathcal{C} \subseteq 2^X$ wird *trivial* genannt, falls \mathcal{C} nur aus einem Konzept oder aus zwei disjunkten Konzepten c, c' mit $c \cup c' = X$ besteht.

Eine triviale Konzeptklasse kann mit höchstens einem Beispiel gelernt werden. Daher wird dieser Fall im weiteren ausgeklammert.

Satz 3.2.15

Sei \mathcal{C} eine nichttriviale, wohlerzogene⁵ Konzeptklasse. Seien $\delta, \epsilon \in \mathbb{R}$.

i. \mathcal{C} ist PAC-lernbar durch \mathcal{C} genau dann, wenn $VC(\mathcal{C}) < \infty$ gilt.

ii. Wenn $VC(\mathcal{C}) < \infty$ ist, dann gilt

(a) für $\epsilon < 1$ und $\delta < 1$, dass jeder konsistente Lernalgorithmus der wenigstens

$$s(\epsilon, \delta) \geq \max\left(\frac{4}{\epsilon} \log\left(\frac{2}{\delta}\right), \frac{8 \cdot VC(\mathcal{C})}{\epsilon} \log\left(\frac{13}{\epsilon}\right)\right)$$

Beispiele anfordert ein PAC-Algorithmus für \mathcal{C} ist, und

(b) für $\epsilon \leq \frac{1}{8}$ und $\delta \leq \frac{1}{100}$, dass jeder PAC-Algorithmus für \mathcal{C} durch \mathcal{H} (für eine beliebige Hypothesenklasse \mathcal{H}) mindestens

$$s(\epsilon, \delta) \geq \max\left(\frac{1 - \epsilon}{\epsilon} \ln\left(\frac{1}{\delta}\right), \frac{VC(\mathcal{C}) - 1}{32\epsilon}\right)$$

Beispiele anfordern muss.

Beweis: BLUMER ET AL. beweisen in [BEHW89] diesen Satz, wobei Ihre untere Schranke in (ii) (b) etwas schlechter ist. Die hier vorgestellte Schranke basiert auf einer Verbesserung durch EHRENFUCHT, HAUSSLER, KEARNS und VALIANT [EHKV89]. \square

Macht man sich klar, dass für eine endliche Konzeptklasse \mathcal{C} die VC-Dimension mit $VC(\mathcal{C}) \leq \log(|\mathcal{C}|)$ abgeschätzt werden kann (wir brauchen $2^{|\mathcal{C}|}$ Konzepte um S zu zertrümmern), dann erhält man mit Satz 3.2.6 eine asymptotisch ähnliche Schranke wie mit Satz 3.2.15. Dabei ist Satz 3.2.6 in vielen Fällen aber leichter anwendbar, da man die VC-Dimension der Konzeptklasse \mathcal{C} nicht berechnen muss, und liefert wenn $VC(\mathcal{C}) \approx \log(|\mathcal{C}|)$ eine etwas bessere Beispielskomplexität. Andererseits ist in einigen Fällen die Kardinalität von \mathcal{C} ein zu ungenaues Maß, um daraus die Beispielskomplexität zu berechnen.

BLUMER ET AL. liefern hierzu als Beispiel die Konzeptklasse \mathcal{C} der “Linear Separators” (die Klasse besitzt Konzepte für jede Schwellwertfunktion) über n Variablen mit Eingaben aus

⁵Die Wohlerzogenheit fordert für die Konzeptklasse gewisse Messbarkeiten, wie etwa die Messbarkeit der symmetrischen Differenz. Wir verzichten darauf, die Wohlerzogenheit formal zu definieren (siehe dazu Anhang A1 in [BEHW89]) und merken nur kurz an, dass die in der Praxis verwandten Konzeptklassen alle wohlerzogen sind.

$\{0, 1\}^n$. Die Größe der Konzeptklasse kann durch $2^{n(n-1)/2} \leq |C| \leq 2^{n^2}$ abgeschätzt werden. Wird dies in Satz 3.2.6 eingesetzt, ergeben sich

$$s(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right) + \frac{n^2}{\epsilon}\right)$$

anzufordernde Beispiele. Andererseits hat C eine VC-Dimension von $VC(C) = n + 1$. Mit Satz 3.2.15 erhält man nur

$$s(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right) + \frac{n}{\epsilon} \log\left(\frac{1}{\epsilon}\right)\right)$$

anzufordernde Beispiele. Für festes ϵ und δ ist dies eine Reduktion der Beispielskomplexität von $O(n^2)$ auf $O(n)$ [BEHW89].

Satz 3.2.16 (Obere und untere Schranke für die Beispielskomplexität)

Gegeben seien eine Konzeptklasse C und eine Hypothesenklasse \mathcal{H} mit $C \subseteq \mathcal{H}$. Die Beispielskomplexität von C durch \mathcal{H} mit einem Fehler von höchstens ϵ und einer Zuverlässigkeit von $1 - \delta$ ist beschränkt durch

$$\text{beispiel}_C(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{\ln(|\mathcal{H}|)}{\epsilon}\right) \quad \text{bzw.} \quad \text{beispiel}_C(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{VC(\mathcal{H})}{\epsilon} \ln\left(\frac{1}{\epsilon}\right)\right)$$

$$\text{beispiel}_C(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{VC(C)}{\epsilon}\right)$$

Beispiele.

Beweis: Aus der Definition der VC-Dimension folgt $VC(C) \leq VC(\mathcal{H})$ für $C \subseteq \mathcal{H}$. Das Ergebnis ergibt sich dann als Konsequenz aus Satz 3.2.6 und Satz 3.2.15. \square

Wird eine Konzeptklasse C durch eine größere Hypothesenklasse \mathcal{H} gelernt, erfordert dies, nach Satz 3.2.16, eine größere Anzahl an Beispielen. Es gibt aber Konzeptklassen, für die der Wechsel zu einer größeren Hypothesenklasse notwendig ist. So existiert für das Lernen von k -Term-DNF $_n$ durch k -Term-DNF $_n$ zwar ein Algorithmus, doch läuft der nicht in polynomieller Zeit. Daher nutzt der beste bekannte effiziente PAC-Algorithmus für k -Term-DNF $_n$ die Hypothesenklasse k -CNF $_n$. Dies erfordert

$$\Omega\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n^k}{\epsilon}\right) \quad \text{statt} \quad \Omega\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n}{\epsilon}\right)$$

Beispiele. Also unterscheidet sich die Zahl der für effizientes Lernen angeforderten Beispiele um einen Faktor von $O(n^{k-1})$ von der Beispielskomplexität von k -Term-DNF $_n$ [EHKV89].

Satz 3.2.17 (Untere Schranken für die Beispielskomplexität konkreter Klassen)

Es gilt:

- i. $\text{beispiel}_C(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n^k}{\epsilon}\right)$ für k -CNF $_n$ und k -DNF $_n$ mit festem k .
- ii. $\text{beispiel}_C(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n}{\epsilon}\right)$ für k -Klausel-CNF $_n$ und k -Term-DNF $_n$ mit festem k .
- iii. $\text{beispiel}_C(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{2^n}{\epsilon}\right)$ für CNF $_n$, DNF $_n$ und BOOLEAN $_n$.

Beweis: Folgt direkt aus Satz 3.2.13 und Satz 3.2.15 \square

3.2.3 Effiziente PAC-Algorithmen

Dieses Kapitel ist Betrachtungen zur Effizienz der in Definition 3.2.3 eingeführten PAC-Algorithmen gewidmet. Bisher war diese Frage ausgeklammert. Es wird sich zeigen, dass effizientes PAC-Lernen in einigen Fällen nicht möglich ist, nämlich dann, wenn das Konsistenzproblem aus Definition 3.2.5 nicht effizient lösbar ist.

So hat GOLD etwa gezeigt, dass die Berechnung des minimalen deterministischen Automaten, der konsistent mit einer Beispielmenge ist, NP-vollständig ist. Solange die allgegenwärtige Annahme $P \neq NP$ gilt, sind reguläre Mengen daher nicht in polynomieller Zeit erlernbar [Gol78]. Dieses Ergebnis wurde durch LI und VAZIRANI noch dahingehend verbessert, dass auch die Konstruktion eines DFA, der um $9/8$ größer als der kleinste konsistente DFA ist, immer noch NP-vollständig ist [Ke90]. Allerdings ist das Ergebnis repräsentationsabhängig in dem Sinne, dass es nur für das Lernen regulärer Mengen durch die Hypothesenklasse der deterministischen endlichen Automaten gilt. Es trifft keine Aussage über andere Hypothesenklassen. Ein anderes repräsentationsabhängiges Resultat etwa ist die Nichtlernbarkeit von k -Term-DNF durch k -Term-DNF; dagegen sind k -Term-DNF durch k -CNF effizient erlernbar.

Definition 3.2.18 (Parametrisierte Konzeptklasse)

Eine parametrisierte Konzeptklasse \mathcal{C} über Σ ist eine Folge $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ von Konzeptklassen über Σ , wobei für jedes n und für jedes $c \in \mathcal{C}_n$ gilt, dass $c \in \Sigma^{\leq n}$ ist. Eine parametrisierte Hypothesenklasse ist eine parametrisierte Konzeptklasse.

Die Konzeptklassen werden in Definition 3.2.18 mit einer Struktur versehen, um asymptotische Aussagen über die Laufzeit eines Lernalgorithmus treffen zu können. Falls es aus dem Zusammenhang ersichtlich ist, wird die Bezeichnung parametrisiert auch fallengelassen. Zudem wird gesagt, dass $c \in \mathcal{C}$ falls ein $n \in \mathbb{N}$ existiert, so dass $c \in \mathcal{C}_n$.

Da das Ziehen eines Beispiels eine Zeiteinheit kostet ist die Zahl der vom Lernalgorithmus angeforderten Beispiele ein wichtiges Kriterium der Effizienz. Diese Zahl hängt von dem Fehlerparameter ϵ , dem Vertrauensparameter δ und der Eingabelänge n ab. Das Ziel ist es, Lernalgorithmen zu finden, deren Zahl an angeforderten Beispielen polynomiell in den Parametern $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ und n ist. Dabei ist die Zeit, die der Lernalgorithmus benötigt, um die Hypothese zu konstruieren und auszugeben, noch nicht berücksichtigt. Um die Beschreibungskomplexität einer Hypothese zu fassen, wird die Namensfunktion einer Hypothese eingeführt.

Definition 3.2.19 (Effiziente Namensfunktion)

Sei $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ eine parametrisierte Konzeptklasse über Σ . Die Funktion

$$\text{name}_{\mathcal{C}} : \bigcup_{n \in \mathbb{N}} \mathcal{C}_n \rightarrow 2^{\{0,1\}^*}$$

heißt Namensfunktion für \mathcal{C} genau dann, wenn für alle $n \in \mathbb{N}$ und alle $c, c' \in \mathcal{C}_n$ mit $c \neq c'$ gilt, dass $\text{name}_{\mathcal{C}}(c) \cap \text{name}_{\mathcal{C}}(c') = \emptyset$. Ein $w \in \text{name}_{\mathcal{C}}(c)$ wird ein Name des Konzeptes c genannt. Die Namensfunktion $\text{name}_{\mathcal{C}}$ wird effizient genannt, wenn für alle $n \in \mathbb{N}$ und alle Konzepte $c \in \mathcal{C}_n$ die Länge eines kürzesten Namens für c , auch kurz mit $\text{länge}(c)$ bezeichnet, polynomiell in n beschränkt ist und es einen deterministischen Algorithmus gibt, der für alle $n \in \mathbb{N}$, alle $w \in \{0,1\}^*$ und jede Eingabe $v \in \Sigma^*$ nachprüft,

- ob w der Name eines Konzeptes in \mathcal{C}_n ist, und
- ob $v \in c$ für das Konzept $c \in \mathcal{C}_n$ mit $w \in \text{name}_{\mathcal{C}}(c)$ gilt.

Dabei muss die Laufzeit des Algorithmus polynomiell in n beschränkt sein.

Eine effiziente Namensfunktion gibt einer Hypothese also nicht zu lange Namen und erlaubt, das Wortproblem (ist $w \in L$) polynomiell zu entscheiden.

Definition 3.2.20 (Effizienter PAC-Algorithmus)

Gegeben seien eine parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ und eine parametrisierte Hypothesenklasse $\mathcal{H} = (\mathcal{H}_n)_{n \in \mathbb{N}}$, sowie die effiziente Namensfunktion $\text{name}_{\mathcal{H}}$ für \mathcal{H} . Ein Lernalgorithmus A ist genau dann ein effizienter PAC-Algorithmus⁶ für \mathcal{C} durch \mathcal{H} mit $\text{name}_{\mathcal{H}}$, falls für alle $\epsilon \in (0, 1]$, für alle $\delta \in (0, 1]$, für alle $n \in \mathbb{N}$, für alle Verteilungen D und für alle Konzepte $c \in \mathcal{C}$ gilt, dass A eine Hypothese $h \in \mathcal{H}$ mit

$$\text{prob}_D \left[\text{fehler}_D(c, h) \leq \epsilon \right] \geq 1 - \delta$$

aufstellt und einen Namen der Hypothese aus $\text{name}_{\mathcal{H}}(h)$ ausgibt. Die Laufzeit des Algorithmus A muss dabei durch ein Polynom

$$\text{poly} \left(\frac{1}{\epsilon}, \frac{1}{\delta}, n, \text{länge}(c) \right)$$

beschränkt sein.

Gibt es einen effizienten PAC-Algorithmus für \mathcal{C} durch \mathcal{H} mit $\text{name}_{\mathcal{H}}$, so wird \mathcal{C} effizient lernbar durch \mathcal{H} mit $\text{name}_{\mathcal{H}}$ genannt. Falls $\mathcal{C} = \mathcal{H}$ wird der Zusatz “durch \mathcal{H} mit $\text{name}_{\mathcal{H}}$ auch weggelassen und \mathcal{C} (effizient) lernbar genannt.

In Definition 3.2.5 wurde bereits das Konsistenzproblem, die Berechnung einer konsistenten Hypothese, angesprochen. Es stellt sich die Frage, ob es einen effizienten PAC-Algorithmus gibt, der lernt, ohne konsistent zu sein. Unsere weiteren Untersuchungen werden diese Frage mit einem klaren nein beantworten: Wenn das Konsistenzproblem nicht effizient lösbar ist, existiert auch kein effizienter PAC-Algorithmus. Allerdings sind unter der Annahme $P = NP$ die meisten “vernünftigen” Konzeptklassen effizient lernbar [PV88]. Um die Nichtlernbarkeit zu zeigen, muss also die allgemein akzeptierte Annahme $P \neq NP$ vorausgesetzt werden. Jedoch sind die hier vorgestellten Algorithmen probabilistisch (und nichtdeterministisch). Daher reicht die Annahme $P \neq NP$ nicht aus. Die Ergebnisse werden vielmehr auf der Annahme $RP \neq NP$ fußen, wobei RP die Klasse der Sprachen ist, die von einem polynomiellen Monte-Carlo-Algorithmus akzeptiert werden [Gil77].

Definition 3.2.21 (Monte-Carlo-Algorithmus)

Gegeben sei eine Sprache L . Einen probabilistischen Algorithmus A nennen wir einen Monte-Carlo-Algorithmus für L , wenn A eine Ausgabe berechnet, so dass für eine Eingabe x

- i. $\text{prob} \left[A(x) = \text{“ja”} \right] \geq \frac{1}{2}$ falls $x \in L$,
- ii. $A(x) = \text{“nein”}$ falls $x \notin L$

gilt. Dabei werden die Wahrscheinlichkeiten über die “internen Münzwürfe” des Algorithmus gebildet.

Definition 3.2.22

RP ist die Klasse aller Sprachen, die durch einen polynomiellen Monte-Carlo-Algorithmus berechnet werden können.

⁶Ein Synonym für effizienter PAC-Algorithmus ist das häufig gebrauchte polynomieller PAC-Algorithmus.

Es gilt $P \subseteq RP \subseteq NP$. $P \subseteq RP$ ist offensichtlich. Betrachten wir $RP \subseteq NP$. Für eine Sprache $L \in RP$ mit zugehörigem Monte-Carlo-Algorithmus A gibt es für $x \in L$ eine akzeptierende Berechnung polynomieller Länge, während für $x \notin L$ keine solche Berechnung existiert. Folglich ist $RP \subseteq NP$. Ob die Inklusionen echt sind, ist ein offenes Problem.

Lemma 3.2.23

Gegeben seien eine Konzeptklasse $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$, ein effizienter PAC-Algorithmus für \mathcal{C} und eine Menge klassifizierter Beispiele. Dann existiert ein probabilistischer Algorithmus, der in polynomieller Zeit mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ eine konsistente Hypothese $h \in \mathcal{C}$ findet.

Beweis: Der Beweis basiert auf [BEHW89]. Sei A der effiziente PAC-Algorithmus für \mathcal{C} . Aufgrund der PAC-Eigenschaften gibt A beim Aufruf mit dem Fehlerparameter ϵ und dem Vertrauensparameter δ mit einer Wahrscheinlichkeit von $1 - \delta$ eine Hypothese aus, deren Fehler höchstens ϵ ist. Dazu fordert er klassifizierte Beispiele an, deren Zahl polynomiell in $\frac{1}{\epsilon}$ und $\frac{1}{\delta}$ beschränkt ist. Sei S eine Menge gemäß einem Konzept $c \in \mathcal{C}_n$, für ein $n \in \mathbb{N}$, klassifizierter Beispiele. Mit Hilfe von A wird ein probabilistischer Algorithmus A^* konstruiert, der mit einer Wahrscheinlichkeit von $\frac{1}{2}$ eine zu S konsistente Hypothese $h \in \mathcal{C}$ findet.

Sei D die Verteilung auf Σ^* , die gleichmäßig auf allen Beispielen $w \in S$ ist und 0 sonst. A^* ruft den PAC-Algorithmus A mit dem Fehlerparameter $\epsilon = 1/(|S|+1)$ und dem Vertrauensparameter $\delta = \frac{1}{2}$ auf. Wenn A ein Beispiel anfordert wird dieses Beispiel unabhängig gemäß der Verteilung D gezogen und identisch zu seiner Klassifizierung in S klassifiziert. A gibt eine Hypothese $h \in \mathcal{C}$ aus, die mit einer Wahrscheinlichkeit von $\frac{1}{2}$ einen Fehler von höchstens ϵ hat.

Die polynomielle Laufzeit des Algorithmus A^* ist offensichtlich. Da jedes gemäß der Verteilung D gezogene Beispiel eine Wahrscheinlichkeit von $\frac{1}{|S|} > \frac{1}{|S|+1} = \epsilon$ aufweist hat jede inkonsistente Hypothese einen Fehler größer als ϵ . Also hat A^* mit einer Wahrscheinlichkeit von wenigstens $\frac{1}{2}$ in polynomieller Zeit eine konsistente Hypothese gefunden. \square

Der in Lemma 3.2.23 vorgestellte probabilistische Algorithmus A^* wird auch polynomieller Hypothesenfinder genannt. BLUMER ET AL. haben in [BEHW89] sogar gezeigt, dass die Existenz eines effizienten PAC-Algorithmus für $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ nicht nur einen polynomiellen Hypothesenfinder bedingt, sondern auch, dass die VC-Dimension von \mathcal{C}_n polynomiell in n wächst. Unter Zuhilfenahme des polynomiellen Hypothesenfinders kann, bei Formulierung des Konsistenzproblems für die Konzeptklasse \mathcal{C} als Sprache $\text{konsistenz}_{\mathcal{C}}$, gezeigt werden, dass wenn $\text{konsistenz}_{\mathcal{C}}$ NP-vollständig ist, es keinen effizienten PAC-Algorithmus für die Konzeptklasse \mathcal{C} gibt.

Definition 3.2.24

Sei $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ eine parametrisierte Konzeptklasse über $\Sigma = \{0, 1\}$. Die Sprache $\text{konsistenz}_{\mathcal{C}}$ wird als

$$\text{konsistenz}_{\mathcal{C}} = \{(n, w_1, b_1, w_2, b_2, \dots, w_s, b_s) \mid \exists c \in \mathcal{C}_n \text{ mit } w_i \in c \Leftrightarrow b_i = 1 \text{ für } 1 \leq i \leq s\}$$

definiert, wobei $n \in \mathbb{N}$ und $b_i \in \{0, 1\}$ für $i = 1, 2, \dots, s$. $\text{konsistenz}_{\mathcal{C}}$ ist die Sprache aller Beispielmengen, für die es ein konsistentes Konzept gibt.

Lemma 3.2.25

Sei $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ eine parametrisierte Konzeptklasse mit effizienter Namensfunktion $\text{name}_{\mathcal{C}}$. Angenommen, es gibt einen effizienten PAC-Algorithmus für \mathcal{C} mit $\text{name}_{\mathcal{C}}$, dann gilt

$$\text{konsistenz}_{\mathcal{C}} \in RP.$$

Beweis: Der Beweis entstammt [Schn97]. Sei A der effiziente PAC-Algorithmus für \mathcal{C} . Sei $(n, x_1, b_1, x_2, b_2, \dots, x_s, b_s)$ die Eingabe für das Konsistenzproblem von \mathcal{C} . Es wird ein Monte-Carlo-Algorithmus A^{**} konstruiert, der durch Anwendung des effizienten PAC-Algorithmus A entscheidet, ob $(n, x_1, b_1, x_2, b_2, \dots, x_s, b_s) \in \text{konsistenz}_{\mathcal{C}}$. Daraus folgt dann, dass $\text{konsistenz}_{\mathcal{C}} \in \text{RP}$.

Sei $S = \{x_1, x_2, \dots, x_s\}$ die Menge klassifizierter Beispiele. Dem Beweis zu Lemma 3.2.23 folgend, wird aus A und S ein probabilistischer Algorithmus A^* konstruiert, der mit einer Wahrscheinlichkeit von $\frac{1}{2}$ eine zu S konsistente Hypothese findet und deren Namen ausgibt. Da die Namensfunktion $\text{name}_{\mathcal{C}}$ effizient ist, kann in polynomieller Zeit überprüft werden, ob der Name tatsächlich der Name einer Hypothese ist; das Gleiche gilt für die Prüfung der Konsistenz. Sei h die Ausgabe von A^* . A^{**} akzeptiert die Eingabe $(n, x_1, b_1, x_2, b_2, \dots, x_s, b_s)$ (mit $x_i \in S$ und $b_i = 1$ genau dann, wenn x_i ein positives Beispiel ist und $b_i = 0$ sonst) genau dann, wenn h eine zu S konsistente Hypothese ist.

Wir müssen zwei Fälle für die Eingabe unterscheiden. Falls $(n, x_1, b_1, x_2, b_2, \dots, x_s, b_s) \notin \text{konsistenz}_{\mathcal{C}}$ gibt es keine mit S konsistente Hypothese. Der Aufruf von A^* liefert in diesem Fall immer eine mit S inkonsistente Hypothese, die A^{**} verwirft. Falls $(n, x_1, b_1, x_2, b_2, \dots, x_s, b_s) \in \text{konsistenz}_{\mathcal{C}}$ gibt es eine mit S konsistente Hypothese. A^* liefert mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ eine mit S konsistente Hypothese. Also akzeptiert A^{**} die Eingabe mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ und verwirft sie mit einer Wahrscheinlichkeit von höchstens $\frac{1}{2}$. Da A^{**} bei einer negativen Antwort nie irrt und bei einer positiven Antwort mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ richtig antwortet erfüllt er die Definition eines Monte-Carlo-Algorithmus.

Zum Abschluss, ist noch die Laufzeit des Algorithmus A^{**} zu untersuchen. Unter der Voraussetzung $c \in \mathcal{C}_n$, läuft A^{**} , da die Namensfunktion $\text{name}_{\mathcal{C}}$ effizient ist, in Zeit

$$\text{poly}(n, |S|, \text{länge}(c)) = \text{poly}(n, |S|).$$

A^{**} dekodiert zunächst die Eingabe. Dann wird die Arbeit von A^* geleistet, der in polynomieller Zeit eine Ausgabe liefert. Als letztes überprüft A^{**} , ob die von A^* berechnete Hypothese h konsistent mit S ist. Dies kann in Zeit $\text{poly}(n, |S|)$ erfolgen, da $|S|$ Beispiele der Eingabelänge n mit h verglichen werden. \square

Satz 3.2.26

Sei $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ eine parametrisierte Konzeptklasse mit effizienter Namensfunktion $\text{name}_{\mathcal{C}}$. Wenn $\text{konsistenz}_{\mathcal{C}}$ NP-vollständig ist, gibt es unter der Annahme $\text{RP} \neq \text{NP}$ keinen effizienten PAC-Algorithmus für \mathcal{C} der \mathcal{C} auch als Hypothesenklasse benutzt.

Beweis: Angenommen A ist ein effizienter PAC Algorithmus für \mathcal{C} mit effizienter Namensfunktion $\text{name}_{\mathcal{C}}$. Nach Lemma 3.2.25 gibt es dann einen polynomiellen Monte-Carlo-Algorithmus für $\text{konsistenz}_{\mathcal{C}}$ und damit gilt $\text{konsistenz}_{\mathcal{C}} \in \text{RP}$, was, unter der Annahme $\text{RP} \neq \text{NP}$, ein Widerspruch zu $\text{konsistenz}_{\mathcal{C}} \in \text{NP}$ ist. \square

Falls also die Annahme $\text{RP} \neq \text{NP}$ gilt, bedingt die Schwierigkeit des Konsistenzproblems für \mathcal{C} , dass die Konzeptklasse \mathcal{C} nicht durch einen effizienten PAC-Algorithmus gelernt werden kann. Wenn also ein NP-vollständiges Problem auf das Konsistenzproblem für \mathcal{C} reduziert werden kann, erhält man eine negative Aussage für das effiziente Lernen der Konzeptklasse. Allerdings ist diese Aussage immer repräsentationsabhängig: Sie gilt nur für das Lernen der Konzeptklasse \mathcal{C} durch eine Hypothesenklasse \mathcal{H} . Nur weil das Konsistenzproblem für eine Hypothesenklasse \mathcal{H} NP-vollständig ist, gilt dies nicht für alle anderen möglichen Hypothesenklassen.

Definition 3.2.27 (Konzeptklassen für aussagenlogische Formeln)

Es werden die folgenden parametrisierten Konzeptklassen definiert:

- i. k -Term-DNF = $(k\text{-Term-DNF}_n)_{n \in \mathbb{N}}$.
- ii. k -Klausel-CNF = $(k\text{-Klausel-CNF}_n)_{n \in \mathbb{N}}$.
- iii. k -DNF = $(k\text{-DNF}_n)_{n \in \mathbb{N}}$.
- iv. k -CNF = $(k\text{-CNF}_n)_{n \in \mathbb{N}}$.

Satz 3.2.28

Unter der Annahme $\text{RP} \neq \text{NP}$ gibt es es keinen effizienten PAC-Algorithmus, der für ein festes $k \geq 2$ die Konzeptklassen

- i. k -Klausel-CNF durch k -Klausel-CNF,
- ii. k -Term-DNF durch k -Term-DNF

mit der Binärcodierung der aussagenlogischen Formel als Namensfunktion erlernt.

Beweis: PITT und VALIANT zeigen in [PV88] das Ergebnis, indem sie das NP-vollständige Problem der k -Färbbarkeit auf das Konsistenzproblem reduzieren. Der Beweis findet sich auch in [KLPV87]. \square

Satz 3.2.29

Es gibt einen effizienten PAC-Algorithmus, der für ein festes $k \geq 1$ die Konzeptklassen

- i. k -Term-DNF durch k -CNF,
- ii. k -Klausel-CNF durch k -DNF,
- iii. k -DNF durch k -DNF,
- iv. k -CNF durch k -CNF

mit der Binärcodierung der aussagenlogischen Formel als Namensfunktion erlernt.

Beweis: PITT und VALIANT beweisen (i) und (ii) in [PV88]. VALIANT beweist (iii) und (iv) in [Val84] und zeigt, dass die Laufzeit des Algorithmus in $O(n^k)$ liegt. BLUMER ET AL. beweisen (iii) und (iv) indem sie zeigen, dass $|\mathcal{C}_n| \leq 2^{(2n)^k}$ gilt und eine Beispielsanzahl von $O(\frac{1}{\epsilon}(n^k + \ln(\frac{1}{\delta})))$ genügt [BEHW89]. \square

Satz 3.2.28 und Satz 3.2.29 drücken aus, wie wichtig die Wahl der Hypothesenklasse ist. Mit dem Wechsel zu einer mächtigeren Hypothesenklasse, statt bei der “natürlichen” Wahl $\mathcal{H} = \mathcal{C}$ zu verbleiben, gelangt man möglicherweise von einem NP-vollständigen Problem zu einem polynomiell berechenbaren. Leider zeigt sich aber auch, wie am Beispiel der k -Term-DNF gesehen, dass eine größere Hypothesenklasse mehr Beispiele erfordert. Daher wird man bei der Entwicklung eines Lernalgorithmus manchmal zwischen Laufzeit und Beispielsanzahl wählen müssen. Dabei ist wichtig zu beachten, dass das Klassifizieren der Beispiele in der Praxis oftmals mit hohen Kosten verbunden ist. Daher muss im Sinne der Praxis solange die Frage der effizienten Lernbarkeit einer Konzeptklasse \mathcal{C} als ungelöst betrachtet werden, solange nicht entweder ein effizienter PAC-Algorithmus gefunden wurde oder gezeigt werden kann, dass das Lernen von \mathcal{C} durch \mathcal{H} für alle Hypothesenklassen \mathcal{H} NP-vollständig ist [Ke90]. Für ein negatives Ergebnis wird also ein repräsentationsunabhängiges Resultat gebraucht. Solch ein repräsentationsunabhängiges

Resultat wird im nächsten Abschnitt für deterministische endliche Automaten und Boolesche Formeln gezeigt.

BLUMER ET AL. geben in [BEHW89] noch eine hinreichende Bedingung für polynomielle Lernbarkeit an, die auf der Bevorzugung von Hypothesen mit geringer Beschreibungskomplexität, das heißt mit kurzem Namen, beruht. (Dabei wird aber nicht nach der kürzesten Hypothese gesucht, da dieses Problem im Allgemeinen NP-vollständig ist [Ke90].) Dieses Prinzip wird abgeleitet aus dem von WILLIAM OF OCKHAM aufgestellten Grundsatz “Non sunt multiplicanda entia praeter necessitatem” (auf Deutsch etwa: Objekte sind nicht unnötig zu multiplizieren) und daher “Occams Razor” genannt. Das Ergebnis zeigt, dass wenn eine konsistente Hypothese effizient konstruiert werden kann und sie genügend kompakter als die Beispielmenge ist, effizient (im PAC-Sinne) gelernt werden kann. Der wesentliche Vorteil der Occam-Algorithmen ist, dass sie sich der Komplexität des gesuchten Konzeptes anpassen und relativ wenige Beispiele für kurze Konzepte ausreichen. Occams Razor wird beispielsweise benutzt, um das Lernen von Konzeptklassen zu studieren, die aus endlichen Vereinigungen oder endlichen Durchschnitten einer festen Basisklasse \mathcal{C} mit endlicher VC-Dimension bestehen. Für diese Klassen kann gezeigt werden, dass sie polynomiell lernbar sind, wenn ein effizienter Algorithmus zur Konstruktion einer konsistenten Hypothese für die Basisklasse \mathcal{C} existiert.

DOMINGO, TSUKIJI und WATANABE zeigen in [DTW97], dass die durch einen Occam-Algorithmus angeforderte Anzahl an Beispielen stark verringert werden kann. Sie führen dazu partielle Occam-Algorithmen ein, die kurze Hypothesen liefern die nur auf einem Teil der Beispiele konsistent sind. Dadurch wird zwar die Genauigkeit der Hypothese etwas verringert, doch können so viele Konzeptklassen mit weniger Aufwand gelernt werden.

3.2.4 Kryptographie und Lernkomplexität

In Abschnitt 3.2.3 wurde bei der Betrachtung effizienter PAC-Algorithmen bereits gezeigt, dass verschiedene Konzeptklassen, wie etwa k -Term-DNF durch k -Term-DNF, nicht effizient gelernt werden können. Dazu wurde ein NP-vollständiges Problem auf das Konsistenzproblem der Hypothesenklasse reduziert. Diese Ergebnisse waren aber repräsentationsabhängig in dem Sinne, das nur bewiesen wurde, dass die Konzeptklasse \mathcal{C} durch die Hypothesenklasse \mathcal{H} nicht effizient gelernt werden kann. Über andere Hypothesenklassen wurde keine Aussage getroffen. Und wie Satz 3.2.29 zeigt, ist es durchaus möglich, k -Term-DNF durch k -CNF effizient zu lernen. In diesem Abschnitt wird ein repräsentationsunabhängiges Ergebnis vorgestellt. Es wird gezeigt, dass es Konzeptklassen gibt, die durch keine Hypothesenklasse effizient gelernt werden können.

Dazu wird nachgewiesen, dass die Existenz eines effizienten PAC-Algorithmus für Boolesche Formeln, deterministische endliche Automaten oder Schaltkreise mit logarithmischer Tiefe erhebliche Konsequenzen für bekannte kryptographische Systeme und die Zahlentheorie hätte: mit diesem Algorithmus wäre es möglich, das RSA-Kryptosystem zu brechen oder quadratische Reste zu entdecken [KV89]. Das Brechen des RSA-Kryptosystems soll exemplarisch näher beleuchtet werden. Dazu wird zuerst das RSA-System vorgestellt. Davon ausgehend wird die Konzeptklasse RSA definiert und gezeigt, dass wenn diese Konzeptklasse von einem effizienten PAC-Algorithmus gelernt werden kann, das RSA-Schema gebrochen wird. Zum Abschluss wird für Schaltkreise logarithmischer Tiefe und $\text{DFA}_{n,\Sigma}$ nachgewiesen, dass sie die Konzeptklasse RSA enthalten. Daher kann es, unter der Bedingung, dass das RSA-Schema nicht gebrochen werden kann, keinen effizienten PAC-Algorithmus für diese beiden Konzeptklassen geben.

Seit alters her waren Menschen daran interessiert, Nachrichten so zu verfassen, dass nur ein kleiner Kreis von Eingeweihten diese verstehen, also entschlüsseln, konnte. Bei der spartanischen Skytale etwa wurde ein Pergamentstreifen in ansteigenden Spiralen um einen Stab gewickelt und beschrieben. Das abgewickelte Schriftstück konnte der Empfänger nur entziffern, wenn er es in

derselben Steigung um einen Stab gleicher Stärke wickelte [ZSG79]. Julius Cäsar dagegen griff in seinen Briefen an Cicero auf das Vertauschen von Buchstaben zurück [Su87]. Im Laufe der Zeit wurden die Verschlüsselungen zwar aufwendiger und ausgefeilter, doch basierten sie immer auf dem folgenden Verfahren. Zuerst einigen sich Sender und Empfänger auf eine Kodierungsfunktion E , eine Dekodierungsfunktion D und einen Schlüssel k , wobei $D(E(x, k), k) = x$ für alle Nachrichten x gefordert wird. Der Sender veröffentlicht die kodierte Nachricht x als $E(x, k)$, welche der Empfänger dank Kenntnis des Schlüssels k mittels $D(E(x, k), k) = x$ entziffern kann. Die Kodierungsfunktion E und die Dekodierungsfunktion D dürfen dabei öffentlich bekannt sein.⁷ Die Sicherheit des Verfahrens basiert also auf der Kenntnis des Schlüssels. Bei Cäsar war der Schlüssel das Wissen darüber welche Buchstaben miteinander vertauscht wurden (D und A, E und B, und so weiter), bei der Skytale die Steigung und die Stabesstärke. Verfahren der eben vorgestellten Art nennt man, da der Schlüssel dem Sender und dem Empfänger bekannt und die Rolle des Senders und des Empfängers austauschbar sind auch symmetrische Verschlüsselungsverfahren. Ihr großer Nachteil ist, dass Sender und Empfänger vorab über einen sicheren Kanal den Schlüssel austauschen müssen, da jeder der den Schlüssel kennt, jede Nachricht entziffern kann.

In ihrer wegweisenden Arbeit [DH76] haben DIFFIE und HELLMAN den Grundstein für das gänzlich andere Konzept der “Public-Key” Verfahren gelegt. Bei diesem System werden für die Kodierungsfunktion E und die Dekodierungsfunktion D die folgenden Eigenschaften verlangt: E und D müssen effizient berechenbar sein, für eine Nachricht x gilt $D(E(x)) = x$ und aus der veröffentlichten Kodierungsfunktion E darf die Dekodierungsfunktion D nicht einfach berechnet werden können. Funktionen die diese drei Eigenschaften erfüllen werden auch “Trapdoor One-Way Funktionen”, kurz Trapdoor-Funktionen, genannt. Sie erhalten das Attribut “One-Way” da sie in die eine Richtung leicht, in die andere aber nur sehr schwer zu berechnen sind. Das Attribut “Trapdoor” dagegen erhalten sie, da die Umkehrfunktion bei Kenntnis einer nichtöffentlichen Trapdoor leicht zu berechnen ist. Mit Hilfe des Public-Key Verfahrens kommunizieren Sender und Empfänger, nennen wir sie Bob und Alice, dann wie folgt:

- i. Alice stellt eine Trapdoor-Funktion E auf und veröffentlicht ein Programm zur Berechnung von E . Die Trapdoor t hält sie geheim.
- ii. Bob kodiert seine Nachricht x mit dem veröffentlichten Programm und veröffentlicht seinerseits $E(x)$.
- iii. Alice kann $D(E(x))$ effizient berechnen, da sie t kennt. Alle anderen kennen t nicht und können deshalb Bobs kodierte Nachricht $E(x)$ nur durch die sehr ineffiziente Methode des Vergleichs mit $E(x')$ für alle x' entschlüsseln.

Die Kodierungsfunktion E besteht üblicherweise aus einer generellen Methode und einem öffentlichen Schlüssel, dem Public-Key. Mittels der generellen Methode wird, gesteuert durch den öffentlichen Schlüssel, eine Nachricht x verschlüsselt. Da jeder die gleiche generelle Methode benutzen kann, basiert auch bei diesem Verfahren die Sicherheit der Verschlüsselung auf der Sicherheit des Schlüssels, das heißt der Trapdoor.

DIFFIE und HELLMAN führen in [DH76] allerdings nur das Konzept der Public-Key Verfahren ein. Eine praktische Implementation stellen erst RIVEST, SHAMIR und ADLEMAN in [RSA78] vor: das nach ihnen benannte RSA-Verfahren (Abbildung 3.1 auf Seite 26). Das RSA-Verfahren gilt als sicheres Verschlüsselungsverfahren und findet beispielsweise Anwendung in Programmen zur

⁷Allgemein wird in der Kryptographie sogar gefordert, dass die mathematischen Verfahren für Verschlüsselung und Entschlüsselung veröffentlicht werden. Eine Verschlüsselung die darauf basiert, dass die zugrundeliegenden Verfahren geheim gehalten wird, gilt, vereinfacht gesagt, als unsicher.

Verschlüsselung von E-Mail (GnuPG), bei verschlüsselten Seiten im World-Wide-Web (openssl) oder bei der Authentifizierung (ssh). Als Trapdoor nutzt das RSA-Verfahren die Eigenschaft, dass das Problem der Primfaktorzerlegung großer natürlicher Zahlen sehr schwer ist [Kn81, 4.5.4], wohingegen das Finden großer Primzahlen und deren Multiplikation relativ leicht ist. So benötigen die bekannten Zerlegungsalgorithmen, wie das Sieb des ERATHOSTENES, für eine Zahl m einen Rechenaufwand von $\Omega(\sqrt{m})$ [AKS03]; selbst moderne Methoden brauchen immer noch

$$e^{\Omega(\sqrt{\ln(m) \cdot \ln(\ln(m))})}$$

[Co93, Kap. 10]. Daher wurde erst am 3. Dezember 2003 durch FRANKE und KLEINJUNG das als RSA-576 bekannte Problem der Faktorisierung einer 174-ziffrigen Zahl (576 Bits) gelöst [We03]. Die Faktorisierung einer Zahl mit 2048 Bits wird ihrer Einschätzung nach noch einige Jahrzehnte warten müssen [Bonn04]. Doch auch wenn im Augenblick kein besserer Faktorisierungsalgorithmus bekannt ist, heißt das nicht, dass ein deartiger Algorithmus nicht existiert. In diesem Fall wäre das RSA-Verfahren, da es dann seine Trapdoor-Eigenschaft verlöre, einfach ausgehebelt.

- i. Alice wählt zwei große Primzahlen p und q , mit $p \neq q$, und berechnet $m = p \cdot q$
- ii. Alice erzeugt eine Zufallszahl $\max(p, q) < d < \varphi(m)$, so dass $\text{ggT}(d, \varphi(m)) = 1$ gilt
- iii. Alice berechnet $2 \leq e < \varphi(m)$ mit $d \cdot e \equiv 1 \pmod{\varphi(m)}$.
- iv. Alice veröffentlicht m und e . (p, q, d) ist die geheimgehaltene Trapdoor.
- v. Bob verschlüsselt Nachricht x , mit $1 \leq x < m$, durch $E(x) = x^e \pmod{m}$ und sendet $E(x)$
- vi. Alice entschlüsselt Bobs Nachricht y durch $D(y) = y^d \pmod{m} = x^{ed} \pmod{m} = x \pmod{m}$

Abbildung 3.1: RSA-Verfahren

Bevor die einzelnen Schritte des in Abbildung 3.1 gegebenen RSA-Verfahrens beleuchtet und auf Korrektheit und Effizienz untersucht werden, muss noch etwas Rüstzeug aus der Zahlentheorie eingeführt werden.

Definition 3.2.30 (Eulersche φ -Funktion)

Die Eulersche φ -Funktion $\varphi : \mathbb{N} \rightarrow \mathbb{N}$ liefert für $\varphi(m)$ die Anzahl der zu $m \in \mathbb{N}$ teilerfremden Zahlen aus $\{1, 2, \dots, m\}$:

$$\varphi(m) = \left| \{x \mid 1 \leq x \leq m \wedge \text{ggT}(x, m) = 1\} \right|.$$

Lemma 3.2.31 (Eigenschaften der Eulerschen φ -Funktion)

Für zwei Primzahlen p und q mit $p \neq q$ gilt:

- i. $\varphi(p) = p - 1$.
- ii. $\varphi(p \cdot q) = (p - 1) \cdot (q - 1)$.

Beweis: “i)” Trivial

“ii)” Es gilt $\varphi(p \cdot q) = p \cdot q - (p - 1) - (q - 1) - 1$, da nur die $p - 1$ Vielfachen von q und die $q - 1$ Vielfachen von p nicht teilerfremd zu $q \cdot p$ sind. Außerdem gilt, da $\text{kgV}(p, q) = p \cdot q$, dass $\{q, 2q, \dots, (p - 1)q\} \cap \{p, 2p, \dots, (q - 1)p\} = \emptyset$. □

Satz 3.2.32 (Satz von Fermat-Euler)

Für alle Zahlen $a \geq 1$ und $m \geq 1$ mit $\text{ggT}(a, m) = 1$ gilt

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

Ist speziell $m = p$ eine Primzahl, so gilt für alle $a \geq 1$ mit $p \nmid a$

$$a^{p-1} \equiv 1 \pmod{p}.$$

Dieses zweite Ergebnis ist auch als Kleiner Fermatscher Satz bekannt.

Beweis: Der Beweis findet sich in verschiedenen Büchern zur Zahlentheorie, z.B. in [RU95].

Ein Modul m der das Produkt zweier verschiedener Primzahlen p und q ist, wird auch RSA-Modul genannt. Die Zahl e aus dem RSA-Verfahren in Abbildung 3.1 wird auch als Kodierungsexponent, die Zahl d entsprechend als Dekodierungsexponent bezeichnet.

“i)” Zum Finden zweier großer Primzahlen p und q wählt man üblicherweise eine Zufallszahl und prüft, ob sie prim ist. Hierzu kann man den probabilistischen Rabin-Miller Test [Rab80] oder den probabilistischen Monte-Carlo-Algorithmus von SOLOVAY und STRASSEN [SS77] einsetzen. Seit August 2002 steht sogar ein deterministisches Verfahren mit polynomieller Laufzeit, der Primzahltest von AGARWAL, KAYAL und SAXENA zur Verfügung [AKS03]. Nach dem großen Primzahlsatz 2.1.3, ist die Chance eine Primzahl zu finden etwa $1 : \ln(n)$. Also müssen im Schnitt für das Finden einer 617-ziffrigen Primzahl (2048 Bits) $\ln(10^{617})/2 = 710$ Zahlen überprüft werden.

“ii)” Der Dekodierungsexponent d kann leicht gewählt werden; so genügt jede Primzahl $d > \max(p, q)$. Er sollte aber nicht zu klein gewählt sein, damit ein Analyst ihn nicht durch direkte Suche aufspüren kann.

“iii)” Der Kodierungsexponent e ist im Ring der ganzen Zahlen modulo $\varphi(m)$ das multiplikative Inverse zu d . Bei Kenntnis von $\varphi(m)$ und d kann e mit dem erweiterten Euklidischen Algorithmus effizient berechnet werden [RSA78], [Kn73, 1.2.2] und [Kn81, 4.5.3].

“v), vi)” Es muss nur noch gezeigt werden, dass die Dekodierungsfunktion und die Kodierungsfunktion effizient berechenbar sind, und dass $D(E(x)) \equiv x \pmod{m}$ gilt. Die effiziente Berechnbarkeit wird mit Lemma 3.2.34 nachgewiesen. Den Nachweis, dass eine kodierte Nachricht x wieder dekodierbar ist erbringt Lemma 3.2.35

Definition 3.2.33

Wir schreiben kurz $\text{pow}(m, x)$ für die Folge natürlicher Zahlen

$$x^{2^0} \bmod m, x^{2^1} \bmod m, x^{2^2} \bmod m, \dots, x^{2^{\lfloor \log(m) \rfloor}} \bmod m.$$

Lemma 3.2.34 (Effizientes Potenzieren ganzer Zahlen)

Die Potenz $x^e \bmod m$, mit $1 \leq x < m$, $e \geq 0$, $m > 0$, ist effizient berechenbar.

Beweis: Aus [Schn97]. Zuerst wird durch fortgesetztes Quadrieren die Folge $\text{pow}(m, x)$ berechnet. Insgesamt wird $\lfloor \log(m) \rfloor + 1$ mal quadriert. Danach wird durch Multiplikation der geeigneten Potenzen, das sind jene, bei denen in der Binärdarstellung von e eine 1 steht, $x^e \bmod m$ berechnet. Sei die Binärdarstellung von e durch $e = \sum_{i=0}^k e_i \cdot 2^i$, $e_i \in \{0, 1\}$, gegeben, dann ist

$$x^e \equiv \prod_{i=0}^k x^{e_i \cdot 2^i} \equiv \prod_{\substack{i=0 \\ e_i=1}}^k x^{2^i} \pmod{m}.$$

$x^e \bmod m$ wird also als Produkt von höchstens $\lfloor \log(m) \rfloor + 1$ vielen Faktoren berechnet. \square

Lemma 3.2.35 (Korrektheit der Kodierung und Dekodierung)

Seien $m = p \cdot q$ ein RSA-Modul, e der Kodierungsexponent und d der Dekodierungsexponent. Sei $0 \leq x < m$ eine Nachricht. Dann gilt für die Kodierungsfunktion $E(x) = x^e \pmod m$ und die Dekodierungsfunktion $D(y) = y^d \pmod m$

$$D(E(x)) \equiv x \pmod m.$$

Beweis: Der Beweis folgt [RSA78]. Sei $\mathbb{Z}_{\varphi(m)}$ der Restklassenring modulo $\varphi(m)$. Da d relativ prim zu $\varphi(m)$ ist, existiert in $\mathbb{Z}_{\varphi(m)}$ ein multiplikatives Inverses e mit $e \cdot d \equiv 1 \pmod{\varphi(m)}$. Also gibt es auch ein k , so dass $e \cdot d = k \cdot \varphi(m) + 1$. Daraus folgt

$$x^{ed} \equiv x^{k \cdot \varphi(m) + 1} \pmod m.$$

Falls $p \nmid x$ gilt nach Satz 3.2.32 $x^{p-1} \equiv 1 \pmod p$ und so erhalten wir

$$x^{k \cdot \varphi(m) + 1} \equiv x^{k \cdot (p-1) \cdot (q-1)} x \equiv x \pmod p.$$

Trivialerweise gilt auch $x \equiv 0 \pmod p$ falls $p \mid x$. Daher gilt die Gleichung für alle x . Mit einer analogen Argumentation für q erhalten wir für alle x

$$x^{k \cdot \varphi(m) + 1} \equiv x \pmod q.$$

Aus den beiden Gleichungen folgt sofort, dass für alle x

$$x^{ed} \equiv x^{k \cdot \varphi(m) + 1} \equiv x \pmod m.$$

Das gewünschte Ergebnis ergibt sich dann durch

$$D(E(x)) \equiv (E(x))^d \equiv (x^e)^d \equiv x^{ed} \equiv x \pmod m.$$

□

RIVEST, SHAMIR und ADLEMAN unterziehen in ihrer Arbeit [RSA78] das RSA-Verfahren noch einer Kryptoanalyse hinsichtlich der Sicherheit der Methode. Sie zeigen, dass das Berechnen von $\varphi(m)$, ohne m zu faktorisieren, das Finden von d , ohne $\varphi(m)$ zu berechnen, oder m zu faktorisieren ebenso schwierig wie die Faktorisierung von m ist. Es ist daher anzunehmen, dass jeder Ansatz das RSA-Verfahren zu brechen zu einem effizienten Faktorisierungsalgorithmus führt. Da die Primfaktorzerlegung aber ein wohluntersuchtes Problem ist und das RSA-Verfahren seit 1978 ungebrochen ist, wagen wir Hypothese 3.2.36 aufzustellen.

Definition 3.2.36 (RSA-Hypothese)

Es gibt keinen probabilistischen Algorithmus A mit folgenden Eigenschaften.

- i. Für jeden gegebenen RSA-Modul m und jeden Exponenten $e > 1$, mit $\text{ggT}(e, \varphi(m)) = 1$ gilt

$$\text{prob} \left[\begin{array}{l} A \text{ berechnet zu gegebenen } m, e \text{ und } E(x) \text{ das} \\ \text{Urbild } x, \text{ sofern } x^e \pmod m \in \mathbb{Z}_m^* \text{ und } 0 \text{ sonst} \end{array} \right] \geq \frac{3}{4},$$

wobei die Wahrscheinlichkeit über die "internen Münzwürfe" von A und die zufällige Wahl von $x \in \{1, 2, \dots, m-1\}$ gebildet wird.

- ii. Die Laufzeit des Algorithmus A ist polynomiell in $\log(m)$

Wir führen nun die parametrisierte Konzeptklasse RSA ein und zeigen, dass es unter der RSA-Hypothese 3.2.36 keinen effizienten PAC-Algorithmus für RSA geben kann. Angenommen, es gibt einen effizienten PAC-Algorithmus für RSA. Dann kann die Dekodierungsfunktion Bit für Bit gelernt werden, indem für jedes Bit $i = 0, 1, 2, \dots, \lfloor \log(m) \rfloor$ eine Hypothese h_i aufgestellt wird, die eine Eingabe $E(x)$ genau dann akzeptiert, wenn $\text{ggT}(x, m) = 1$ und Bit i in x gesetzt ist. Dazu werden unter der Gleichverteilung zufällige Beispiele x_j gezogen und $y_j = E(x_j)$ berechnet. y_j wird genau dann als positives Beispiel klassifiziert, wenn Bit i in x gesetzt ist. Da die Kodierungsfunktion $E(x) = x^e \bmod m$ veröffentlicht wird, ist dies möglich. Mit Hilfe des effizienten PAC-Algorithmus kann also eine Hypothese h_i gelernt werden, die mit einer Zuverlässigkeit von $1 - \delta$ und einem Fehler von höchstens ϵ das Bit i des Urbildes der Eingabe $E(x)$ bestimmt.

KEARNS und VALIANT weisen in [KV89], unter Zuhilfenahme eines Ergebnisses von ALEXI, CHOR, GOLDREICH und SCHNORR, übrigens darauf hin, dass bereits die Bestimmung des “least significant bit” so schwierig wie die Dekodierung aller Bits ist.

Definition 3.2.37 (Konzeptklasse RSA)

Zu einem RSA-Modul m , einem Kodierungsexponenten $e \in \mathbb{Z}_{\varphi(m)}^*$ und einem i , mit $1 \leq i \leq \lfloor \log(m) \rfloor + 1$, wird das Konzept $c_{m,e,i}$ als

$$c_{m,e,i} = \left\{ (m, e, \text{pow}(m, x^e)) \mid 0 \leq x < m \wedge \text{ggT}(m, x) = 1 \wedge \text{Bit } i \text{ von } x \text{ ist } 1 \right\}$$

definiert. Die Konzeptklasse RSA_n enthält alle Konzepte der RSA-Moduln mit einer Bitlänge von höchstens n

$$\text{RSA}_n = \left\{ c_{m,e,i} \mid m < 2^n \text{ ist RSA-Modul, } e \in \mathbb{Z}_{\varphi(m)}^*, 1 \leq i \leq \lfloor \log(m) \rfloor + 1 \right\}.$$

Die parametrisierte Konzeptklasse RSA ist dann gegeben als $\text{RSA} = (\text{RSA}_n)_{n \in \mathbb{N}}$.

Satz 3.2.38

Die RSA Hypothese 3.2.36 möge gelten. Gibt es für eine gemäß der Beispiellänge parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n)_{n \in \mathbb{N}}$ ein Polynom p , so dass $\text{RSA}_n \subseteq \mathcal{C}_{p(n)}$, dann existiert kein effizienter PAC-Algorithmus für \mathcal{C} .

Beweis: Der Beweis folgt [Schn97]. Sei A ein effizienter PAC-Algorithmus für \mathcal{C} . Aufgrund der PAC-Eigenschaften gibt A beim Aufruf mit dem Fehlerparameter ϵ und dem Vertrauensparameter δ mit einer Wahrscheinlichkeit von $1 - \delta$ eine Hypothese aus, deren Fehler höchstens ϵ ist. Dazu fordert A $s(\epsilon, \delta)$ viele Beispiele an, wobei $s(\epsilon, \delta)$ polynomiell in $\frac{1}{\epsilon}$ und $\frac{1}{\delta}$ beschränkt ist. Unter Einsatz von A wird ein polynomieller probabilistischer Algorithmus A^* konstruiert, der, im Widerspruch zur RSA-Hypothese, für den RSA-Modul m und den Kodierungsexponenten e aus einer Eingabe $\hat{y} = E(\hat{x}) = \hat{x}^e \bmod m$ das Urbild \hat{x} mit einer Wahrscheinlichkeit von wenigstens $\frac{3}{4}$ berechnet.

Sei $1 \leq i \leq \lfloor \log(m) \rfloor + 1$ beliebig aber fest. Wir beschreiben nun wie A^* das Bit i des Urbildes berechnet (wobei für jedes Bit die gleiche Zahlenfolge x_1, x_2, \dots verwendet wird). A^* ruft A mit $\epsilon = 1/(8 \cdot (p(n) + 1))$ und $\delta = 1/(8 \cdot (p(n) + 1))$ auf. Wenn A ein Beispiel anfordert, ziehen wir gleichverteilt und unabhängig ein x , mit $0 \leq x < m$, und liefern als Beispiel $(m, e, \text{pow}(m, x^e))$, wobei wir das Beispiel genau dann positiv klassifizieren, wenn Bit i von x gesetzt ist. A gibt mit einer Zuverlässigkeit von $1 - \delta$ eine Hypothese h_i aus, die einen Fehler von höchstens ϵ aufweist. Um Bit i für das Urbild \hat{x} von $\hat{y} = \hat{x}^e \bmod m$ zu berechnen, prüfen wir, ob h_i die Eingabe $(m, e, \text{pow}(m, \hat{y}))$ akzeptiert. Falls ja, setzen wir Bit i in unserer Dekodierung von \hat{y} .

Seien nun alle Hypothesen h_i berechnet. Es können zwei Fehler gemacht werden: Es wird eine Hypothese h_i berechnet, die einen größeren Fehler als ϵ aufweist oder die Eingabe wird,

obwohl alle Hypothesen einen Fehler von höchstens ϵ aufweisen, falsch dekodiert. Aus der PAC-Eigenschaft von A folgt, dass die Wahrscheinlichkeit, dass eine Hypothese h_i einen Fehler größer ϵ aufweist durch δ beschränkt ist. Daher gilt

$$\text{prob}\left[\exists i : \text{fehler}_D(h_i, c_{m,e,i}) > \epsilon\right] \leq \sum_{i=1}^{\lfloor \log(m) \rfloor + 1} \text{prob}\left[\text{fehler}_D(h_i, c_{m,e,i}) > \epsilon\right] \leq (\lfloor \log(m) \rfloor + 1) \cdot \delta$$

und weiter, da nach Voraussetzung $\text{RSA}_n \subseteq \mathcal{C}_{p(n)}$ und $\log(m) \leq n$ ist, dass $\lfloor \log(m) \rfloor + 1 \leq p(n) + 1$ ist. Also erhalten wir

$$\text{prob}\left[\exists i : \text{fehler}_D(h_i, c_{m,e,i}) > \epsilon\right] \leq (p(n) + 1) \cdot \delta = \frac{1}{8}.$$

Sei nun $\text{fehler}_D(h_i, c_{m,e,i}) \leq \epsilon$ für alle $i = 1, 2, \dots, \lfloor \log(m) \rfloor + 1$. Wir haben als Eingabe für Algorithmus A^* nur Werte $0 \leq \hat{y} < m$. Algorithmus A weist einen Fehler von höchstens ϵ auf. Also wird für ein festes i das Bit i nur bei $m \cdot \epsilon$ Eingaben falsch dekodiert. Damit erhalten wir, dass höchstens

$$(\lfloor \log(m) \rfloor + 1) \cdot m \cdot \epsilon = (\lfloor \log(m) \rfloor + 1) \cdot \frac{m}{8 \cdot (p(n) + 1)} \leq \frac{m}{8}$$

Eingaben falsch dekodiert werden. Aus m Eingaben und höchstens $\frac{m}{8}$ falsch dekodierten Eingaben ergibt sich eine Wahrscheinlichkeit von $\frac{1}{8}$ für eine falsch dekodierte Eingabe unter der Voraussetzung, dass alle Hypothesen einen Fehler von höchstens ϵ aufweisen.

Somit weist A^* einen Fehler von $\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$ auf, was nichts anderes heißt, als dass die RSA-Kodierung mit einer Wahrscheinlichkeit von wenigstens $\frac{3}{4}$ gebrochen wird. \square

Satz 3.2.38 liefert ein repräsentationsunabhängiges Ergebnis für nicht effizient lernbare Konzeptklassen. Es verbleibt die Frage, welche konkreten Konzeptklassen mächtig genug sind, um RSA zu enthalten. KEARNS und VALIANT zeigen in [KV89], dass die Konzeptklassen der Schaltkreise logarithmischer Tiefe, der Booleschen Formeln über n Variablen und die azyklischer deterministischer endlicher Automaten RSA enthalten. Wir wollen das Thema PAC-Lernen mit dem Nachweis abschließen, dass die bereits eingeführte Konzeptklasse $\text{DFA}_{n,\Sigma}$ RSA enthält. Dabei folgen wir SCHNITGER in [Schn97] und reduzieren die Konzeptklasse LOG-SCHALTKREIS auf die Konzeptklasse $\text{DFA}_{n,\Sigma}$.

Definition 3.2.39 (Schaltkreis)

Ein Schaltkreis ist ein azyklischer, gerichteter Graph $G = (V, E)$. Die Menge der Knoten V zerfällt in zwei disjunkte Teilmengen: die Menge der Eingabeknoten (Knoten ohne eingehende Kanten) und die Menge der Gatter. Ein Gatter berechnet eine der booleschen Funktionen AND, OR oder NOT auf den Ausgaben seiner direkten Vorgänger. Einen Knoten ohne ausgehende Kanten bezeichnen wir als Ausgabeknoten. Die Tiefe des Schaltkreises ist die Anzahl der Kanten auf dem längsten Pfad in G von einem Eingabeknoten zu einem Ausgabeknoten.

Definition 3.2.40 (Schaltkreis logarithmischer Tiefe)

Ein Schaltkreis logarithmischer Tiefe mit n Eingabeknoten ist ein Schaltkreis gemäß Definition 3.2.39, der die folgenden Eigenschaften aufweist:

- i. die Anzahl der eingehenden Kanten eines Gatters ist höchstens 2,
- ii. die Tiefe ist höchstens $\log(n)$,
- iii. es gibt genau einen Ausgabeknoten.

Der Schaltkreis S akzeptiert eine Eingabe $x \in \{0, 1\}^n$ genau dann, wenn der Ausgabeknoten den Wert 1 berechnet.

Der Schaltkreis berechnet das Ergebnis wie folgt. Zuerst sind alle Eingabeknoten aktiv. Ein Gatter berechnet seine Ausgabe wenn alle seine direkten Vorgänger aktiv sind. Dabei wendet es die ihm zugeordnete boolsche Funktion auf die Ausgaben seiner Vorgänger an. Danach wird das Gatter selbst aktiv. Der Schaltkreis hat das Ergebnis berechnet, wenn die Ausgabe aktiv ist.

Definition 3.2.41

Die Konzeptklasse LOG-SCHALTKREIS_n besitzt für jeden Schaltkreis logarithmischer Tiefe S mit n Eingabeknoten ein Konzept

$$\text{wahr}(S) = \{x \mid x \in \{0, 1\}^n \wedge S \text{ akzeptiert } x\}.$$

Satz 3.2.42

Die RSA-Hypothese 3.2.36 möge gelten. Dann gibt es keinen effizienten PAC-Algorithmus für LOG-SCHALTKREIS_n .

Beweis: Der Beweis entstammt [Schn97]. Gemäß Satz 3.2.38 reicht es aus zu zeigen, dass ein Polynom p existiert, so dass $\text{RSA}_n \subseteq \text{LOG-SCHALTKREIS}_{p(n)}$ gilt. Für jedes Konzept $c_{m,e,i} \in \text{RSA}_n$ muss ein Schaltkreis S mit einer Tiefe $O(\log(m))$ konstruiert werden, der genau die Menge $c_{m,e,i}$ akzeptiert. Wir beschreiben die Konstruktion des Schaltkreises S .

Schaltkreis S überprüft, ob seine Eingabe eine Binärkodierung der Form $(m, e, \text{pow}(m, y))$ mit $y \notin \mathbb{Z}_m^*$ oder $y = x^e \in \mathbb{Z}_m^*$ ist. Falls $y \in \mathbb{Z}_m^*$ ist, soll S Bit i von x und sonst 0 ausgeben. Im Schaltkreis S werden die Bitposition i , der RSA-Modul m , der Kodierungsexponent e , die Faktoren p und q des Moduls m und der Dekodierungsexponent d , mit $(x^e)^d \equiv x \pmod{m}$ für alle $x \in \mathbb{Z}_m^*$ kodiert.

Da \mathbb{Z}_m^* eine Gruppe ist, gilt $y^d \in \mathbb{Z}_m^* \Leftrightarrow y \in \mathbb{Z}_m^*$. Wegen $\text{ggT}(y, m) \neq 1 \Leftrightarrow p \mid y \vee q \mid y$ genügt es zu testen, ob p oder q Teiler von y sind. Dies ist in einem Schaltkreis logarithmischer Tiefe möglich [BCH86]. Die Prüfung wird parallel vorgenommen.

Einfaches Quadrieren ist für einen Schaltkreis logarithmischer Tiefe ebenfalls möglich. Da alle Potenzen $\text{pow}(m, y)$ als Eingabe vorliegen, überprüft S nur, ob $(y^{2^i})^2 \equiv (y^{2^{i+1}}) \pmod{m}$ ist. Dies wird parallel $\lfloor \log(m) \rfloor$ -mal in Tiefe $O(\log(\log(m)))$ durchgeführt, beziehungsweise höchstens n -mal in Tiefe $O(\log(n))$.

Für die Berechnung von $y^d \pmod{m}$ nutzen wir die bereits als Eingaben zur Verfügung gestellten Potenzen $y^{2^i} \pmod{m}$. Gemäß unserer Konstruktion ist die Binärdarstellung von $d = \sum d_i \cdot 2^i$, mit $d_i \in \{0, 1\}$, dem Schaltkreis S bekannt. Es gilt

$$x \equiv y^d \equiv y^{\sum_{i=0}^k d_i \cdot 2^i} \equiv \prod_{d_i=1} y^{2^i} \pmod{m}$$

und $y^d \pmod{m}$ ist somit ein Produkt von höchstens $\lfloor \log(m) \rfloor + 1 \leq n + 1$ Potenzen. Auch das Wissen welche $d_i = 1$, das heißt, welche Potenzen zu wählen sind, "programmieren" wir direkt in S . Nach [BCH86] existiert ein Schaltkreis, der höchstens $\lfloor \log(m) \rfloor + 1 \leq n + 1$ viele Zahlen mit $\lfloor \log_2(m) \rfloor \leq n + 1$ Bits in Tiefe $O(\log(n))$ miteinander multipliziert.

Wenn die Tiefe von S also höchstens $k \cdot \log(n)$ beträgt, gilt für das Polynom $p(n) = n^k$ $\text{wahr}(S) \in \text{LOG-SCHALTKREIS}_{p(n)}$. \square

Zur speichereffizienten Auswertung der Berechnung des Schaltkreises definieren wir das Pebble-Spiel. Damit kann zu guter Letzt gezeigt werden, dass auch für $\text{DFA}_{n,\Sigma}$ kein effizienter PAC-Algorithmus existiert.

Definition 3.2.43 (Pebble-Spiel)

Gegeben sei ein Schaltkreis $S \in \text{LOG-SCHALTKREIS}_n$. Ziel des Pebble-Spieles ist, den Ausgabebaustein des Schaltkreises S mit einem Pebble (“Spielstein”) zu besetzen. Dabei sind folgende Regeln zu beachten:

- i. Ein Pebble darf stets auf einen Eingabeknoten des Schaltkreises gesetzt werden.
- ii. Ein Pebble darf jederzeit gelöscht werden.
- iii. Sind alle direkten Vorgänger eines Gatters mit einem Pebble besetzt darf das Gatter selbst mit einem Pebble besetzt werden.

Bei dieser Version des Pebble-Spieles ist es nicht erlaubt, einen Pebble zu verschieben. Daher werden drei Pebbles gebraucht, um ein AND oder OR Gatter mit einem Pebble zu besetzen.

Lemma 3.2.44

Gegeben sei ein Schaltkreis $S \in \text{LOG-SCHALTKREIS}_n$ mit einer Tiefe t . Dann kann das Pebble-Spiel aus Definition 3.2.43 mit $t + 2$ Pebbles so abgeschlossen werden, das höchstens $2^{t+1} - 1$ mal ein Pebble gesetzt werden muss.

Beweis: Der Graph des Schaltkreises logarithmischer Tiefe kann in einen binären Baum “transformiert” werden. Unter Umständen müssen dabei Teilbäume neu erzeugt werden, doch dadurch ändert sich nicht die Tiefe. Bekanntermaßen hat ein vollständiger binärer Baum höchstens $2^{t+1} - 1$ viele Knoten. Also muß höchstens $2^{t+1} - 1$ -mal ein Pebble gesetzt werden.

Der Beweis für die Zahl der Pebbles erfolgt über vollständige Induktion. Für einen AND oder OR Baustein mit Tiefe $t = 1$ benötigen wir $t + 2 = 3$ Pebbles. Sei das Gewünschte bereits für Tiefe $t - 1$ gezeigt. Wir betrachten einen Knoten mit Tiefe t . Eine Eingabe sei bereits berechnet. Die andere Seite hat Tiefe $t - 1$ und braucht damit $t - 1 + 2$ Pebbles. Also erhalten wir insgesamt $(t - 1 + 2) + 1 = t + 2$. \square

Satz 3.2.45

Die RSA Hypothese 3.2.36 möge gelten. Dann gibt es keinen effizienten PAC-Algorithmus für die Konzeptklasse $\text{DFA}_{n,\Sigma}$.

Beweis: SCHNITGER reduziert in [Schn97] LOG-SCHALTKREIS_n auf $\text{DFA}_{p(n),\{0,1\}}$, wobei p ein Polynom ist. Wir skizzieren den Beweis. Zuerst ist zu klären, wie ein Schaltkreis S durch einen endlichen Automaten M mit wenigen Zuständen ausgewertet werden kann. Sei t die Tiefe von S . Aus Lemma 3.2.44 wissen wir, dass das Pebble-Spiel aus Definition 3.2.43 dann mit $t + 2$ Pebbles und höchstens $2^{t+1} - 1$ Zügen abgeschlossen werden kann. Darauf aufbauend, wird ein DFA M mit $p(2^t) \leq p(n)$ Zuständen konstruiert (aus quasi “hintereinander” geschalteten DFA zur Berechnung jedes Pebbles), der die Eingabe $w^{2^{t+1}-1}$ genau dann akzeptiert, wenn $w \in \{0,1\}^n$ von S akzeptiert wird. Damit kann abschließend gezeigt werden, dass aus der Existenz eines effizienten PAC-Algorithmus für $\text{DFA}_{p(n),\{0,1\}}$ die Existenz eines effizienten PAC-Algorithmus für LOG-SCHALTKREIS_n folgt, was ein Widerspruch zu Satz 3.2.42 ist. \square

3.3 Lernen regulärer Sprachen mit Element- und Äquivalenzfragen

Beim PAC-Lernen blieb der Lernalgorithmus passiv. Er forderte nur eine beliebig große Menge an klassifizierten Beispielen an und erstellte auf diesen eine konsistente Hypothese. Im Gegensatz zum PAC-Lernen wird dem Lernalgorithmus beim Lernen mit Element- und Äquivalenzfragen eine aktivere Rolle gestattet. Das hier vorgestellte aktive Lernmodell basiert auf der Arbeit von ANGLUIN [An87].

Gegeben seien die Konzeptklasse \mathcal{C} über Σ und die Hypothesenklasse \mathcal{H} . Die Konzeptklasse \mathcal{C} sei bekannt. Der Lernalgorithmus versucht zu einem unbekanntem Konzept $c \in \mathcal{C}$ eine äquivalente Hypothese $h \in \mathcal{H}$ aufzustellen. Dazu darf der Lernalgorithmus zwei Arten von Fragen stellen.

- i. Bei einer Elementfrage fordert der Lernalgorithmus das Orakel auf, ein Beispiel $w \in \Sigma^*$ gemäß dem gesuchten Konzept als positiv ($w \in c$) oder negativ ($w \notin c$) zu klassifizieren.
- ii. Bei einer Äquivalenzfrage stellt der Lernalgorithmus eine Hypothese h auf und erhält vom Orakel die Antwort “die Hypothese ist richtig”, falls c äquivalent zu h ist, oder ein Gegenbeispiel w aus der symmetrischen Differenz von Hypothese und gesuchtem Konzept.

Als Effizienzkriterium wird in diesem Modell die Zahl der Äquivalenz- und Elementfragen herangezogen.

Es muss natürlich hinterfragt werden, wie praxisnah die Annahme ist, dass das Orakel die Fragen des Lernalgorithmus beantworten kann. Diese Fähigkeit billigt dem Orakel implizit eine unbeschränkte Berechnungskraft zu. Deshalb wird bei der Analyse der Komplexität der Lernalgorithmen ausgeklammert, wie schwer die Antwort auf eine Frage zu berechnen ist. Ansonsten müsste etwa für die Beantwortung einer Äquivalenzfrage gefordert werden, dass das Problem der Äquivalenz von Konzept und Hypothese (effizient) entscheidbar ist. Aber selbst wenn die Äquivalenz entscheidbar wäre, wird in verschiedenen Lernsituationen das Orakel gar kein Gegenbeispiel konstruieren können und auf einen menschlichen Experten zurückgreifen müssen. Andererseits beschränkt sich diese Arbeit auf das Lernen regulärer Sprachen mit deterministischen endlichen Automaten als Hypothesenklasse. Dank der Abschlusseigenschaften der regulären Mengen und der Entscheidbarkeit der Äquivalenz zweier deterministischer endlicher Automaten (Satz 2.2.4) kann das Orakel sowohl Element- als auch Äquivalenzfragen ohne Rückgriff auf einen Experten effizient beantworten.

Ein weiterer Kritikpunkt ist, dass ein böses Orakel⁸ Äquivalenzfragen mit wenig aussagekräftigen Gegenbeispielen beantwortet. Bei dem Beweis für die untere Schranke für das Lernen regulärer unärer Sprachen mit Äquivalenzfragen, Lemma 6.1.10, werden wir solch ein böses Orakel kennenlernen. Die Existenz eines möglicherweise bösen Orakels erfordert auch, dass der Lernalgorithmus immer eine konsistente Hypothese als Äquivalenzfrage ausgibt, da ansonsten das Orakel als Antwort eines der bereits bekannten Beispiele gibt, zu denen die Hypothese nicht konsistent ist. Diese Antwort wäre sogar nichtssagend. Daher muss von den Lernalgorithmen gefordert werden, dass sie eine konsistente Hypothese (effizient) konstruieren können.

Ein Lernalgorithmus der Äquivalenzfragen stellt, kann übrigens in einen PAC-Algorithmus transformiert werden. Das Lernen mit Äquivalenzfragen impliziert also einen PAC-Algorithmus für die Konzeptklasse. Der umgekehrte Fall gilt allerdings nicht, da es Konzeptklassen gibt, die PAC-lernbar sind, aber nicht exakt identifiziert werden können [An88]. Gleichfalls kann ein Lernalgorithmus der Äquivalenz- und Elementfragen in einen PAC-Algorithmus mit Elementfragen transformiert werden. Dies wird in Satz 3.3.13 am Beispiel Angluins Algorithmus vorgestellt.

⁸Bösartig nicht im moralischen oder theologischen Sinne, sondern in dem, dass es die für den Lernalgorithmus ungünstigsten Beispiele, also solche aus denen er nur wenig Information über das gesuchte Konzept gewinnt, zurückgibt.

3.3.1 Lernen mit Äquivalenz- oder Elementfragen

Angluins Algorithmus lernt die regulären Sprachen mit Äquivalenz- und Elementfragen in polynomieller Zeit. Der Einsatz beider Arten von Fragen ist dabei notwendig, da eine Art von Fragen alleine nicht ausreicht, um in polynomieller Zeit die regulären Sprachen zu lernen.

Wir formalisieren die Lernsituation. Gegeben sei ein endliches Alphabet Σ , $|\Sigma| > 1$, und eine reguläre Sprache $L \subseteq \Sigma^*$, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Der Lernalgorithmus darf Element- oder Äquivalenzfragen stellen und muss eine äquivalente Hypothese M , mit $L(M) = L$, finden. Die Konzeptklasse ist somit die Menge der regulären Sprachen, die Hypothesenklasse die Menge der deterministischen endlichen Automaten. Da keine Eingabe als Bezugsgröße für die Komplexität des Algorithmus vorliegt, wird die Komplexität als Funktion in der Zahl der Zustände des minimalen DFA für L und der Länge des längsten Gegenbeispiels gemessen.

Satz 3.3.1 (Elementfragen reichen nicht aus)

Sei L eine reguläre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Dann gibt es keinen polynomiellen Lernalgorithmus, der L mit Elementfragen durch die Hypothesenklasse der deterministischen endlichen Automaten lernt.

Beweis: [An82]. □

Dieses Ergebnis gilt bereits für die einfachere Konzeptklasse der Sprachen die genau ein Wort der Länge n enthalten. Diese Sprachen werden von einem minimalen DFA mit $O(n)$ Zuständen akzeptiert. Um aber die Sprache zu lernen, müssen im schlimmsten Fall $\Omega(|\Sigma|^n)$ viele Elementfragen gestellt werden. Ein Lernalgorithmus der nur Elementfragen stellen darf hat noch ein weiteres Defizit. Er braucht, wenn die Konzeptklasse nicht endlich ist, immer eine Zusatzinformation, um zu terminieren, da er in diesem Falle natürlich nicht alle Hypothesen bis auf eine "ausschalten" kann.

Satz 3.3.2 (Äquivalenzfragen reichen nicht aus)

Sei L eine reguläre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird, dann gibt es keinen polynomiellen Lernalgorithmus, der L mit Äquivalenzfragen durch die Hypothesenklasse der deterministischen endlichen Automaten lernt.

Beweis: [An90]. □

Satz 3.3.2 ist repräsentationsabhängig: Es wird versucht die regulären Mengen durch DFA zu lernen. Er basiert aber nicht auf einer Vermutung wie der RSA-Hypothese. ANGLUIN zeigt, dass es Gegenbeispiele mit schwacher Aussagekraft gibt, die nur wenige Konzepte "ausschalten". In einem "Worst-Case" Szenario, von dem ausgegangen werden muss, da das Orakel die Gegenbeispiele frei wählen darf, gibt ein bösartiges Orakel immer genau diese nichtssagenden Gegenbeispiele zurück. In diesem Fall ist eine nichtpolynomielle Anzahl von Äquivalenzfragen nötig, um die Menge der konsistenten Konzepte auf genau eines zu reduzieren, das heißt L zu erlernen. Das Ergebnis wirft natürlich gleich die Frage auf, was geschieht, wenn die Äquivalenzfragen erweitert oder die Hypothesenklasse geändert werden. In [An90] zeigt ANGLUIN aber, dass selbst mit den nichtdeterministischen endlichen Automaten als Hypothesenklasse kein polynomieller Algorithmus für das Lernen mit Äquivalenzfragen existiert.

3.3.2 Angluins Algorithmus

In ihrer Arbeit [An87] stellte ANGLUIN den nach ihr benannten Algorithmus zum Lernen regulärer Mengen mit Äquivalenz- und Elementfragen vor. Der Algorithmus führt eine Beobachtungstabelle, in der er sein Wissen über die Menge der klassifizierten Beispiele organisiert. Diese

Tabelle füllt er mit Elementfragen. Wann immer aus dieser Beobachtungstabelle eine konsistente Hypothese gewonnen werden kann, stellt der Algorithmus diese als Äquivalenzfrage. Dadurch lernt Angluins Algorithmus, wie in Satz 3.3.12 gezeigt wird, eine unbekannte reguläre Sprache in polynomieller Zeit (polynomiell in der Zustandszahl des minimalen DFA für die gesuchte Sprache und der Länge des längsten Beispiels).

Definition 3.3.3 (Beobachtungstabelle)

Eine Beobachtungstabelle ist ein Tripel $B = (S, E, A)$, wobei $S \subseteq \Sigma^*$ und $E \subseteq \Sigma^*$ zwei endliche, nichtleere Mengen sind, mit der Eigenschaft, dass jeder Präfix (Suffix) jedes $u \in S$ ($v \in E$) ebenfalls in S (E) enthalten ist. Mit A wird die mit 0 und 1 vollständig besetzte Beobachtungsmatrix bezeichnet. A hat eine Zeile für jedes Element aus $S \cup S\Sigma$ und eine Spalte für jedes Element aus E . Für Zeile $u \in S \cup S\Sigma$ und Spalte $v \in E$ gilt stets

$$A[u, v] = \begin{cases} 1 & \text{falls } uv \in L \\ 0 & \text{sonst} \end{cases}.$$

Für $u \in S \cup S\Sigma$ steht $A[u]$ kurz für die gesamte Zeile u . Weiter wird für $u, u' \in S \cup S\Sigma$ definiert, dass $A[u] = A[u']$ genau dann, wenn für alle $v \in E$ $A[u, v] = A[u', v]$.

Angluins Algorithmus erstellt anhand der Beobachtungstabelle einen mit den in der Beobachtungsmatrix organisierten klassifizierten Beispielen konsistenten DFA als Hypothese. Die Zeilen der Beobachtungsmatrix A , die mit $u \in S$ bezeichnet sind, werden dabei zu Zuständen; daher werden sie auch Zustandszeilen genannt. Analog gilt für die Zeilen die mit $u' \in S\Sigma$ bezeichnet sind, dass aus Ihnen die Übergangsfunktion erstellt wird; daher werden sie auch δ -Zeilen genannt. Die Spalten $v \in E$ nutzt der Algorithmus, um die Zustände (gemäß Definition 2.2.6) zu unterscheiden.

Definition 3.3.4 (Abgeschlossene Beobachtungstabelle)

Eine Beobachtungstabelle $B = (S, E, A)$ wird abgeschlossen genannt, wenn für jede δ -Zeile $u \in S \cup S\Sigma$ eine Zustandszeile $u' \in S$ existiert, so dass $A[u] = A[u']$.

Falls zu einer δ -Zeile u keine passende Zustandszeile in der Beobachtungsmatrix vorhanden ist, fehlt der Zustand, in den der Automat durch Eingabe u übergeht. In diesem Fall wird S um u ergänzt und A mit Elementfragen vervollständigt. Die Definition 3.3.3 der Beobachtungstabelle impliziert übrigens, dass für jede Zustandszeile $u \in S$ und jedes $\sigma \in \Sigma$ eine δ -Zeile $u\sigma$ vorhanden sein muss. Das bedeutet aber auch, dass durch Hinzufügen von u zu S unter Umständen auch neue δ -Zeilen mit Elementfragen besetzt werden müssen.

Definition 3.3.5 (Konsistente Beobachtungstabelle)

Eine Beobachtungstabelle $B = (S, E, A)$ wird konsistent genannt, wenn für zwei beliebige Zustandszeilen $u, u' \in S$ mit $A[u] = A[u']$ für alle $\sigma \in \Sigma$ gilt, dass $A[u\sigma] = A[u'\sigma]$.

Wenn es zwei Zustände u und u' gibt, die hinsichtlich ihres bekannten Akzeptanzverhaltens auf den Worten aus E nicht unterschieden werden, aber gemäß ihrer unterschiedlichen δ -Zeilen unterschiedliche Nachfolgezustände haben, fügt der Algorithmus eine neue Spalte ein. Durch die anschließende Besetzung der neuen Spalte durch Elementfragen wird die Inkonsistenz aufgehoben.

Definition 3.3.6 (Konsistenter DFA)

Gegeben sei eine abgeschlossene, konsistente Beobachtungstabelle $B = (S, E, A)$. Der durch die Beobachtungstabelle gegebene deterministische endliche Automate $M(B) = (Q, \Sigma, \delta, q_0, F)$ wird definiert durch $Q = \{A[u] \mid u \in S\}$, $F = \{A[u] \mid u \in S \wedge A[u, \epsilon] = 1\}$, $q_0 = A[\epsilon]$ und $\delta(A[u], \sigma) = A[u\sigma]$ für $u \in S$ und $\sigma \in \Sigma$

Der DFA $M(B)$ ist wohldefiniert. Da S nichtleer ist und ϵ enthält, ist q_0 definiert. Ebenso gilt, da E nichtleer ist und ϵ enthält, dass für $u, u' \in S$ mit $A[u] = A[u']$ $A[u, \epsilon]$ und $A[u', \epsilon]$ definiert und gleich sind. Somit ist F wohldefiniert. Um zu sehen, dass δ wohldefiniert ist, sei angenommen, dass $u, u' \in S$ mit $A[u] = A[u']$ sind. Da $B = (S, E, A)$ konsistent ist, gilt für jedes $\sigma \in \Sigma$ $A[u\sigma] = A[u'\sigma]$ und da B abgeschlossen ist, existiert ein $s \in S$ mit $A[s] = A[u\sigma]$.

Es wird nun gezeigt, dass der zu einer abgeschlossenen und konsistenten Beobachtungstabelle $B = (S, E, A)$ aufgestellte DFA $M(B)$ konsistent mit A ist, also die Beispiele richtig klassifiziert. Außerdem wird gezeigt, dass $M(B)$ der kleinste mit A konsistente DFA ist. Jeder andere DFA M' der konsistent mit A aber nicht äquivalent zu $M(B)$ ist, muss mehr Zustände als $M(B)$ haben.

Lemma 3.3.7

Gegeben sei eine abgeschlossene, konsistente Beobachtungstabelle $B = (S, E, A)$. Für $M(B)$ und für jedes $u \in S \cup S\Sigma$ gilt $\delta(q_0, u) = A[u]$

Beweis: Der Beweis erfolgt mittels vollständiger Induktion über die Länge von u . Natürlich gilt für $|u| = 0$, das heißt für $u = \epsilon$, dass $q_0 = A[\epsilon]$. Angenommen, die Behauptung $\delta(q_0, u) = A[u]$ sei für jedes $u \in S \cup S\Sigma$ mit $|u| \leq k$ bereits gezeigt. Es sei $v \in S \cup S\Sigma$ mit $|v| = k + 1$. Dann gilt $v = u\sigma$ für ein $\sigma \in \Sigma$ und ein Wort u mit $|u| = k$. Falls $v \in S\Sigma$ muss $u \in S$ gelten. Falls $v \in S$, muss $u \in S$ sein, da S alle Präfixe von v enthält. Also gilt

$$\delta(q_0, v) = \delta(\delta(q_0, u), \sigma) = \delta(A[u], \sigma) = A[u\sigma] = A[v]$$

was zu zeigen war. □

Lemma 3.3.8

Es sei $B = (S, E, A)$ eine abgeschlossene, konsistente Beobachtungstabelle. Dann ist der DFA $M(B)$ konsistent mit der Beobachtungsmatrix A . Das heißt, für jedes $s \in S \cup S\Sigma$ und $e \in E$, $\delta(q_0, se) \in F \Leftrightarrow A[s, e] = 1$.

Beweis: Der Beweis wird mittels vollständiger Induktion über die Länge von e gezeigt. Wenn $e = \epsilon$ und $s \in S \cup S\Sigma$ gilt nach Lemma 3.3.7 $\delta(q_0, se) = A[s\epsilon] = A[s]$. Ist $s \in S$, dann gilt nach Definition von F : $A[s] \in F \Leftrightarrow A[s, \epsilon] = 1$. Andernfalls folgt aus $s \in S\Sigma$ und der Abgeschlossenheit der Beobachtungstabelle $A[s] = A[s']$ für ein $s' \in S$ und $A[s'] \in F \Leftrightarrow A[s', \epsilon] = 1$. Da $A[s] = A[s']$ ist, gilt natürlich auch $A[s', \epsilon] = A[s, \epsilon]$ und damit $A[s] \in F \Leftrightarrow A[s, \epsilon] = 1$.

Angenommen die Behauptung gilt für alle $e \in E$ mit $|e| \leq k$. Gegeben sei ein $e \in E$ mit $|e| = k + 1$. Da E alle Suffixe von e enthält, gibt es ein $e' \in E$ mit $|e'| = k$ und ein $\sigma \in \Sigma$, so dass $e = \sigma e'$. Da B abgeschlossen ist, existiert weiter ein $s' \in S$ mit $A[s] = A[s']$. Dann gilt

$$\begin{aligned} \delta(q_0, se) &= \delta(\delta(q_0, s), ae') \\ &= \delta(A[s], ae') = \delta(A[s'], ae') \\ &= \delta(\delta(A[s'], a), e') \\ &= \delta(A[s'a], e') \\ &= \delta(\delta(q_0, s'a), e') \\ &= \delta(q_0, s'ae') \quad . \end{aligned}$$

Gemäß der Induktionsannahme gilt $\delta(q_0, s'ae') \in F$ genau dann, wenn $A[s'ae', \epsilon] = 1$. Da $A[s] = A[s']$ und $ae' = e \in E$ gilt $A[s'ae'] = A[sae'] = A[se]$. Also folgt $\delta(q_0, se) \in F$ genau dann, wenn $A[se, \epsilon] = 1$. □

Lemma 3.3.9

Sei $B = (S, E, A)$ eine abgeschlossene, konsistente Beobachtungstabelle. Angenommen der DFA $M(B)$ habe n Zustände, dann ist jeder DFA M' mit n oder weniger Zuständen, der konsistent mit der Beobachtungsmatrix A ist, isomorph zu $M(B)$.

Beweis: Angluin stellt in [An87] für diesen Beweis einen Isomorphismus auf und zeigt, dass M' mindestens n , also genau n Zustände haben muss, wenn M' konsistent mit der Beobachtungsmatrix A ist. \square

Satz 3.3.10

Sei $B = (S, E, A)$ eine abgeschlossene, konsistente Beobachtungstabelle. Dann ist der DFA $M(B)$ konsistent mit der Beobachtungsmatrix A . Jeder andere mit M konsistente DFA M' der nicht äquivalent zu A ist, muss mehr Zustände als $M(B)$ haben.

Beweis: Lemma 3.3.8 zeigt, dass $M(B)$ konsistent mit M ist. Lemma 3.3.9 zeigt dass jeder andere mit M konsistente DFA entweder isomorph zu M ist oder mindestens einen Zustand mehr aufweist. Also ist $M(B)$ der minimale mit M konsistente DFA. \square

Angluins Algorithmus ist als Algorithmus 3.3.1 auf Seite 38 aufgeführt. Er führt eine Beobachtungstabelle $B = (S, E, A)$ und beginnt mit $S = \{\epsilon\}$, $E = \{\epsilon\}$. Um die Beobachtungsmatrix A zu besetzen, stellt er Elementfragen für ϵ und jedes $\sigma \in \Sigma$. In der Schleife ab Zeile 3 prüft Angluins Algorithmus, ob die Beobachtungstabelle abgeschlossen und konsistent ist. Falls nicht, fügt er, wie oben bereits dargestellt, Zeilen oder Spalten in die Beobachtungsmatrix A ein. Wenn die Beobachtungstabelle dann abgeschlossen und konsistent ist, stellt der Algorithmus den DFA $M(B)$ als Äquivalenzfrage. Falls das Orakel mit "ja" antwortet terminiert er. Andernfalls erhält er ein Gegenbeispiel w . Er fügt w und alle Präfixe von w zu S hinzu und füllt die Beobachtungsmatrix entsprechend mit Elementfragen auf. Die Korrektheit und die Anzahl der Fragen des Algorithmus 3.3.1 werden in Satz 3.3.12 gezeigt. Vorab wird aber noch ein Lemma benötigt.

Lemma 3.3.11

Sei $B = (S, E, A)$ eine Beobachtungstabelle und n bezeichne die Anzahl verschiedener Zustandszeilen in A , also $n = |\{A[s] \mid s \in S\}|$. Jeder mit M konsistente DFA muss mindestens n Zustände haben.

Beweis: Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein mit A konsistenter DFA. Wir definieren $f(s) = \delta(q_0, s)$ für $s \in S$. Angenommen es gibt zwei $s, s' \in S$ mit $A[s] \neq A[s']$. Dann existiert ein $e \in E$ so dass $A[s, e] \neq A[s', e]$. Da M konsistent mit A ist, muss entweder $\delta(q_0, se)$ oder $\delta(q_0, s'e)$ in F sein. Also sind $\delta(q_0, se)$ und $\delta(q_0, s'e)$ zwei verschiedene Zustände. Dann muss $f(s)$ für $s \in S$ mindestens n verschiedene Werte annehmen. Daraus folgt, dass M wenigstens n Zustände haben muss. \square

Satz 3.3.12 (Korrektheit und Laufzeit von Angluins Algorithmus)

Gegeben seien das Alphabet Σ und eine unbekannt reguläre Sprache $L \subseteq \Sigma^*$, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Algorithmus 3.3.1 lernt die Sprache L und stellt dabei höchstens n Äquivalenzfragen und höchstens

$$n^2 \cdot (1 + \ell) \cdot (1 + |\Sigma|)$$

Elementfragen, wobei ℓ die Länge des längsten Gegenbeispiels ist. Die Laufzeit des Lernalgorithmus ist polynomiell in ℓ und n .

Beweis: Zuerst wird die Korrektheit des Algorithmus verifiziert. Aus Satz 3.3.10 folgt, dass wenn der Algorithmus terminiert, er einen DFA M mit $L(M) = L$ ausgibt. Es muss also nur geprüft werden, ob der Algorithmus terminiert. Dazu wird gezeigt, dass die Zahl unterschiedlicher

Eingabe: Eingabealphabet Σ , unbekannte reguläre Sprache $L \subseteq \Sigma^*$

Ausgabe: Minimaler DFA für L

- 1: $S \leftarrow \{\epsilon\}; E \leftarrow \{\epsilon\}$
- 2: Stelle Elementfragen für ϵ und alle $\sigma \in \Sigma$ und besetze A
- 3: **repeat**
- 4: **while** $B = (S, E, A)$ nicht abgeschlossen oder nicht konsistent ist **do**
- 5: /* siehe Definition 3.3.5 */
- 6: **if** $B = (S, E, A)$ ist nicht konsistent **then**
- 7: Finde $u, u' \in S, \sigma \in \Sigma$ mit $A[u] = A[u']$ und $A[u\sigma] \neq A[u'\sigma]$ und
- 8: $v \in E$, so dass $A[u\sigma, v] \neq A[u'\sigma, v]$
- 9: Füge σv zu E hinzu /* dies erzeugt neue Spalte σv in A */
- 10: Stelle Elementfragen für $u\sigma v$ für alle Zeilen $u \in S \cup S\Sigma$ und besetze A
- 11: **end if**
- 12: /* siehe Definition 3.3.4 */
- 13: **if** $B = (S, E, A)$ ist nicht abgeschlossen **then**
- 14: Finde $u \in S, \sigma \in \Sigma$, so dass kein $u' \in S$ mit $A[u\sigma] = A[u']$ existiert
- 15: Füge $u\sigma$ zu S hinzu /* dies erzeugt neue Zustandszeile und u.U. neue δ -Zeilen */
- 16: Stelle Elementfragen für die neuen Zeilen und besetze A
- 17: **end if**
- 18: **end while**
- 19: Stelle $M(B)$ als Äquivalenzfrage /* siehe Definition 3.3.6 */
- 20: **if** Orakel antwortet mit Gegenbeispiel $w \in \Sigma^*$ **then**
- 21: Füge w und alle Präfixe von w zu S hinzu /* neue Zustandszeilen und neue δ -Zeilen */
- 22: Stelle Elementfragen für die neuen Zeilen und besetze A
- 23: **end if**
- 24: **until** Orakel antwortet auf die Äquivalenzfrage $M(B)$ mit "ja"
- 25: Gib $M(B)$ aus

Algorithmus 3.3.1: Angluins Algorithmus zum Lernen regulärer Sprachen

Zustandszeilen, das heißt $|\{A[s] \mid s \in S\}|$, während des Laufs des Algorithmus monoton wächst und durch n beschränkt ist.

Angenommen, die Beobachtungstabelle (S, E, A) ist inkonsistent, dann wird ein Wort zu E hinzugefügt. Dies bedingt, dass die Zahl der unterschiedlichen Zustandszeilen um mindestens eins wächst, da zwei vorher gleiche Zustandszeilen nun ungleich sind und die vorher ungleichen Zeilen auch durch Hinzunahme einer weiteren Spalte ungleich bleiben.

Angenommen, die Beobachtungstabelle (S, E, A) ist nicht abgeschlossen, dann wird ein Wort $v\sigma \in S\Sigma$ zu S hinzugefügt und es gilt $A[v\sigma] \neq A[u]$ für alle $u \in S \setminus \{v\sigma\}$. Also wird die Zahl unterschiedlicher Zustandszeilen um mindestens eins erhöht.

Da der Algorithmus nur n verschiedene Zustandszeilen finden kann und in der Anfangsphase bereits eine gesetzt wird, werden die beiden oben angesprochenen Operationen höchstens $n - 1$ mal ausgeführt. Der Algorithmus findet immer eine abgeschlossene konsistente Beobachtungstabelle $B = (S, E, A)$, auf der er seine Hypothese $M(B)$ aufstellt.

Es verbleibt die Frage, wieviele unterschiedliche Hypothesen $M(B)$ der Algorithmus aufstellt. Angenommen, das Orakel gibt als Antwort auf die Äquivalenzfrage $M(B)$ das Gegenbeispiel w . Dann sind der minimale DFA für L und $M(B)$ nicht äquivalent. Andererseits sind sowohl der minimale DFA für L als auch $M(B)$ konsistent mit der Beobachtungsmatrix A . Also muss nach Satz 3.3.10 der minimale DFA für L mindestens einen Zustand mehr als $M(B)$ haben. Daraus folgt sofort, dass $M(B)$ höchstens $n - 1$ Zustände hat.

Der Algorithmus muss auf jeden Fall eine nächste Hypothese $M(B')$, $B' = (S', E', A')$ aufstellen, die, da A' eine Erweiterung von A ist, konsistent mit A sein muss und zudem das Gegenbeispiel w wie der minimale DFA für L klassifiziert. Daher ist $M(B')$ nicht äquivalent zu $M(B)$. Dann muss $M(B')$ aber nach Satz 3.3.10 mindestens einen Zustand mehr als $M(B)$ haben. Das zeigt, dass der Algorithmus nacheinander eine Folge von höchstens $n - 1$ falschen Hypothesen aufstellt, da die Zahl der Zustände der Hypothesen monoton wächst und mit einem Zustand beginnt und spätestens die Hypothese mit n Zuständen korrekt sein muss.

Insgesamt ergibt sich, dass Algorithmus 3.3.1 nach höchstens n Hypothesen terminiert und dabei die Schleife ab Zeile 3 höchstens $n - 1$ Mal durchläuft.

Als nächstes wird die Laufzeit untersucht. Diese hängt von der Länge der Gegenbeispiele ab. Sei also ℓ die Länge des längsten Gegenbeispiels. Es wird gezeigt, dass Algorithmus 3.3.1 polynomiell in n und ℓ läuft.

Zu Anfang haben S und E je ein Element. Jedesmal wenn die Beobachtungstabelle $B = (S, E, A)$ als nicht abgeschlossen erkannt wird, wird ein Wort zu S hinzugefügt. Jedesmal wenn $B = (S, E, A)$ als nicht konsistent erkannt wird, wird ein Element zu E hinzugefügt. Für jedes Gegenbeispiel w mit $|w| \leq \ell$ werden höchstens ℓ Worte zu S hinzugefügt (da ϵ bereits in S enthalten ist).

Folglich gilt $|E| \leq n$, da die Schleife ab Zeile 3 nur $n - 1$ mal durchlaufen wird. Die Länge des längsten Wortes in E ist anfangs 0 und erhöht sich um höchstens 1 mit jedem Wort, das hinzugefügt wird. Also ist die Länge des längsten Wortes in E durch $n - 1$ beschränkt. Weiter gilt $|S| \leq 1 + (n - 1) + \ell(n - 1) = n + \ell(n - 1)$. Die Beobachtungstabelle $B = (S, E, A)$ wird höchstens $n - 1$ mal als nicht abgeschlossen erkannt und es gibt höchstens $n - 1$ Gegenbeispiele, von denen jedes höchstens ℓ Worte zu S hinzufügt. Die Länge des längsten Wortes in S erhöht sich ebenfalls höchstens um 1 mit jedem Wort, das hinzugefügt wird weil B nicht abgeschlossen ist. Also ist die Länge des längsten Wortes in S durch $\ell + n - 1$ beschränkt. Zusammengefasst ergibt sich, dass die Beobachtungstabelle

$$(|\Sigma| + 1) \cdot (n + \ell(n - 1)) \text{ Zeilen und } n \text{ Spalten} \quad (3.1)$$

besitzt. Dazu ist die maximale Länge eines Wortes $w \in (S \cup S\Sigma)E$ durch $\ell + 2n - 1$ beschränkt. Also kann die Beobachtungstabelle $B = (S, E, A)$ in einer endlichen Tabelle mit polynomieller Größe in ℓ und n dargestellt werden.

Betrachten wir nun noch die Operationen die Algorithmus 3.3.1 ausführt. Die Prüfung, ob die Beobachtungstabelle abgeschlossen oder konsistent ist, muss höchstens $n - 1$ Mal vorgenommen werden. Sie erfordert polynomielle Zeit in der Größe der Beobachtungstabelle. Nach Hinzunahme eines Wortes zu S oder E , werden höchstens $O(\ell n)$ Elementfragen nach Worten mit einer Länge von höchstens $O(\ell + n)$ gebraucht, um die Beobachtungsmatrix A vollständig zu besetzen. Wenn die Beobachtungstabelle $B = (S, E, A)$ abgeschlossen und konsistent ist, kann $M(B)$ in polynomieller Zeit in Größe der Beobachtungstabelle konstruiert werden. Diese Konstruktion wird höchstens n Mal vorgenommen. Ein Gegenbeispiel erfordert die Hinzunahme von höchstens ℓ Worten mit einer Länge von höchstens ℓ zu S . Da die letzte Hypothese richtig ist, wird das Orakel maximal $n - 1$ Mal ein Gegenbeispiel liefern. Die Laufzeit des Algorithmus 3.3.1 ist also polynomiell in ℓ und n .

Zuletzt betrachten wir noch die Zahl der gestellten Fragen. Der Algorithmus stellt höchstens n Mal eine Äquivalenzfrage. Die Zahl der Elementfragen ist durch die Größe der Beobachtungstabelle beschränkt. Wegen (3.1) ergibt sich eine obere Schranke von

$$n \cdot (n + \ell(n - 1)) \cdot (|\Sigma| + 1) \leq n \cdot (n + \ell n) \cdot (|\Sigma| + 1) = n^2 \cdot (\ell + 1) \cdot (|\Sigma| + 1)$$

Elementfragen. □

Satz 3.3.12 zeigt, dass die Kombination aus Äquivalenz- und Elementfragen die Lernkraft der Algorithmen mit polynomieller Laufzeit gegenüber denen mit nur einer Frageart merklich erhöht. Außerdem gibt es noch das Ergebnis aus Satz 3.2.45, wonach unter kryptographischen Annahmen die regulären Mengen nicht PAC-lernbar sind. Erweitern wir das PAC-Modell um Elementfragen, dann läßt sich, wie bereits erwähnt, Angluins Algorithmus in einen PAC-Algorithmus mit Elementfragen transformieren. Dazu wird Algorithmus 3.3.1 so modifiziert, dass die Äquivalenzfrage mit der Hypothese h durch die Anforderung klassifizierter Beispiele w_1, w_2, \dots, w_s ersetzt wird. Wenn ein Beispiel w_i , mit $1 \leq i \leq s$, von der Hypothese h anders als vom Orakel klassifiziert wird, dann wird w_i ein Gegenbeispiel genannt und der modifizierte Algorithmus passt seine Hypothese in der gleichen Weise an, wie auch Algorithmus 3.3.1. Wenn es kein Gegenbeispiel gibt, hält der modifizierte Algorithmus mit der Ausgabe h .

Satz 3.3.13

Sei L eine reguläre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird, dann kann Algorithmus 3.3.1 in einen PAC-Algorithmus mit Elementfragen mit Fehlerparameter ϵ und Vertrauensparameter δ transformiert werden. Dazu muss er

$$O\left(n + \frac{1}{\epsilon} \cdot \left(n \ln\left(\frac{1}{\delta}\right) + n^2\right)\right)$$

Beispiele anfordern.

Beweis: [An87]. □

Angluins Algorithmus ist inzwischen hinsichtlich der Zahl der Elementfragen erheblich verbessert worden. Ausgangspunkt war die Aufgabe eines Roboters, die Karte einer ihm unbekanntem Umgebung zu erlernen. Dieses “Map-Learning” genannte Problem kann als Lernen eines endlichen Automaten modelliert werden. Ein wichtiger Unterschied ist aber, dass es beim “Map-Learning” keine “reset” Operation gibt. Beim Lernen eines endlichen Automaten führt jede Elementfrage einen “reset” aus, der den DFA in den Startzustand versetzt, von dem aus die Eingabe eingelesen wird. Beim “Map-Learning” dagegen startet der Roboter in einem unbekanntem Zustand des zu lernenden DFA und bewegt sich von Zustand zu Zustand fort. Ein Startzustand zu dem der Roboter jederzeit zurückkehren kann (“reset”) fehlt in dieser Lernsituation.⁹ RIVEST und SCHAPIRE gehen in ihrer Arbeit [RS93] davon aus, dass die Umgebung durch einen stark zusammenhängenden Graphen (zwischen je zwei Knoten existiert ein Pfad vom ersten zum zweiten Knoten) beschrieben ist. Darauf aufbauend geben Sie einen Algorithmus mit polynomieller Laufzeit an, der mit hoher Wahrscheinlichkeit die Umgebung durch Erkundung und Äquivalenzfragen lernt. Dieser von RIVEST und SCHAPIRE vorgestellte Algorithmus kann auch auf die Situation angewandt werden, wenn ein “reset” erlaubt ist. In diesem Falle braucht er signifikant weniger Elementfragen als Algorithmus 3.3.1.¹⁰

Satz 3.3.14

Sei L eine reguläre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Dann existiert ein gegenüber Algorithmus 3.3.1 verbesserter Lernalgorithmus der L lernt und dabei höchstens n Äquivalenzfragen und höchstens

$$n^2 \cdot (1 + |\Sigma|) + (n - 1) \cdot (\log_2(\ell) + 1)$$

Elementfragen stellt, wobei ℓ die Länge des längsten Gegenbeispieles ist. Die Laufzeit des Lernalgorithmus ist polynomiell in ℓ und n .

Beweis: [Schn97]. □

⁹Das “Map-Learning” wird daher in der Literatur auch “Lernen ohne Reset” genannt.

¹⁰Weitere Ergebnisse zum “Map-Learning” und dem Lernen endlicher Automaten finden sich in [FKRRSS93].

Kapitel 4

Reguläre unäre Sprachen

Von nun ab beschäftigen wir uns mit dem algorithmischen Lernen regulärer unärer Sprachen. Reguläre unäre Sprachen sind, wie bereits erwähnt, reguläre Sprachen, die über einem Alphabet mit nur einem Buchstaben erklärt sind. Wir verwenden in dieser Arbeit das Alphabet $\Sigma = \{a\}$. Als Hypothesenklasse kommt ausschließlich die “natürliche” Wahl, die Klasse der (unären) deterministischen endlichen Automaten, zum Einsatz. Eine andere Hypothesenklasse werden wir nicht untersuchen, da reguläre unäre Sprachen durch (unäre) deterministische endliche Automaten mit Äquivalenzfragen oder Elementfragen (mit einer kleinen Zusatzinformation) effizient gelernt werden können.

4.1 Klassifizierung der unären deterministischen endlichen Automaten

Sei M ein deterministischer endlicher Automat, der eine reguläre unäre Sprache akzeptiert. Wir nennen M auch einen unären DFA. Der Graph eines unären DFA muss, sofern wir nicht erreichbare Zustände außer Acht lassen, aus einem Anfangspfad gefolgt von einem Zyklus bestehen. Der Graph ist also bereits durch die Zahl der Zustände und die Zahl der Zustände im Zyklus, auch Zykluslänge genannt, gegeben und entspricht, bis auf Isomorphismen, Abbildung 4.1. Für unsere Betrachtungen fassen wir die Automaten mit gleichem Graphen zu einer Klasse zusammen.

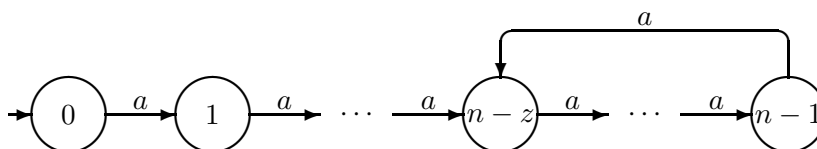


Abbildung 4.1: Unärer DFA mit n Zuständen und Zykluslänge z

Definition 4.1.1 (DFA mit n Zuständen und Zykluslänge z)

Ein DFA M mit n Zuständen und Zykluslänge z , $1 \leq z \leq n$, wird formal beschrieben durch $M = (Q, \Sigma, \delta, q_0, F)$, wobei $Q = \{q_0, q_1, \dots, q_{n-1}\}$ die Zustandsmenge, $\Sigma = \{a\}$ das Eingabealphabet und $F \subseteq Q$ die Menge der akzeptierenden Zustände ist und dessen Übergangsfunktion gegeben wird durch

$$\delta(q_i, a) = \begin{cases} q_{i+1} & \text{falls } 0 \leq i < n-1 \\ q_{n-z} & \text{falls } i = n-1 \end{cases}.$$

Mit $M(n, z)$ bezeichnen wir die Klasse aller DFA mit n Zuständen und Zykluslänge z .

Die Automaten in der Klasse $M(n, z)$ unterscheiden sich nur durch ihr Akzeptanzverhalten, also die Menge der akzeptierenden Zustände. Wir schreiben daher kurz $M \in M(n, z)$ mit F falls wir von einem bestimmten Automaten oder $M \in M(n, z)$ falls wir von einem beliebigen Automaten aus der Klasse $M(n, z)$ sprechen. Im Allgemeinen sind wir an Aussagen über die Klasse, das heißt, über alle Automaten der Klasse interessiert. Naturgemäß sind dies Aussagen, die darauf beruhen, dass ein Wort $w \in \Sigma^*$ in $M \in M(n, z)$ einen bestimmten Zustand erreicht. Wir benennen daher die Zustände der Automaten aus $M(n, z)$ nach ihrer kanonischen Nummerierung $0, 1, 2, \dots, n-1$. Außerdem definieren wir die Funktion $\hat{\delta}_{n,z}(k, w)$, die den kanonischen Namen eines Zustandes und ein Wort $w \in \Sigma^*$ als Eingabe nimmt und den kanonischen Namen des Zustandes $\delta(q_k, w)$ eines DFA $M \in M(n, z)$ liefert. Da die Übergangsfunktionen der Automaten einer Klasse identisch sind, ist die Wahl von M dabei unerheblich. Falls es aus dem Zusammenhang ersichtlich ist, lassen wir den Index der $\hat{\delta}$ Funktion auch fallen.

Aus der identischen Übergangsfunktion aller Automaten der Klasse $M(n, z)$ folgt natürlich sofort, dass die Namen der in $M, M' \in M(n, z)$ durch w erreichten Zustände gleich (nämlich $\hat{\delta}(0, w)$) sind.

Definition 4.1.2 (DFA mit vorgegebener Zykluslänge)

Sei $z > 0$ beliebig aber fest. Einen Automaten $M \in M(n, z)$, mit $n \geq z$, nennen wir einen DFA mit vorgegebener Zykluslänge z .

Beobachtung 4.1.3

Gegeben seien ein DFA $M \in M(n, z)$ und zwei Eingabeworte w und w' mit $|w|, |w'| \geq n - z$. Dann erreichen beide Worte einen Zustand im Zyklus und es gilt

$$\delta(q_0, w) = \delta(q_0, w') \iff |w| \equiv |w'| \pmod{z}.$$

Beweis: Falls $w = w'$ ist dies trivialerweise richtig. Also sei $w \neq w'$. Wie leicht nachgeprüft werden kann, gilt für die Übergangsfunktion

$$\delta(q_0, a^m) = \begin{cases} q_m & \text{falls } m < n - z \\ q_{n-z+(m-(n-z) \bmod z)} & \text{sonst} \end{cases}.$$

Der Beweis ergibt sich durch einfaches Anwenden der Übergangsfunktion.

“ \Leftarrow ” Wir setzen $r = |w| \bmod z$. Dann können wir w als $w = a^{r+sz}$, mit $s > 0$, und w' als $w' = a^{r+s'z}$, mit $s' > 0$, schreiben. Eingesetzt in die Übergangsfunktion erhalten wir

$$\delta(q_0, w) = q_{n-z+(r+sz-n+z \bmod z)} = q_{n-z+(r-n \bmod z)} = q_{n-z+(r+s'z-n+z \bmod z)} = \delta(q_0, w').$$

“ \Rightarrow ” Damit $\delta(q_0, w) = \delta(q_0, w')$ gilt, muss gemäß Übergangsfunktion

$$n - z + (|w| - n + z \bmod z) = n - z + (|w'| - n + z \bmod z)$$

sein. Daraus folgt direkt $|w| \equiv |w'| \pmod{z}$. □

4.2 Konsistenzproblem der Klasse $M(n, z)$

Sei nun $n > 0$ beliebig aber fest. Sei z , mit $1 \leq z \leq n$, beliebig aber fest. Außerdem sei P eine Menge positiver Beispiele und N eine Menge negativer Beispiele. Wir untersuchen nun das Konsistenzproblem der Klasse $M(n, z)$, das Problem der Konstruktion eines mit P und N konsistenten DFA $M \in M(n, z)$. Wir betrachten in welchen Fällen das Konsistenzproblem der Klasse $M(n, z)$ lösbar ist, und welche Schlüsse aus der Unlösbarkeit des Konsistenzproblems der Klasse $M(n, z)$ auf die Lösbarkeit des Konsistenzproblems einer Klasse $M(n', z')$ gezogen werden können.

Definition 4.2.1 (Beispiel)

Wir bezeichnen $w \in \Sigma^*$ als (klassifiziertes) Beispiel, wenn bekannt ist, ob es in der zu lernenden Sprache L enthalten ist. Falls $w \in L$ nennen wir w ein positives, ansonsten ein negatives Beispiel.

Definition 4.2.2 (Konsistenter DFA)

Sei P die Menge positiver Beispiele und N die Menge negativer Beispiele. Der DFA M wird konsistent zu P und N genannt, falls er alle positiven Beispiele akzeptiert und alle negativen verwirft, also

$$M \text{ konsistent} \iff L(M) \cap P = P \wedge L(M) \cap N = \emptyset.$$

Wir nennen M den minimalen konsistenten DFA, wenn es keinen anderen konsistenten DFA mit weniger Zuständen gibt.

Es gilt natürlich $L(M) \cap P = P \iff P \subseteq L(M)$; ein konsistenter DFA muss also alle positiven Beispiele akzeptieren. Falls die Beispielmengen klar sind, lassen wir den Zusatz “zu P und N ” fallen. Außerdem erweitern wir Definition 4.2.2 (bezüglich des minimalen konsistenten DFA) in der natürlichen Weise auf die DFA mit vorgegebener Zykluslänge.

Definition 4.2.3 (Konsistenzproblem)

Sei P die Menge positiver Beispiele und N die Menge negativer Beispiele. Als Konsistenzproblem der Klasse $M(n, z)$ bezeichnen wir das Problem der Konstruktion eines konsistenten DFA $M \in M(n, z)$.

Haben wir ein klassifiziertes Beispiel w erhalten, ist unsere Wahlfreiheit ob wir den durch w in einem DFA $M \in M(n, z)$ erreichten Zustand in die Menge der akzeptierenden Zustände aufnehmen eingeschränkt. Um konsistent mit dem Beispiel zu sein, darf dies nur für ein positives Beispiel geschehen. Daher bezeichnen wir einen solchen Zustand als festgelegt.

Definition 4.2.4 (Festgelegter Zustand)

Den Zustand q eines DFA $M \in M(n, z)$ bezeichnen wir als festgelegt, falls es ein klassifiziertes Beispiel w mit $\delta(q_0, w) = q$ gibt.

Definition 4.2.5 (Widersprüchlich festgelegter Zustand)

Den Zustand q eines DFA $M \in M(n, z)$ bezeichnen wir als widersprüchlich festgelegt, wenn er durch ein positives Beispiel w und ein negatives Beispiel w' festgelegt wird. Die beiden Beispiele w und w' nennen wir entsprechend auch widersprüchliche Beispiele.

Aufgrund der identischen Übergangstabelle aller Automaten einer Klasse $M(n, z)$, folgt aus der Festlegung eines Zustandes durch ein klassifiziertes Beispiel w im DFA $M \in M(n, z)$, dass in jedem DFA der Klasse $M(n, z)$ der Zustand mit dem kanonischen Namen $\hat{\delta}(0, w)$ festgelegt ist. Wir sprechen daher kurz davon, dass in der Klasse $M(n, z)$ der Zustand mit dem kanonischen Namen $\hat{\delta}(0, w)$, oder noch kürzer, dass in der Klasse $M(n, z)$ der Zustand $\hat{\delta}(0, w)$ (widersprüchlich) festgelegt sei.

Definition 4.2.6 (Am ehesten konsistenter DFA)

Sei P die Menge positiver Beispiele und N die Menge negativer Beispiele. Den DFA $M \in M(n, z)$ mit $F = \{\delta(q_0, v) \mid v \in P\}$ nennen wir den am ehesten konsistenten DFA.

Lemma 4.2.7 (Konsistenz des am ehesten konsistenten DFA)

Sei P die Menge positiver Beispiele und N die Menge negativer Beispiele. Sei $M \in M(n, z)$ mit $F = \{\delta(q_0, v) \mid v \in P\}$ der am ehesten konsistente DFA für $M(n, z)$. M ist genau dann nicht konsistent, wenn ein Zustand widersprüchlich festgelegt wird.

Beweis: “ \Rightarrow ” Angenommen M ist nicht konsistent, dann gilt entweder $L(M) \cap P \neq P$ oder $L(M) \cap N \neq \emptyset$. Der Fall $L(M) \cap P \neq P$ kann aber nicht eintreten, da gemäß Definition 4.2.6 der am ehesten konsistente DFA alle Worte aus P akzeptiert. Also muss $L(M) \cap N \neq \emptyset$ sein. Dann gibt es ein Wort $w \in L(M) \cap N$. Aus $w \in L(M)$ folgt $\delta(q_0, w) \in F$ und weiter, dass es auch ein $w' \in P$ mit $\delta(q_0, w) = \delta(q_0, w')$ geben muss. Somit sind w und w' unsere widersprüchlichen Beispiele.

“ \Leftarrow ” Es seien $w \in P$ und $w' \in N$ die beiden Beispiele mit $\delta(q_0, w) = \delta(q_0, w')$. Dann gilt $w, w' \in L(M)$ und $w' \in L(M) \cap N$ und damit $L(M) \cap N \neq \emptyset$. Also ist M nicht konsistent. \square

Korollar 4.2.8

Sei wieder P die Menge positiver und N die Menge negativer Beispiele. Der DFA $M \in M(n, z)$ ist nicht konsistent, falls ein Zustand widersprüchlich festgelegt wird.

Beweis: Sei q der widersprüchlich festgelegte Zustand. Seien w und w' die beiden widersprüchlichen Beispiele mit $q = \delta(q_0, w) = \delta(q_0, w')$. O.B.d.A. sei $w \in P$ und $w' \in N$. Den Fall $q \in F$ haben wir in der Rückrichtung des Beweises des Lemma 4.2.7 bereits gezeigt. Angenommen es gilt $q \notin F$. Dann gilt für unsere beiden Beispiele, dass $w, w' \notin L(M)$ und $w \notin L(M) \cap P$. Also ist M nicht konsistent. \square

Die Umkehrung des Korollars 4.2.8 gilt im Allgemeinen natürlich nicht. Allerdings folgt aus der Inkonsistenz des am ehesten konsistenten DFA die widersprüchliche Festlegung eines Zustandes. Und somit können wir aus der Inkonsistenz des am ehesten konsistenten DFA der Klasse $M(n, z)$ eine Aussage über die Lösbarkeit des Konsistenzproblems der Klasse $M(n, z)$ ableiten.

Lemma 4.2.9 (Lösbarkeit des Konsistenzproblems)

Das Konsistenzproblem der Klasse $M(n, z)$ ist genau dann unlösbar, wenn der am ehesten konsistente DFA der Klasse nicht konsistent ist.

Beweis: “ \Leftarrow ” Gemäß Lemma 4.2.7 folgt aus der Inkonsistenz des am ehesten konsistenten DFA, dass ein Zustand durch zwei Beispiele w und w' widersprüchlich festgelegt wird. Aufgrund der für alle DFA der Klasse identischen Übergangsfunktion legen w und w' in jedem DFA $M \in M(n, z)$ den gleichen Zustand fest. Also sind gemäß Korollar 4.2.8 alle DFA der Klasse, unabhängig von ihrer Menge akzeptierender Zustände, nicht konsistent.

“ \Rightarrow ” Wenn das Konsistenzproblem der Klasse $M(n, z)$ unlösbar ist, existiert kein konsistenter DFA $M \in M(n, z)$. Also ist auch der am ehesten konsistente DFA der Klasse nicht konsistent. \square

Zum Abschluss stellen wir mit Lemma 4.2.12 ein wichtiges Hilfsmittel bereit, das Aufschluss darüber gibt, was wir aus der Unlösbarkeit des Konsistenzproblems einer Klasse auf die Lösbarkeit des Konsistenzproblems einer anderen Klasse und die Zahl der festgelegten Zustände schließen können. Ausgehend von Beobachtung 4.1.3, wonach für zwei hinreichend lange Eingabeworte w und w' , mit $|w|, |w'| \geq n - z$, und für alle DFA $M \in M(n, z)$

$$\delta(q_0, w) = \delta(q_0, w') \iff |w| \equiv |w'| \pmod{z}$$

gilt, treffen wir aber erst einmal zwei weitere Beobachtungen.

Beobachtung 4.2.10

Sei $M(n', z')$ gegeben. Für $z \leq z'$ mit $z \mid z'$ und $n \leq n' + z - z'$ gilt, dass zwei Worte w und w' , mit $w \neq w'$, die in $M(n', z')$ den gleichen Zustand festlegen, auch in $M(n, z)$ den gleichen Zustand festlegen.

Beweis: O.B.d.A. sei $|w| < |w'|$. Nur wenn beide Worte den Zyklus erreichen, können sie in einem DFA $M \in M(n', z')$ den gleichen Zustand festlegen. Also gilt $|w| \geq n' - z'$ und aufgrund von Beobachtung 4.1.3 gilt $|w| \equiv |w'| \pmod{z'}$. Daraus folgt, wegen $z \mid z'$, aber sofort $|w| \equiv |w'| \pmod{z}$. Weiter gilt $|w| \geq n' - z' \geq n - z$. Also legen beide Worte auch in $M(n, z)$ den gleichen Zustand fest. \square

Beobachtung 4.2.11

Sei $M(n, z)$ gegeben. Für $z \leq z'$ mit $z \mid z'$ und $n \leq n' + z - z'$ gilt, dass zwei Worte w und w' , die in $M(n, z)$ zwei unterschiedliche Zustände festlegen, auch in $M(n', z')$ zwei unterschiedliche Zustände festlegen.

Beweis: Diese Beobachtung ist die Kontraposition zu Beobachtung 4.2.10. \square

Lemma 4.2.12

Gegeben sei eine Menge klassifizierter Beispiele. Seien $n > 0$ und $1 \leq z \leq n$ beliebig aber fest. Für $z' \geq z$ mit $z \mid z'$ und $n' \geq n - z + z'$ gilt:

- i. In $M(n', z')$ sind mindestens so viele Zustände festgelegt wie in $M(n, z)$.
- ii. Ist das Konsistenzproblem für $M(n, z)$ unlösbar, dann ist das Konsistenzproblem für $M(n', z')$ ebenfalls unlösbar oder es wird in $M(n', z')$ mindestens ein Zustand mehr festgelegt, als in $M(n, z)$.
- iii. Ist das Konsistenzproblem für $M(n', z')$ unlösbar, dann ist das Konsistenzproblem für $M(n, z)$ ebenfalls nicht lösbar.
- iv. Ist das Konsistenzproblem für $M(n, z)$ lösbar, dann ist auch das Konsistenzproblem für $M(n - z + z', z')$ lösbar.
- v. Ist das Konsistenzproblem $M(n, z')$ unlösbar, dann ist das Konsistenzproblem für $M(n - z' + z, z)$ ebenfalls nicht lösbar.

Beweis: “i)” Dies folgt direkt aus Beobachtung 4.2.11.

“ii)” Seien w, w' , $w \neq w'$, zwei Beispiele, die in $M(n, z)$ einen Zustand widersprüchlich festlegen. O.B.d.A. sei $|w| < |w'|$. Gilt $|w| < n' - z'$, dann erreicht w in $M(n', z')$ den Zyklus nicht und es wird in $M(n', z')$ ein Zustand mehr festgelegt als in $M(n, z)$. Sei also $|w| \geq n' - z'$. Falls $|w| \equiv |w'| \pmod{z'}$ wird in $M(n', z')$ ein Zustand widersprüchlich festgelegt und das Gewünschte ist gezeigt. Sei also $|w| \not\equiv |w'| \pmod{z'}$. Dann erreichen w, w' in $M(n', z')$ verschiedene Zustände. Beide Zustände können aber nur von einem Wort w'' festgelegt werden, für das $|w''| \equiv |w'| \equiv |w| \pmod{z}$ gilt. Also wird in $M(n', z')$ ein Zustand mehr festgelegt als in $M(n, z)$.

“iii)” Folgt direkt aus Beobachtung 4.2.10, da die beiden widersprüchlichen Beispiele auch in der Klasse mit weniger Zuständen den gleichen Zustand festlegen.

“iv)” Trivial. Nur die Beispiele die in $M(n, z)$ den Zyklus erreichen, erreichen auch den Zyklus in $M(n - z + z', z')$.

“v)” Seien w und w' zwei Beispiele, die in der Klasse $M(n, z')$ einen Zustand widersprüchlich festlegen. O.B.d.A. sei $|w| < |w'|$. Es gilt $|w| \geq n - z'$ und damit erreicht w in $M(n - z' + z, z)$ einen Zustand im Zyklus. Wegen $|w| \equiv |w'| \pmod{z'}$ und $z \mid z'$ erreichen w und w' in $M(n - z' + z, z)$ den gleichen Zustand; also ist das Konsistenzproblem nicht lösbar. \square

Bei den Aussagen (iv) und (v) des Lemma 4.2.12 erweist sich unsere Notation der Klasse $M(n, z)$ als etwas schwerfällig. Leider sieht man deshalb nicht sofort, dass wir hier zwei einfache Aussagen über Klassen treffen, die einen gleichlangen Anfangspfad haben.

4.3 VC-Dimension und Beispielskomplexität

Bei der Vorstellung des PAC-Lernens haben wir gesehen, wie wichtig die VC-Dimension einer Konzeptklasse für die Zahl der anzufordernden Beispiele ist. Wir werden mit Satz 4.3.5 für die VC-Dimension der regulären unären Sprachen eine obere und eine untere Schranke angeben, die sich nur um einen additiven Term von $O(\log(\ln(n)))$ unterscheiden. Damit können wir dann auch sofort eine obere Schranke der Beispielskomplexität bestimmen.

Definition 4.3.1

Wir definieren die Konzeptklassen

$$L_{n,z} = \left\{ L(M) \mid M \in M(n, z) \right\} \quad \text{und} \quad L_n = \bigcup_{1 \leq z \leq n} L_{n,z}.$$

Die Klasse L_n ist also die Menge aller regulären unären Sprachen, die von einem DFA mit höchstens n Zuständen akzeptiert werden.

Lemma 4.3.2 (VC-Dimension von $L_{n,z}$)

Es gilt

$$\text{VC}(L_{n,z}) = n.$$

Beweis: $\text{VC}(L_{n,z}) \geq n$ ist offensichtlich, da die Menge $S = \{a^0, a^1, a^2, \dots, a^{n-1}\}$ von $L_{n,z}$ zertrümmert wird. Wir müssen also nur noch zeigen, dass keine Menge mit $n+1$ Elementen durch $L_{n,z}$ zertrümmert wird. Sei $S = \{w_1, w_2, \dots, w_{n+1}\}$, mit $w_i \in \Sigma^*$. Dann existieren zwei Worte $w_{i_1}, w_{i_2} \in S$, die in $M(n, z)$ den gleichen Zustand erreichen. Sei $M \in M(n, z)$ ein beliebiger DFA der Klasse. Es gilt $w_{i_1} \in L(M) \Leftrightarrow w_{i_2} \in L(M)$. Also ist die Menge $\{w_{i_1}\} \subseteq 2^S$ kein Konzept. \square

Lemma 4.3.3

Es gilt für $n \geq 5$

$$\text{VC}(L_n) \geq n + 1.$$

Beweis: Sei $n \geq 5$ beliebig aber fest. Wir beweisen die stärkere Aussage $\text{VC}(L_{n,2} \cup L_{n,3} \cup L_{n,5}) \geq n + 1$. Sei $S = S' \cup S''$ mit $S' = \{a^0, a^1, a^2, \dots, a^{n-3}\}$ und $S'' = \{w_1, w_2, w_3\}$. Zur Wahl der w_i kommen wir später. Wir fordern aber $\delta(0, w) \in \{n-1, n-2\}$ für $w \in S''$ und alle $M \in M(n, z)$, mit $z \in \{2, 3, 5\}$. Wir müssen nach Definition 3.2.11 zeigen, dass

$$2^S = \{c \cap S \mid c \in L_{n,2} \cup L_{n,3} \cup L_{n,5}\}$$

gilt. Dies können wir aufgrund der Definition der Konzeptklassen auch als

$$2^S = \{L(M) \cap S \mid M \in M(n, z) \text{ für } z \in \{2, 3, 5\}\}$$

formulieren. Da die Worte aus S'' in allen DFA $M \in M(n, z)$, mit $z \in \{2, 3, 5\}$, nur die Zustände $n-1$ und $n-2$ erreichen, sehen wir sofort, dass für ein beliebiges aber festes z

$$2^{S'} = \{L(M) \cap S \mid M \in M(n, z) \text{ mit } F \subseteq \{0, 1, \dots, n-3\}\}$$

gilt. Uns fehlen noch die Teilmengen aus $2^S \setminus 2^{S'}$. Angenommen, wir wählen $w_1, w_2, w_3 \in S''$ so, dass in den drei Klassen $M(n, 2)$, $M(n, 3)$ und $M(n, 5)$ bei Eingabe dieser Worte die folgenden Zustände erreicht werden

	w_1	w_2	w_3
$M(n, 2)$	$n-1$	$n-2$	$n-2$
$M(n, 3)$	$n-2$	$n-1$	$n-2$
$M(n, 5)$	$n-2$	$n-2$	$n-1$

dann erhalten wir für die Teilmengen von S''

$$\begin{aligned} \{\emptyset, \{w_1\}, \{w_2, w_3\}, \{w_1, w_2, w_3\}\} &\subseteq L_{n,2}, \\ \{\emptyset, \{w_2\}, \{w_1, w_3\}, \{w_1, w_2, w_3\}\} &\subseteq L_{n,3}, \\ \{\emptyset, \{w_3\}, \{w_1, w_2\}, \{w_1, w_2, w_3\}\} &\subseteq L_{n,5}. \end{aligned}$$

Außerdem können wir zu jeder dieser Teilmengen von S'' noch beliebige Teilmengen von S' “zuschalten” (da wir für S'' die Zustände $n - 2$ und $n - 1$, für S' die Zustände $0, 1, \dots, n - 3$ “benutzen”). Damit können wir zeigen, dass für jede Teilmenge $c \subseteq S$ ein DFA $M \in M(n, z)$, mit $z \in \{2, 3, 5\}$, existiert, so dass $L(M) \cap S = c$ ist.

Wir müssen zum Abschluss nur noch zeigen, dass diese Wahl von w_1, w_2 und w_3 auch möglich ist. Diese Anforderung läßt sich aber direkt als System simultaner linearer Kongruenzen ausdrücken

$$\begin{array}{lll} |w_1| \equiv n - 1 \pmod{2} & |w_1| \equiv n - 2 \pmod{3} & |w_1| \equiv n - 2 \pmod{5} \\ |w_2| \equiv n - 2 \pmod{2} & |w_2| \equiv n - 1 \pmod{3} & |w_2| \equiv n - 2 \pmod{5} \\ |w_3| \equiv n - 2 \pmod{2} & |w_3| \equiv n - 2 \pmod{3} & |w_3| \equiv n - 1 \pmod{5} \end{array}$$

dessen Lösung für $|w_1|, |w_2|$ und $|w_3|$ nach dem dem Chinesischen Restsatz, Satz 2.1.4, existiert und modulo 30 eindeutig ist. \square

Wir wollen das Ergebnis aus Lemma 4.3.3 am Beispiel von L_5 verdeutlichen. Wie man an Abbildung 4.2 sehr schön sieht, wird die Menge $S = \{a^0, a^1, a^2, a^{43}, a^{69}, a^{78}\}$ mit $|S| = 6$ von L_5 zertrümmert. Dazu werden die einelementigen Teilmengen von $S'' = \{a^{43}, a^{69}, a^{78}\}$ im letzten Zustand, die zweielementigen im vorletzten Zustand kodiert.

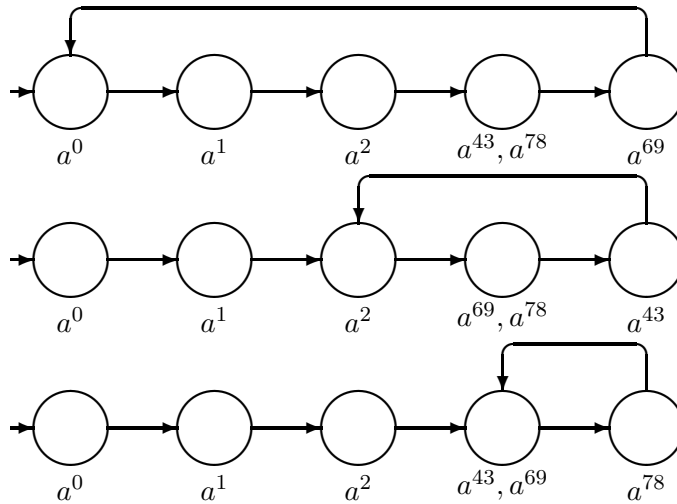


Abbildung 4.2: Beispiel der Zertrümmerung von $S = \{a^0, a^1, a^2, a^{43}, a^{69}, a^{78}\}$ durch L_5

Natürlich sind wir nicht darauf beschränkt, nur eine Menge mit drei Worten in den beiden letzten Zuständen zu kodieren.

Lemma 4.3.4

Gegeben sei eine Menge S . Dann existieren $s = 2^{|S|-1} - 1$ Paare von disjunkten Teilmengen $S_i, \bar{S}_i \subseteq S$, $1 \leq i \leq s$, mit $S_i \cup \bar{S}_i = S$ und $S_i \cap \bar{S}_i = \emptyset$, so dass

$$\bigcup_{1 \leq i \leq s} \{S_i, \bar{S}_i\} = 2^S \setminus \{S, \emptyset\}.$$

Beweis: Es gibt genau $2^{|S|}$ Teilmengen von S . Außerdem können wir zu jedem $S_i \subseteq S$ ein $\bar{S}_i = S \setminus S_i$ finden, dass die Anforderung $S_i \cup \bar{S}_i = S$ und $S_i \cap \bar{S}_i = \emptyset$ erfüllt. Streichen wir die leere Menge und S , erhalten wir $2^{|S|-1} - 1$ Paare. \square

Satz 4.3.5 (VC-Dimension für L_n)

Es gilt für $n \geq 5$

$$\text{VC}(L_n) \geq n - 1 + \lfloor \log(\pi(n) + 1) \rfloor$$

und

$$\text{VC}(L_n) \leq n + \log(n),$$

wobei $\pi(n)$ die Anzahl der Primzahlen kleiner oder gleich n ist.

Beweis: Wir zeigen zuerst $\text{VC}(L_n) \geq n - 1 + \lfloor \log(\pi(n) + 1) \rfloor$. Dazu bauen wir auf der Konstruktion im Beweis von Lemma 4.3.3 auf. Wir zertrümmern die Menge $S = S' \cup S''$, mit $S' \cap S'' = \emptyset$. Dabei setzen wir $S' = \{a^0, a^1, \dots, a^{n-3}\}$. Nach Lemma 4.3.4 müssen wir für eine Menge S'' $2^{|S''|-1} - 1$ Paare unterbringen, um alle Teilmengen von S'' zu kodieren. Für jedes Paar brauchen wir eine Klasse $M(n, z)$. Wir verwenden dabei nur die Klassen $M(n, z)$ mit primärer Zykluslänge z , damit das System simultaner linearer Kongruenzen das wir erhalten, um die Worte in S'' zu berechnen, sicher lösbar ist. Dies sind insgesamt $\pi(n)$ verschiedene Klassen. Für die Kardinalität von S'' ergibt sich also

$$\pi(n) = 2^{|S''|-1} - 1 \Leftrightarrow \log(\pi(n) + 1) + 1 = |S''|.$$

Insgesamt erhalten wir eine Kardinalität von S mit

$$|S| = |S'| + |S''| = n - 2 + \log(\pi(n) + 1) + 1 = n - 1 + \log(\pi(n) + 1).$$

Nun zeigen wir noch $\text{VC}(L_n) \leq n + \log(n)$. Es gilt $|L_n| \leq n \cdot 2^n$. Da die VC-Dimension mit $\text{VC}(L_n) \leq \log(|L_n|)$ abgeschätzt werden kann [BEHW89], ergibt sich $\text{VC}(L_n) \leq \log(n \cdot 2^n) = n + \log(n)$. \square

Schätzen wir $\pi(n)$ nach Satz 2.1.3 mit $n/\ln(n)$ ab, so erhalten wir, dass sich die obere und die untere Schranke aus Satz 4.3.5 nur durch einen additiven Term von $O(\log(\ln(n)))$ unterscheiden. Die Abschätzung von $|L_n|$ für die obere Schranke ist übrigens optimal, da nach [DKS02] $|L_n| = \Omega(n \cdot 2^n)$ gilt.

Mit dem Ergebnis aus Satz 4.3.5 und mit Satz 3.2.16 können wir dann auch die Beispielskomplexität für das PAC-Lernen regulärer unärer Sprachen durch deterministische endliche Automaten bestimmen mit

$$\text{beispiel}_{\mathcal{C}}(\epsilon, \delta) = O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) + \frac{n + \log(n)}{\epsilon} \ln\left(\frac{1}{\epsilon}\right)\right)$$

für $\mathcal{C} = L_n$. Da die VC-Dimension polynomiell in n wächst und, wie wir noch in Kapitel 5 zeigen werden, ein polynomieller Algorithmus zur Konstruktion einer konsistenten Hypothese existiert, sind die regulären unären Sprachen sogar effizient PAC-lernbar.

Kapitel 5

Konsistente Hypothesen für reguläre unäre Sprachen

Wir haben bereits bei der Vorstellung der Lernmodelle gesehen, wie wichtig die Berechenbarkeit der konsistenten Hypothese für das PAC-Lernen und das Lernen mit Äquivalenz- und Elementfragen ist. So ist etwa jeder Lernalgorithmus für die Konzeptklasse \mathcal{C} , der immer eine konsistente Hypothese ausgibt und mehr als $\text{beispiel}_{\mathcal{C}}(\epsilon, \delta)$ viele Beispiele anfordert ein PAC-Algorithmus (für ϵ und δ). Beim Lernen mit Äquivalenzfragen ist das Aufstellen einer konsistenten Hypothese notwendig, da das Orakel, welches in der Wahl seines Gegenbeispiels ja völlig frei ist, sonst immer eines der bereits bekannten Beispiele als Antwort geben kann.

Wir betrachten daher das Berechnen einer konsistenten Hypothese für reguläre unäre Sprachen. Zuerst stellen wir eine obere Schranke für die Zahl der Zustände des minimalen mit den klassifizierten Beispielen konsistenten DFA auf. Alsdann entwickeln wir diese Betrachtung weiter zu einem Algorithmus zur Berechnung der minimalen konsistenten Hypothese, der auf einer Registermaschine mit uniformen Kostenmaß [AHU74] $O(n \cdot |S|)$ viele Operationen braucht, wobei n die Zahl der Zustände des minimalen DFA für die zu erlernende Sprache L und S die Menge der klassifizierten Beispiele ist.

5.1 Obere Schranke der Zustandszahl des minimalen konsistenten DFA

Wir wissen, dass das Konsistenzproblem der Klasse $M(n, z)$ genau dann nicht lösbar ist, wenn der am ehesten konsistente DFA $M \in M(n, z)$ nicht konsistent ist. Dazu muss es nach Lemma 4.2.7 und Beobachtung 4.1.3 zwei widersprüchliche Beispiele geben, die in M den gleichen Zustand (im Zyklus) erreichen und hinsichtlich ihrer Länge kongruent modulo z sind. Wählen wir also die Zustandszahl n so groß, dass in Automaten der Klasse $M(n, z)$ entweder nur positive oder nur negative Beispiele den Zyklus erreichen, ist der am ehesten konsistente DFA der Klasse trivialerweise konsistent. Davon ausgehend erhalten wir eine erste Abschätzung für die maximale Zustandszahl des minimalen konsistenten DFA.

Definition 5.1.1

Gegeben sei eine endliche Menge $S \subseteq \Sigma^*$. Mit $\lambda(S)$ bezeichnen wir die Länge des längsten Wortes aus S

$$\lambda(S) = \begin{cases} \max(\{m \mid m = |s| \wedge s \in S\}) & \text{falls } S \neq \emptyset \\ -1 & \text{sonst} \end{cases}.$$

Wir setzen $\lambda(\emptyset) = -1$, da $\lambda(\{\epsilon\}) = 0$ und es uns, wie wir später sehen, einige Berechnungen vereinfacht.

Lemma 5.1.2 (Obere Schranke der Zustandszahl)

Sei P die Menge positiver Beispiele und N die Menge negativer Beispiele. Dann ist die Zahl der Zustände des minimalen konsistenten DFA beschränkt durch

$$\leq \min(\lambda(P), \lambda(N)) + 2.$$

Beweis: Sei $n = \min(\lambda(P), \lambda(N)) + 2$. Wir wählen den am ehesten konsistenten DFA $M \in M(n, 1)$. Der Zustand $n - 1$, der einzige im Zyklus, wird nur durch Beispiele aus einer der beiden Mengen festgelegt, da die Beispiele der anderen Menge aufgrund der Wahl von n zu kurz sind. Somit gibt es keine widersprüchlichen Beispiele die den gleichen Zustand in M festlegen. Nach Lemma 4.2.7 folgt daraus die Konsistenz von M . \square

Korollar 5.1.3

Sei P die Menge positiver Beispiele und N die Menge negativer Beispiele. Sei $z > 0$ beliebig aber fest. Dann ist die Zahl der Zustände des minimalen konsistenten DFA mit vorgegebener Zykluslänge z beschränkt durch

$$\leq \min(\lambda(P), \lambda(N)) + z + 1.$$

Beweis: Sei $n = \min(\lambda(P), \lambda(N)) + z + 1$. Wir wählen den am ehesten konsistenten DFA $M \in M(n, z)$. Wiederum werden alle Zustände im Zyklus nur durch die Beispiele einer Menge festgelegt, da die Beispiele aus der zweiten Menge zu kurz sind. Also ist M konsistent. \square

Man erliegt leicht der Vermutung, dass $n \leq \min(\lambda(P), \lambda(N)) + 1$ bei Wahl des am ehesten konsistenten DFA $M \in M(n, n)$ eine bessere obere Schranke ist. Auch wenn dies in einzelnen Fällen zutrifft, so gilt dies nicht allgemein, wie man anhand der Mengen $P = \{a^1\}$ und $N = \{a^5\}$ leicht nachprüfen kann. Eine bessere obere Schranke muss die Restklasseneigenschaften der Beispiele berücksichtigen; damit sind wir aber dann auch gleich bei der genauen Zustandszahl des minimalen konsistenten DFA, die in Lemma 5.2.2 und Korollar 5.2.3 gezeigt wird. Tatsächlich können wir sogar nachweisen, dass unsere Schranke scharf in dem Sinne ist, dass es Beispielmengen gibt, für die die Zahl der Zustände des minimalen konsistenten DFA die Schranke erreicht.

Lemma 5.1.4 (Schärfe der oberen Schranke)

Es existiert eine Menge positiver Beispiele P und eine Menge negativer Beispiele N , so dass der minimale konsistente DFA

$$\min(\lambda(P), \lambda(N)) + 2$$

Zustände benötigt.

Beweis: Sei $n > 0$ beliebig aber fest. Wir konstruieren eine Menge positiver Beispiele P und eine Menge negativer Beispiele N und zeigen, dass der am ehesten konsistente DFA $M \in M(n, 1)$ mit $F = \{\delta(0, v) \mid v \in P\}$ der minimale konsistente DFA ist. Wir werden gleich zwei Beweise angeben. Der erste basiert auf einer großen Zahl von Beispielen, der zweite auf zwei sehr langen Beispielen.

“a)“ Wir setzen $P = \{a^0, a^1, a^2, \dots, a^{n-2}\}$ und $N = \{a^{n-1}\}$. Zuerst zeigen wir, dass M konsistent ist. Die positiven Beispiele legen die Zustände $0, 1, 2, \dots, n - 2$, das negative Beispiel aber den Zustand $n - 1$, den Zustand im Zyklus, fest. Es wird also kein Zustand durch ein positives und ein negatives Beispiel festgelegt. Nach Lemma 4.2.7 ist M dann konsistent. Wir

zeigen nun, dass es keinen kleineren konsistenten DFA gibt. Sei $n' < n$ beliebig aber fest. Dann gilt für $1 \leq z' \leq n'$, dass jeder Zustand durch wenigstens ein positives Beispiel festgelegt wird. Also muss das negative Beispiel einen Zustand festlegen, der bereits durch ein positives festgelegt wird. Somit kann kein $M' \in M(n', z')$ konsistent sein. Also ist M der minimale konsistente DFA.

“b)” Wir setzen $P = \{a^{n-2}\}$ und $N = \{a^{n-2+n!}\}$. M ist trivialerweise konsistent. Sei nun $n' < n$ beliebig aber fest. Dann gilt für alle DFA $M' \in M(n', z')$, mit $1 \leq z' \leq n'$, dass $|a^{n-2}|, |a^{n-2+n!}| \geq n' - z'$ und $|a^{n-2}| \equiv |a^{n-2+n!}| \pmod{z'}$. Nach Lemma 4.2.7 und Beobachtung 4.1.3 ist M' also nicht konsistent. Also muss M der minimale konsistente DFA sein. \square

Die in Lemma 5.1.4 angeführten Beispielmengen weisen eine interessante Gemeinsamkeit auf: beide enthalten das Beispiel a^{n-2} , das den Zustand direkt vorm Zyklus festlegt. Tatsächlich kann man, wie später in Beobachtung 5.2.5, zeigen, dass aus der Minimalität und der Konsistenz des DFA $M \in M(n, z)$ folgt, dass a^{n-z-1} ein Beispiel ist, und dass $q_{n-z-1} \in F \Leftrightarrow q_{n-1} \notin F$ gilt.

5.2 Berechnung des minimalen konsistenten DFA

Wir untersuchen zuerst, wieviele Zustände der minimale konsistente DFA mit vorgegebener Zykluslänge haben muss. Dadurch erhalten wir mit Lemma 5.2.2 sogleich auch eine Berechnungsvorschrift. Darauf aufbauend können wir dann mit Algorithmus 5.2.1 die Berechnung des minimalen konsistenten DFA beschreiben.

Wie bereits mehrfach erwähnt, folgt die Unlösbarkeit des Konsistenzproblems der Klasse $M(n, z)$ letztlich aus zwei widersprüchlichen Beispielen w, w' mit $|w| \equiv |w'| \pmod{z}$. Daher bietet es sich an, statt nur die Länge der Beispiele, auch die Restklasse der Länge modulo z heranzuziehen.

Definition 5.2.1

Sei $S \subseteq \Sigma^*$ eine endliche Menge. Sei $z > 0$ beliebig aber fest. Mit $S_{r,z}$, für $0 \leq r < z$, bezeichnen wir die Menge der Worte aus S , die ihrer Länge nach modulo z in die Restklasse r fallen, also

$$S_{r,z} = \{s \mid s \in S \wedge |s| \bmod z = r\}.$$

Lemma 5.2.2 (Zustandszahl des minimalen konsistenten DFA mit vorgeg. Zyklusl.)

Sei P die Menge positiver Beispiele und N die Menge negativer Beispiele. Sei $z > 0$ beliebig aber fest. Der minimale konsistente DFA mit fester Zykluslänge z benötigt

$$\max \left(\min \left(\lambda(P_{0,z}), \lambda(N_{0,z}) \right) + z + 1, \dots, \min \left(\lambda(P_{z-1,z}), \lambda(N_{z-1,z}) \right) + z + 1 \right)$$

Zustände, wobei $P_{r,z}$ und $N_{r,z}$, mit $0 \leq r < z$, Teilmengen von P und N gemäß Definition 5.2.1 sind.

Beweis: Sei $n = \max(\min(\lambda(P_{0,z}), \lambda(N_{0,z})) + z + 1, \dots, \min(\lambda(P_{z-1,z}), \lambda(N_{z-1,z})) + z + 1)$. Sei M der am ehesten konsistente DFA für $M(n, z)$. Wir zeigen, dass M der minimale konsistente DFA mit vorgegebener Zykluslänge z ist.

Angenommen, M ist nicht konsistent. Dann gibt es zwei widersprüchliche Beispiele w, w' mit $|w|, |w'| \geq n - z$ und $|w| \equiv |w'| \pmod{z}$. O.B.d.A. gelte $|w| < |w'|$. Sei $r = |w| \bmod z$. Es gilt $\lambda(P_{r,z}) \geq |w|$ und $\lambda(N_{r,z}) \geq |w|$. Also gilt auch $\min(\lambda(P_{r,z}), \lambda(N_{r,z})) \geq |w|$. Also wurde $n \geq \min(\lambda(P_{r,z}), \lambda(N_{r,z})) + z + 1 \geq |w| + z + 1$ gewählt. Damit erhalten wir aber wegen $|w| \geq n - z$ mit $n \geq |w| + z + 1 \geq n + 1$ einen Widerspruch zur Wahl von n . Also ist M konsistent.

Für $n = z$ ist M trivialerweise minimal. Also gelte $n > z$. Gemäß der Wahl von n muss es zwei widersprüchliche Beispiele w, w' mit $|w| \equiv |w'| \pmod{z}$ und $n = \min(|w|, |w'|) + z + 1$

geben. O.B.d.A. sei $|w| < |w'|$. Sei $n' < n$ beliebig aber fest. Es gilt $|w| = n - z - 1 \geq n' - z$. Dann legen w und w' in $M(n', z)$ den gleichen Zustand fest. Also ist M' nicht konsistent. Daraus folgt, dass M der minimale konsistente DFA mit vorgegebener Zykluslänge z ist. \square

Korollar 5.2.3 (Zustandszahl des minimalen konsistenten DFA)

Sei n die Zahl der Zustände des minimalen DFA für die zu lernende Sprache L . Sei S eine Menge klassifizierter Beispiele. Dann benötigt der minimale mit S konsistente DFA

$$\min(k_1, k_2, k_3, \dots, k_n)$$

Zustände, wobei k_z , für $1 \leq z \leq n$, die Zustandszahl des minimalen konsistenten DFA mit vorgegebener Zykluslänge z ist.

Beweis: Trivial. Wenn der minimale DFA für L n Zustände hat, dann hat der minimale mit S konsistente DFA höchstens n Zustände, da der minimale DFA für L ein mit S konsistenter DFA ist. \square

Korollar 5.2.3 folgend, zählen wir zur Berechnung des minimalen konsistenten DFA die minimalen konsistenten DFA mit vorgegebener Zykluslänge z für $z = 1, 2, 3, \dots$ auf.

Eingabe: Menge positiver Beispiele P , Menge negativer Beispiele N

Ausgabe: Minimaler mit P und N konsistenter DFA

- 1: $z \leftarrow 0$
- 2: **repeat**
- 3: /* Alle M_i , mit $1 \leq i \leq z$, haben wenigstens $z + 1$ Zustände */
- 4: $z \leftarrow z + 1$
- 5: /* Berechne den minimalen konsistenten DFA mit fester Zykluslänge z (Lemma 5.2.2) */
- 6: Berechne die Mengen $P_{r,z}$ und $N_{r,z}$ für $0 \leq r < z$
- 7: Berechne $n_r = \min(\lambda(P_{r,z}), \lambda(N_{r,z})) + z + 1$ für $0 \leq r < z$
- 8: $n \leftarrow \max(n_0, n_1, n_2, \dots, n_{z-1})$
- 9: /* Wähle den am ehesten konsistenten DFA $M_z \in M(n, z)$ */
- 10: $M_z \leftarrow M \in M(n, z)$ mit $F = \{\delta(0, v) \mid v \in P\}$
- 11: /* M_z hat mindestens z Zustände */
- 12: /* Alle M_i , mit $1 \leq i \leq z$, haben wenigstens z Zustände */
- 13: **until** es existiert ein i , mit $1 \leq i \leq z$, so dass M_i z Zustände hat
- 14: $M \leftarrow M_i$

Algorithmus 5.2.1: Berechnung des minimalen konsistenten DFA

Satz 5.2.4 (Korrektheit und Laufzeit des Algorithmus 5.2.1)

Sei $L \subseteq \Sigma^*$ eine reguläre unäre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Sei $P \subseteq L$ die Menge positiver Beispiele und $N \subseteq \Sigma^* - L$ die Menge negativer Beispiele. Algorithmus 5.2.1 berechnet den minimalen konsistenten DFA. Hierzu sind auf einer Registermaschine mit uniformen Kostenmaß, bei der die Längenberechnung eines Wortes als Operation zählt,

$$n \cdot 3 \cdot (|P| + |N|) = O(n \cdot (|P| + |N|))$$

Berechnungsschritte nötig.

Beweis: Der minimale DFA für L muss in einer der Klassen $M(n, 1), M(n, 2), \dots, M(n, n)$ sein. Damit ist n eine obere Schranke für Zustandszahl und Zykluslänge des minimalen konsistenten DFA. Algorithmus 5.2.1 berechnet daher nacheinander den minimalen konsistenten DFA

mit vorgegebener Zykluslänge für die Zykluslänge $1, 2, 3, \dots, n$ bis einer der bereits berechneten konsistenten minimalen DFA mit vorgegebener Zykluslänge kleiner als jeder noch nicht berechnete ist (was wir leicht prüfen können, da der minimale konsistente DFA mit vorgegebener Zykluslänge z wenigstens z Zustände haben muss).

Die Korrektheit der Berechnung des minimalen konsistenten DFA mit vorgegebener Zykluslänge folgt aus Lemma 5.2.2. Wir müssen also nur noch zeigen, dass wir den minimalen konsistenten DFA finden. Wir betrachten den Schleifendurchlauf in den Zeilen 5 bis 12 für ein beliebiges aber festes z . Als Vorbedingung der Schleife gilt, dass M_i , der minimale konsistente DFA für die vorgegebene Zykluslänge i , mit $1 \leq i < z$, wenigstens z Zustände hat. Der minimale konsistente DFA mit vorgegebener Zykluslänge z muss ebenfalls mindestens z Zustände aufweisen. Vor dem Ende der Schleife in Zeile 12 gilt also, dass alle M_i , für $1 \leq i \leq z$, wenigstens z Zustände haben. Gibt es einen mit genau z Zuständen, terminieren wir (spätestens wenn wir M_n berechnet haben). Ansonsten gilt als Nachbedingung, dass alle M_i , für $1 \leq i < z + 1$, wenigstens $z + 1$ Zustände haben.

Die Laufzeitabschätzung ergibt sich im zweiten Faktor $3 \cdot (|P| + |N|)$ aus der Berechnung des minimalen konsistenten DFA mit vorgegebener Zykluslänge. Dazu muss in Zeile 6 zur Berechnung der Mengen $P_{r,z}$ und $N_{r,z}$ und in Zeile 7 bei der Berechnung der λ -Funktion für jedes Beispiel genau eine Längenberechnung durchgeführt werden. Abschließend muss noch in Zeile 11 bei der Bestimmung der Menge der akzeptierenden Zustände jedes (positive) Beispiel nochmals "angefasst" werden. Wir müssen also nur noch abschätzen, wie oft die Schleife durchlaufen wird. Dies ist natürlich durch die Zustandszahl des minimalen DFA für L oder unsere obere Schranke aus Lemma 5.1.2 beschränkt. \square

Die Laufzeit des Algorithmus 5.2.1 beruht auf der Berechnung des minimalen konsistenten DFA mit vorgegebener Zykluslänge. Dessen Berechnung wird sich asymptotisch wohl nicht verbessern lassen, da wir, wenn wir nicht alle Beispiele berücksichtigen, nicht notwendigerweise konsistent sind. Daher bleibt für eine Verbesserung der Laufzeit des Algorithmus 5.2.1 nur, die Zahl der zu berechnenden minimalen konsistenten DFA mit vorgegebener Zykluslänge zu reduzieren.

Beobachtung 5.2.5

Sei S eine Menge klassifizierter Beispiele. Sei $M \in M(n, z)$ der minimale zu S konsistente DFA. Dann existieren ein positives und ein negatives Beispiel $w, w' \in S$ mit $w = a^{n-z-1}$, $w' = a^{n-1+iz}$, mit $i \geq 0$, und $|w| \equiv |w'| \pmod{z}$.

Beweis: Angenommen, a^{n-z-1} ist kein Beispiel. Dann ist der Zustand $n - z - 1$, der Zustand direkt vor dem Zyklus, nicht festgelegt. Dann ist aber das Konsistenzproblem für $M(n - 1, z)$ lösbar und M nicht der minimale konsistente DFA. Sei nun $w = a^{n-z-1}$ ein Beispiel. Angenommen es existiert kein zu w widersprüchliches Beispiel $w' = a^{n-1+iz}$, mit $i \geq 0$, das den Zustand $n - 1$ erreicht, dann wäre wieder das Konsistenzproblem für $M(n - 1, z)$ lösbar und M nicht der minimale konsistente DFA. \square

Beobachtung 5.2.6

Sei S eine Menge klassifizierter Beispiele. Sei $M \in M(n, z)$ der minimale zu S konsistente DFA mit vorgegebener Zykluslänge z . Seien w, w' das positive und das negative Beispiel aus Beobachtung 5.2.5. Dann gilt für $z' \geq z$, mit $z \mid z'$, und den minimalen konsistenten DFA $M' \in M(n', z')$ für die vorgegebene Zykluslänge z' , dass

- i. M' hat höchstens $n' \leq n - z + z'$ Zustände, und
- ii. M' braucht mindestens $n' \geq n - z + z'$ Zustände, falls $|w| \equiv |w'| \pmod{z'}$.

Beweis: “*i*” Gemäß Lemma 4.2.12 folgt aus der Lösbarkeit des Konsistenzproblems der Klasse $M(n, z)$ die Lösbarkeit des Konsistenzproblems der Klasse $M(n - z + z', z')$.

“*ii*” Falls $|w| \equiv |w'| \pmod{z'}$ muss der minimale konsistente DFA mit Zykluslänge z' so groß sein, dass das kürzere Wort $w = a^{n-z-1}$ den Zyklus nicht erreicht, da ansonsten ein Zustand durch w und w' widersprüchlich festgelegt wird. Also muss $n' - z' > |w| = n - z - 1$ und damit $n' \geq n - z + z'$ sein. \square

Modifizieren wir Algorithmus 5.2.1 derart, dass er als Zusatzinformation die beiden Beispiele w, w' aus Beobachtung 5.2.5 protokolliert, dann kann er vor der Berechnung des minimalen konsistenten DFA mit vorgegebener Zykluslänge z' mit wenig Aufwand prüfen, ob diese Berechnung sich “lohnt”. Dazu prüft er für jedes $z < z'$ mit $z \mid z'$, ob $|w| \equiv |w'| \pmod{z'}$. Falls dies zutrifft, braucht er den minimalen konsistenten DFA mit vorgegebener Zykluslänge z' nicht berechnen, da der nach Beobachtung 5.2.6 mindestens soviele Zustände aufweisen muss wie der minimale konsistente DFA mit vorgegebener Zykluslänge z . Allerdings ergibt sich daraus erstmal keine asymptotische Verbesserung der Laufzeit. Daher wollen wir auf den modifizierten Algorithmus nicht weiter eingehen.

Kapitel 6

Aktives Lernen regulärer unärer Sprachen

In Abschnit 3.3 haben wir Angluins Algorithmus 3.3.1 vorgestellt, der eine reguläre Sprache mit Äquivalenz- und Elementfragen lernt. Wir haben dort auch gesehen, dass reguläre Sprachen alleine mit Äquivalenz- oder alleine mit Elementfragen nicht effizient gelernt werden können. In diesem Kapitel werden wir daher das Lernen regulärer unärer Sprachen untersuchen. Im ersten Abschnitt widmen wir uns dem Lernen mit Äquivalenzfragen. Wir werden sehen, dass wir höchstens $O(n^2)$ und mindestens $\Omega(n^2/\ln(n))$ Äquivalenzfragen stellen müssen, um eine reguläre unäre Sprache L zu erlernen, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Im nächsten Abschnitt beschäftigen wir uns dann mit dem Lernen mit Elementfragen. Ist uns n , die Zahl der Zustände eines (nicht notwendigerweise minimalen) DFA für die zu lernende Sprache L , bekannt, können wir zeigen, dass $2n - 1$ Elementfragen genügen, um L zu lernen. Ist n dagegen nicht bekannt, benötigen wir, da wir eine unendliche Menge von Konzepten haben, eine weitere Frageart, um zu terminieren, also alle Konzepte bis auf ein letztes auszuschalten. Dazu definieren wir die “abgeschwächte Äquivalenzfrage”, die uns bestätigt, ob unsere Hypothese äquivalent zur gesuchten Sprache L ist aber kein Gegenbeispiel liefert. Dann können wir zeigen, dass $2n$ Elementfragen und n abgeschwächte Äquivalenzfragen genügen, um L zu lernen.

6.1 Lernen mit Äquivalenzfragen

Wir versuchen nun, eine uns unbekannte reguläre unäre Sprache L , die von einem minimalen DFA mit n Zuständen akzeptiert wird, mit Äquivalenzfragen zu lernen. Dabei suchen wir nach dem minimalen DFA M für L .

Definition 6.1.1 (Äquivalenzfrage)

Gegeben sei eine reguläre Sprache $L \subseteq \Sigma^*$. Als Äquivalenzfrage bezeichnen wir einen DFA M mit $L(M) \subseteq \Sigma^*$, auf die uns das Orakel entweder mit “dieser Automat akzeptiert L ” oder einem Wort aus der symmetrischen Differenz $L(M) \oplus L$ antwortet.

Ein als Antwort auf die Äquivalenzfrage M erhaltenes Wort w , vermögen wir entsprechend Definition 4.2.1 zu klassifizieren. Wird w von unserem DFA M akzeptiert, gilt also $w \in L(M)$, dann gilt natürlich $w \notin L$ und so nennen wir w ein negatives Beispiel. Entsprechend nennen wir w ein positives Beispiel, wenn w von M verworfen wird. Daher erweitern wir Definition 6.1.1 in dem Sinne, dass die Antwort w ein klassifiziertes Beispiel ist. Als erlernt gilt L , wenn das Orakel uns auf eine Äquivalenzfrage M mit “dieser Automat akzeptiert L ” geantwortet hat.

6.1.1 Obere Schranke für die Zahl der Äquivalenzfragen

Angenommen, der minimale deterministische Automat für die zu lernende Sprache L habe n' Zustände. Dann wird ein Algorithmus, der stur nacheinander alle DFA mit $1, 2, 3, \dots, n'$ Zuständen als Äquivalenzfrage stellt die Sprache L sicher erlernen. Übertragen in die uns bekannte Notation, kann man dieses Vorgehen als Aufzählung aller DFA in den Klassen $M(n, z)$ ansehen.

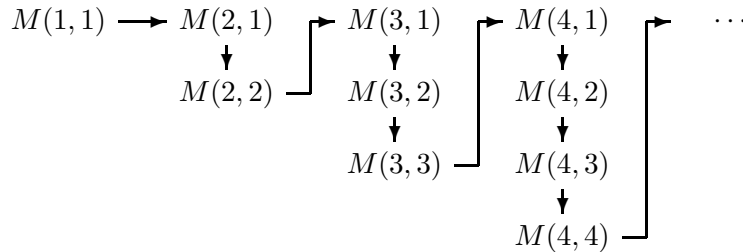


Abbildung 6.1: Aufzählung zum Erlernen der Sprache L mittels Äquivalenzfragen

Wir wissen bereits aus Lemma 4.2.9, dass kein DFA der Klasse $M(n, z)$ konsistent ist, falls der am ehesten konsistente DFA der Klasse $M(n, z)$ nicht konsistent ist. Ein etwas ausgefeilterer Algorithmus wird daher nicht alle DFA der Klasse $M(n, z)$ stur nacheinander als Äquivalenzfrage stellen, sondern die als Antwort erhaltenen Beispiele nutzen, um die Konsistenz des am ehesten konsistenten DFA der Klasse $M(n, z)$ zu prüfen. Gemäß Lemma 4.2.7 ist der am ehesten konsistente DFA der Klasse $M(n, z)$ genau dann nicht konsistent, wenn ein Zustand durch zwei Beispiele widersprüchlich festgelegt wird. Darauf aufbauend können wir zeigen, dass $n + 1$ Fragen genügen, um die Unlösbarkeit des Konsistenzproblems der Klasse $M(n, z)$ nachzuweisen.

Lemma 6.1.2

Sei L die zu erlernende Sprache. Sei $n > 0$ beliebig aber fest. Sei $1 \leq z \leq n$ beliebig aber fest. Zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klasse $M(n, z)$ genügen $n + 1$ Äquivalenzfragen.

Beweis: Wir zeigen, dass das als Antwort auf eine Äquivalenzfrage erhaltene Beispiel entweder einen bisher noch nicht festgelegten Zustand festlegt oder zu einem widersprüchlich festgelegten Zustand führt. Daraus folgt, dass nach n Fragen alle Zustände festgelegt sind, so dass die nächste Frage unweigerlich zu einer widersprüchlichen Festlegung führen muss (sofern nicht gerade die gesuchte Sprache L durch einen DFA M aus der Klasse akzeptiert wird).

Gegeben seien die Menge positiver Beispiele P und die Menge negativer Beispiele N , die wir durch frühere Äquivalenzfragen gewonnen haben. Wir betrachten den am ehesten konsistenten DFA $M \in M(n, z)$ mit $F = \{\delta(q_0, v) \mid v \in P\}$. Ist M nicht konsistent, ist das Konsistenzproblem der Klasse nicht lösbar. Andernfalls stellen wir M als Äquivalenzfrage und erhalten als Antwort das Beispiel w aus der symmetrischen Differenz zwischen L und $L(M)$. Legt w einen noch nicht festgelegten Zustand fest, ist das gewünschte gezeigt. Andernfalls sei w' das Beispiel, das den von w erreichten Zustand bereits festlegte. Dann muss eines ein positives und das andere ein negatives Beispiel und das Konsistenzproblem der Klasse unlösbar sein. Denn angenommen, w und w' sind beide positive Beispiele, dann gilt $w' \in L(M)$. Aus $\delta(q_0, w) = \delta(q_0, w')$ folgt aber auch $w \in L(M)$ und somit $w \notin L(M) \oplus L$, was ein Widerspruch zur Wahl des Beispiels ist. Mit analoger Argumentation können wir folgern, dass w und w' nicht beide negative Beispiele sind. \square

Erinnern wir uns nun der grundsätzlichen Beobachtungen und des Lemmas 4.2.12. Wir können die als Antwort auf Äquivalenzfragen erhaltenen Beispiele natürlich für die Prüfung der Lösbarkeit des Konsistenzproblems anderer Klassen wiederverwerten; insbesondere für Klassen mit gleicher Zykluslänge, wie uns das folgende Lemma zeigt.

Lemma 6.1.3

Sei L die zu erlernende Sprache. Sei $n > 0$ beliebig aber fest. Sei $1 \leq z \leq n$ beliebig aber fest. Dann genügen zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klassen $M(z, z)$, $M(z + 1, z)$, \dots , $M(n, z)$ $n + 1$ Äquivalenzfragen.

Beweis: Der Beweis erfolgt per vollständiger Induktion über die Zahl der Klassen. Die Induktionsannahme von $z + 1$ Fragen für $M(z, z)$ folgt sofort aus Lemma 6.1.2. Betrachten wir die Klasse $M(z + i, z)$, mit $i > 0$. Für den Nachweis der Unlösbarkeit des Konsistenzproblems der Klassen $M(z, z), \dots, M(z + i - 1, z)$ hatten bereits $k \leq z + i$ Fragen genügt. Dann ist nach Lemma 4.2.12 das Konsistenzproblem von $M(z + i, z)$ entweder unlösbar oder es werden bereits k Zustände festgelegt. Also brauchen wir, nach Lemma 6.1.2, noch höchstens $z + i + 1 - k$ neue Äquivalenzfragen, um die Unlösbarkeit des Konsistenzproblems der Klasse $M(z + i, z)$ nachzuweisen; somit genügen $z + i + 1$ Fragen insgesamt. \square

Algorithmus 6.1.1 auf Seite 58 lernt L dadurch, dass er für die Klassen $M(n, z)$, in der in Abbildung 6.1 angegebenen Reihenfolge, die Unlösbarkeit des Konsistenzproblems nachweist.

Satz 6.1.4 (Obere Schranke für die Zahl der Äquivalenzfragen)

Sei L eine reguläre unäre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Algorithmus 6.1.1 genügen zum Lernen der Sprache L

$$(n + 1) \cdot n = n^2 + n = O(n^2)$$

Äquivalenzfragen.

Beweis: Algorithmus 6.1.1 zählt nacheinander die Äquivalenzklassen wie in Abbildung 6.1 angeführt auf. Erst wenn die Unlösbarkeit des Konsistenzproblems der gerade untersuchten Klasse gezeigt ist, geht er über zur nächsten. Zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klasse $M(n', z)$ wendet er das in Lemma 6.1.2 aufgezeigte Verfahren an, indem er den gerade am ehesten konsistenten DFA $M \in M(n', z)$, sofern er konsistent ist, als Äquivalenzfrage stellt und die Antwort auf diese Frage vermerkt. Also prüft der Algorithmus 6.1.1 implizit alle DFA mit n oder weniger Zuständen. Somit wird er L sicher erlernen.

Die Zahl der benötigten Äquivalenzfragen ergibt sich wie folgt. Nach Lemma 6.1.3 genügen für ein festes $z > 0$ zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klassen $M(z, z)$, $M(z + 1, z)$, \dots , $M(n, z)$ $n + 1$ Fragen. Da es genau n mögliche Zykluslängen gibt, erhalten wir das obige Ergebnis. \square

Die Abschätzung der Zahl der Äquivalenzfragen in Satz 6.1.4 verwertet die Fragen nur beim Nachweis der Unlösbarkeit des Konsistenzproblems der Klassen mit gleicher Zykluslänge wieder. Mit Beobachtung 6.1.5 kann auch die implizite Wiederverwertung der für den Nachweis der Unlösbarkeit des Konsistenzproblems einer Klasse mit kürzerer Zykluslänge gestellten Fragen gezeigt werden. Allerdings reicht dies noch nicht für eine asymptotische Verbesserung der Laufzeitabschätzung des Algorithmus 6.1.1 aus.

Beobachtung 6.1.5

Sei $z > 0$ beliebig aber fest. Sei $z' > z$ mit $z \mid z'$ beliebig aber fest. Dann gilt für ein beliebiges aber festes $i \geq 0$, dass zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klassen $M(z + i, z)$ und $M(z' + i, z')$ $z' + i + 1$ Äquivalenzfragen genügen.

Eingabe: Keine

Ausgabe: Minimaler DFA für die gesuchte Sprache L

```

1:  $P \leftarrow \emptyset$  /* Menge positiver Beispiele */
2:  $N \leftarrow \emptyset$  /* Menge negativer Beispiele */
3:  $n \leftarrow 1; z \leftarrow 1$  /* Untersuchte Klasse  $M(n, z)$  */
4: while Sprache  $L$  nicht erlernt do
5:   /* Berechne den zu  $P$  und  $N$  am ehesten konsistenten DFA  $M \in M(n, z)$  */
6:    $M \leftarrow M \in M(n, z)$  mit  $F = \{\delta(0, v) \mid v \in P\}$ 
7:   if  $M$  ist konsistent then
8:     /* Stelle den am ehesten konsistenten DFA als Äquivalenzfrage */
9:      $w \leftarrow$  Antwort auf Äquivalenzfrage  $M$ 
10:    if  $w$  ist "dieser Automat akzeptiert  $L$ " then
11:      Sprache  $L$  wurde erlernt
12:    else if  $w \notin L(M)$  then
13:       $P \leftarrow P \cup \{w\}$  /*  $w$  ist ein positives Beispiel */
14:    else if  $w \in L(M)$  then
15:       $N \leftarrow N \cup \{w\}$  /*  $w$  ist ein negatives Beispiel */
16:    end if
17:  else
18:    /* Das Konsistenzproblem für  $M(n, z)$  ist unlösbar. Weiter mit der Klasse */
19:    /*  $M(n, z + 1)$  beziehungsweise  $M(n + 1, 1)$  falls  $n = z$  */
20:     $z \leftarrow z + 1$ 
21:    if  $z > n$  then
22:       $n \leftarrow n + 1$ 
23:       $z \leftarrow 1$ 
24:    end if
25:  end if
26: end while
    
```

Algorithmus 6.1.1: Erlernen der Sprache L mittels Äquivalenzfragen

Beweis: Zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klasse $M(z+i, z)$ genügen nach Lemma 6.1.2 $z+i+1$ Fragen. Dabei wird durch jede Frage, bis auf die letzte, die einen Widerspruch erzeugt, ein noch nicht festgelegter Zustand festgelegt. Gemäß Lemma 4.2.12 ist das Konsistenzproblem für $M(z'+i, z')$ nicht lösbar, wenn das Konsistenzproblem für $M(z+i, z)$ unlösbar ist, oder aber es wird in $M(z'+i, z')$ wenigstens ein Zustand mehr festgelegt als in $M(z+i, z)$. Ist das Konsistenzproblem für $M(z'+i, z')$ nicht lösbar, ist das Gewünschte bereits gezeigt. Andernfalls legt in $M(z'+i, z')$ jede Äquivalenzfrage einen Zustand fest. Fragen wir nun entsprechend der Vorschrift in Lemma 6.1.2 weiter, muss jede weitere Frage einen noch nicht festgelegten Zustand festlegen oder den Widerspruch erzeugen. Demgemäß genügen zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klasse $M(z'+i, z')$ insgesamt $z'+i+1$ Fragen. \square

Zum Abschluss möchten wir, auch zum besseren Vergleich mit dem Lernen mit Elementfragen, noch das Lernen einer endlichen Sprache anreißen.

Satz 6.1.6 (Obere Schranke für endliche Sprachen)

Sei L eine endliche Sprache. Das L endlich ist sei bekannt. Dann genügen $|L|+1$ Äquivalenzfragen um L zu erlernen.

Beweis: Ist $L \neq \emptyset$, dann wird L von einem DFA $M \in M(\lambda(L) + 1, 1)$ akzeptiert. Andernfalls ist $M_\emptyset \in M(1, 1)$ mit $F = \emptyset$ der gesuchte DFA.

Wir stellen als erste Äquivalenzfrage M_\emptyset und erhalten ein positives Beispiel als Antwort. Sei P die Menge der bisher als Antwort erhaltenen positiven Beispiele. Dann stellen wir den am ehesten konsistenten DFA $M \in M(\lambda(P) + 1, 1)$ mit $F = \{\delta(0, v) \mid v \in P\}$ als Äquivalenzfrage. Die Antwort muss wiederum ein positives Beispiel sein. Nach $|L|$ Fragen kennen wir alle Wörter in L und erhalten somit auf die letzte Frage die Antwort, dass wir L erlernt haben. \square

6.1.2 Untere Schranke für die Zahl der Äquivalenzfragen

Für das Lernen mit Äquivalenzfragen lässt sich anhand der VC-Dimension der Konzeptklasse schnell eine untere Schranke für die Zahl der Fragen bestimmen. Sei $s = VC(\mathcal{C})$. Dann existiert eine Menge S mit $|S| = s$ Elementen, die von \mathcal{C} zertrümmert wird und für jede Teilmenge von S existiert (eingeschänkt auf S) ein Konzept. Wenn das Orakel also die Gegenbeispiele aus der Menge S wählt, muss jeder Lernalgorithmus wenigstens s Äquivalenzfragen stellen [Schn97].

Aus Satz 4.3.5 wissen wir bereits, dass $VC(L_n) \geq n + 1$. Wir werden in diesem Abschnitt eine Strategie entwickeln, die, allerdings eingeschränkt auf die Hypothesenklasse der unären deterministischen endlichen Automaten mit primärer Zykluslänge, diese untere Schranke fast quadratisch vergrößert. Dazu wählt das Orakel die zu lernende Sprache aus der Klasse $M(q, q)$, mit q prim, und zwingt durch seine Wahl der Beispiele jeden Algorithmus dazu, die Unlösbarkeit des Konsistenzproblems der Klassen $M(p, p)$, mit p prim, $p \neq q$, nachzuweisen.

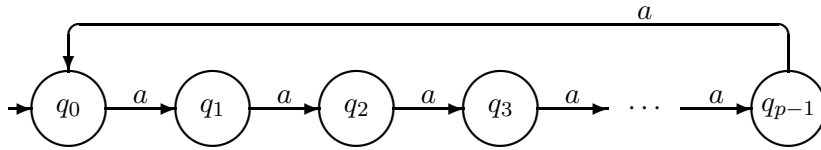


Abbildung 6.2: DFA aus der Klasse $M(p, p)$

Der Zwang zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klassen $M(p, p)$, p prim, ergibt sich, da alle Automaten $M \in M(p, p)$, mit Ausnahme der beiden mit $F = Q$ und $F = \emptyset$, minimal sind. Dazu zeigen wir, dass bei den Automaten der Klasse $M(p, p)$, mit p prim, zwei beliebige Zustände nur dann äquivalent gemäß Definition 2.2.6 sind, wenn alle Zustände äquivalent sind. Also sind alle Zustände unterscheidbar und damit die Automaten nach dem Satz von Nerode, Satz 2.2.5, minimal.

Lemma 6.1.7

Gegeben sei ein $m > 0$. Für eine beliebige Zahl s relativ prim zu m , also mit $\text{ggT}(s, m) = 1$, gilt $\{s \cdot k \bmod m \mid 1 \leq k \leq m\} = \mathbb{Z}_m$.

Beweis: Wir zeigen, dass wir m unterschiedliche Restklassen erhalten. Da \mathbb{Z}_m unter Produktbildung abgeschlossen ist, erhalten wir das obige Ergebnis. Angenommen es gilt $s \cdot k \equiv s \cdot k' \pmod{m}$, dann dürfen wir kürzen, da $\text{ggT}(s, m) = 1$. Somit folgt $k \equiv k' \pmod{m}$. Also erhalten wir m unterschiedliche Restklassen. \square

Lemma 6.1.8

Der DFA $M \in M(p, p)$, p prim, ist minimal, falls $L(M) \neq \Sigma^*$ und $L(M) \neq \emptyset$.

Beweis: Wir zeigen, dass zwei beliebige Zustände q_i und q_j mit $i \neq j$ gemäß Definition 2.2.6 unterscheidbar sind und somit unser Automat minimal ist. Dazu definieren wir folgende Relation für zwei Zustände unseres DFA M

$$q \sim q' \iff q \in F \iff q' \in F.$$

O.B.d.A sei $i < j$. Falls $q_i \not\sim q_j$ sind die Zustände trivialerweise unterscheidbar. Also gelte $q_i \sim q_j$. Um nicht unterscheidbar zu sein, muss für alle Wörter $v \in \Sigma^*$ gelten, dass $\delta(q_i, v) \sim \delta(q_j, v)$. Insbesondere muss dann natürlich für $k \geq 1$

$$\delta(q_i, a^{k(j-i)}) = \delta(q_j, a^{(k-1)(j-i)}) \sim \delta(q_j, a^{k(j-i)})$$

gelten, oder anders aufgeschrieben

$$q_{j+(k-1)(j-i) \bmod p} \sim q_{j+k(j-i) \bmod p}$$

Da $1 \leq j - i < p$ und somit $\text{ggT}(j - i, p) = 1$ ist, gilt nach Lemma 6.1.7

$$\{j + k \cdot (j - i) \bmod p \mid k > 0\} = \mathbb{Z}_p.$$

Also müssen alle Zustände das gleiche Akzeptanzverhalten aufweisen wie q_i und q_j . Das aber ist ein Widerspruch zur Annahme $L \neq \Sigma^* \wedge L \neq \emptyset$. \square

Definition 6.1.9

Das Produkt aller Primzahlen zwischen 1 und n bezeichnen wir mit $\phi(n)$.

Lemma 6.1.10

Sei $n > 0$ beliebig aber fest. Sei L_n die Menge der regulären unären Sprachen, die von einem minimalen DFA mit höchstens n Zuständen akzeptiert werden. Zum Lernen der Konzeptklasse L_n durch die Hypothesenklasse der unären deterministischen endlichen Automaten mit primärer Zykluslänge muss jeder Lernalgorithmus wenigstens

$$\sum_{\substack{p \text{ prim} \\ p \leq n}} (p - 1)$$

Äquivalenzfragen stellen.

Beweis: Wir zeigen die Gegenstrategie des Orakels auf. Das Orakel beschränkt sich auf die Sprachen, die von einem minimalen DFA $M \in M(p, p)$, p prim, akzeptiert werden, bei dem der Zustand 0 verwirft und der Zustand 1 akzeptiert. Diese Einschränkung kann dem Lernalgorithmus durchaus bekannt sein. Nach Lemma 6.1.8 sind alle DFA $M \in M(p, p)$ mit $F \cap \{0, 1\} = \{1\}$, p prim, minimal — sie müssen also alle “ausgeschaltet” werden.

Sei $p \leq n$, p prim, beliebig aber fest. Selbst wenn dem Lernalgorithmus bekannt ist, dass die gesuchte Sprache L von einem DFA $M \in M(p, p)$ akzeptiert wird, muss er, um alle bis auf einen DFA “auszuschalten”, jeden Zustand festlegen. Dazu sind, da die Festlegung der Zustände 0 und 1 bekannt ist, $p - 2$ Äquivalenzfragen nötig. Zum Abschluss muss auch noch der letzte konsistente DFA $M \in M(p, p)$ als Äquivalenzfrage gestellt werden, um L zu lernen. Insgesamt müssen also $p - 2 + 1 = p - 1$ Fragen gestellt werden. Wählt das Orakel seine Antworten nun so, dass ein positives (negatives) Beispiel, das es zum “Ausschalten” der DFA in $M(p, p)$, p prim, gab, nicht für das “Ausschalten” der DFA in $M(q, q)$, q prim, $q \neq p$, “wiederverwertet” werden kann, weil es in $M(q, q)$ den Zustand 1 (0) festlegt, dann sind auch für $M(q, q)$ nochmals $q - 1$ Äquivalenzfragen nötig und das Ergebnis dieses Lemma folgt.

Wir zeigen zuerst, dass Beispiele mit diesen Eigenschaften existieren. Dazu erweitern wir unsere Definition der Festlegung eines Zustandes eines DFA in der natürlichen Weise auf die Restklassen modulo seiner Zykluslänge und sprechen künftig davon, dass die Restklasse r modulo der Zykluslänge z festgelegt sei, wenn ein Beispiel w mit $|w| \equiv r \pmod{z}$ existiert, dass einen Zustand im Zyklus festlegt. Für ein $p > 0$, p prim, definieren wir

$$A_{p,b} = \bigcup_{n' \geq \max(n,p)} \left\{ \frac{\phi(n')}{p} \cdot k + b \mid 1 \leq k \leq p \right\}.$$

Wie leicht nachgeprüft werden kann, legt ein Wort a^m , mit $m \in A_{p,b}$, in einem DFA mit höchstens n Zuständen und primer Zykluslänge q , mit $q \neq p$, $q \leq n$, immer die Restklasse b fest. Andererseits gilt, da $\phi(n)/p$ und p relativ prim sind, nach Lemma 6.1.7 $\{m \bmod p \mid m \in A_{p,b}\} = \mathbb{Z}_p$. Also existiert für ein beliebiges r , $0 \leq r < p$, ein $m \in A_{p,b}$, so dass a^m in einem DFA mit höchstens n Zuständen und Zykluslänge p die Restklasse r festlegt.

Wir betrachten erneut die Eigenschaften der Beispiele. Sei $p \leq n$, p prim, beliebig aber fest. Das Orakel kann seine Antworten so wählen, dass ein Wort a^m , mit $m \in A_{p,0} \cup A_{p,1}$, in $M(p,p)$ eine Restklasse r , $0 \leq r < p$, in $M(q,q)$, $q \leq n$, q prim, $q \neq p$, aber die Restklasse 0 oder 1 festlegt. Wählt es für ein positives Beispiel a^k immer $k \in A_{p,1}$ und für ein negatives Beispiel a^ℓ immer $\ell \in A_{p,0}$, dann “bündelt” es in $M(q,q)$ die positiven Beispiele in Restklasse 1 und die negativen in Restklasse 0. Dadurch zwingt es den Lernalgorithmus aber auch, zur “Ausschaltung” der DFA in der Klasse $M(q,q)$, weitere $q - 1$ Äquivalenzfragen zu stellen.

Wir kommen nun zur Strategie des Orakels. Angenommen, der Lernalgorithmus stellt einen DFA $M \in M(y,p)$, p prim, als Äquivalenzfrage. Sei S die Menge der bereits vom Orakel klassifizierten Beispiele. Ist M nicht konsistent mit S antwortet das Orakel mit einem entsprechenden Beispiel aus S . Daher gehen wir im Weiteren davon aus, dass M konsistent ist. Aus der Konsistenz von M folgt, dass keine Restklasse widersprüchlich festgelegt ist. Die Restklassen 0 und 1 betrachten wir für dabei für jede prime Zykluslänge als bereits festgelegt, da wir in der Restklasse 0 die negativen und in Restklasse 1 die positiven Beispiele aus S “bündeln”.

Falls es noch eine nicht festgelegte Restklasse gibt, wählt das Orakel eine beliebige. Sei r die noch nicht festgelegte Restklasse. Akzeptiert der r zugeordnete Zustand des DFA M (der Lernalgorithmus ist frei in seiner Wahl, ob er nicht festgelegte Zustände akzeptierend oder verwerfend setzt), dann wählt das Orakel als Antwort das negative Beispiel a^ℓ , mit $\ell \in A_{p,0}$, $\ell \geq y$ und $\ell \equiv r \pmod{p}$, andernfalls das positive Beispiel a^k , mit $k \in A_{p,1}$, $k \geq y$ und $k \equiv r \pmod{p}$. Die Existenz der beiden Beispiele folgt aus der Definition der Mengen $A_{p,b}$. Falls bereits alle Restklassen festgelegt sind, wählt das Orakel eine beliebige Restklasse r und erzeugt in dieser einen Widerspruch. Das entsprechende Beispiel wählt es wie für eine nicht festgelegte Restklasse. Hierdurch wird der Nachweis der Unlösbarkeit des Konsistenzproblems für $M(y,p)$ erbracht. Daraus folgt sofort, dass auch das Konsistenzproblem für $M(p,p)$ unlösbar ist (und damit werden alle DFA in dieser Klasse “ausgeschaltet”). Hierzu waren $p - 1$ Äquivalenzfragen nötig. \square

Satz 6.1.11 (Untere Schranke für die Zahl der Äquivalenzfragen)

Sei $n > 0$ beliebig aber fest. Sei L_n die Menge der regulären unären Sprachen, die von einem minimalen DFA mit höchstens n Zuständen akzeptiert werden. Zum Lernen der Konzeptklasse L_n durch die Hypothesenklasse der unären deterministischen endlichen Automaten mit primer Zykluslänge muss jeder Algorithmus wenigstens

$$\Omega\left(\frac{n^2}{\ln(n)}\right)$$

Äquivalenzfragen stellen, um L zu lernen.

Beweis: Der Beweis folgt aus Lemma 6.1.10. Die dort angegebene Zahl der Fragen können wir mit dem großen Primzahlsatz, Satz 2.1.3, durch $\pi(n) \approx n/\ln(n)$ abschätzen. Außerdem gilt nach [GKP94] für die i -te Primzahl p_i , dass $p_i \geq i \ln(i)$ ist.

$$\sum_{\substack{p \leq n \\ p \text{ prim}}} p - 1 \geq \sum_{i=1}^{\pi(n)} i \ln(i) - \sum_{i=1}^{\pi(n)} 1$$

$$\begin{aligned}
 &\geq \int_{x=1}^{\pi(n)} x \ln(x) dx - \pi(n) = \left[\frac{1}{2}x^2 \ln(x) - \frac{1}{4}x^2 \right]_{x=1}^{\pi(n)} - \pi(n) \\
 &= \frac{1}{2}\pi(n)^2 \ln(\pi(n)) - \frac{1}{4}\pi(n)^2 + \frac{1}{4} - \pi(n) \\
 &\geq \frac{\pi(n)^2}{4} \ln(\pi(n)) - \pi(n) \\
 &\approx \frac{n^2}{4 \ln(n) \ln(n)} \ln\left(\frac{n}{\ln(n)}\right) - \frac{n}{\ln(n)} = \frac{n^2}{4 \ln(n) \ln(n)} \cdot \left(\ln(n) - \ln(\ln(n)) \right) - \frac{n}{\ln(n)} \\
 &= \frac{n^2 \ln(n) - 4n \ln(n)}{4 \ln(n) \ln(n)} - \frac{n^2 \ln(\ln(n))}{4 \ln(n) \ln(n)} \\
 &\geq \frac{n^2 \ln(n) - 4n \ln(n)}{4 \ln(n) \ln(n)} - \frac{2n^2}{4 \ln(n) \ln(n)} \\
 &= \frac{n^2(\ln(n) - 2) - 4n \ln(n)}{4 \ln(n) \ln(n)} = \frac{n^2 \ln\left(\frac{n}{e^2}\right) - 4n \ln(n)}{4 \ln(n) \ln(n)} \\
 &= \Omega\left(\frac{n^2}{\ln(n)}\right).
 \end{aligned}$$

□

6.2 Lernen mit Elementfragen

Beim Lernen mit Elementfragen darf der Lernende gezielt nachfragen, ob ein bestimmtes Wort in der vom Orakel gewählten Sprache L enthalten ist. ANGLUIN hat in [An82] gezeigt, dass bei Kenntnis von n , der Zahl der Zustände des minimalen DFA für eine reguläre Sprache L , die obere und untere Schranke für die Zahl der Elementfragen exponentiell in n ist. Für reguläre unäre Sprachen können wir eine bessere obere Schranke von $2n - 1$ angeben.

Eine Sprache L gilt in diesem Modell als erlernt, wenn nur noch eine konsistente Hypothese übrig ist. Daher ist es ohne eine Zusatzinformation, wie etwa die Zahl der Zustände des minimalen DFA, mit Elementfragen alleine nicht möglich, die regulären unären Sprachen zu lernen. Statt der Zahl der Zustände betrachten wir daher noch eine andere Zusatzinformation: die abgeschwächte Äquivalenzfrage. Im Gegensatz zur Äquivalenzfrage erhalten wir bei der abgeschwächten Äquivalenzfrage kein Gegenbeispiel. Unter diesem Ansatz können wir die regulären unären Sprachen mit $2n$ Element- und n abgeschwächten Äquivalenzfragen erlernen.

Definition 6.2.1 (Elementfrage)

Gegeben sei eine reguläre Sprache $L \subseteq \Sigma^*$. Als Elementfrage bezeichnen wir ein Wort $w \in \Sigma^*$, auf die uns das Orakel mit “ja” antwortet, falls $w \in L$, oder mit “nein” antwortet falls $w \notin L$.

Aufgrund der Antwort des Orakels können wir das als Elementfrage gestellte Wort gemäß Definition 4.2.1 klassifizieren: war die Antwort “ja” so ist es ein positives Beispiel, ansonsten ein negatives. Daher erweitern wir die Definition 6.2.1 in der natürlichen Weise und bezeichnen künftig das als Elementfrage gestellte Wort auch als Beispiel.

Beweis: Da M, M' konsistent zu S sind, gilt natürlich $L(M) \cap S = L(M') \cap S$ und auch $a^i \in L(M) \Leftrightarrow a^i \in L(M')$ für $i < 2n$. Wir definieren für zwei Worte $w, w' \in \Sigma^*$ folgende Äquivalenzrelation in M

$$w \sim w' \iff w \in L(M) \Leftrightarrow w' \in L(M).$$

Direkt aus der Übergangsfunktion des DFA M und der Konsistenz beider DFA folgt

$$\begin{aligned} a^{n-z+i} &\sim a^{n+i} && \text{für } i \geq 0 \\ a^{n-z+i} &\sim a^{n-z'+i} && \text{für } 0 \leq i < n. \end{aligned}$$

Die erste Beziehung ist trivial, die zweite ergibt sich direkt aus $\delta(0, a^{n-z+i}) = \delta(0, a^{n+i})$ und $\delta'(0, a^{n-z'+i}) = \delta'(0, a^{n+i})$. Wir haben zwei Fälle zu unterscheiden.

“ $z' > z$ ” Wir setzen $\ell = z' - z$. Betrachten wir nun die Worte $a^{n+i+\ell \cdot m}$ und $a^{n+i+\ell \cdot (m-1)}$ für ein $m > 0$. Beide Worte erreichen in M einen Zustand im Zyklus. Daher existiert ein j , mit $0 \leq j < z$, so dass in M $\delta(0, a^{n-z+j}) = \delta(0, a^{n+i+\ell \cdot (m-1)})$ und $\delta(0, a^{n-z+j+\ell}) = \delta(0, a^{n+i+\ell \cdot m})$. Aus $a^{n-z+j} \sim a^{n-z'+j}$ und $a^{n-z+j} = a^{n-(z'-\ell)+j}$ folgt $a^{n-z'+\ell+j} \sim a^{n-z'+j}$. Da $\ell + j < n$ ist, gilt auch $a^{n-z'+\ell+j} \sim a^{n-z+\ell+j}$ und damit $a^{n-z+\ell+j} \sim a^{n-z+j}$. Dann muss natürlich, da die gleichen Zustände erreicht werden, auch $a^{n+i+\ell \cdot (m-1)} \sim a^{n+i+\ell \cdot m}$ sein. Hieraus lässt sich letztlich herleiten, dass $a^{n+i+\ell \cdot k} \sim a^{n+i+\ell \cdot (k-1)} \sim a^{n+i+\ell \cdot (k-2)} \sim \dots \sim a^{n+i+\ell \cdot (k-k)}$ für $k \geq 0$.

“ $z' < z$ ” Wir setzen $\ell = z - z'$. Betrachten wir wieder die Worte $a^{n+i+\ell \cdot m}$ und $a^{n+i+\ell \cdot (m-1)}$ für ein $m > 0$. Beide Worte erreichen in M einen Zustand im Zyklus. Daher existiert ein j , mit $0 \leq j < z$, so dass in M $\delta(0, a^{n-z+j}) = \delta(0, a^{n+i+\ell \cdot (m-1)})$ und $\delta(0, a^{n-z+j+\ell}) = \delta(0, a^{n+i+\ell \cdot m})$. Aus $a^{n-z+j} \sim a^{n-z'+j}$ und $a^{n-z'+j} = a^{n-(z-\ell)+j}$ folgt $a^{n-z+j} \sim a^{n-z+\ell+j}$. Der Rest ergibt sich wie oben. \square

Lemma 6.2.3

Sei L eine reguläre unäre Sprache, die von einem DFA mit n Zuständen akzeptiert wird. Sei $S = \{a^0, a^1, a^2, \dots, a^{2n-1}\}$ die Menge klassifizierter Beispiele. Jeder mit S konsistente DFA mit n Zuständen akzeptiert L .

Beweis: Wir zeigen, dass für zwei beliebige konsistente DFA M und M' $L(M) \subseteq L(M')$ gilt. Daraus folgt $L(M) = L(M')$. Außerdem ist einer der DFA der, der L akzeptiert, also akzeptieren alle L . Sei nun $M \in M(n, z)$ und $M' \in M(n, z')$. Ist $z = z'$ ist es trivial, also gelte $z \neq z'$. Angenommen $L(M) \not\subseteq L(M')$, dann existiert ein Wort $w \in L(M)$ mit $w \notin L(M')$. Da beide Automaten mit S konsistent sind, muss $|w| \geq 2n$ sein.

“ $z' < z$ ” Wir können das Wort w als $w = a^{n-z+r+zt}$ mit $0 \leq r < z$ und $t > 0$ schreiben. Aus $\delta(0, w) = \delta(0, a^{n+r})$ folgt sofort $a^{n+r} \in L(M)$ und aufgrund der Konsistenz beider Automaten auch $a^{n+r} \in L(M')$. Setzen wir $z = z' + \ell$, so ergibt sich durch Einsetzen und Umformen $w = a^{n-(z'+\ell)+r+(z'+\ell)t} = a^{n-z'+r+z't+\ell(t-1)}$. In M' erreicht w also den Zustand, den auch $a^{n+r+\ell(t-1)} = a^{n+r+|z-z'|\cdot(t-1)}$ erreicht. Aus $w \notin L(M')$ und Lemma 6.2.2 folgt dann $a^{n+r} \notin L(M')$, was einen Widerspruch zur Konsistenz darstellt. Also gilt $L(M) \subseteq L(M')$.

“ $z' > z$ ” Der Beweis erfolgt analog obigen Überlegungen. \square

Aus Lemma 6.2.3 folgt direkt, dass wenn n , die Zahl der Zustände des minimalen DFA für die zu erlernende Sprache L bekannt ist, $2n$ Elementfragen genügen, um die Sprache L zu erlernen. Wir wollen aber mit Satz 6.2.5 noch einen Beweis für die marginal bessere Schranke $2n - 1$ angeben.

Definition 6.2.4

Gegeben seien zwei DFA M und M' . Wir definieren für $m \geq 0$ folgende Äquivalenzrelationen

$$M \equiv_m M' \iff a^m \in L(M) \Leftrightarrow a^m \in L(M')$$

Satz 6.2.5 (Obere Schranke für die Zahl der Fragen bei bekannter Zustandszahl)

Sei L eine reguläre unäre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Ist n bekannt, so genügen $2n - 1$ Elementfragen, um L zu lernen.

Beweis: Wir stellen als erstes die Worte $a^0, a^1, a^2, \dots, a^{n-1}$ als Elementfrage. Dadurch werden in den Klassen $M(n, z)$, für $1 \leq z \leq n$, alle Zustände festgelegt, so dass in jeder Klasse genau ein konsistenter DFA übrigbleibt. Einer dieser DFA muss der minimale DFA für L sein. Wir werden zeigen, dass wir nach weiteren $n - 1$ Elementfragen entweder alle Wörter aus $\Sigma^{<2n}$ als Beispiel klassifiziert oder für alle bis auf eine Klasse $M(n, z)$ die Unlösbarkeit des Konsistenzproblems nachgewiesen haben. In beiden Fällen haben wir L erlernt.

Wir betrachten einmal das Wort a^m , mit $n \leq m < 2n$. Es seien A_m und A'_m die beiden Äquivalenzklassen der auf alle konsistenten DFA $M \in M(n, z)$, mit $1 \leq z \leq n$, angewandten Äquivalenzrelation \equiv_m aus Definition 6.2.4. Das heißt, eine enthält alle a^m akzeptierenden, die andere alle a^m verwerfenden DFA. Gilt $A_m = \emptyset$ oder $A'_m = \emptyset$ brauchen wir a^m natürlich nicht als Elementfrage stellen, da das Akzeptanzverhalten aller konsistenten DFA für a^m gleich (und der minimale DFA für L einer von ihnen) ist. In diesem Falle klassifizieren wir a^m entsprechend als Beispiel. Andernfalls werden durch die Elementfrage a^m wenigstens $\min(|A_m|, |A'_m|) \geq 1$ DFA als inkonsistent nachgewiesen. \square

Bisher sind wir davon ausgegangen, dass die Zahl der Zustände des minimalen DFA für die zu erlernende Sprache L bekannt ist. In diesem Fall gelingt es uns sehr effizient, L zu erlernen. Es wäre eine natürliche Annahme, davon auszugehen, dass beim Fehlen dieser Zusatzinformation nur ein etwas höherer Aufwand nötig wird. Doch leider zeigt es sich, dass L mit Elementfragen alleine nicht erlernt werden kann.

Beobachtung 6.2.6

Sei L eine reguläre unäre Sprache. Ein Algorithmus der nur Elementfragen stellen darf, kann L nicht erlernen.

Beweis: Trivial. Der Algorithmus erhält vom Orakel keine Bestätigung, dass die Sprache L erlernt wurde. Er darf also erst dann aufhören weiterzufragen, wenn er alle Konzepte bis auf eines ausgeschaltet hat. Da die Konzeptklasse unendlich viele Konzepte enthält, kann er also niemals terminieren. \square

Als Folge der Beobachtung 6.2.6 erweitern wir unser Modell um eine abgeschwächte Form der Äquivalenzfrage, bei der wir zwar kein Beispiel als Antwort erhalten, die uns aber bestätigt, ob ein DFA die zu erlernende Sprache akzeptiert. Alsdann betrachten wir mit Algorithmus 6.2.1 auf Seite 66 einen Lernalgorithmus, der eine reguläre Sprache L mit $2n$ Element- und n abgeschwächten Äquivalenzfragen erlernt.

Definition 6.2.7 (abgeschwächte Äquivalenzfrage)

Sei L eine reguläre unäre Sprache. Als abgeschwächte Äquivalenzfrage bezeichnen wir einen DFA M , auf den uns das Orakel mit “dieser Automat akzeptiert L ” falls $L = L(M)$, oder “dieser Automat akzeptiert L nicht” antwortet.

Satz 6.2.8 (Korrektheit und Laufzeit des Algorithmus 6.2.1)

Sei L eine reguläre unäre Sprache, die von einem minimalen DFA mit n Zuständen akzeptiert wird. Um L zu lernen, stellt Algorithmus 6.2.1 $2n$ Elementfragen und höchstens n abgeschwächte Äquivalenzfragen.

Beweis: Der Algorithmus prüft nacheinander alle DFA mit $1, 2, 3, \dots$ Zuständen. Als Vorbedingung der Schleife in den Zeilen 4 bis 24 gilt, dass kein DFA mit $\leq n$ Zuständen L akzeptiert. Außerdem wurden bereits alle Worte aus $\Sigma^{<2n}$ als Elementfrage gestellt. Der Algorithmus

```

1:  $P \leftarrow \emptyset$  /* Menge der positiven Beispiele */
2:  $N \leftarrow \emptyset$  /* Menge der negativen Beispiele */
3:  $n \leftarrow 0$ 
4: repeat
5:    $n \leftarrow n + 1$ 
6:   /* Stelle Elementfragen  $a^{2n-2}$  und  $a^{2n-1}$  */
7:   for  $i = 1$  to 2 do
8:     if Antwort auf Elementfrage  $a^{2n-i}$  ist ja then
9:        $P \leftarrow P \cup \{a^{2n-i}\}$  /* positives Beispiel */
10:    else
11:       $N \leftarrow N \cup \{a^{2n-i}\}$  /* negatives Beispiel */
12:    end if
13:  end for
14:  /* Erstelle  $H$ , die Menge aller konsistenten DFA mit  $n$  Zuständen */
15:  /* Alle DFA  $M \in H$  akzeptieren nach Lemma 6.2.3 die gleiche Sprache */
16:   $H \leftarrow \emptyset$ 
17:  for  $z = 1$  to  $n$  do
18:    /* Berechne den am ehesten konsistenten DFA  $M \in M(n, z)$  */
19:     $M \leftarrow M \in M(n, z)$  mit  $F = \{\delta(0, v) \mid v \in P\}$ 
20:    if  $M$  ist konsistent then
21:       $H \leftarrow H \cup \{M\}$ 
22:    end if
23:  end for
24: until  $H \neq \emptyset$  und abgeschwächte Äquivalenzfrage bestätigt beliebigen  $M \in H$ 
    
```

Algorithmus 6.2.1: Lernen mit Elementfragen und abgeschwächten Äquivalenzfragen

stellt die zwei Elementfragen a^{2n} und a^{2n+1} und prüft, ob ein konsistenter DFA M mit $n + 1$ Zuständen existiert. Nach Lemma 6.2.3 akzeptieren alle konsistenten DFA mit n Zuständen die gleiche Sprache. Falls M existiert wird er als abgeschwächte Äquivalenzfrage gestellt. Gilt $L(M) = L$ terminieren wir, ansonsten gilt als Nachbedingung der Schleife, dass kein DFA mit $n + 1$ Zuständen L akzeptiert und dass alle Worte aus $\Sigma^{<2(n+1)}$ als Elementfrage gestellt wurden.

Die Zahl der Fragen lässt sich direkt ablesen. In jedem Durchlauf der Schleife werden genau 2 Elementfragen und höchstens eine abgeschwächte Äquivalenzfrage gestellt. \square

Erinnern wir uns des Satzes 6.1.6. Er besagt, dass wir mit $|L| + 1$ Äquivalenzfragen L erlernen, wenn bekannt ist, dass L endlich ist. Das beruht letztlich darauf, dass wir immer einen DFA M mit $L(M) \subseteq L$ als Äquivalenzfrage stellen und daher immer ein positives Beispiel als Antwort erhalten.

Satz 6.2.9

Sei L eine endliche reguläre unäre Sprache. Dass L endlich ist sei bekannt. Dann genügen $\lambda(L)$ Elementfragen und $\lambda(L)$ abgeschwächte Äquivalenzfragen, um L zu lernen.

Beweis: Da L endlich ist, wird es von einem DFA $M \in M(\lambda(L) + 1, 1)$ akzeptiert. Ein Algorithmus muss jeden der Zustände außerhalb des Zyklus mittels einer Elementfrage festlegen. Um korrekt zu terminieren, muss er nach der Elementfrage den DFA der Form $M(n, 1)$ als abgeschwächte Äquivalenzfrage stellen.

Kapitel 7

Ausblick

Wir haben einige für das PAC-Lernen regulärer unärer Sprachen wichtige Fragen beantwortet. So wurde ein Algorithmus entwickelt, der den minimalen zu einer Beispielmengemenge S konsistenten DFA mit $O(n \cdot |S|)$ Schritten berechnet. Außerdem wurde die VC-Dimension der von einem DFA mit n Zuständen akzeptierten regulären unären Sprachen bis auf einen additiven Term von $O(\log(\ln(n)))$ zwischen der oberen und der unteren Schranke bestimmt. Dadurch war es möglich, eine obere Schranke für die Beispielskomplexität der regulären unären Sprachen anzugeben. Für das Lernen mit Äquivalenzfragen und das Lernen mit Elementfragen wurden Algorithmen vorgestellt, deren Anzahl an Fragen polynomiell in n , der Zahl der Zustände des minimalen DFA der zu lernenden Sprache L , ist. Dies ist eine signifikante Verbesserung zum Lernen der regulären Sprachen, für die eine polynomielle Anzahl an Fragen nicht ausreicht.

Einige Fragen mussten jedoch offen bleiben. Auf diese soll zum Abschluss noch genauer eingegangen werden. Es ist zu vermuten, dass viele dieser Fragen einen “ähnlichen” Lösungsansatz haben, und dass aus der Beantwortung einer Frage die Lösung weiterer Fragen folgt. Daher betrachten wir offene Fragen zu allen Kapiteln, selbst wenn sie auf den ersten Blick nicht so interessant erscheinen mögen.

7.1 VC-Dimension und Beispielskomplexität

Bei der VC-Dimension haben wir nahe beieinander liegende untere und obere Schranken für L_n gefunden. Die obere Schranke ist wegen $|L_n| = \Omega(n \cdot 2^n)$ [DKS02] wahrscheinlich optimal (zumindest kann sie über die Größe der Konzeptklasse nicht besser abgeschätzt werden). Die untere Schranke kann für große n vermutlich noch etwas verbessert werden, indem man die Teilmengen von S'' in mehr Klassen und mehr Zustände (beispielsweise vier oder acht Zustände) kodiert. Dadurch wäre es möglich, bei gleicher Zahl nutzbarer Klassen, mehrere “Paare” von Teilmengen unterzubringen. Außerdem verlangt der chinesische Restsatz nur, dass die Zykluslängen der verwendeten Klassen relativ prim zueinander sind; wir haben nur Primzahlen betrachtet. Bereits für L_5 kann man die Klassen $M(5, 3)$, $M(5, 4)$ und $M(5, 5)$ verwenden und statt zweier drei Zustände für die Kodierung von S'' einsetzen (wobei nicht verifiziert wurde, ob das wirklich eine Verbesserung bringt).

7.2 Konsistente Hypothesen für reguläre unäre Sprachen

Beim Berechnen des minimalen konsistenten DFA wäre es interessant zu untersuchen, wieviele minimale konsistente DFA mit fester Zykluslänge tatsächlich berechnet werden müssen. Aus Lemma 5.1.4 wissen wir bereits, dass zwei Beispiele reichen, um für die minimale konsistente

Hypothese eine gewisse Größe zu erzwingen. Adaptiert man die Überlegungen zur Gegenstrategie des Orakels in Lemma 6.1.10, wird es wahrscheinlich möglich sein, unendlich viele Beispielmengen zu konstruieren, so dass für das Berechnen der minimalen konsistenten Hypothese eine untere Schranke von $\Omega(\pi(n) \cdot (|P| + |N|))$ Schritten gezeigt werden kann. Hilfreich könnte hierbei auch die Arbeit von DOMARATZKI, KISMAN und SHALLIT [DKS02] sein, da dort Bedingungen für die Minimalität eines unären DFA formuliert werden.

7.3 Aktives Lernen regulärer unärer Sprachen

Bei der unteren Schranke der Zahl der Äquivalenzfragen mussten wir die Hypothesenklasse einschränken, da die Gegenstrategie des Orakels bei beliebigen Hypothesen zusammenzuberechnen scheint. Dies zeigen wir mit der folgenden Beobachtung.

Beobachtung 7.3.1

Sei $n > 0$ beliebig aber fest. Sei L_n die Menge der regulären unären Sprachen, die von einem minimalen DFA mit höchstens n Zuständen akzeptiert werden. Zum Lernen der Konzeptklasse L_n benötigt jeder Lernalgorithmus, der immer nur einen am ehesten konsistenten DFA als Äquivalenzfrage stellt, wenigstens

$$\sum_{\substack{p \text{ prim} \\ p \leq n}} (p - 1)$$

Äquivalenzfragen.

Beweis: Wir zeigen wiederum die Gegenstrategie des Orakels auf; die grundsätzlichen Betrachtungen aus Lemma 6.1.10 führen wir hier allerdings nicht nochmals auf. Das Orakel beschränkt sich wiederum auf die Sprachen, die von einem minimalen DFA $M \in M(p, p)$, p prim, akzeptiert werden, bei dem der Zustand 0 verwirft und der Zustand 1 akzeptiert. Diese Einschränkung kann dem Lernalgorithmus durchaus bekannt sein.

Für ein $z > 0$ mit der kanonischen Primzerlegung $z = p_1^{j_1} p_2^{j_2} \cdots p_s^{j_s}$ definieren wir

$$A_{z,b} = \bigcup_{n' \geq \max(n, p_s)} \left\{ \frac{\phi(n')}{p_1 \cdot p_2 \cdot \dots \cdot p_s} \cdot k + b \mid 1 \leq k \leq z \right\}.$$

Wie leicht nachgeprüft werden kann, legt ein Wort a^m , mit $m \in A_{z,b}$, in einem DFA mit höchstens n Zuständen und primen Zykluslänge q , mit $q \nmid z$, $q \leq n$, immer die Restklasse b fest. Andererseits gilt, da $\phi(n')/(p_1 \cdot p_2 \cdot \dots \cdot p_s)$ und z relativ prim sind, nach Lemma 6.1.7 $\{m \bmod z \mid m \in A_{z,b}\} = \mathbb{Z}_z$. Also existiert für ein beliebiges r , $0 \leq r < z$, ein $m \in A_{z,b}$, so dass a^m in einem DFA mit höchstens n Zuständen und Zykluslänge z die Restklasse r festlegt.

Wir kommen nun zur Strategie des Orakels. Angenommen, der Lernalgorithmus stellt einen DFA $M \in M(y, z)$ als Äquivalenzfrage. Sei S die Menge der bereits vom Orakel klassifizierten Beispiele. Ist M nicht konsistent mit S antwortet das Orakel mit einem entsprechenden Beispiel aus S . Daher gehen wir im weiteren davon aus, dass M konsistent ist. Sei $z = p_1^{j_1} \cdot p_2^{j_2} \cdot \dots \cdot p_s^{j_s}$ die kanonische Primzerlegung der Zykluslänge z . Das Orakel bestimmt den ersten offenen Primfaktor p_i . Einen Primfaktor p_i nennen wir dabei offen, solange keine Restklasse modulo p_i widersprüchlich festgelegt ist. Die Restklassen 0 und 1 betrachten wir für jeden Primfaktor bereits als festgelegt, da wir in der Restklasse 0 die negativen und in Restklasse 1 die positiven Beispiele aus S "bündeln".

Sei p_i ein offener Primfaktor. Falls es noch eine nicht festgelegte Restklasse gibt, wählt das Orakel eine beliebige. Sei r die noch nicht festgelegte Restklasse. Akzeptieren oder verwerfen alle r zugeordneten Zustände des DFA M , dann wählt das Orakel als Antwort das negative

Beispiel a^ℓ , mit $\ell \in A_{p_i,0}$, $\ell \geq y$ und $\ell \equiv r \pmod{p_i}$, andernfalls das positive Beispiel a^k , mit $k \in A_{p_i,1}$, $k \geq y$ und $k \equiv r \pmod{p_i}$. Das Beispiel existiert, da für ein festes b gilt, dass $\mathbb{Z}_{p_i} = \{m \bmod p_i \mid m \in A_{p_i,b}\}$. Außerdem ist M nach Voraussetzung ein am ehesten konsistenter DFA. Da keiner der r zugeordneten Zustände in M festgelegt ist (denn dann wäre r bereits festgelegt) müssen diese verwerfen. Falls bereits alle Restklassen festgelegt sind, wählt das Orakel eine beliebige Restklasse r und erzeugt in dieser einen Widerspruch, wodurch der Nachweis der Unlösbarkeit des Konsistenzproblems für $M(p_i, p_i)$ erbracht wird (und damit alle DFA in dieser Klasse “ausgeschaltet” werden). Dieser Widerspruch kann, muss aber nicht, auch zum Nachweis der Unlösbarkeit des Konsistenzproblems für $M(y, z)$ führen.

Ist keiner der Primfaktoren mehr offen, wählt das Orakel eine beliebige Restklasse r , $0 \leq r < z$, und gibt bis zu zwei (nicht gezählte) Beispiele a^k (positiv) und a^ℓ (negativ), mit $\ell \in A_{z,0}$, $k \in A_{z,1}$ und $k \equiv \ell \equiv r \pmod{z}$ als Antwort, die die Restklasse r widersprüchlich festlegen. Die Existenz der beiden Beispiele folgt, da für ein festes b gilt, dass $\mathbb{Z}_z = \{m \bmod z \mid m \in A_{z,b}\}$. Dieser Widerspruch führt zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klasse $M(y, z)$ und, da für einen beliebigen Primfaktor p_i aus $k \equiv \ell \equiv r \pmod{z}$ sofort $k \equiv \ell \equiv r \pmod{p_i}$ folgt, auch zum Nachweis der Unlösbarkeit des Konsistenzproblems der Klasse $M(p_i, p_i)$. Da aber keiner der Primfaktoren mehr offen war und zum Füllen eines Primfaktors p_i bereits $p_i - 2 + 1$ Äquivalenzfragen nötig waren, kann dies vernachlässigt werden. \square

Wenn der Lernalgorithmus nun beliebige DFA als Äquivalenzfrage stellen darf, dann scheint der Beweis der Beobachtung 7.3.1 zusammenzuberechnen, da der Algorithmus das Akzeptanzverhalten der der gewählten Restklasse r zugeordneten Zustände beliebig wählen darf. Dadurch wäre es möglich, dass negative Beispiele a^ℓ mit $\ell \in A_{p_i,0}$ (positive Beispiele a^k mit $k \in A_{p_i,1}$) nur Zustände erreichen, die verwerfen (akzeptieren). Dann kann das Orakel aber kein widersprüchliches Gegenbeispiel als Antwort geben. Ob allerdings wirklich für eine beliebige nicht festgelegte Restklasse r keine zwei Zahlen $\ell \in A_{p_i,0}$ und $k \in A_{p_i,1}$ mit $k \equiv \ell \pmod{z}$ und $k \equiv \ell \equiv r \pmod{p_i}$ existieren, oder ob durch eine etwas andere Definition der Längen in den Mengen $A_{p_i,b}$ die nötigen Beispiele gefunden werden können, verbleibt als offene Frage.

Beim Lernen mit Elementfragen bei bekannter Zustandszahl n wurde eine obere Schranke von $2n - 1$ Elementfragen gezeigt. Wir vermuten allerdings eine obere Schranke von $O(n + \log(n))$. Daher zeigen wir in Beobachtung 7.3.2 einen Ansatz, mit dem nachgewiesen werden kann, dass es keine reguläre unäre Sprache gibt, so dass wir tatsächlich $2n - 1$ Elementfragen stellen müssen. Man sieht beispielsweise sehr schnell, dass

$$\left| \{ \hat{\delta}_{n,z}(0, a^{n+i}) \mid 1 \leq z \leq n \} \right| \geq n - i$$

für ein festes n und $1 \leq i < n$ gilt. Daher läßt sich vermuten, dass ein binäres Suchverfahren anwendbar ist, woraus sich $O(n + \log(n))$ Elementfragen ergeben.

Beobachtung 7.3.2

Es existiert keine reguläre unäre Sprache L , die von einem minimalen DFA mit n Zuständen akzeptiert wird, so dass tatsächlich $2n - 1$ Elementfragen gestellt werden müssen, um L zu erlernen.

Beweis: Wir stellen wie in Satz 6.2.5 zuerst die Worte $a^0, a^1, a^2, \dots, a^{n-1}$ als Elementfrage. Dadurch verbleibt in jeder Klasse $M(n, z)$, mit $1 \leq z \leq n$, genau ein konsistenter DFA. Bezeichne im Weiteren M_z den am ehesten konsistenten DFA aus der Klasse $M(n, z)$. Außerdem seien A_m und A'_m die beiden Äquivalenzklassen der Äquivalenzrelation \equiv_m aus Definition 6.2.4 auf allen DFA für ein $m \geq 0$. Um weitere $n - 1$ Fragen zu erzwingen, darf jede Elementfrage nur genau einen DFA M_z “ausschalten”. Also muss für alle $m \geq n$ entweder $|A_m| = 1$ oder $|A'_m| = 1$ sein (genau ein DFA muss sich hinsichtlich der Akzeptanz des Wortes a^m von allen anderen

unterscheiden), denn andernfalls müssen wir entweder a^m nicht als Elementfrage stellen, oder a^m weist die Inkonsistenz bei mehr als einem DFA nach.

Wir zeigen nun, wie wir mit zwei Elementfragen die Inkonsistenz von wenigstens drei DFA nachweisen (und daher keine weiteren $n-1$ Elementfragen brauchen). Wir stellen a^n als Elementfrage. Es gilt $\hat{\delta}_{n,z}(0, a^n) = n - z \in \{0, 1, 2, \dots, n-1\}$. Damit genau eine Klasse als inkonsistent nachgewiesen wird, muss es genau ein i , mit $0 \leq i < n$, geben so dass a^i ein zu a^n widersprüchliches Beispiel ist. Das heißt natürlich auch, dass der Zustand i anders festgelegt ist, als die restlichen Zustände.

Wir müssen drei Fälle unterscheiden. Angenommen $i < n-2$. Dann sind M_1 und M_2 äquivalent und wir sparen die Elementfrage, um die beiden zu unterscheiden. Angenommen $i = n-1$. Dann ist M_1 inkonsistent und Zustand $n-1$ weist ein anderes Akzeptanzverhalten als die übrigen Zustände auf. Als nächste Frage stellen wir a^{n+3} . Es gilt $\hat{\delta}_{n,2}(0, a^{n+3}) = \hat{\delta}_{n,4}(0, a^{n+3}) = n-1$ und $\hat{\delta}_{n,3}(0, a^{n+3}) = n-3$ und $\hat{\delta}_{n,4}(0, a^{n+3}) = n-2$. Gemäß Voraussetzung gilt $n-3 \in F \Leftrightarrow n-2 \in F$ und $n-1 \in F \Leftrightarrow n-2 \notin F$. Daraus folgt, dass $M_2 \equiv_{n+3} M_4 \not\equiv_{n+3} M_3 \equiv_{n+3} M_5$. Also wird für mindestens zwei DFA die Inkonsistenz nachgewiesen. Angenommen $i = n-2$. Dann ist M_2 inkonsistent und Zustand $n-2$ weist ein anderes Akzeptanzverhalten als die übrigen Zustände auf. Als nächste Frage stellen wir a^{n+10} , da aufgrund analoger Überlegungen wie für $i = n-1$, $M_3 \equiv_{n+10} M_4 \not\equiv_{n+10} M_1 \equiv_{n+10} M_5$. Wiederum weist das Beispiel dann die Inkonsistenz von mindestens zwei Klassen nach. Also ist das Gewünschte gezeigt. \square

Eine untere Schranke für das Lernen mit Elementfragen bei bekannter Zustandszahl konnte nicht gezeigt werden. Trivialerweise liegt sie bei mindestens n , da jeder Zustand des DFA mindestens einmal festgelegt werden muss. Vermutlich kann aber mit dem gleichen Ansatz wie für eine bessere obere Schranke eine untere Schranke von $\Omega(n + \log(\pi(n)))$ nachgewiesen werden.

Die untere Schranke für die Anzahl der Element- und abgeschwächten Äquivalenzfragen fehlt uns auch. Trivialerweise müssen wieder mindestens n Elementfragen gestellt werden. Die Schwierigkeit hierbei liegt aber in der Freiheit des Lernalgorithmus, nach beliebigen Beispielen fragen zu können. Möglicherweise kann ein Algorithmus der mit Elementfragen lernt, einen zu Lemma 6.1.10 “umgekehrten” Weg gehen und durch geschicktes “ansteuern” bestimmter Zustände in den einzelnen Klassen einiges an Fragen (sowohl Element- als auch abgeschwächte Äquivalenzfragen) einsparen.

7.4 Aktives Lernen anderer Sprachklassen

Wie Eingangs erwähnt war die Motivation dieser Arbeit, eine Teilklasse der regulären Sprachen zu finden, die mit Äquivalenzfragen lernbar ist. Neben den regulären unären Sprachen gibt es sicher weitere interessante Klassen, für die, durch eine geschickte Klassifizierung der Hypothesen, die Lernbarkeit in gleicher Weise nachgewiesen werden kann.

Betrachten wir einmal für $\Sigma = \{0, 1\}$ die deterministischen endlichen Automaten, deren Graph ein vollständiger binärer Baum ist, bei dem von jedem “Blatt” und für jede Eingabe ein Rücksprung zur Wurzel erfolgt. Wird zusätzlich gefordert, dass diese Automaten nur in einem “Blatt” akzeptieren, so scheinen die Ergebnisse für reguläre unäre Sprachen direkt übertragbar zu sein, wenn wir statt der Zykluslänge die Höhe des Baumes heranziehen (allerdings wurde dies nicht vollständig verifiziert). Eine andere Sprachklasse wäre eine Konkatenation regulärer unärer Sprachen. Sei $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$ ein endliches Alphabet. Sei $L_i \subseteq \{\sigma_i\}^*$ eine reguläre unäre Sprache. Die Konkatenation $L_1 \cdot L_2 \cdot \dots \cdot L_k$ ist wahrscheinlich mit den gleichen Ansätzen wie für die regulären unären Sprachen lernbar (wobei auch das nicht verifiziert wurde). Diesen Ideen folgend lassen sich dann noch weitere mit Äquivalenzfragen lernbare Teilklassen der regulären Sprachen

beschreiben. Eine wirklich interessante Frage wäre daher, Bedingungen für die Lernbarkeit einer Teilklasse der regulären Sprachen auszuarbeiten.

Eine andere interessante Frage wäre, ob die in dieser Arbeit vorgestellten Ideen auf unäre Sprachen oder auf reguläre unäre Sprachen mit Kellerautomaten mit unärem Eingabealphabet (aber mehr als einem Kellersymbol) als Hypothesenklasse übertragbar sind. Falls nicht, sind hieraus vermutlich Bedingungen für die Lernbarkeit einer Teilklasse der regulären Sprachen ableitbar.

Literaturverzeichnis

- [AKS03] Manindra Agarwal, Neeraj Kayal, Nitin Saxena: PRIMES is in P. Überarbeiteter Vorabdruck.
http://www.cse.iitk.ac.in/news/primes_v3.ps
- [AHU74] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman: The design and analysis of computer algorithms. Addison-Wesley (1974).
- [An82] Dana Angluin: A note on the number of queries needed to identify regular languages. Information and Control, Band 51 (1981), Seiten 76 bis 87.
- [An87] Dana Angluin: Learning Regular Sets from Queries and Counterexamples. Information and Computation, Band 75 (1987), Seiten 87 bis 106.
- [An88] Dana Angluin: Queries and concept learning. Machine Learning (1988), Seiten 319 bis 342.
- [An90] Dana Angluin: Negative results for equivalence queries. Machine Learning (1990), Seiten 121 bis 150.
- [An93] Dana Angluin: Learning with Queries. Computational Learning and and Cognition (1993), Seiten 1 bis 28.
- [BCH86] P. W. Beame, S. A. Cook und H. J. Hoover: Log depth circuits for division and related problems. S.I.A.M. Journal on Computing, Band 15 (1986), Seiten 994 bis 1003.
- [BEHW87] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler und Manfred K. Warmuth: Occam's Razor, Information Processing Letters, Band 24, (1987), Seiten 377 bis 380.
- [BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler und Manfred K. Warmuth: Learnability and the Vapnik Chervonenkis Dimension. Journal of the Association for Computing Machinery, Band 36 (1989), Seiten 929 bis 965.
- [Bonn04] Pressestelle der Rheinischen Friedrich-Wilhelms-Universität (Hg.): Bonner Mathematiker knacken Weltrekordzahl. Pressemitteilung (2004).
http://www.uni-bonn.de/Aktuelles/Presseinformationen/2004/039_druck.html
- [Co93] Henri Cohen: A Course in Computational Algebraic Number Theory. Springer Verlag (1993).

- [Den03] Klaus Denecke: Algebra und Diskrete Mathematik für Informatiker. B.G. Teubner (2003).
- [DH76] Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography. IEEE Transactions on Information Theory, Band 22 (1976), Seiten 644 bis 654.
<http://citeseer.nj.nec.com/diffie76new.html>
- [DKS02] Michael Domaratzki, Derek Kisman, Jeffrey Shallit: On the Number of Distinct Languages Accepted by Finite Automata with n States. Journal of Automata, Languages and Combinatorics, Band 7 (2002), Seiten 469 bis 486.
http://www.cs.queensu.ca/home/domaratz/enum_jalc.ps
- [DTW97] Carlos Domingo, Tatsui Tsukiji, Osamu Watanabe: Partial Occam's Razor and its Applications. Proceedings Workshop on Algorithmic Learning Theory (1997), Seiten 85 bis 99.
<http://citeseer.ist.psu.edu/article/domingo96partial.htm>
- [Du96] Dudenredaktion (Hg.): Duden Deutsches Universalwörterbuch (3. Auflage). Dudenverlag (1996).
- [EHKV89] Andrzej Ehrenfeucht, David Haussler, Michael Kearns, Leslie G. Valiant: A General Lower Bound on the Number of Examples Needed for Learning. Information and Computation, Band 82 (1989), Seiten 247 bis 261.
<http://www.cis.upenn.edu/~mkearns/papers/lower.ps>
- [FKRRSS93] Yoav Freund, Michael Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie: Efficient learning of typical finite automata from random walks. Proceedings of the 24th Annual ACM Symposium on Theory of Computing (1993), Seiten 315 bis 324.
<http://www.cis.upenn.edu/~mkearns/papers/typical.ps>
- [Gil77] J. Gill: Computational complexity of probabilistic Turing machines. SIAM Journal on Computing, Band 6 (1977), Seiten 675 bis 695.
- [Gol78] E. M. Gold: Complexity of automaton identification from given data. Information and Control Band 37 (1978), Seiten 302 bis 320.
- [GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik: Concrete Mathematics. Addison-Wesley (1994).
- [Har78] Michael A. Harrison: Introduction to Formal Language Theory. Addison-Wesley (1978).
- [HU88] John E. Hopcroft, Jeffrey D. Ullmann: Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie. Addison-Wesley Deutschland (1988).
- [Ke90] Michael Kearns: The Computation Complexity of Machine Learning. MIT Press (1990).
<http://www.cis.upenn.edu/~mkearns/papers/thesis.ps>

- [KLPV87] Micheal Kearns, Ming Li, Leonard Pitt, Leslie G. Valiant: On the Learnability of Boolean Formulae. Proceedings of the nineteenth annual ACM conference on Theory of computing (1987), Seiten 285 bis 295.
<http://www.cis.upenn.edu/~mkearns/papers/klvp.ps>
- [KV89] Michael Kearns, Lesslie G. Valiant: Cryptographic limitations on learning Boolean formulae and finite automata. Proceedings of the 21st Annual ACM Symposium on Theory of Computing (1989), Seiten 433 bis 444.
<http://citeseer.nj.nec.com/kearns89cryptographic.html>
- [Kn73] Donald E. Knuth: Fundamental Algorithms (second edition). Addison-Wesley (1973).
- [Kn81] Donald E. Knuth: Seminumerical Algorithms (second edition). Addison-Wesley (1981).
- [PV88] Leonard Pitt, Leslie G. Valiant: Computational Limitations on Learning from Examples. Journal of the Association for Computing Machinery, Band 35 (1988), Seiten 965 bis 984.
- [Rab80] Michael O. Rabin: Probabilistic algorithm for primality testing. Journal of Number Theory, Band 12 (1980), Seiten 128 bis 138.
- [RU95] Reinhold Remmert, Peter Ulrich: Elementare Zahlentheorie. Birkhäuser Verlag (1995).
- [RSA78] Ronald L. Rivest, A. Shamir und Len Adleman: A method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, Band 21 (1978), Seiten 120 bis 126.
<http://theory.lcs.mit.edu/~rivest/rsapaper.ps>
- [RS93] Ronald L. Rivest und R. Shapire: Inference of finite automata using homing sequences. Information and Computation, Band 103 (1993), Seiten 299 bis 347.
- [Schn97] Georg Schnitger: Skript zur Vorlesung "Algorithmisches Lernen". Johann Wolfgang Goethe-Universität (1997).
- [SS77] Robert Solovay, Volker Strassen: A fast Monte-Carlo test for primality. SIAM Journal on Computing, Band 6 (1977), Seiten 84 bis 85.
- [Su87] Gaius Suetonius Tranquillus: Kaiserbiographien. Aus C. Suetonius Tranquillus Sämtliche erhaltene Werke. Phaidon Verlag (1987).
- [Val84] Leslie G. Valiant: A theory of the learnable. Communications of the ACM 27 (1984), Seiten 1134 bis 1142.
- [We03] Eric W. Weisenstein: RSA-576 Factored. mathworld.wolfram.com (2003).
<http://mathworld.wolfram.com/news/2003-12-05/rsa/>
- [ZSG79] Konrat Ziegler, Walter Sontheimer und Hans Gärtner: Der kleine Pauly - Lexikon der Antike in fünf Bänden. Deutscher Taschenbuch Verlag (1979).

Die angegebenen Internetressourcen waren am 31. März 2004 aufrufbar. Eine Verfügbarkeit nach diesem Datum, kann aufgrund der Flüchtigkeit des Internets nicht garantiert werden.