

Lernalgorithmen

SoSe 2012 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Lernalgorithmen

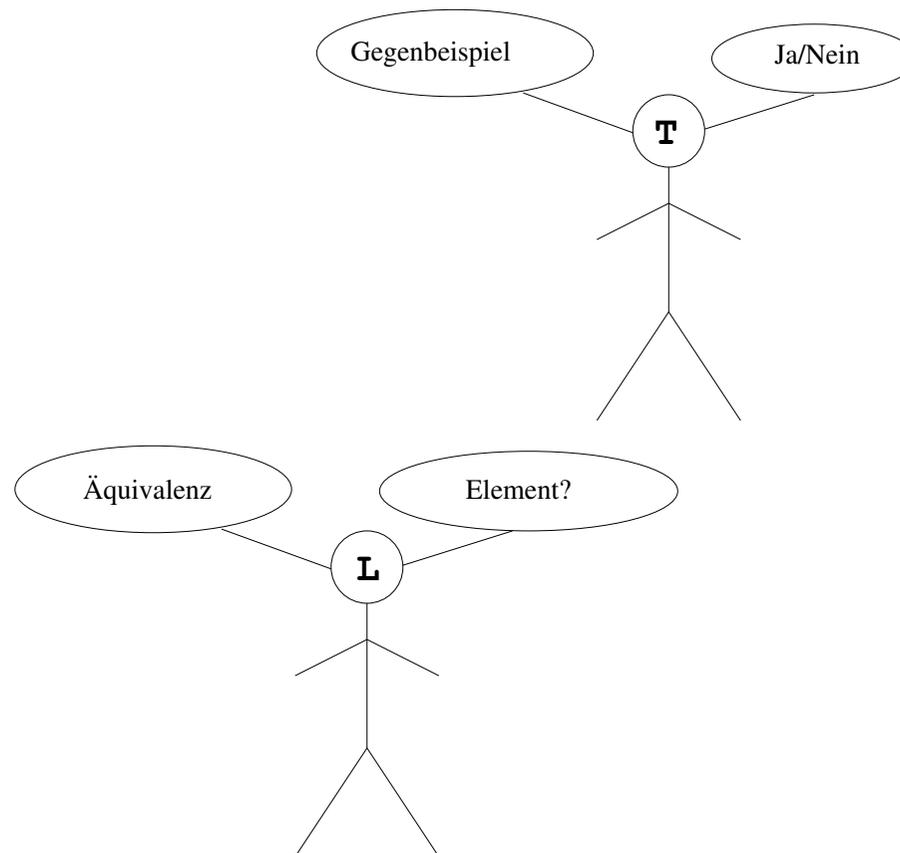
Gesamtübersicht

0. Einführung
1. Identifikation (aus positiven Beispielen)
2. Zur Identifikation regulärer Sprachen, mit XML-Anwendung
3. HMM — Hidden Markov Models
4. Lernen mittels Anfragen & zur Roboterorientierung
5. Lernen mit negativen Beispielen
6. PAC-Lernen

Gliederung des Foliensatzes

- Anfrage-Lernen im Allgemeinen
- Der L^* Algorithmus von Angluin für reguläre Sprachen
- Eine Anwendung zur Roboterorientierung

Schema des Anfrage-Lernens



Polynomialzeitlernen mit einem Lehrer

“minimally adequate teacher” MAT Modell

Anfragetypen

Im Bereich des Lernens regulärer Sprachen und ähnlicher Objekte sind hauptsächlich folgende Typen von Anfragen betrachtet worden:

Elementaranfragen: Ist $x \in L$?

Antwort: “Ja” oder “Nein”.

Äquivalenzanfragen: Erzeugt der Automat M die Sprache L ?

Antwort: “Ja” oder ein Gegenbeispiel x , auf dem M einen Fehler macht.

Teilmengenanfragen: Erzeugt M eine Teil-/Ober-Menge von L ?

Antwort: “Ja” oder “Nein”.

Gegenbeispiele sind notwendig bei Äquivalenzanfragen

Sonst: exponentiell viele Anfragen möglich!

Um eine Sprache der Form $\{x\}$ zu lernen (die von einem endlichen Automaten mit $|x| + 1$ Zuständen erkannt wird), müssen ggf. alle Strings y der Länge $|x|$ durchprobiert werden;

das heißt: der Lernalgorithmus muss fragen, ob $y \in L$ ist (oder ob $L = \{y\}$ ist).

Dieser Aufwand ist exponentionell.

Umgekehrt genügt aber auch ein exponentioneller Aufwand:

Der Lernalgorithmus erzeugt in der Reihenfolge ihrer Größe alle endlichen Automaten und stellt Äquivalenzanfragen.

Eie polynomiale Zeitbeschränkung ist so die **wesentliche Hürde** beim Algorithmenentwurf.

Literatur

D. Angluin. Learning regular sets from queries and counterexamples. *Information and Control* 75 (1987), pp. 87–106.

R. L. Rivest und R. E. Schapire. Inference of finite automata using homing sequences. *Information and Control* 103 (1993), pp. 299–347.

H. Fernau und J. M. Sempere. Permutations and control sets for learning non-regular language families. IN: *Proceedings of the International Conference on Grammatical Inference ICGI 2000*.

Vergleiche auch das entsprechende Kapitel von Frank Stephans Skriptum.

Anfrage-Lernen von regulären Sprachen

Satz: [Angluin] Es gibt einen Algorithmus (genannt L^*),
der für jeden unbekanntem, endlichen Automaten M der Größe n
in Zeit $p(n, m)$
mittels Äquivalenzanfragen und Elementanfragen
einen Automaten N lernt, der zu M äquivalent ist.
Dabei ist p ein Polynom und m die größte Länge eines Gegenbeispiels.

Hilfsdefinitionen und -aussagen

Der Algorithmus L^* braucht eine spezielle Datenstruktur:

Die Beobachtungstabelle (S, E, T) ist gegeben wie folgt:

E (*Experimente*) ist eine endliche, unter Suffixbildung abgeschlossene Menge von Wörtern, d.h.: $\forall te \in E : e \in E$.

S (*potentielle Zustände*) ist eine endliche, unter Präfixbildung abgeschlossene Menge von Wörtern, d.h.: $\forall st \in S : s \in S$.

T ist eine endliche Funktion $T : S \cdot (\Sigma \cup \{\lambda\}) \cdot E \rightarrow \{0, 1\}$, die für jedes $s \cdot t \cdot e \in S \cdot (\Sigma \cup \{\lambda\}) \cdot E$ speichert, ob der gesuchte Automat M die Eingabe ste akzeptiert oder nicht.

Bezeichnung: $\text{row}(s) = \{(e, T(se)) \mid e \in E\}$.

(S, E, T) heißt *konsistent*, wenn:

$\forall s_1, s_2 \in S \forall a \in \Sigma : \text{row}(s_1) = \text{row}(s_2) \rightsquigarrow \text{row}(s_1 a) = \text{row}(s_2 a)$.

(S, E, T) ist *geschlossen*, wenn:

$\forall s_1 \in S \forall a \in \Sigma \exists s_2 \in S : \text{row}(s_2) = \text{row}(s_1 a)$.

T_4	λ	a
λ	1	0
a	0	1
b	0	0
bb	1	0
aa	1	0
ab	0	0
ba	0	0
$bb a$	0	1
bbb	0	0

Automaten aus Beobachtungstabellen

Zu jeder konsistenten und geschlossenen Tabelle (S, E, T) definiert man den Automaten $M(S, E, T)$ wie folgt:

Zustandsmenge ist $Q = \{\text{row}(s) \mid s \in S\}$.

Startzustand ist $q_0 = \text{row}(\lambda)$.

Akzeptierende Zustände F sind alle $\text{row}(s)$ mit $T(s) = 1$.

Übergangsfunktion δ : $\delta(\text{row}(s), a) = \text{row}(sa)$.

Die Übergangsfunktion ist **wohldefiniert**:

Sind $s_1, s_2 \in S$ mit $\text{row}(s_1) = \text{row}(s_2)$, so gilt für jedes $a \in \Sigma$ wegen der Konsistenz der Beobachtungstabelle $\text{row}(s_1a) = \text{row}(s_2a)$, und da die Tabelle geschlossen ist, ist $\text{row}(s_1a) = \text{row}(s)$ für ein $s \in S$.

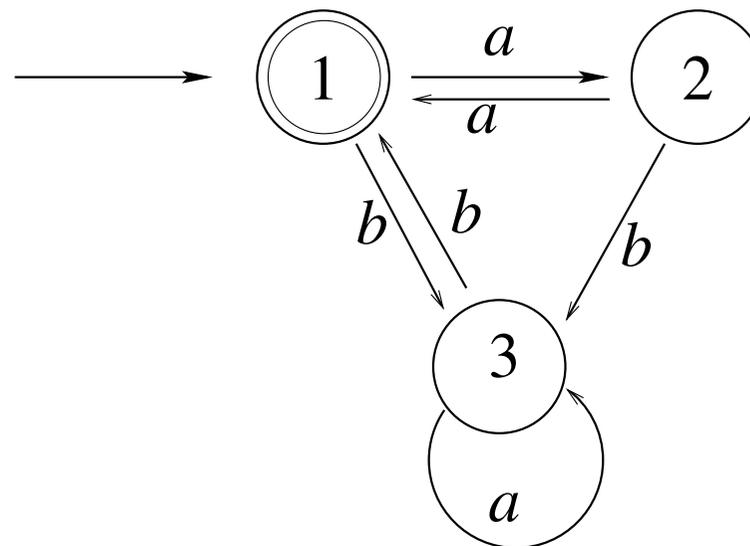
Durch Induktion über die Länge der Eingaben lässt sich zeigen:

Lemma: Für jeden Automaten $M(S, E, T)$ einer geschlossenen konsistenten Beobachtungstabelle (S, E, T) gilt: Ist s aus $S \cup S\Sigma$, so gilt $\delta^*(q_0, s) = \text{row}(s)$.

Ein Beispiel

T_4	λ	a
λ	1	0
a	0	1
b	0	0
bb	1	0
aa	1	0
ab	0	0
ba	0	0
bba	0	1
bbb	0	0

T_4 entspricht der Automat:



Lemma: Für jeden Automaten $M(S, E, T)$ einer geschlossenen konsistenten Beobachtungstabelle (S, E, T) gilt: $M(S, E, T)$ ist konsistent mit T , d.h., ist s aus $S \cup S\Sigma$ und e aus E , so ist $se \in L(M(S, E, T))$ genau dann, wenn $T(se) = 1$.

BEWEIS. Wir zeigen das Lemma durch Induktion über die Länge von e .

Ist $e = \lambda$ und $s \in (S \cup S\Sigma)$, so gilt $\delta^*(q_0, se) = \text{row}(s)$ mit dem voranstehenden Lemma.

Liegt $s \in S$, so: $\text{row}(s)$ Endzustand $\iff T(s) = 1$.

Liegt $s \in S\Sigma$, so: $\exists s' \in S : \text{row}(s) = \text{row}(s')$, denn die Beobachtungstabelle ist geschlossen.

$\text{row}(s) = \text{row}(s')$ ist Endzustand $\iff T(s') = 1 \iff T(s) = 1$.

IV: Die Behauptung gilt für alle $e \in E$ der Länge höchstens k .

Betrachte ein $e \in E$ der Länge $k + 1$.

Da E suffixabgeschlossen ist, gilt $e = ae'$ für einen Buchstaben a und ein $e' \in E$.

Sei s ferner ein beliebiges Element aus $S \cup S\Sigma$.

Da die Beobachtungstabelle geschlossen ist, gibt es ein $s' \in S$ mit $\text{row}(s) = \text{row}(s')$. \rightsquigarrow

$$\begin{aligned} \delta^*(q_0, se) &= \delta^*(\delta^*(q_0, s), ae') && \text{wegen des vorigen Lemmas} \\ &= \delta^*(\text{row}(s), ae') && \text{denn } \text{row}(s) = \text{row}(s') \\ &= \delta^*(\text{row}(s'), ae') && \\ &= \delta^*(\delta(\text{row}(s'), a), e') && \text{per def. von } \delta \\ &= \delta^*(\text{row}(s'a), e') && \text{wegen des vorigen Lemmas} \\ &= \delta^*(\delta^*(q_0, s'a), e') && \\ &= \delta^*(q_0, s'ae'). \end{aligned}$$

Wir können IV auf e' anwenden:

$\delta^*(q_0, s'ae')$ ist ein Endzustand genau dann, wenn $T(s'ae') = 1$ ist.

Wegen $\text{row}(s') = \text{row}(s)$ und $ae' = e \in E$ gilt $T(s'ae') = T(sae') = T(se)$.

Also ist $\delta^*(q_0, se)$ Endzustand genau dann, wenn $T(se) = 1$. □

Lemma: Für jeden Automaten $M(S, E, T)$ einer geschlossenen konsistenten Beobachtungstabelle (S, E, T) gilt: Jeder mit T konsistente DEA $M' = (\Sigma, Q', q'_0, F', \delta')$ mit höchstens soviel Zuständen wie $M(S, E, T)$ ist isomorph zu $M(S, E, T)$.

BEWEIS. Wir konstruieren solch einen Isomorphismus:

Definiere $\text{row} : Q' \times E \rightarrow \{0, 1\}$ mit $\text{row}(q')(e) = 1 \Leftrightarrow \delta'^*(q', e) \in F'$.

Da M' konsistent mit T ist, gilt:

$\forall s \in (S \cup S\Sigma) \forall e \in E : \delta'^*(q_0, se) \in F' \Leftrightarrow T(se) = 1.$

$\leadsto \delta'^*(\delta'^*(q_0, s), e) \in F' \iff T(se) = 1,$

d.h., $\text{row}(\delta'^*(q_0, s))$ ist gleich $\text{row}(s)$ in $M(S, E, T)$.

Da $s \in S$ beliebig, ist $\{\text{row}(\delta'^*(q_0, s)) \mid s \in S\} \supseteq Q$.

Also hat M' genauso viele Zustände wie $M(S, E, T)$.

Es gibt daher eine Funktion ϕ , die jedem $\text{row}(s) \in Q$ in eindeutiger Weise ein $q' \in Q'$ zuordnet mit $\text{row}(s) = \text{row}(q')$;

setze nämlich $\phi(\text{row}(s)) = \delta'^*(q'_0, s)$. ϕ ist bijektiv.

Wir müssen noch zeigen: Automatenstrukturerhaltung.

Anfangszustand: $\phi(q_0) = \phi(\text{row}(\lambda)) = \delta'^*(q'_0, \lambda) = q'_0$.

Endzustände: z.z.: $\phi(F) = F'$.

Liegt $\text{row}(s)$ in F , so gilt $T(s) = 1$;

also $\phi(\text{row}(s)) \in F'$ wegen $\text{row}(\phi(\text{row}(s))) = \text{row}(s)$.

Liegt $\phi(\text{row}(s)) \in F'$, so $T(s) = 1$, d.h., $\text{row}(\phi(\text{row}(s))) = \text{row}(s) \in F$.

Als Übungsaufgabe zeige man, dass für alle $s \in S$ und Zeichen $a \in \Sigma$ gilt:

$\phi(\delta^*(\text{row}(s), a)) = \delta'^*(\phi(\text{row}(s)), a)$.

Dabei benutze man, dass man o.E. voraussetzen kann, dass aus $\text{row}(q') = \text{row}(p')$ für zwei Zustände $q', p' \in Q'$ ihre Gleichheit folgt. \square

Satz: Es sei (S, E, T) eine konsistente geschlossene Beobachtungstabelle.

Dann ist $M(S, E, T)$ der bis auf Isomorphie eindeutig bestimmte kleinste DEA (Minimalautomat), der konsistent mit T ist.

Beweis des Satzes von Angluin: Der *Lernalgorithmus* L^* :

Initialisiere $S = \{\lambda\}$ und $E = \{\lambda\}$.

(1) % Beginn Schleife 1, um M zu lernen.

(2) % Mache (S, E, T) konsistent und geschlossen:

Solange T auf einem $t \in S \cdot (\Sigma \cup \{\lambda\}) \cdot E$ undefiniert,

bestimme $T(t)$ mittels einer *Elementanfrage* bei t .

Wenn (S, E, T) nicht konsistent ist,

dann finde $s_1, s_2 \in S$, $a \in \Sigma$ und $e \in E$ mit $\text{row}(s_1) = \text{row}(s_2) \wedge T(s_1ae) \neq T(s_2ae)$;

addiere ae zu E und gehe zu (2).

Wenn (S, E, T) nicht geschlossen ist,

dann finde $s_1 \in S$ und $a \in \Sigma$ mit $\text{row}(s_1a) \neq \text{row}(s_2)$ für alle $s_2 \in S$;

addiere s_1a zu S und gehe zu (2).

% *Äquivalenzanfrage*:

Wenn $L(M(S, E, T)) \neq L(M)$, **dann** addiere alle Präfixe des Gegenbeispiels zu S ; gehe zu (1).

Trivialerweise ist der Algorithmus korrekt, sofern er terminiert.

Warum terminiert L^* ?

Es seien L die zu lernende reguläre Sprache und n die Zustandszahl von $A(L)$. Beachte, dass wegen des vorigen Satzes nie ein Automat mit mehr als n Zuständen konstruiert werden wird.

Wir zeigen: Schleife (2) kann höchstens $n - 1$ -mal durchlaufen werden, denn die Anzahl verschiedener Zeilen ist gerade die Zustandszahl von $M(S, E, T)$.

Angenommen, ein Wort wird zu E hinzugefügt, weil die Tabelle inkonsistent ist. Dann wächst die Zahl der verschiedenen Zeilen $\text{row}(s)$ um wenigstens 1, da zwei bislang gleiche Zeilen $\text{row}(s_1)$ und $\text{row}(s_2)$ nach der Erweiterung von E verschieden sein müssen. Außerdem bleiben zwei verschiedene Zeilen trivialerweise verschieden nach irgendeiner E -Erweiterung.

Angenommen, ein Wort $s'a$ wird zu S hinzugefügt, weil die Tabelle nicht geschlossen ist. Per Def. ist $\text{row}(s'a) \neq \text{row}(s)$ für alle "alten" s . Also wächst die Zahl der verschiedenen Zeilen $\text{row}(s)$ auch in diesem Fall um wenigstens 1. \square

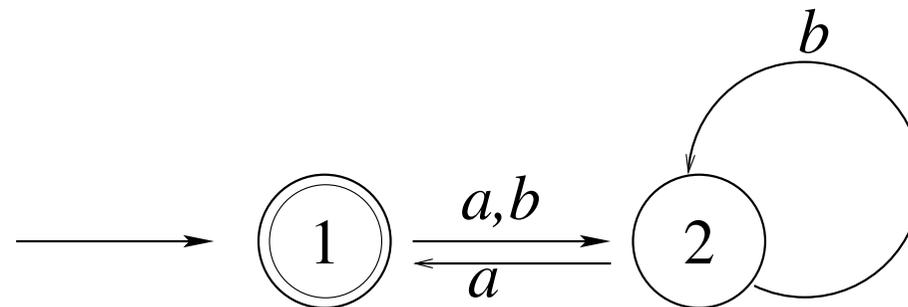
Mitteilung: Genau genommen benötigt der Algorithmus L^* höchstens $n - 1$ Äquivalenzanfragen und $O(|\Sigma|mn^2)$ viele Elementaranfragen.

Rivest und Schapire haben eine Modifikation vorgeschlagen, die mit $O(|\Sigma|n^2 + n \log m)$ vielen Elementaranfragen auskommt.

Beispiel: Betrachte $L \subseteq \Sigma^*$ mit $\Sigma = \{a, b\}$, deren Wörter alle eine gerade Anzahl von a s und eine gerade Anzahl von b s enthalten.

Die Anfangsbeobachtungstabelle T_1 enthält (nach drei Elementanfragen) die Werte $T_1(\lambda) = 1$, $T_1(a) = T_1(b) = 0$ und es ist $S = E = \{\lambda\}$. T_1 ist konsistent, aber nicht geschlossen, da (z.B.) $\text{row}(a) \neq \text{row}(\lambda)$.

L^* fügt daher a zu S hinzu und fragt nach der Mitgliedschaft von aa und ab , i.e., $T_2(aa) = 1$ und $T_2(ab) = 0$. T_2 (das außerdem die Werte von T_1 enthält) ist konsistent und geschlossen und führt zur Äquivalenzanfrage für den Automaten:



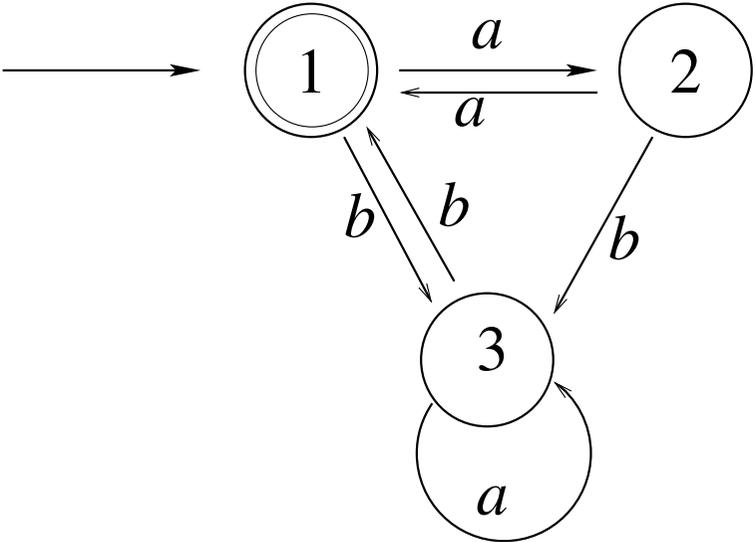
Angenommen, der Lehrer wählt als Gegenbeispiel bb .
 L^* fügt daher b und bb zu S hinzu (λ ist ja schon enthalten) und erfragt die Wörter ba , bba und bbb , um die Beobachtungstabelle T_3 zu erhalten.

Wegen $\text{row}(a) = \text{row}(b)$, aber $\text{row}(aa) \neq \text{row}(ba)$ ist diese Tabelle inkonsistent.

Deshalb fügt L^* a zu E hinzu und konstruiert mittels von fünf weiteren Elementanfragen die folgende Tabelle T_4 :

T_4	λ	a
λ	1	0
a	0	1
b	0	0
bb	1	0
aa	1	0
ab	0	0
ba	0	0
bba	0	1
bbb	0	0

T_4 entspricht der Automat:

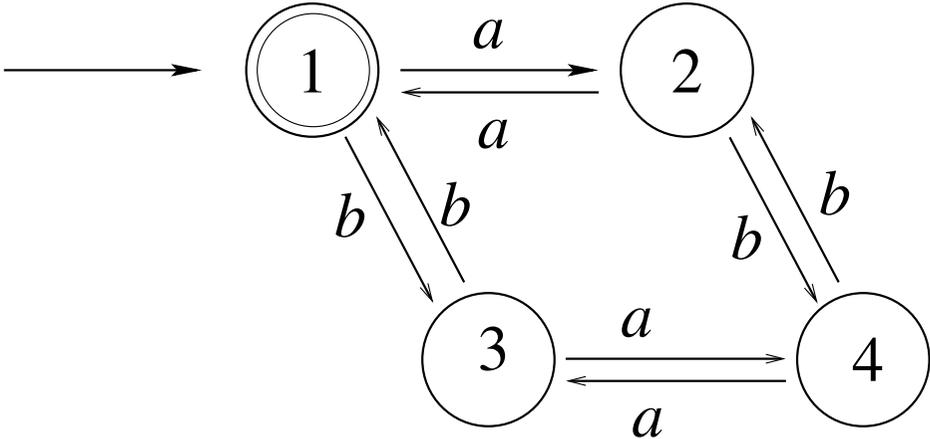


Nehmen wir nun an, der Lehrer gebe das Gegenbeispiel abb .
 Daraufhin fügt L^* ab und abb zu S hinzu.
 Mit fünf weiteren Elementanfragen erhält L^* :

T_5	λ	a	b für T_6
λ	1	0	0
a	0	1	0
b	0	0	1
bb	1	0	0
ab	0	0	0
abb	0	1	0
aa	1	0	0
ba	0	0	0
bba	0	1	0
bbb	0	0	1
aba	0	0	1
$abba$	1	0	0
$abbb$	0	0	0

Die Tabelle ist geschlossen, aber inkonsistent, da $\text{row}(b) = \text{row}(ab)$, aber $\text{row}(bb) \neq \text{row}(abb)$.
 $\rightsquigarrow T_6$

Automat zu T_6 :



Naheliegende Fragen

Man könnte im Grunde zu jeder Beobachtungstabelle einen sie verallgemeinernden endlichen Automaten angeben (mit Zuständen $\text{row}(s)$ für $s \in S$).

Welche Eigenschaften hätte so ein Automat, wenn:

- die Tabelle nicht vollständig mit Elementanfragen ausgefüllt wäre?
- die Tabelle inkonsistent wäre? (Bsp.: T_3)
- die Tabelle nicht geschlossen wäre? (Bsp.: T_1)

Satz: Eine reguläre Sprache kann in Zeit $O(n^3 + m^3)$ mittels Äquivalenzanfragen gelernt werden, wobei n die Anzahl der Zustände des kleinsten Automaten für diese Sprache und m die maximale Länge eines Gegenbeispiels ist.

BEWEIS. Wir beschränken uns beim folgenden Argument auf Sprachen über $\{a, b\}$.

Es sei M_0, M_1, \dots eine Liste aller endlichen Automaten, geordnet nach der Anzahl ihrer Zustände. L_0, L_1, \dots seien die von ihnen erkannten Sprachen.

Daher hat M_k höchstens $\log(k)$ Zustände.

Die Grundidee des Algorithmus ist, die Automaten M_0, M_1, \dots in der Reihenfolge ihrer Größe durchzuprobieren und gleichzeitig mit den Anfragen N_0, N_1, \dots das Gegenbeispiel so hochzutreiben, dass die polynomiale Schranke in n und m erhalten bleibt.

Der Algorithmus läuft wie folgt:

Setze $k = 0$.

(*) % Schleifenstart

Produziere einen Automaten N_k für $L_k \triangleq \{a^k\}$.

Frage, ob N_k die gesuchte Sprache erzeugt.

Bei Antwort "Ja" gib N_k aus und terminiere.

Bei Antwort "Nein" mit Gegenbeispiel $x \neq a^k$,
setze $k = k + 1$ und geh nach (*).

Bei Antwort "Nein" mit Gegenbeispiel a^k :

frage, ob M_k die Sprache erzeugt.

Wenn "Ja", gib M_k aus und terminiere.

Wenn "Nein", setze $k = k + 1$ und geh nach (*).

N_0, N_1, \dots werden zuerst gefragt, damit in dem Fall, wenn M_k die gesuchte Sprache erzeugt, noch gerade eben vorher ein entsprechend langes Gegenbeispiel ausgegeben wird.

Antwortet der Lehrer mit einem anderen Gegenbeispiel als a^k , so weiß man, dass auch M_k falsch ist.

Der Algorithmus terminiert nach endlich vielen Schritten gemäß einem der beiden folgenden Fälle:

- $L_k \triangle \{a^k\}$ ist die gesuchte Sprache, die von N_k erzeugt wird. M_k hat $\log(k)$ Zustände und $L_k \triangle \{a^k\}$ kann von einem Automaten mit n Zuständen erzeugt werden. Der Produktautomat akzeptiert $\{a^k\} = L_k \triangle (L_k \triangle \{a^k\})$ mit $n \log(k)$ Zuständen. Es folgt $n \geq \frac{k+1}{\log(k)}$, da ein endlicher Automat, der $\{a^k\}$ akzeptiert, mindestens $k + 1$ Zustände benötigt.

- L_k ist die gesuchte Sprache. Das letzte Gegenbeispiel hatte die Länge k und daher ist $m \geq k$.

Jeder Schleifendurchlauf benötigt $O(k \cdot \log(k))$ Schritte.

(Im Wesentlichen um den Automaten N_k hinzuschreiben und zu vergleichen, ob das Antwortwort a^k ist. Die Rechnungen mit M_k fallen dagegen kaum ins Gewicht.)

Da k Schleifendurchläufe stattgefunden haben, ist der Gesamtaufwand $O(k^2 \cdot \log(k))$. In beiden Fällen gilt außerdem $n + m \geq \frac{k}{\log(k)}$ und der Gesamtaufwand des Lernalgorithmus kann mit $O(n^3 + m^3)$ nach oben abgeschätzt werden. \square

L^* erfüllt *Uniformitätsbedingung*:

Bei der k -ten Anfrage ist die Rechenzeit bis zu dieser Anfrage sowie auch ihre Länge polynomial beschränkt in n und der maximalen Länge m' der Gegenbeispiele zu den Anfragen $1, 2, \dots, k - 1$.

Das verhindert den **Trick** aus dem vorigen Satz.

Mitteilung: Es gibt einen Algorithmus und ein Polynom p , der jede unbekannte, von einem Automaten mit n Zuständen erzeugte, reguläre Sprache in Zeit $p(n)$ mittels Teilmengenanfragen an einen Lehrer lernt.

Zur Orientierung von Robotern

Eine *Umgebung* E ist ein Tupel $(Q, \Sigma, \delta, q_0, F)$, wobei
 Q eine endliche Zustandsmenge,
 Σ das Alphabet der möglichen Aktionen,
 $\delta : Q \times \Sigma \rightarrow Q$ die Transitionsfunktion,
 $q_0 \in Q$ der Startpunkt und
 $F : Q \rightarrow \{0, 1\}$ die Beobachtungsfunktion (Endzustand ?) sind.

Schreibweise für die Beobachtungsfolge als Reaktion auf eine Aktionenfolge, anfangend in einem gewissen Zustand $q \in Q$:

wir setzen $q\langle\lambda\rangle = F(q)$ und

$q\langle v\alpha\rangle = q\langle v\rangle F(\delta^*(q, v\alpha))$ für beliebige $v \in \Sigma^*$ und $\alpha \in \Sigma$.

Ferner sei $Q\langle w\rangle = \{q\langle w\rangle \mid q \in Q\}$ für $w \in \Sigma^*$.

Näheres finden Sie in der Arbeit von Rivest und Schapire.

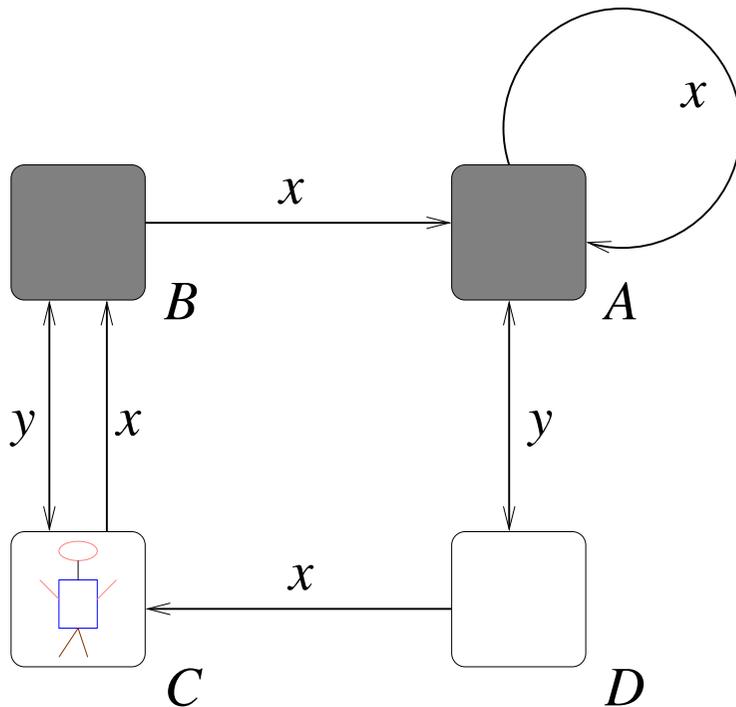
Wir sagen, dass die Folge w zwei Zustände q_1, q_2 *unterscheidet*, wenn $q_1 \langle w \rangle \neq q_2 \langle w \rangle$.

Erinnerung: In einem deterministischen Minimalautomaten mit n Zuständen gibt es zu jedem Zustandspaar stets (mindestens) eine unterscheidende Eingabe der Länge höchstens n .

Eine Aktionsfolge h heiÙe *Ortungsfolge* (engl.: homing sequence), falls für alle $q_1, q_2 \in Q$ aus $q_1 \langle h \rangle = q_2 \langle h \rangle$ die Zustandsgleichheit $\delta^*(q_1, h) = \delta^*(q_2, h)$ folgt.

Bemerkung: Ortungsfolgen sind ursprünglich im Zusammenhang mit dem Testen von Schaltkreisen betrachtet worden, s. das 13. Kapitel in Z. Kohavi: *Switching and Finite Automata Theory*, McGrawHill, 1978.

Beispiel: Ein Roboter muss sich in folgender einfachen Umgebung orientieren:



Als Antwort auf die Aktionsfolge xy beobachtet der Roboter, ausgehend von Zustand C, $C\langle xy \rangle = 010$, wenn wir 0 mit “weißer Zustand” und 1 mit “grauer Zustand” interpretieren.

xy unterscheidet die Zustände C und D, aber nicht die Zustände A und B.

Die Aktion x ist Ortungsfolge:

Aus $q\langle x \rangle = 00$ folgt $\delta(q, x) = C$,

$q\langle x \rangle = 01$ impliziert $\delta(q, x) = B$ und

$\delta(q, x) = 11$ liefert $\delta(q, x) = A$.

Satz: [Moore] Jeder deterministische Minimalautomat mit n Zuständen hat eine Ortungsfolge der Maximallänge n^2 .

BEWEIS. Anstelle eines formalen Beweises betrachte man folgendes Programm-fragment:

Setze $h = \lambda$

Solange es $q_1 \neq q_2$ in Q gibt mit:

$q_1 \langle h \rangle = q_2 \langle h \rangle$ aber $q'_1 = \delta^*(q_1, h) \neq \delta^*(q_2, h) = q'_2$ tue:

Bestimme $x \in \Sigma^*$, das q'_1 und q'_2 unterscheidet.

Setze $h = hx$.

Man überlegt sich leicht, dass die Schleife höchstens n mal durchlaufen wird. \square

Mitteilung: Trotz dieses einfachen Konstruktionsverfahrens für Ortungsfolgen ist es NP-vollständig, eine kürzeste Ortungsfolge zu finden.

Ein Orientierungsalgorithmus (von Rivest und Schapire)

Eingabe: Eine Ortungsfolge h der zu lernenden Umgebung E .

Ausgabe: Der E entsprechende minimale DEA.

1. Führe h aus und beobachte die Folge σ .

Wenn σ noch nicht betrachtet wurde: Erzeuge L_σ^* .

Simuliere die nächste Anfrage von L_σ^* :

Falls L_σ^* eine Elementanfrage für w stellt, so:

gib (durch Experimentieren) L_σ^* die Ausgabe des durch die Aktionsfolge w erreichten Zustands.

Falls L_σ^* eine Äquivalenzanfrage stellt, so:

Ist der mutmaßliche DEA korrekt, so gib ihn aus,
sonst: liefere ein Gegenbeispiel für L_σ^* ; geh zu 1.

Der Algorithmus hat noch mehrere **Nachteile**:

1. Woher soll die Ortungsfolge h kommen?
2. Woher weiß der Roboter, wann "sein" DEA korrekt ist?
3. Woher erhält der Roboter ein Gegenbeispiel?

Zum Lernen nicht-regulärer Sprachen

Unter einer Permutation Ψ wollen wir hier genauer eine Familie von Permutationen ψ_n der Menge $\{1, \dots, n\}$ verstehen.

Ψ lässt sich als Abbildung $\Psi : \Sigma^* \rightarrow \Sigma^*$ deuten, die Wörter der Länge n gemäß ψ_n umordnet, d.h., für $a_1, \dots, a_n \in \Sigma$ gilt:

$$\Psi(a_1 \dots a_n) = a_{\psi_n(1)} \dots a_{\psi_n(n)}.$$

Die Permutation $\Psi(L)$ einer Sprache $L \subseteq \Sigma^*$ ist dann die Menge aller durch Ψ permutierten Wörter von L .

Für eine Sprachfamilie \mathcal{L} ist $\Psi(\mathcal{L})$ die Menge aller durch Ψ permutierten Sprachen aus \mathcal{L} .

Betrachte $(\psi_n^{\text{EL}})^{-1}(1) = 1$, $(\psi_n^{\text{EL}})^{-1}(n) = 2$ und
 $(\psi_n^{\text{EL}})^{-1}(k) = 2 + (\psi_{n-2}^{\text{EL}})^{-1}(k-1)$ für $n > 1$ und $1 < k < n$.
 $\Psi^{\text{EL}} = \{\psi_n^{\text{EL}} \mid n \in \mathbb{N}\} \rightsquigarrow \Psi^{\text{EL}}(\{(ab)^n \mid n \in \mathbb{N}\}) = \{a^n b^n \mid n \in \mathbb{N}\}$.

Satz: [Amar/Putzolu] Es bezeichne REG die regulären Sprachen. Dann gilt: $\text{REG} \subsetneq \Psi^{\text{EL}}(\text{REG})$.

BEWEIS. Das Beispiel liefert die Striktheit der Inklusion.

Weshalb gibt zu jeder regulären Sprache L eine andere reguläre Sprache L' mit $L = \Psi^{\text{EL}}(L')$?

Betrachte eine induzierte Zustandsfolge eines Automaten A für L:

a_1	a_2	a_3	a_4	a_5	\dots	a_{n-3}	a_{n-2}	a_{n-1}	a_n
q_1	q_2	q_3	q_4	q_5	\dots	q_{n-3}	q_{n-2}	q_{n-1}	q_n
a_1	a_n	a_2	a_{n-1}	a_3	a_{n-2}	a_4	a_{n-3}	a_5	\dots
q_1	q_n	q_2	q_{n-1}	q_3	q_{n-2}	q_4	q_{n-3}	q_5	\dots

Der simulierende Automat (für L') muss daher A abwechselnd vorwärts und rückwärts simulieren. \rightsquigarrow Übungsaufgabe ! □

Weitere Literaturhinweise

H. Fernau. Even linear simple matrix languages: formal language properties and grammatical inference. *Theoretical Computer Science*, 289 (2002), Seiten 425–456.

Y. Takada. Learning formal languages based on control sets. In K. P. Jantke und S. Lange, Herausgeber, *Algorithmic Learning for Knowledge-Based Systems*, Band 961 aus der Reihe LNCS, Seiten 317–339. Springer-Verlag, 1995.

Y. Takada. A hierarchy of language families learnable by regular language learning. *Information and Computation* 123 (1995), Seiten 138–145.

Die entsprechenden Grammatik-Formalismen werden auch in der Master-Vorlesung “Formale Sprachen” vorgestellt.