Einführung in die Lerntheorie

Privatdozent Dr. Frank Stephan

Rupprechts-Karls-Universität Heidelberg *

Wintersemester 2002/2003

In dieser Vorlesung wird eine Einführung in die verschiedenen theoretischen Modelle der Lerntheorie gegeben und im besonderen das Gebiet der induktiven Inferenz behandelt. Die induktive Inferenz ist das allgemeinste Konzept der Lerntheorie und untersucht verschiedene Varianten des Lernen im Limes, d.h. die lernende Maschine erhält als Eingabe nach und nach alle Funktionswerte und gibt nach ggf. mehreren Fehlversuchen schliesslich ein Programm aus, das die gesuchte Funktion berechnet.

Die Vorlesung richtet sich an Mathematikstudenten des Hauptstudiums. Kenntnisse aus der Vorlesung "Einführung in die Theoretische Informatik" oder anderweitig erworbene Grundkenntnisse der Berechenbarkeitstheorie sind von Vorteil. Die Vorlesung stellt die Grundlagen der Lerntheorie dar und präsentiert auch einige neue Ergebnisse aus der aktuellen Forschung.

$f(0) = 1$
$f(1) = 2$
$f(2) = 4$
$f(3) = 8$

$\Longrightarrow$ $\boxed{\text{Induktive Inferenz Maschine}}$ $\Longrightarrow$

$f(x) = 1 + x + \frac{x(x-1)}{2} + \frac{x(x-1)(x-2)}{6}$
$f(4) = 15$

"Lernen ist das Finden einer Formel oder das Vorhersagen neuer Funktionswerte aus den bisher gesehenen Daten."

*Mathematisches Institut der Universität Heidelberg, Im Neuenheimer Feld 294, 69120 Heidelberg, Germany, `fstephan@math.uni-heidelberg.de`.

Leichte Überarbeitetung der gleichnamigen Vorlesung aus dem Jahr 1996.

Inhaltsverzeichnis

1 Allgemeine Modellbildung

Lernen im abstrakten Sinn bedeutet, dass man eine Funktion oder Menge aus einer gegebenen Ansammlung von Funktionen oder Mengen identifiziert, entweder durch explizite Angabe eines Programms oder durch Vorhersage des jeweils nächsten, noch unbekannten Funktionswert. Der Lerner – meistens eine berechenbare Maschine – erhält dabei nach und nach Zugang zu dem zu lernenden Objekt und rekonstruiert es aus diesen Daten. Den meisten sind schon einige Beispiele des Lernens bekannt:

- Vorhersage des nächsten Wertes (oft bei Intelligenztests):
  Gegeben $f(0), f(1), \ldots, f(n)$.
  Gesucht $f(n+1)$.

- Interpolation von Polynomen:
  Gegeben $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.
  Gesucht Formel für ein Polynom p (vom Grad n) mit $p(x_m) = y_m$ für $m = 0, 1, \ldots, n$. z.B. $p(x) = \frac{(x-x_0)y_1+(x_1-x)y_0}{x_1-x_0}$ bei $n = 1$.

Diese Beispiele zeigen bereits, dass die Probleme nur unter bestimmten Randbedingungen lösbar sind; im ersten Fall muss die Funktion durch $f(0), f(1), \ldots, f(n)$ eindeutig beschrieben sein, im zweiten darf das Polynom höchstens den Grad n haben. Das heisst zu einer Lernaufgabe gehört neben den jeweils aktuellen Daten noch ein Kontext, eine Klasse S aus welcher die zu lernenden Objekte genommen sind. Die Klasse $S = \{\text{Polynome } p\text{: Grad}(p) \leq n\}$ ist dieser Kontext im zweiten Fall. Wenn man nun alle Polynome lernen will, also nicht nur die eines festen Grades, ist dies nur noch im Limes möglich. Ein solches Verfahren liest in jedem Schritt eine neue Date und gibt auf der Grundlage aller bisher gelesenen Daten eine Hypothese aus:

> Schritt n:
> Lese (x_n, y_n) ein; die (x_m, y_m) mit $m < n$ sind bereits bekannt.
> Gebe das p des kleinsten Grades mit $p(x_m) = y_m$ für alle $m \leq n$ aus.

M liest Schritt um Schritt Daten ein und gibt Hypothesen aus:

Daten				Hypothesen
(x_0, y_0)				$p_0(x) = y_0$
(x_1, y_1)	$\Longrightarrow$	$\boxed{M}$	$\Longrightarrow$	$p_1(x) = \frac{(x-x_0)y_1+(x_1-x)y_0}{x_1-x_0}$
$\ldots$				$\ldots$

Hier ein Beispiel:

Daten	Hypothesen
$(0,0)$	$p(x) = 0$
$(1,1)$	$p(x) = x$
$(2,4)$	$p(x) = x^2$
$(3,9)$	$p(x) = x^2$
$(4,4)$	$p(x) = x^2 - \frac{x(x-1)(x-2)(x-3)}{2}$

Dieses Lernverfahren hat einige Besonderheiten:

- Die Maschine gibt eine Folge von Hypothesen aus.
- Ab einem gewissen n sind alle Hypothesen gleich und korrekt.
- Die Maschine weiss aber nicht, ab wann ihre Hypothesen korrekt sind.

Gold [30] bezeichnete dies als Lernen im Limes. Er definierte allgemein:

> Eine Klasse S von Funktionen ist Ex-lernbar wenn es eine berechenbare Maschine M gibt, so dass die Folge $M(f(0)f(1)\dots f(n))$ gegen einen festen Wert e konvergiert und e (der Code für) ein Programm von f ist.

Hierbei heisst eine Maschine berechenbar, wenn es einen Algorithmus für sie gibt, der M ohne Rücksicht auf Beschränkungen von Rechenzeit und Speicherplatz berechnet. Der Algorithmus muss aber für jede Eingabe terminieren. Ex ist die Abkürzung für "Explanatory", d.h. der Lerner erzeugt (im Limes) ein Programm, das die Funktion erklärt.

Beispiel 1.1 *Die Klassen* $S_0 = \{f : f = \varphi_{f(0)}\}$ *und* $S_1 = \{f : (\forall^\infty x)\,[f(x) = 0]\}$ *sind* Ex-*lernbar.*

Beweis Im ersten Fall ist $M(f(0)f(1)\dots f(n)) = f(0)$ da $f(0)$ immer ein Programm für die zu lernende Funktion f ist. Im zweiten Fall berechnet die Maschine bei jedem n einen kanonischen Index der Funktion $f(0)f(1)\dots f(n)0^\infty$ wie folgt:

Es sei $F = \{(x, f(x)) : f(x) > 0\}$. Dann setze:

$$\varphi_{s(F)}(x) = \begin{cases} y & \text{falls } (x,y) \in F; \\ 0 & \text{sonst.} \end{cases}$$

Der Index $s(F)$ hängt nur von der Menge F und nicht von n ab.

Das mit dem kanonischen Index ist wichtig, damit M nicht bei jedem n eine andere, aber gleichwertige Hypothese erzeugt. ∎

Im folgenden Satz wird gezeigt, dass $S_0 \cup S_1$ von keiner berechenbaren Maschine gelernt wird. Daraus folgt dann insbesondere Gold's Ergebnis [30], dass die Klasse REC aller berechenbaren Funktionen von keiner berechenbaren Maschine gelernt wird.

Satz 1.2 [11, 12] *Die Klassen* REC *und* $S_0 \cup S_1$ *können nicht unter dem Kriterium* Ex *gelernt werden.*

Beweis Es reicht, dies für $S_0 \cup S_1$ zu zeigen. Angenommen eine berechenbare Maschine M lernt S_1. Dann produziert folgender Algorithmus eine totale Funktion $\varphi_{s(e)} = \lim_n \sigma_n$:

> $\sigma_0 = e$; $\sigma_{n+1} = \sigma_n \tau$ für die erste Folge τ endlich vieler natürlicher Zahlen mit $M(\sigma_n \tau) \neq M(\sigma_n)$.

Die Suche nach dem τ terminiert, weil M jede Funktion $\sigma_n y 0^\infty$ lernt. Fast alle dieser Funktionen haben ein anderes Programm als $M(\sigma_n)$ und da M diese Funktionen lernt, gibt es ein τ mit $M(\sigma_n \tau) \neq M(\sigma_n)$. Weil M berechenbar ist, ist die Folge aller σ_n berechenbar. Des weiteren ist kein τ der leere String, also enthält jeder String σ_n mindestens n Zahlen und die Funktion $\varphi_{s(e)} = \lim_n \sigma_n$ ist total und uniform in e berechenbar. Nach dem Fixpunktsatz gibt es ein e mit $\varphi_{s(e)} = \varphi_e$ und der Fixpunkt $f = \varphi_{s(e)}$ erfüllt die Bedingung $f = \varphi_{f(0)}$. Also ist $f \in S_0$ aber M gibt auf f unendlich viele verschiedene Hypothesen aus. Somit ist $S_0 \cup S_1 \notin$ Ex. ∎

Ein wichtiger Gegenstand der Lerntheorie ist das Vergleichen von verschiedenen Lernkriterien. Im folgenden werden neben Ex noch einige weitere gebräuchliche Kriterien angegeben:

Name	Konvergenzkriterium	Nebenbedingung
Ex	endlich viele Hypothesen und letzte Hypothese ist korrekt	keine
Fin	eine korrekte Hypothese und "?" für "keine Hypothese"	keine
PEx	endlich viele Hypothesen und letzte Hypothese ist korrekt	alle Hypothesen sind totale Funktionen
PFin	eine korrekte Hypothese und "?" für "keine Hypothese"	alle Hypothesen sind totale Funktionen

Der Sinn dieser Definitionen ist der folgende: Fin erfordert, dass die lernende Maschine nur eine Hypothese ausgibt und die richtig ist. Dazu darf jedoch ein Zeichen "?"

ausgegeben, solange die Daten nicht für die Hypothese ausreichen. PEx bedeutet, dass die lernende Maschine bei jeder Eingabe (auch bei solcher die zu keiner Funktion aus der zu lernenden Klasse S gehört) entweder ein "?" oder das Programm einer totalen Funktion ausgibt. PFin ist die Kombination beider Konzepte: Der Lerner darf höchstens ein Programm ausgeben und dies muss stets überall definiert sein. "Fin" steht für "Finite" oder "endliches Lernen", "P" in den Zusammensetzungen "PEx" und "PFin" für "Popperean" nach dem Philosophen Popper.

Den folgenden Satz entdeckten van Leeuwen und Barzdin. Sie zeigten, dass PEx äquivalent ist zur Vorhersage des jeweils nächsten Wertes [14, Theorem 2.19].

Satz 1.3 *Eine Klasse S ist* PEx*-lernbar genau dann wenn es eine Maschine M gibt, die für jede Eingabe definiert ist und für jedes $f \in S$ den jeweils nächsten Wert fast immer vorhersagt, d.h.,* $(\forall f \in S)\,(\exists x)\,(\forall y > x)\,[f(y+1) = M(f(0)f(1)\dots f(y))]$.

Beweis Es sei $S \in$ PEx mittels M. Nun definiert man

$$N(f(0)f(1)\dots f(n)) = \varphi_{M(f(0)f(1)\dots f(n))}(n+1).$$

N ist immer definiert, weil M für jede Eingabe den Index einer totalen Funktion ausgibt. Des weiteren ist $M(f(0)f(1)\dots f(n))$ für fast alle n ein Programm für f, die Vorhersage $N(f(0)f(1)\dots f(n))$ ist folglich fast immer korrekt.

Umgekehrt sei nun N eine Maschine, die für jedes $f \in S$ den nächsten Wert fast immer korrekt vorhersagt. Man betrachte nun folgende Funktion

$$F_\sigma(n) = \begin{cases} \sigma(n) & \text{falls } n \in dom(\sigma); \\ N(f(0)f(1)\dots f(n-1)) & \text{sonst.} \end{cases}$$

F_σ ist total, weil N total ist. Wenn $f \in S$, so gilt für fast alle n, dass $F_{f(0)f(1)\dots f(n)} = f$ ist. Somit hat man den folgenden Lernalgorithmus:

> Zu Eingabe $f(0)f(1)\dots f(n)$ berechnet M das kleinste $m \leq n$ mit
>
> $$(\forall k \leq n)\,[F_{f(0)f(1)\dots f(m)}(k) = f(k)].$$
>
> Dann gibt N einen Index $s(f(0)f(1)\dots f(m))$ für $F_{f(0)f(1)\dots f(m)}$ aus, der nur von m und $f(0), f(1), \dots, f(m)$ abhängt.

Diese Normierung ist notwendig, damit M für jedes $f \in S$ nur endlich viele verschiedene Indizes ausgibt. Im Limes konvergiert die Ausgabe von M bei Funktion f gegen $s(f(0)f(1)\dots f(m))$ für das kleinste m mit $F_{f(0)f(1)\dots f(m)} = f$. ∎

Satz 1.4 *Zwischen den vier Lernkriterien besteht folgendes Verhältnis:*

$$
\begin{tikzcd}
& \mathrm{Ex} & \\
\mathrm{Fin} \arrow[ur] & & \mathrm{PEx} \arrow[ul] \\
& \mathrm{PFin} \arrow[ul] \arrow[ur] \arrow[uu] &
\end{tikzcd}
$$

Ein Pfeil wie PFin $\rightarrow$ PEx bedeutet dass jede PFin-lernbare Klasse S auch PEx-lernbar ist. Wo keine Pfeile sind, besteht keine solche Beziehung. Transitive Pfeile wie der von PFin $\rightarrow$ Ex werden bei späteren Diagrammen weggelassen.

Solch eine Beziehung heisst Inklusionsbeziehung. Der Name Inklusionsbeziehung kommt daher, dass man Lernkriterien wie Ex mit der Menge aller Ex-lernbaren Klassen identifiziert. PFin $\rightarrow$ Ex ist daher gleichbedeutend mit PFin $\subseteq$ Ex.

Beweis Die Inklusionen sind offenkundig und ergeben sich direkt aus den Definitionen. Zu zeigen bleibt, dass keine weiteren gelten, hierzu ist es ausreichend zu zeigen, dass Fin und PEx unvergleichbar sind.

$S_0 \in$ Fin $-$ PEx: S_0 ist offenkundig Fin-lernbar, da eine lernende Maschine nur eine einzige Hypothese, nämlich $f(0)$, ausgibt. Nach vorstehendem Satz ist $S \in$ PEx genau dann, wenn es eine totale Maschine gibt, die jedes $f \in S$ fast immer richtig vorhersagt. Nun definiert man – unter Verwendung des Fixpunktsatzes – f wie folgt:

$$
\begin{aligned}
f(0) &= \text{ein Index für } f \\
f(n+1) &= 1 + M(f(0)f(1)\ldots f(n))
\end{aligned}
$$

Nach der Wahl von $f(0)$ ist $f \in S_0$. Andererseits sagt M die Funktion f nie korrekt vorher. Also ist $S_0 \notin$ PEx (mittels diesem M).

$S_1 \in$ PEx $-$ Fin: In Beispiel 1.1 wurde gezeigt, dass $S_1 \in$ Ex ist. Alle von diesem Lernalgorithmus erzeugten Programme sind jedoch total, also zeigt der Beweis bereits $S_1 \in$ PEx. Angenommen nun $S_1 \in$ Fin mittels M. M muss mindestens eine Hypothese auf irgendeiner Eingabe ausgeben, sei also $M(\sigma) \neq ?$ für ein σ. Die Funktionen $\sigma 10^\infty$ und $\sigma 20^\infty$ sind beide in S_1, aber da M seine Hypothese nicht mehr wechseln kann, gibt M für beide das selbe Programm aus. Also kann M eine von ihnen nicht erkennen. ∎

Hier eine Übersicht über die wichtigsten Notationen:

Terminologie	Symbol, Beispiel	Erklärung
Lernkriterium	Ex, Fin, ...	Ansammlung lernbarer Klassen
Klasse 	$S, S_0, S_1, \ldots$ REC	Klasse von berechenbaren Funktionen oder aufzählbaren Mengen Klasse aller berechenbaren Funktionen
Lerner	M, N	Berechenbare Maschine, die im Limes oder endlich lernt
Hypothese	e	Index für das zu lernende Objekt: Programm für Funktion φ_e, Grammatik für Sprache W_e.
Konvergenzkriterium endliche Konvergenz Konvergenz im Limes	Ex, Fin, ... Fin Ex	Definition, wann eine Folge von Hypothesen konvergiert: genau eine Hypothese, letzte von endlich vielen Hypothesen.
Objekt	$f, L \in S$	zu lernende Funktion oder Sprache

Übung 1.5 Beweisen Sie: Jede endliche Klasse $S \subseteq \text{REC}$ kann endlich gelernt werden, d.h. es gilt $S \in \text{Fin}$.

Übung 1.6 Geben Sie eine Klasse $S \in \text{Ex} - \text{Fin}$ von Funktionen an mit der Nebenbedingung, dass ein Lerner M existiert, der auf jedem $f \in S$ höchstens zwei verschiedene Hypothesen ausgibt.

2 Sprachlernen

Sprachen bezeichnen in der Lerntheorie stets aufzählbare Mengen. Sie werden meistens durch eine Grammatik geschrieben, die sie erzeugt, oder aber durch ein Programm, dass eine Aufzählung aller Elements erlaubt. Beim Sprachlernen unterscheidet man zwei Grundtypen:

- Lernen von Informant (positive und negative Daten):
  Die Eingabe für die Lernmaschine ist die charakteristische Funktion der zu lernenden Menge.

- Lernen von Text (nur positive Daten):
  Die Eingabe für die Lernmaschine ist eine beliebige Folge von Zahlen, in der jedes Element der Menge vorkommt, aber sonst nichts. Das Symbol "#" steht dabei für "keine Eingabe", da sonst z.B. die leere Menge keinen Text hätte.

Gerade der zweiten Begriff des Lernen von Text wurde in Anlehnung an das Lernen natürlicher Sprachen geprägt: Wenn ein Archäologe irgendwo die Überreste einer bisher unbekannten Kultur ausgräbt, wird er immer wieder auf Beispiele der Schriftsprache dieser Kultur treffen. So wird er sich nach einiger Zeit ein Bild machen, welche (geschriebenen) Wörter zu der Sprache dieser Kultur gehörten und welche nicht. Dieser Lernvorgang liefert also nur Beispiele für das zu lernende Konzept, aber keine Gegenbeispiele der Art "das Wort XUYPOEUR gehört nicht zu der Sprache." Im folgenden ein Informant und ein Text für die Menge der geraden Zahlen:

- $1, 0, 1, 0, 1, 0, 1, 0, 1, 0, \ldots,$
  d.h., 0 ist gerade, 1 ist ungerade, 2 ist gerade, 3 ist ungerade, ...;

- $2, 4, 8, 16, 32, 64, \#, \#, 0, 6, 4, 10, \ldots,$
  d.h., $2, 4, 8, \ldots$ sind gerade Zahlen.

Während ein Informant in seiner Reihenfolge eindeutig festgelegt ist, kann der Text die Elemente in beliebiger Reihenfolge liefern und braucht auch nicht berechenbar zu sein. Eine Maschine liest immer jeweils eine weitere Date von der (unendlichen) Eingabe und macht dann eine Hypothese, an Hand dieser Folgen von Hypothesen lassen sich die Lernkriterien Fin und Ex auf das Lernen von Sprachen übertragen:

Name	Konvergenzkriterium	Art der Daten
InfEx	endlich viele Hypothesen und letzte Hypothese ist korrekt	Informant
InfFin	eine einzige Hypothese	Informant
TxtEx	endlich viele Hypothesen und letzte Hypothese ist korrekt	Text
TxtFin	eine einzige Hypothese	Text

Die Klasse S_{fin} aller endlichen Mengen ist z.B. sowohl von Text als auch von Informant lernbar. Ein Text-Lerner merkt sich immer alle bisher gesehenen Elemente und gibt dann die endliche Menge aus, die gerade diese Elemente enthält:

Daten	Hypothesen
#	$\emptyset$
0	$\{0\}$
#	$\{0\}$
1	$\{0,1\}$
0	$\{0,1\}$
5	$\{0,1,5\}$
...	...

Diese Klasse kann jedoch nicht endlich gelernt werden, weder von Text noch von Informant. Ein Informant-Lerner muss z.B. die leere Menge erkennen und deshalb ist $M(0^n) \neq ?$ für irgendein n. Mit dem alten Argument, dass es verschiedene endliche Mengen gibt, deren charakteristische Funktion mit 0^n anfängt, zeigt man dann, dass einige Mengen in S_{fin} nicht gelernt werden. Das Lernen von Text ist sogar noch restriktiver:

Satz 2.1 *Wenn* $L_1 \subset L_2$ *dann ist* $\{L_1, L_2\} \notin \text{TxtFin}$.
Wenn L *unendlich ist, dann ist* $S_{fin} \cup \{L\} \notin \text{TxtEx}$.

Beweis Zu einer Menge L bezeichne L^* die Menge aller endlichen Folgen von Elementen aus L sowie #, z.B. $2\,2\,5\,\#\,2,\ 2\,3\,5 \in \{2,3,5,7\}^*$. Wenn eine Maschine L_1 endlich lernt, dann gibt es $\sigma \in L_1^*$ so dass $M(\sigma) = e$ für eine Grammatik e von L_1. Nun ist $L_1 \subset L_2$, also ist σ auch der Anfang eines Textes für L_2. Da M seine Hypothese aber nicht mehr ändern darf, kann M L_2 nicht lernen.

Im zweiten Fall ist der Beweis etwas umfangreicher. Man zeigt zuerst folgendes: Wenn eine Maschine M die Menge L erkennt, dann gibt es ein $\sigma \in L^*$ so dass bei keiner endlichen Fortsetzung $\tau \in L^*$ mehr ein Hypothesenwechsel auftritt, d.h., $M(\sigma\tau) = M(\sigma)$ für alle solche τ. Dazu nimmt man das Gegenteil an und führt es mittels folgender induktiv definierten Folge $\sigma_0, \sigma_1, \ldots$ auf einen Widerspruch:

> Sei $L = \{a_0, a_1, \ldots\}$ und $\sigma_0 = \#$.
> Zu $\sigma_n a_n$ sucht man ein $\tau \in L^*$ mit $M(\sigma_n a_n \tau) \neq M(\sigma_n a_n)$.
> Man setzt $\sigma_{n+1} = \sigma_n a_n \tau$ für dieses τ.

Wenn alle diese σ_n existieren, ist der Limes $T = \lim_n \sigma_n$ eine unendliche Folge, die alle Elemente $a_0, a_1, \ldots$ von L enthält aber keine weiteren. M aber macht auf dieser Folge unendlich viele Hypothesenwechsel und daher wird L nicht von M im Limes gelernt.

Also bricht die Konstruktion bei einem n ab und $\sigma = \sigma_n a_n$ erfüllt die gewünschte Bedingung.

Es gibt also ein σ so dass $M(\sigma\tau) = e$ für einen Index e von L und alle $\tau \in L^*$ ist. Insbesondere ist dann $M(\sigma\#^n) = e$ für alle n. Nun ist

$$range(\sigma) = \{x : x \text{ kommt in } \sigma \text{ vor und } x \neq \#\}$$

endlich und daher ist $range(\sigma) \in S_{fin}$ und $range(\sigma) \neq L$. Also gilt, dass keine Maschine, die L erkennt, auch alle Mengen in S_{fin} erkennt. ▮

Ein solches σ wie im zweiten Teil des Beweises vorkommt, nennt man eine Locking-Sequenz für L. Diese spielen in der induktiven Inferenz eine grosse Rolle. Satz 2.3 fast die wesentlichen Eigenschaften zusammen. Doch zuerst noch mal die formale Definition:

Definition 2.2 $\sigma \in L^*$ heisst Locking-Sequenz für L genau dann wenn
(1) $M(\sigma)$ ist ein Index für L;
(2) $M(\sigma\tau) = M(\sigma)$ für alle $\tau \in L^*$.

Satz 2.3 *Wenn M eine Menge L* TxtEx*-lernt, dann hat L eine Locking-Sequenz. Zu jedem $S \in$* TxtEx *gibt es eine Maschine M die S lernt so dass jeder Text T für jede Sprache $L \in S$ einen Anfang $\sigma \preceq T$ hat, der eine Locking-Sequenz für L ist* [46].

Satz 2.4 *Die vier Sprachlernkriterien haben folgende Inklusionsstruktur:*

$$\text{TxtFin} \to \text{InfFin} \to \text{TxtEx} \to \text{InfEx}$$

Beweis Von allen Inklusionen muss nur InfFin $\to$ TxtEx bewiesen werden, in dem man zu gegebenem InfFin-Lerner M einen TxtEx-Lerner N wie folgt konstruiert:

> Eingabe $\sigma \preceq T$ für einen Text T für L.
> Man setzt $D = range(\sigma)$.
> Falls $M(D(0)D(1)\ldots D(m)) = e \neq ?$ für ein $m \leq |\sigma|$
> - dann Ausgabe e
> - sonst Ausgabe ?.

Es sei $L \in S$ die zu lernende Menge. Zu L gibt es ein n mit $M(L(0)L(1)\ldots L(n)) = e$ für einen Index e von L. Da T ein Text von L ist, gilt für alle hinreichend langen $\sigma \preceq T$, dass $|\sigma| \geq n$ und dass sich $D = range(\sigma)$ und L nicht unterhalb von n unterscheiden, d.h., $D(0) = L(0)$, $D(1) = L(1)$, $\ldots$, $D(n) = L(n)$. Also gibt es $m \leq |\sigma|$ mit $M(D(0)D(1)\ldots D(m)) = e$. Da M ein endlicher Lerner ist, gibt M

für keine Menge zwei verschiedene Hypothesen aus, auch nicht für D (eine zweite Hypothese könnte ggf. unterdrückt werden, da man dann weiss, dass die Eingabe von keiner Menge in S stammt). Also ist $N(\sigma) = e$ und so konvergiert N auf dem Text T zu e.

Zu zeigen bleiben die Nichtinklusionen. Die Klasse $\{\{0\}, \{0, 1\}\}$ ist InfFin-lernbar, da die charakteristische Funktion an der Stelle 1 verrät, welche der beiden Möglichkeiten vorliegt. Sie ist aber – wie bereits gezeigt – nicht TxtFin-lernbar. Die Klasse S_{fin} ist TxtEx-lernbar, aber nicht InfFin-lernbar. Die Klasse $S_{fin} \cup \{\mathbb{N}\}$ ist nicht TxtEx-lernbar. Aber sie ist InfEx-lernbar: Der Lerner gibt solange einen Index für $\{\mathbb{N}\}$ aus bis der Informant durch eine 0 in der Eingabe anzeigt, dass es sich um eine endliche Menge handelt. Danach gibt der Lerner bei Eingabe $\sigma \in \{0, 1\}^*$ immer einen kanonischen Index für die endliche Menge $\{x < |\sigma| : \sigma(x) = 1\}$ aus. Nach endlich vielen Hypothesenwechseln wird die gesuchte endliche Menge erkannt. ∎

Satz 2.5 *Es sei S eine Klasse von Funktionen. Zu jeder Funktion f sei $L_f = \{\langle x, f(x)\rangle : x = 0, 1, \ldots\}$ und $S' = \{L_f : f \in S\}$. Es gilt: $S \in \mathrm{Ex} \Leftrightarrow S' \in \mathrm{InfEx} \Leftrightarrow S' \in \mathrm{TxtEx}$ und $S \in \mathrm{Fin} \Leftrightarrow S' \in \mathrm{InfFin} \Leftrightarrow S' \in \mathrm{TxtFin}$.*

Beweis $S \in \mathrm{Ex} \Rightarrow S' \in \mathrm{TxtEx}$: Es sei $S \in Ex$ mittels M. Folgende Maschine N TxtEx-lernt S'.

> Zu Eingabe σ berechnet N das grösste n und geeignete $y_0, y_1, \ldots, y_n$, so dass $\langle 0, y_0\rangle$, $\langle 1, y_1\rangle$, ..., $\langle n, y_n\rangle \in range(\sigma)$. Die Werte $y_0, y_1, \ldots, y_n$ sind eindeutig bestimmt.
> Sei nun $e = M(y_0 y_1 \ldots y_n)$.
> Dann ist $N(\sigma) = s(e)$ wobei $W_{s(e)} = \{\langle x, \varphi_e(x)\rangle : x \in dom(\varphi_e)\}$.

Die Funktion s kann berechenbar und total gewählt werden, daher ist N eine berechenbare Maschine. Des weiteren ist $y_x = f(x)$ sofern definiert, da $range(\sigma)$ keine Paare (x, y) mit $y \neq x$ enthält. Da T jedes Paar $\langle x, f(x)\rangle$ enthält, geht mit wachsendem $\sigma \preceq T$ auch $n \to \infty$ und $M(y_0 y_1 \ldots y_n)$ konvergiert gegen einen Index für f. N konvergiert dann gegen einen Index für L_f.

$S' \in \mathrm{InfEx} \Rightarrow S \in \mathrm{Ex}$: Es sei $S' \in \mathrm{InfEx}$ mittels M. Die charakteristische Funktion von L_f kann aus der f berechnet werden. Die Paarfunktion ist so definiert, dass $\langle x, y\rangle \geq x$ für alle x gilt. Daher kann $L_f(0)L_f(1)\ldots L_f(n)$ aus $f(0)f(1)\ldots f(n)$ berechnet werden. Also setzt man

> $N(f(0)f(1)\ldots f(n)) = s(M(L_f(0)L_f(1)\ldots L_f(n)))$
> wobei $\varphi_{s(e)}(x) = y$ für das erste y mit $\langle x, y\rangle \in W_e$.

s ist eine berechenbare Funktion und kodiert folgenden Algorithmus: bei gegebenem x wartet der Algorithmus so lange bis ein Paar $\langle x, y\rangle$ nach W_e aufgezählt wird. Dann gibt es y aus. Sobald M konvergiert hat gegen e konvergiert hat und $W_e = L_f$ ist, gilt, dass W_e für jedes x genau ein Paar $\langle x, y \rangle$ enthält. In diesem Fall ist dann $\varphi_{s(e)}$ eine totale Funktion, die zu jedem x dieses $y = f(x)$ ausgibt. Also ist $s(e)$ ein Programm für f.

Der letzte Fall $S' \in \mathrm{InfEx} \Rightarrow S' \in \mathrm{TxtEx}$ folgt umittelbar daraus, das jede unter Text lernbare Menge auch unter Informant gelernt werden kann. Somit schliesst sich der Kreis. Der Beweis für endliches Lernen ist fast wortwörtlich identisch zu dem für Lernen im Limes. ▮

Übung 2.6 Bestimmen Sie für die folgenden Klassen, gemäss welchen Lernkriterien sie gelernt werden können:

(a) $\{L : |L| = 2\}$, d.h. die Klasse aller zweielementigen Sprachen.

(b) $\{L : |L| \leq 2\}$.

(c) $\{L : |\overline{L}| = 1\}$, d.h. die Klasse aller L deren Komplement genau ein Element hat.

(d) $\{L : |\overline{L}| \leq 1\}$, d.h. die Klasse von (c) zuzüglich der Menge $\mathbb{N}$.

3 BC-Lernen und Lernen mit Fehlern

Es ist unentscheidbar, ob eine Funktion φ_e an einer bestimmten Stelle definiert ist. Dies führt dazu, dass bestimmte Klassen von Funktionen nicht unter dem Kriterium Ex gelernt werden können, z.B. die Klasse

$$S_{0,1} = \{f : (\exists y)\,(\forall x \neq y)\,[\varphi_{f(0)}(x)\!\downarrow = f(x)]\,\}$$

aller Funktionen, die bei allen bis auf einer Stelle mit $\varphi_{f(0)}$ übereinstimmen. Um diesem Umstand Rechnung zu tragen, wurden die Begriffe des BC-Lernens und des Lernens mit Fehlern geschaffen.

BC heisst "Behaviorally Correct" und ist ein neues Konvergenzkriterium wie etwa das endliche Lernen und das Lernen im Limes. M lernt eine Funktion f unter dem Kriterium BC wenn $M(\sigma)$ bei fast allen Eingaben $\sigma \preceq f$ ein Programm von f ist.

Zu jedem Kriterium wie z.B. Ex kann man die Variante Ex^a definieren. Dabei bedeutet Ex^a dass die lernende Maschine gegen ein Programm konvergiert, das die zu lernende Funktion an allen bis auf a Stellen richtig berechnet. Dabei kann a sowohl eine natürliche Zahl aus auch das Symbol $*$ für endlich viele Fehler bedeuten. Die Klasse $S_{0,1}$ ist in Ex^1, da $\varphi_{f(0)}$ bis auf eine Stelle korrekt ist. Allgemein definiert man nun $S_{0,a} = \{f : f =^a \varphi_{f(0)}\}$ als die Klasse aller Funktionen, die mit der durch $f(0)$ definierten Funktion an allen bis auf a Stellen übereinstimmen.

Satz 3.1 [14] $\mathrm{Ex} \to \mathrm{Ex}^1 \to \mathrm{Ex}^2 \to \ldots \to \mathrm{Ex}^* \to \mathrm{BC} \to \mathrm{BC}^1 \to \mathrm{BC}^2 \to \ldots \to \mathrm{BC}^*$.

Beweis Die Nichtinklusionen werden nicht bewiesen, sondern es werden nur Beispielklassen angegeben.

- $S_{0,a} = \{f : f =^a \varphi_{f(0)}\} \in \mathrm{Ex}^b \;\Leftrightarrow\; a \leq b$ (wobei $a \leq *$ für alle a).
  D.h. $S_{0,1} \in \mathrm{Ex}^1 - \mathrm{Ex}$, $S_{0,2} \in \mathrm{Ex}^2 - \mathrm{Ex}^1$, … und $S_{0,*} \in \mathrm{Ex}^* - \mathrm{Ex}^a$ für alle $a \in \mathbb{N}$.
- $\{g : (\exists f \in S_{0,1})\,(\forall x, y)\,[g(\langle x, y\rangle) = f(x)]\,\} \in \mathrm{BC} - \mathrm{Ex}^*$.
  Diese Menge erhält man durch das Einführen einer "Dummy-Variable". So wird aus jedem einfachen Fehler eine Menge von unendlich vielen Fehlern, die dann jedes Ex^*-Lernverfahren zerstören.
- $\{f : (\forall^\infty x)\,[\varphi_{f(x)} =^a f]\,\} \in \mathrm{BC}^b \;\Leftrightarrow\; a \leq b$.

Alle Inklusionen folgen aus ihren Definitionen ausser $\mathrm{Ex}^* \to \mathrm{BC}$. Es sei $S \in \mathrm{Ex}^*$ via M. Nun lernt N die Klasse S unter dem Kriterium BC wie folgt:

$$\varphi_{N(\sigma)}(x) = \begin{cases} \sigma(x) & \text{falls } x \in dom(\sigma); \\ \varphi_{M(\sigma)}(x) & \text{sonst.} \end{cases}$$

Irgendwann konvergiert M gegen ein Programm e. e hat nur endlich viele Fehler, sei y die grösste Eingabe, bei der $\varphi_e(y)$ von der zu lernenden Funktion f abweicht. Also gilt für fast alle x, dass $M(f(0)f(1)\dots f(x)) = e$ und – wegen $x > y$ – $N(f(0)f(1)\dots f(x))$ ein Index für f ist. Somit ist die Inklusion bewiesen. ∎

Es bleibt anzumerken, dass $S_{0,a} \in \mathrm{Fin}^a$ ist. Damit sind die Nichtinklusionen noch stärker: $\mathrm{Fin}^a \not\to \mathrm{Ex}^b$ wann immer $a > b$.

Satz 3.2 [14, Leo Harrington] $\mathrm{REC} \in \mathrm{BC}^*$.

Beweis Zu Eingabe σ gibt die lernende Maschine M einen Index der folgenden partiellen Funktion ψ_σ aus:

$$\psi_\sigma(x) = \begin{cases} \varphi_e(x) & \text{für das kleinste } e \leq x \text{ so dass} \\ & \varphi_{e,x}(y)\downarrow = \sigma(y) \text{ für alle } y \in dom(\sigma), \\ 0 & \text{falls es kein solches } e \leq x \text{ gibt.} \end{cases}$$

Hierbei ist $\varphi_{e,x}(y)$ das nach x Rechenschritten vom e-ten Programm bei Eingabe y erzielte Ergebnis, sofern es existiert.

Es sei $f \in \mathrm{REC}$. Dann hat f einen minimalen Index e und es gibt ein n so dass sich alle φ_i mit $i < e$ an einer Stelle $m \leq n$ von f unterscheiden: entweder ist $\varphi_i(m)$ undefiniert oder $\varphi_i(m)\downarrow \neq f(m)$. Also gilt für alle $\sigma = f(0)f(1)\dots f(k)$ mit $k \geq n$:

- Wenn $\varphi_{j,x}(y)\downarrow = \sigma(y)$ für alle $y \in dom(\sigma)$, dann ist $j \geq e$;
- $\varphi_{e,x}(y)\downarrow = \sigma(y)$ für alle $y \in dom(\sigma)$ und alle hinreichend grossen x.

Dann ist $\psi_\sigma(x)\downarrow = \varphi_e(x) = f(x)$ für fast alle x. Folglich gilt $\psi_\sigma =^* f$ für fast alle $\sigma \preceq f$, nämlich für diejenigen ab der Länge n, und M lernt alle berechenbaren Funktionen unter dem Kriterium BC^*. ∎

Es wurde gezeigt, dass eine Klasse genau dann PEx-lernbar ist, wenn ihre Funktionen von einer totalen Maschine fast überall vorhergesagt werden. Die entsprechende Verallgemeinerung charakterisiert die Klasse BC:

Satz 3.3 [14] *Eine Klasse $S \subseteq \mathrm{REC}$ ist genau dann* BC-*lernbar wenn es eine partielle rekursive Maschine N gibt, die jedes $f \in S$ fast überall vorhersagt, d.h., wenn* $(\forall f)\,(\forall^\infty x)\,[N(f(0)f(1)\dots f(x-1))\downarrow = f(x)]$.

Beweis Beide Richtungen werden durch direkte Übersetzung der jeweiligen Lernverfahren gezeigt. Angenommen, $S \in \mathrm{BC}$ mittels M. Dann setzt man

$$N(\sigma) = \varphi_{M(\sigma)}(|\sigma|).$$

N ist partiell-rekursiv. Wenn $f \in S$ ist, dann ist $M(\sigma)$ für fast alle $\sigma \preceq f$ ein Index für f; insbesondere ist die Funktion an der Stelle $|\sigma|$ definiert und gleich $f(|\sigma|)$; also wird jedes $f \in S$ fast überall korrekt vorhergesagt.

Umgekehrt sei N nun eine Maschine, die jedes $f \in S$ fast überall korrekt vorhersagt. Man definiert M so, dass

$$\varphi_{M(\sigma)}(x) = \begin{cases} \sigma(x) & \text{falls } x \in dom(\sigma); \\ N(\varphi_{M(\sigma)}(0)\varphi_{M(\sigma)}(1)\ldots\varphi_{M(\sigma)}(x-1)) & \text{sonst.} \end{cases}$$

Wenn diese induktive Definition an einer Stelle x undefiniert ist (weil N undefiniert ist), dann ist auch $\varphi_{M(\sigma)}(y)$ für alle $y \geq x$ undefiniert. Wenn jedoch σ so lang ist, dass N die Funktion f an jeder Stelle $x \geq |\sigma|$ korrekt vorhersagt, dann ist $\varphi_{M(\sigma)} = f$. Somit sind die Hypothesen fast immer richtig und M lernt jedes $f \in S$ verhaltenskorrekt. ∎

Die Konzepte des Lernens mit Fehlern und des verhaltenskorrekten Lernens lassen sich auch auf das Lernen von Sprachen übertragen. Dabei sind die Inklusionsbeziehungen im Fall des Lernens von Informant die gleichen wie beim Funktionslernen. Beim Lernen von Text gibt es allerdings weniger Inklusionen, insbesondere sind TxtEx^* und TxtBC unvergleichbar.

Satz 3.4 *Beim Lernen von Text gelten folgende Inklusionen:*

$$\begin{tikzcd}
\mathrm{TxtBC} \arrow[rr] & & \mathrm{TxtBC}^1 \arrow[rr] & & \mathrm{TxtBC}^2 \arrow[r] & \ldots \arrow[r] & \mathrm{TxtBC}^* \\
\mathrm{TxtEx} \arrow[u] \arrow[r] & \mathrm{TxtEx}^1 \arrow[r] & \mathrm{TxtEx}^2 \arrow[u] \arrow[r] & \mathrm{TxtEx}^3 \arrow[r] & \mathrm{TxtEx}^4 \arrow[u] \arrow[r] & \ldots \arrow[r] & \mathrm{TxtEx}^* \arrow[u] \\
\mathrm{TxtFin} \arrow[u] \arrow[r] & \mathrm{TxtFin}^1 \arrow[u] \arrow[r] & \mathrm{TxtFin}^2 \arrow[u] \arrow[r] & \mathrm{TxtFin}^3 \arrow[u] \arrow[r] & \mathrm{TxtFin}^4 \arrow[u] \arrow[r] & \ldots \arrow[r] & \mathrm{TxtFin}^* \arrow[u]
\end{tikzcd}$$

Beweis Bis auf die Inklusionen $\mathrm{TxtEx}^{2a} \to \mathrm{BC}^a$ folgen alle Inklusionen direkt aus den Definitionen. Es sei $a < *$ und $S \in \mathrm{TxtEx}^{2a}$ mittels M. Nun definiert man den neuen Algorithmus N durch

$$W_{N(\sigma)} = W_{M(\sigma)} \cup range(\sigma) - \{ \text{ die } a \text{ kleinsten Elemente in } (W_{M(\sigma)} - range(\sigma))\}$$

Es sei $L \in S$. M konvergiert auf einem Text für L gegen e. Dann ist $L = W_e \cup D_1 - D_2$ wobei $|D_1| + |D_2| \leq 2a$. Wenn σ ein hinreichend grosses Anfangsstück des Textes für L ist, dann ist M bereits gegen e konvergiert und alle Elemente der endlichen Menge $D_1 \subseteq L$ sind im Text aufgetaucht. Also gibt N stets Indizes von Mengen $W_e \cup D_1 - D_3$ aus, wobei $|D_3| \leq a$ ist. Man trifft eine Fallunterscheidung.

- $|D_2| \geq a$. Dann ist D_3 bei fast alle Hypothesen von N die Menge der a kleinsten Elemente von W_e, die nicht in L sind. Insbesondere ist $D_3 \subseteq D_2$. L und $W_e \cup D_1 - D_2$ unterscheiden sich an höchstens $2a$ Stellen. a dieser Unterschiede sind die a Elemente von D_3, die in W_e aber nicht in L sind. Somit unterscheidet sich $W_e - D_3$ nur noch an höchstens a Stellen von L und ebenso $W_e \cup D_1 - D_3$ da $D_1 \subseteq L$ ist. Also stimmen fast alle Hypothesen mit L an allen bis auf maximal a Stellen überein.

- $|D_2| < a$. Nun kann man argumentieren, dass fast immer $D_2 \subseteq D_3$ gilt: nach hinreichend langer Zeit sind alle Elemente in $W_e - L$, die kleiner als $\max(D_2)$ sind, im Text aufgetaucht und die a kleinsten Elemente aus $W_e - range(\sigma)$ umfassen die Elemente von D_2. Es folgt, dass sich L von fast allen Hypothesen $W_e \cup D_1 - D_3$ nur an den Elementen der jeweiligen Menge $D_3 - D_2$ unterscheidet. Da $|D_3| \leq a$ ist, sind auch jetzt fast alle Hypothesen modulo a Fehler richtig.

Somit gilt in beiden Fällen, dass N fast immer höchstens a Fehler macht. Also lernt N jedes $L \in S$ und N belegt, dass $S \in \mathrm{TxtBC}^a$ gilt.

Von den Nichtinklusionen zeigt man zuerst diejenigen, die aus der Fehlerzahl folgen. Dazu betrachtet man die Klasse

$$S_{cofin,a} = \{L : |\overline{L}| \leq a\}.$$

Da alle Mengen dieser Klasse von $\mathbb{N}$ nur um höchstens a Fehler abweichen, genügt es, wenn ein Fin^a-Lerner stets einen Index von $\mathbb{N}$ ausgibt. Umgekehrt sei b eine natürliche Zahl mit $2b < a$ und man nehme an, dass M zumindest die Menge $\mathbb{N}$ mit b Fehlern lernt. Nun hat M eine Locking-Sequenz σ für $\mathbb{N}$, d.h., $M(\sigma\tau)$ ist stets eine Menge, die sich an höchstens b Stellen von $\mathbb{N}$ unterscheidet. Es gibt eine Menge L mit $range(\sigma) \subseteq L \subseteq \mathbb{N}$ derart, dass $\mathbb{N} - L$ genau $2b + 1 \leq a$ Elemente hat. $T \succeq \sigma$ sei ein Text von L. Auf fast allen Eingaben $\tau \preceq T$ gibt M einen Index aus, der sich von $\mathbb{N}$ an nur b Stellen unterscheidet und daher von L an mindestens $b + 1$ Stellen unterscheiden muss. Da M die Menge L nicht lernt, lernt M auch nicht die Klasse S.

Des weiteren ist $S_{cofin,a} \notin \mathrm{TxtEx}^b$ für alle $b < a$. b ist eine natürliche Zahl und $a \geq b + 1$. Wieder betrachtet man eine Locking-Sequenz σ. Diesmal muss M gegen einen festen Index e konvergieren, W_e ist unendlich. Also gibt es eine $b+1$-elementige Menge $D \subseteq W_e - range(\sigma)$. Nun ist $L = \overline{D} \in S_{cofin,a}$, aber M lernt L nicht, da M auf einem Text für L gegen e konvergiert und $L \neq^b W_e$ ist.

Die Nichtinklusion $\mathrm{TxtEx}^a \not\to \mathrm{TxtFin}^b$ wird durch die Klasse $\{\emptyset, \mathbb{N}\}$ belegt. Ein endlicher Lerner muss auf dem Text $\#\#\#\ldots$ irgendwann den Index einer endlichen Menge ausgeben. Da aber auch ein Text für $\mathbb{N}$ mit beliebig vielen Symbolen # beginnen kann, wird diese Menge nicht von allen Texten endlich gelernt. Andererseits kann diese Klasse ohne Probleme im Limes gelernt werden, der Lerner macht beim Auftauchen des ersten, von # verschiedenen Symbols einen Hypothesenwechsel von $\emptyset$ zu $\mathbb{N}$.

Die Nichtinklusion $\mathrm{TxtBC}^a \not\to \mathrm{TxtEx}^b$ ist etwas schwieriger zu beweisen. Man betrachtet hierzu die Klasse S aller Mengen

$$L_x = \{\langle y, z\rangle : y \in \mathbb{N} \wedge z \in K \cup \{x\}\}.$$

OBdA sei $0 \in K$. Dann gilt $L_0 \subseteq L_x$ für alle x. Die Klasse S ist in TxtBC mittels folgendem Lernalgorithmus:

$$W_{M(\sigma)} = \{\langle y, z\rangle : y \in \mathbb{N} \wedge (z \in K \vee (\exists u)\,[\langle u, z\rangle \in range(\sigma)]\,)\,\}.$$

Irgendwann wird das zu L_x gehörige x in einem Paar in $range(\sigma)$ auftauchen und ab da ist der ausgegebene Index jeweils einer von L_x. Also ist $S \in$ TxtBC. Umgekehrt sei N eine Maschine, die zumindest L_0 TxtEx^*-lernt. Dann hat N eine Locking-Sequenz σ für L_0. Da die L_x uniform aufzählbar sind, gibt es eine Familie von berechenbaren Texten T_x für die jeweiligen L_x, welche alle mit σ beginnen. Wenn N eine Menge L_x mit $x \in K$ lernt, dann macht N nach σ auf T_x keinen Hypothesenwechsel mehr. Wenn N eine Menge L_x mit $x \notin K$ lernt, dann macht N nach σ auf T_x einen Hypothesenwechsel, da sich L_0 und L_x an unendlich vielen Stellen unterscheiden. Also gibt es einige L_x die N nicht lernt, da sonst K nach folgender Formel co-aufzählbar wäre:

$$x \notin K \;\Leftrightarrow N \text{ macht auf } T_x \text{ nach } \sigma \text{ einen Hypothesenwechsel.}$$

Somit ist $S \notin \mathrm{TxtEx}^b$. ∎

Übung 3.5 S_{fin} bezeichnet die Klasse aller endlichen Mengen. Unter welchen der hier untersuchten Kriterien können die Klassen S_{fin} und $S_{fin} \cup \{\mathbb{N}\}$ von Text gelernt werden? Zeigen Sie ausserdem: $S_{fin} \cup \{\mathbb{N}\} \in \mathrm{InfEx} - \mathrm{InfFin}^*$.

4 Lernen mit Orakeln

Ein Orakel ist eine Art unendliche Datenbank oder CD-ROM, die nützliche Information zur Verfügung stellt, aber diese Information ist unabhängig von der zu lernenden Funktion. Orakel sind bereits aus der Berechenbarkeitstheorie bekannt und mit dem Begriff der relativen Berechenbarkeit verknüpft:

> Eine Menge B (oder eine Funktion f) ist relativ zu A berechenbar, wenn es eine Maschine M gibt, die jeden Wert $B(x)$ (bzw. $f(x)$) unter Verwendung von Datenbankanfragen der Form "Ist y in A" berechnet.

Man schreibt hierfür auch "$B \leq_T A$" oder "B ist Turing reduzierbar auf A". In der Berechenbarkeitstheorie gibt es eine ganze Menge von Klassifizierungen von Orakeln:

Halteproblem
Die Menge $K = \{x : \varphi_x(x)\downarrow\}$ heisst "diagonales Halteproblem". Sie ist Turing äquivalent zum allgemeinen Halteproblem $\{\langle x, y\rangle : \varphi_x(y)\downarrow\}$, d.h. das diagonale Halteproblem kann relativ dem allgemeinen berechnet werden und umgekehrt. Daher spricht man im allgemeinen nur von dem Halteproblem.

high Orakel
Ein Orakel A heisst high, wenn es eine relativ zu A berechenbare Funktion $f = \varphi_e^A$ gibt, die jede berechenbare Funktion dominiert, d.h.

$$(\forall g \in \mathrm{REC})\,(\exists x)\,(\forall y \geq x)\,[g(y) \leq f(y)].$$

Die high Orakel kann man auch mittels dem relativen Halteproblem ausdrücken: Die Menge $K^A = \{x : \varphi_x^A(x)\downarrow\}$ beinhaltet alle x, für welche die x-te Funktion bei Benutzung des Orakels A und bei Eingabe x definiert ist. K^A heisst das Halteproblem relativ zu A und A ist high genau dann wenn das Halteproblem relativ zu K auf das relativ zu A Turing reduzierbar ist, d.h. wenn $K^K \leq_T K^A$ gilt.

low Orakel
Ein Orakel A heisst low, wenn $K^A \leq_T K$ ist. Low Orakel gibt es daher nur unterhalb von K.

aufzählbare Orakel
Ein Orakel ist aufzählbar, wenn es als Menge aufzählbar ist, d.h. der Definitionsbereich oder der Wertebereich einer partiell-rekursiven Funktion ist. K ist ein aufzählbares Orakel, ebenso jede entscheidbare Menge. Es gibt auch low und high aufzählbare Orakel, die echt zwischen diesen liegen bezüglich der Turing Reduktion.

1-generic Orakel

Ein Orakel A heisst 1-generic, wenn es zu jeder aufzählbaren Menge W von Strings einen String σ der Form $\sigma = A(0)A(1)\ldots A(n)$ gibt so dass entweder $\sigma \in W$ ist oder keine Verlängerung $\tau \succeq \sigma$ in W ist. Jedes 1-generic Orakel unterhalb von K ist low. Andererseits gibt es aber auch 1-generic Orakel, die high sind. Kein 1-generic Orakel ist oberhalb eines nichtberechenbaren aber aufzählbarem Orakel.

PA-vollständige Orakel

Ein Orakel A heisst PA-vollständig wenn es zu jeder partiell-rekursiven $\{0,1\}$-wertigen Funktion ψ eine totale Funktion $f \leq_T A$ gibt mit $f(x) = \psi(x)$ für alle $x \in dom(\psi)$. Alle aufzählbaren PA-vollständigen Mengen sind Turing-äquivalent zu K. Aber es gibt auch low PA-vollständige Orakel.

Diese Typen von Orakeln werden später noch benötigt, um ihren Nutzen für Lernalgorithmen zu untersuchen. Jedes Kriterium kann mit einem Orakel kombiniert werden, so bezeichnet Fin[A] z.B. die Gesamtheit aller Klassen $S \subseteq$ REC, die unter Zuhilfenahme des Orakels A endlich gelernt werden können.

Satz 4.1 *Es gilt* Fin[A] $\rightarrow$ Fin[B] *genau dann wenn* $A \leq_T B$ *ist.*

Beweis Es sei $S \in$ Fin[A] und $A \leq_T B$. Dann kann jede Anfrage an A mit endlich vielen Anfragen an B effektiv beantwortet werden, also ist jeder mit Orakel A ausführbare Lernalgorithmus auch mit Orakel B ausführbar. Umgekehrt sei nun Fin[A] $\rightarrow$ Fin[B]. Man betrachtet folgendes S:

$$S = \{xa^n b^\infty : a, n \in \mathbb{N}, b = A(x)\}.$$

Hierbei bedeutet $f = xa^n b^\infty$, dass $f(0) = x$, $f(1) = f(2) = \ldots = f(n) = a$ und $f(y) = b$ für $y > n$ ist. Man beachte, dass $a = b$ sein kann. S kann mit Orakel A endlich gelernt werden: Man wartet bis die Eingabe die Form $x\tau b$ mit $b = A(x)$ hat. Dann gibt man einen Index für $f = x\tau b^\infty$ aus, da man weiss, dass alle weiteren Funktionswerte b sind. Umgekehrt muss ein solches b auch auftauchen, da jedes $f \in S$ fast überall den Wert $b = A(f(0))$ ausgibt. Nach Voraussetzung lernt eine Maschine $N \leq_T B$ ebenfalls S. Nun gilt:

$$A(x) = a \;\Leftrightarrow\; (\exists n)\,[N(xa^n) \neq ?].$$

Wenn $A(x) = a$ ist, muss N die Funktion xa^∞ erkennen und bei irgendeiner Eingabe xa^n eine Hypothese ausgeben. Wenn $A(x) = b \neq a$ ist, darf N bei Eingabe xa^n noch keine Hypothese ausgeben, da N nicht weiss, ob die zu lernende Funktion gleich $xa^n b^\infty$ oder anders ist, z.B. gleich $xa^{n+1}b^\infty$. Somit kann man durch paralleles Untersuchen des Verhaltens von N auf den Eingaben $x0^\infty$ und $x1^\infty$ den Wert $A(x)$ bestimmen. Also ist $A \leq_T B$. ▮

Bei den Lernkriterien Ex und BC gibt es Orakel, mit deren Hilfe ganz REC gelernt werden kann. Ein solches Orakel heisst allwissend. Andererseits gibt es Orakel, die keine Unterstützung darstellen, da man ohne sie das selbe lernen kann. Solche Orakel heissen trivial. Während beim Fin-Lernen nur berechenbare Orakel trivial sind, ist dies bei Ex und BC nicht mehr der Fall.

Satz 4.2 [1, Adleman und Blum] *Ein Orakel ist genau dann allwissend bezüglich* Ex *wenn es high ist.*

Beweis (nur eine Richtung) A sei ein high Orakel. Dann gibt es eine Funktion $f \leq_T A$ die jede berechenbare Funktion dominiert. Folgender Algorithmus M lernt REC:

> Es sei e das kleinste $j \leq n$ so dass $\varphi_{j,f(n)}(m)\downarrow = \sigma(m)$ für alle $m \in dom(\sigma)$. Nun setzt man
>
> $$M(a_0 a_1 \ldots a_n) = \begin{cases} e & \text{falls ein solches } e \leq n \\ & \text{gefunden wurde;} \\ 0 & \text{sonst.} \end{cases}$$

M ist total, denn M wertet von allen betrachteten Ausdrücke $\varphi_j(m)$ nur die ersten $f(n)$ Rechenschritte aus.

Nun betrachtet man das Verhalten von M auf einer Funktion $g \in$ REC. i sei der kleinste Index von g und $h(n)$ sei die Anzahl der Rechenschritte, um die Werte $\varphi_i(0)$, $\varphi_i(1)$, …, $\varphi_i(n)$ zu berechnen. f dominiert h, daher wird für fast alle n bei Eingabe $g(0)g(1)\ldots g(n)$ ein Index $e \leq i$ gefunden. Des weiteren unterscheidet sich g von jeder der Funktionen φ_0, φ_1, …, φ_{i-1}, folglich kann das Zwischenergebnis e nur bei endlich vielen n einen Wert kleiner i annehmen. Also ist $M(g(0)g(1)\ldots g(n)) = i$ für fast alle n und M lernt alle berechenbaren Funktionen unter dem Kriterium Ex. ▮

Satz 4.3 [49, Slaman and Solovay] *Ein Orakel A ist genau dann trivial, wenn es ein 1-generic Orakel B mit $A \leq_T B \leq_T K$ gibt.*

Es wurde gezeigt, dass PEx äquivalent ist dazu, dass eine totale Maschine jedes $f \in S$ fast überall vorhersagt. Letzteres gilt nicht relativ zu Orakeln. Man kann einerseits zeigen, dass PEx$[A]$ = PEx für alle $A \leq_T K$ gilt, und andererseits, dass mit einem high Orakel jede berechenbare Funktion fast überall vorhergesagt werden kann.

Satz 4.4 *Zu einem Orakel A gibt es genau dann eine totale Maschine $M \leq_T A$ die alle berechenbaren Funktionen fast überall vorhersagt wenn A high ist.*

Beweis Die eine Richtung ist ähnlich zu der des Satzes von Adleman und Blum: Es eine Funktion $f \leq_T A$ eine Funktion, die jede berechenbare Funktion dominiert. M ist nun wie folgt definiert:

> Es sei e das kleinste $j \leq n$ so dass $\varphi_{j,f(n)}(m)\downarrow = \sigma(m)$ für alle $m \in dom(\sigma)$ und $\varphi_{j,f(n)}(n+1)\downarrow$. Nun setzt man
>
> $$M(a_0a_1\dots a_n) = \begin{cases} \varphi_e(n+1) & \text{falls ein solches } e \leq n \\ & \text{gefunden wurde;} \\ 0 & \text{sonst.} \end{cases}$$

M ist total, denn M wertet von allen betrachteten Ausdrücke $\varphi_j(m)$ nur die ersten $f(n)$ Rechenschritte aus.

Nun betrachtet man das Verhalten von M auf einer Funktion $g \in$ REC. i sei der kleinste Index von g und $h(n)$ sei die Anzahl der Rechenschritte, um die Werte $\varphi_i(0)$, $\varphi_i(1)$, …, $\varphi_i(n+1)$ zu berechnen. f dominiert h, daher wird für fast alle n bei Eingabe $g(0)g(1)\dots g(n)$ ein Zwischenergebnis $e \leq i$ gefunden. Des weiteren unterscheidet sich g von jeder der Funktionen φ_0, φ_1, …, φ_{i-1}, folglich kann das Zwischenergebnis e nur bei endlich vielen n einen Wert kleiner i annehmen. Also ist $e = i$ und $M(g(0)g(1)\dots g(n)) = g(n+1)$ für fast alle n. Somit sagt M jede berechenbare Funktion fast überall vorher.

Umgekehrt sei nun $M \leq_T A$ für irgendein Orakel A. Nun kann man folgende Familie totaler Funktionen $F_\sigma \leq_T A$ definieren:

$$F_\sigma(n) = \begin{cases} \sigma(n) & \text{falls } n \in dom(\sigma); \\ M(F_\sigma(0)F_\sigma(1)\dots F_\sigma(n-1)) & \text{sonst.} \end{cases}$$

Die Funktionen sind total, da M total ist, aber sie benötigen ein A-Orakel und sind daher nicht immer rekursiv. Wohl aber ist jede rekursive Funktion gleich einem F_σ da M jede rekursive Funktion fast überall vorhersagt. Es sei $\sigma_0, \sigma_1, \dots$ eine Aufzählung aller Strings. Nun setzt man

$$f(n) = F_{\sigma_0}(n) + F_{\sigma_1}(n) + \dots + F_{\sigma_n}(n)$$

und erhält so eine A-rekursive Funktion, die jede rekursive Funktion dominiert. Also ist A high. ∎

Zur Struktur der Ex-Grade liegen auch einige Ergebnisse vor, die hier ohne Beweis referiert werden.

Satz 4.5 [40] *Wenn A und B aufzählbare Orakel sind und B nicht high ist, dann gilt* $\text{Ex}[A] \to \text{Ex}[B] \;\Leftrightarrow\; \text{BC}[A] \to \text{BC}[B] \;\Leftrightarrow\; A \leq_T B$.
Wenn $K^A \not\leq_T K^B$ und B nicht high ist, dann gilt $\text{Ex}[A] \not\to \text{Ex}[B]$.

Diese Ähnlichkeit zwischen Ex und BC auf den aufzählbaren Orakeln besteht nicht im allgemeinen. In der Tat gilt, dass es bereits ein low Orakel gibt, das für BC allwissend ist: Jockusch und Soare zeigten mit ihrem Low Basis Theorem [44, Theorem V.5.32], dass es ein PA-vollständiges low Orakel gibt, und Satz 4.6 zeigt, dass jedes PA-vollständige Orakel allwissend für BC ist.

Satz 4.6 *Jedes PA-vollständige Orakel ist allwissend für* BC.
Ein Orakel A ist genau dann trivial bezüglich BC *wenn $A \leq_T B \leq_T K$ für ein 1-generic Orakel B gilt.*

Folgende Tabelle fast die Bedingungen zusammen, unter welchen A trivial oder allwissend ist. Dabei sind in zwei Fällen nur hinreichende Bedingungen angegeben. A ist z.B. allwissend für BC wenn A high oder wenn A PA-vollständig ist. Vielleicht gibt es noch einige weitere Fälle von BC-allwissenden Orakeln.

Kriterium	trivial	allwissend
Fin	A ist entscheidbar	nie
PEx	$A \leq_T K$	$A \oplus K \geq_T K^K$
Ex	$(\exists$ 1-generic $B)\,[A \leq_T B \leq_T K]$	A ist high
Vorhersage	$\mathrm{Ex}[A] = \mathrm{Ex}$, A ist 2-generic, ...	A ist high
BC	$(\exists$ 1-generic $B)\,[A \leq_T B \leq_T K]$	A ist high, PA-vollständig, ...

Fortnow et al. [21] geben eine ausführliche Auflistung der trivialen und allwissenden Grade unter vielen verschiedenen Kriterien.

Übung 4.7 Mit jedem PA-vollständigen Orakel können alle berechenbaren $\{0, 1\}$-wertigen Funktionen fast überall vorhergesagt werden.

5 Lernen mittels Anfragen an einen Lehrer

Orakel beantworten Fragen unabhängig von der zu lernenden Funktion, z.B. ob die Funktionen φ_i und φ_j verschieden sind oder ob $x \in K$ ist. Diese Informationen können nützlich sein und ggf. erlauben, mehr zu lernen. Lehrer dagegen beantworten Fragen über die zu lernende Funktion [26, 27, 28]. Dazu werden dann Anfragesprachen eingeführt und der Lehrer beantwortet in dieser Sprache über die zu lernende Funktion f gestellten Fragen mit “Ja” oder “Nein”. Im Folgenden werden hauptsächlich vier Anfragesprachen untersucht:

Anfragen mit Nachfolgersymbol $L[\mathrm{Succ}]$
: Diese Anfragesprache erlaubt z.B. festzustellen, ob f zwei hintereinanderfolgende Nullstellen hat: $(\exists x)\,[f(x) = 0 \wedge f(\mathrm{Succ}(x)) = 0]$? Dabei ist $\mathrm{Succ}(x) = x + 1$.

Anfragen mit Kleinersymbol $L[<]$
: Diese Anfragesprache erlaubt z.B. festzustellen, ob f unendlich viele Werte annimmt: $(\forall x)\,(\exists y)\,[x < f(y)]$?

Anfragen mit Addition $L[+]$
: Diese Anfragesprache erlaubt z.B. festzustellen, ob alle Funktionswerte von f gerade sind: $(\forall x)\,(\exists y)\,[f(x) = y + y]$?

Anfragen mit Addition und Multiplikation $L[+, *]$
: Diese Anfragesprache erlaubt z.B. festzustellen, ob f durch eine diophantische Menge definiert ist, d.h. ob $f(x) = y$ genau dann wenn es $z_1, \ldots, z_k$ gibt mit $p(x, y, z_1, \ldots, z_k) = 0$ für ein Polynom p. Die Anfrage ist:

$$
\begin{aligned}
(\forall x)\,(\forall y)\,[&(f(x) = y \wedge (\exists z_1, \ldots, z_k)\,[p(x, y, z_1, \ldots, z_k) = 0]) \vee \\
&(f(x) \neq y \wedge (\forall z_1, \ldots, z_k)\,[p(x, y, z_1, \ldots, z_k) \neq 0])]
\end{aligned}
$$

Matjiasevich [44, Seite 135] hat einen Algorithmus angegeben, mit dem zu jeder berechenbaren Funktion φ_e ein Polynom p_e konstruiert werden kann, so dass $f = \varphi_e$ ist genau dann wenn f obige Formel mit p_e erfüllt. Damit kann in der Anfragesprache $L[+, *]$ geprüft werden, ob $f = \varphi_e$ für ein gegebenes e ist.

Wie aus den Beispielen ersichtlich, dürfen die Sprachen $L[\mathrm{Succ}]$, $L[<]$ und $L[+]$ neben dem angegebenen Symbol noch Quantoren über die natürlichen Zahlen, Zahlkonstanten, das Gleichheitssymbol, Klammern, das Symbol f für die zu lernende Funktion und Boole’sche Verknüpfungen für die elementaren Terme enthalten.

Definition 5.1 M lernt eine Funktion f in endlicher Zeit mittels L-Fragen, wenn M Anfragen an den Lehrer in der Sprache L stellt und dann nach endlicher Zeit einen Index für f ausgibt. Entsprechend kann Lernen im Limes und verhaltenskorrektes Lernen definiert werden. Eine Klasse S wird gelernt, wenn eine Maschine M jede Funktion $f \in S$ lernt. Hier eine Tabelle über die üblichen Lernbegriffe:

	endliches Lernen	Lernen im Limes
L[Succ]	QFin[Succ]	QEx[Succ]
$L[<]$	QFin[<]	QEx[<]
$L[+]$	QFin[+]	QEx[+]
$L[+,*]$	QFin[+, *]	QEx[+, *]

Beispiel 5.2 REC $\in$ QFin[+, *]. Das Lernverfahren erfragt, ob $f = \varphi_0$, ob $f = \varphi_1$, ... bis ein e mit $f = \varphi_e$ gefunden wurde. Daraus folgt direkt QEx[+, *] = QFin[+, *].

Eine weiter wichtige Eigenschaft ist: $L[+] = L[+,<]$ und $L[<] = L[\text{Succ},<]$. Dies folgt daraus, dass das zweite Symbol jeweils durch das erste ausdrückbar ist. Z.B. ist $x < y$ genau dann wenn $(\exists z)\,[y = x + z + 1]$. Ebenso ist $x = \text{Succ}(y)$ genau dann wenn $(x < y) \wedge (\forall z)\,[z < x \vee z = x \vee z = y \vee y < z]$. Hieraus folgt QFin[Succ] $\subseteq$ QFin[<] $\subseteq$ QFin[+] und QEx[Succ] $\subseteq$ QEx[<] $\subseteq$ QEx[+].

Satz 5.3 $S_1 \in$ QFin[Succ].
QFin[Succ] $\not\subseteq$ Fin[A] *für alle Orakel A.*

Beweis Der Algorithmus fragt für jedes n, ob $(\forall x > n)\,[f(x) = 0]$ zutrifft. Diese Frage kann in der Form

$$(\forall x)\,[x = 0 \vee x = 1 \vee \ldots \vee x = n \vee f(x) = 0]$$

gestellt werden, so dass das Symbol $>$ nicht mehr auftaucht. Irgendwann wird diese Frage bejaht. Dann werden $f(0)$ bis $f(n)$ erfragt (oder von der Eingabe eingelesen) und ein Index der Funktion $f(0)f(1)\ldots f(n)0^\infty$ ausgegeben, die mit f übereinstimmt.

Die zweite Behauptung ergibt sich daraus, dass S_1 nicht relativ zu irgendeinem irgendeinem Orakel endlich gelernt werden kann. ∎

Da man jedoch eine Folge von Anfragen stellen muss, man also nicht mit einer einzigen Anfrage auskommt, gilt:

Satz 5.4 $S_0 \cup S_1 \notin$ QFin[Succ].
$S_0 \cup S_1 \in$ QFin[<].

Das zweite Statement zeigt dann QFin[Succ] $\subset$ QFin[<]. Der Lernalgorithmus kann mit Anfragen in der Sprache $L[<]$ die beiden Klassen trennen, denn die Anfrage $(\exists x)\,(\forall y > x)\,[f(y) = 0]$? wird genau für die Funktionen in S_0 bejaht. Danach gibt er je nach Fall $f(0)$ aus oder simuliert obigen Algorithmus um Funktionen in S_1 mittels Anfragen endlich zu lernen.

Also trennt $S_0 \cup S_1$ das Inferenzkriterium QFin[<] von QFin[Succ]. Ebenso kann man die anderen beiden trennen:

Satz 5.5 $S_2 = \{f : f = \varphi_{\max(\{f(y):y \text{ ist gerade}\})}\} \in$ QFin[+] $-$ QFin[<]. $S_0 \cup S_2 \in$ QFin[+, $*$] $-$ QFin[+].

Satz 5.6 *Die Klasse S_{pol} aller Polynome ist in* QFin[+].

Beweis Der Algorithmus basiert auf der Tatsache, dass abgefragt werden kann, ob der Grad eines Polynoms $< n$ für ein gegebenes n ist. Dazu betrachtet man die Operatoren $D^0(f,x) = f$ und $D^{n+1}(f,x) = D^n(f,x+1) - D^n(f,x)$. Für $f = ax^3 + bx$ nehmen diese Operatoren folgende Werte an:

$$
\begin{aligned}
D^0(f,x) &= f(x) \\
&= ax^3 + bx \\
D^1(f,x) &= f(x+1) - f(x) \\
&= 3ax^2 + 3ax + a + b \\
D^2(f,x) &= f(x+2) - 2f(x+1) + f(x) \\
&= 6ax + 6a \\
D^3(f,x) &= f(x+3) - 3f(x+2) + 3f(x+1) - f(x) \\
&= 6a \\
D^4(f,x) &= f(x+4) - 4f(x+3) + 6f(x+2) - 4f(x+1) + f(x) \\
&= 0
\end{aligned}
$$

Also gilt: Ist der Grad von f kleiner als n, so ist $D^n(f) = 0$. Diese Gleichung kann durch Umlegen der negativen Terme auf die andere Seite in eine Anfrage in $L[+]$ umgewandelt werden. Z.B. gilt:

$$(\forall x)\,[D^3(f,x) = 0] \quad \Leftrightarrow \quad (\forall x)\,[f(x+3) + 3f(x+1) = 3f(x+2) + f(x)].$$

Auf diesem Weg kann durch Anfragen an den Lehrer der Grad n des Polynoms bestimmt werden. Sobald man diesen kennt (oder eine obere Schranke dafür hat), kann durch Interpolation der Funktionswerte $f(0), f(1), \ldots, f(n)$ das Polynom bestimmt werden. ∎

Als nächstes werden die entsprechenden Ergebnisse für das QEx-Lernen vorgestellt.

Satz 5.7 *$S = \{f \in \text{REC} : range(f) = K\} \in \text{QEx[Succ]} - \text{Ex}[A]$ für alle Orakel A, die nicht high, also nicht allwissend sind.*

Beweis Es sei M die Maschine, die alle berechenbaren Funktionen durch Befragen eines K-Orakel lernt. Man kann nun jede Anfrage "Ist $x \in K$?" durch die Anfrage "$(\exists y)\,[f(y) = x]$?" ersetzen. Für alle $f \in S$ sind die Antworten des Lehrers identisch mit dem des Orakels K und daher liefert der Algorithmus für diese Antworten das gleiche Ergebnis.

Umgekehrt nehme man nun an, dass $S \in \text{Ex}[A]$ für ein Orakel A. Nun betrachte man die folgende Operation Γ auf den berechenbaren Funktionen:

$$\begin{aligned}\Gamma(f)(2n) &= a_n;\\ \Gamma(f)(2n+1) &= a_{f(n)};\end{aligned}$$

wobei $a_0, a_1, \ldots$ eine bijektive Aufzählung von K ist. $\Gamma(\text{REC}) \subseteq S$ und daher ist $\Gamma(\text{REC}) \in \text{Ex}[A]$ mittels der selben Maschine M. Man konstruiert nun eine Maschine N, die zeigt, dass $\text{REC} \in \text{Ex}[A]$ ist:

$N(b_0 b_1 \ldots b_n)$ berechnet $e_n = M(a_0 a_{b_0} a_1 a_{b_1} \ldots a_n a_{b_n})$ und gibt $s(e_n)$ aus wobei $\varphi_{s(e)}(x) = \min\{y : \varphi_e(2x+1) = a_y\}$.

Es ist leicht zu sehen, dass $\varphi_{s(e)} = f$ aus $\varphi_e = \Gamma(f)$ folgt. Da $\Gamma(f) \in S$ ist für alle f und die Folge $a_0 a_{b_0} a_1 a_{b_1} \ldots a_n a_{b_n}$ gegen $\Gamma(f)$ konvergiert für $b_0 b_1 \ldots b_n \to f$, konvergiert die Folge $e_0, e_1, \ldots$ gegen einen Index für $\Gamma(f)$ und die Folge $s(e_0), s(e_1), \ldots$ gegen einen für f. Also lernt M ganz REC und das Orakel A ist allwissend. ∎

Man kann ähnliche Übersetzungen [27, 52] definieren mit folgenden Eigenschaften:

$$\begin{aligned} S \in \text{Ex} &\Leftrightarrow \Gamma_1(S) \in \text{Ex}\\ \Gamma_1(S) \in \text{QEx}[<] &\Leftrightarrow \Gamma_1(S) \in \text{Ex}\\ \Gamma_1(S) \in \text{QEx}[+] & \end{aligned}$$

sowie

$$\begin{aligned} S \in \text{Ex} &\Leftrightarrow \Gamma_2(S) \in \text{Ex}\\ \Gamma_2(S) \in \text{QEx}[+] &\Leftrightarrow \Gamma_2(S) \in \text{Ex} \end{aligned}$$

Wenn man $\text{REC}_{0,1}$ einsetzt, erhält man im ersten Fall $\Gamma_1(\text{REC}_{0,1}) \in \text{QEx}[+] - \text{QEx}[<]$ und im zweiten Fall $\Gamma_2(\text{REC}_{0,1}) \notin \text{QEx}[+]$. Insgesamt lässt sich zeigen:

Satz 5.8 $\text{Ex} \to \text{QEx[Succ]} \to \text{QEx}[<] \to \text{QEx}[+]$.

Übung 5.9 Eine Funktion heisst periodisch, wenn es ein n gibt, so dass für alle x die Gleichung $f(x+n) = f(x)$ gilt. Geben Sie einen QFin[Succ]-Algorithmus an, um alle periodischen Funktionen zu lernen.

6 Team-Lernen

Ein $[m,n]$-Team ist eine Gruppe von n Maschinen, die parallel versuchen, die selbe Funktion oder Menge zu lernen. Das Team ist erfolgreich, wenn mindestens m der Maschinen unter dem gegebenen Kriterium die gesuchte Funktion oder Menge lernen. Smith [51] gibt eine Einführung in diese Theorie.

Beispiel 6.1 Die Klasse $\{f : (\forall x)\,[f(x) \leq f(x+1) \leq 2]\}$ ist $[1,3]$Fin-lernbar.

Hierzu betrachtet man ein Team aus drei Maschinen. Die erste Maschine gibt sofort einen Index für die Funktion 0^∞ aus. Wenn nach irgendeinem Zeitpunkt eine 1 auftaucht, d.h. wenn die Eingabe die Form 0^n1 hat, gibt die zweite Maschine einen Index für 0^n1^∞ aus. Wenn irgendwann eine 2 auftaucht, d.h. wenn die Eingabe die Form 0^n1^m2 hat, gibt die dritte Maschine einen Index für die Funktion $0^n1^m2^\infty$ aus. Da alle Funktionen in der zu lernenden Klasse gleich 0^∞, 0^n1^∞ oder $0^n1^m2^\infty$ für irgendwelche n und m sind, lernt das $[1,3]$-Team diese Klasse von Funktionen.

Diese Klasse ist jedoch nicht endlich lernbar: Da 0^∞ gelernt wird, muss ein Lerner bei irgendeiner Eingabe 0^n eine Hypothese 0^∞ ausgeben und erkennt dann nicht mehr die Funktion 0^n1^∞. Auf diesem Weg kann man sogar zeigen, dass selbst ein $[1,2]$-Team die Klasse nicht lernt, da eine zweite Maschine dann bei irgendeiner Eingabe 0^n1^m einen Index für 0^n1^∞ ausgibt und die Funktion $0^n1^m2^\infty$ von beiden Maschinen nicht erkannt wird.

Satz 6.2 $[m,n]\mathrm{Ex} = \mathrm{Ex}$ *genau dann wenn* $\frac{m}{n} > \frac{1}{2}$.

Beweis Wenn $\frac{m}{n} \leq \frac{1}{2}$ ist, so ist $S_0 \cup S_1 \in [m,n]\mathrm{Ex} - \mathrm{Ex}$: Nach Beispiel 1.1 sind $S_0 = \{f : f = \varphi_{f(0)}\}$ und $S_1 = \{f : (\forall^\infty x)\,[f(x) = 0]\}$ beide Ex-lernbar: Wenn $\frac{m}{n} \leq \frac{1}{2}$ ist, dann lernen m Maschinen die Funktionen aus S_0 und m weitere Maschinen die aus S_1. Somit sind für jedes $f \in S_0 \cup S_1$ mindestens m Maschinen erfolgreich. Nach Satz 1.2 ist die Vereinigung $S_0 \cup S_1$ jedoch nicht Ex-lernbar.

Umgekehrt sei nun $M_1,\ldots,M_n$ ein $[m,n]$-Team für eine Klasse S und $\frac{m}{n} > \frac{1}{2}$. OBdA geht man davon aus, dass jede Maschine M_i des Teams eine nachweislich falsche Hypothese e ($\varphi_e(x)\!\downarrow \neq f(x)$ für ein x) irgendwann auswechselt: Bevor $M_i(\sigma a)$ ausgegeben wird, simuliert M die Funktion $\varphi_{M_i(\sigma)}$ auf allen $x \leq |\sigma|$ bis zu $|\sigma|$ Schritten und falls M_i einen Fehler findet, wird die Hypothese – sofern nicht sowieso bereits ein Hypothesenwechsel anliegt – durch eine neue, bei x korrigierte, ersetzt.

Man ordnet die Maschinen bei jedem σ nach dem Alter ihrer Hypothesen, d.h. die Hypothese $M_1(\sigma)$ ist älter als die von $M_2(\sigma)$ usw.; daher gilt, dass die Maschinen $M_1,\ldots,M_m$ gegen Werte $j_1,\ldots,j_m$ konvergieren, da sie die jeweils m ältesten

Hypothesen ausgeben. Nun definiert man

$$\begin{array}{rcl} M(\sigma) & = & s(M_1(\sigma), \ldots, M_m(\sigma)) \quad \text{wobei} \\ \varphi_{s(e_1,\ldots,e_m)} & = & \begin{cases} \varphi_{e_i}(x) & \text{für das erste } i \text{ wo dieser Ausdruck konvergiert;} \\ \uparrow & \text{sonst, d.h., } \varphi_{e_1}(x)\uparrow, \ldots, \varphi_{e_m}(x)\uparrow. \end{cases} \end{array}$$

Für fast alle $\sigma \preceq f$ gilt $M_1(\sigma) = j_1$, …, $M_m(\sigma) = j_m$. Daher ist $M(\sigma)$ fast immer gleich $s(j_1, \ldots, j_m)$ und M konvergiert syntaktisch. Da $n - m < m$ und nur $n - m$ der Maschinen nicht gegen einen Index für f konvergieren brauchen, konvergiert mindestens eine der Maschinen $M_1, \ldots, M_m$ gegen einen Index für f und $s(j_1, \ldots, j_m)$ ist der Index einer totalen Funktion. Da des weiteren jede von f nachweisbar verschiedene Hypothese ausgewechselt wird, ist keine der Hypothesen $j_1, \ldots, j_m$ nachweisbar von f verschieden und $s(j_1, \ldots, j_m)$ ist ein Index für f. ∎

Smith [50] zeigte noch eine weitergehende Beziehung:

$$[m, n]\text{Ex} = [1, k]\text{Ex} \quad \Leftrightarrow \quad \tfrac{1}{k+1} < \tfrac{m}{n} \leq \tfrac{1}{k}.$$

So führte er alle $[m, n]$-Teams auf $[1, k]$-Teams zurück. Diese Klassen sind übrigens alle verschieden, d.h. die Hierarchie $[1, 1]\text{Ex} \subset [1, 2]\text{Ex} \subset [1, 3]\text{Ex} \subset \ldots$ hat unendlich viele Ebenen. Die Ergebnisse für das Team-Lernen unter dem Kriterium Ex und unter dem Kriterium BC sind gleich.

Anders verhält es sich beim endlichen Lernen: Hier sind die Verhältnisse wesentlich komplizierter und bis heute noch nicht vollständig aufgeklärt. Zuerst werden einige einfache, aber dennoch typische Fälle vorgestellt.

Satz 6.3 $[2, 3]\text{Fin} \neq \text{Fin}$.

Beweis Zu jedem $x \in K$ bezeichne $a(x)$ den Zeitpunkt, zu dem x nach K aufgezählt wird. Die Klasse S habe folgende Funktionen:

$x0^\infty$ für alle x;
$x0^{a(x)}1^\infty$ für alle $x \in K$.

Angenommen, $S \in \text{Fin}$ mittels einer Maschine M. $b(x)$ sei das erste n so dass $M(x0^n) \neq ?$ ist. Es folgt für alle $x \in K$, dass $b(x) > a(x)$ sein muss, da M bei Eingabe $x0^{b(x)}$ wissen muss, ob die Eingabe zu $x0^\infty$ oder zu $x0^{a(x)}1^\infty$ gehört. Somit ist $x \in K$ genau dann wenn x innerhalb von $b(x)$ Schritten nach K aufgezählt wird. Damit können die Funktion b und auch die Maschine M nicht berechenbar sein: $S \notin \text{Fin}$ (aber es gilt $S \in \text{Fin}[K]$).

Umgekehrt bleibt zu zeigen, dass $S \in [2, 3]\text{Fin}$ ist. a ist zwar keine totale Funktion, aber die Funktionenfamilie

$$f_{x,c}(y) = \begin{cases} x & \text{falls } y = 0; \\ 0 & \text{falls } x \notin K \text{ oder } y \leq a(x); \\ c & \text{falls } x \in K \text{ und } y > a(x). \end{cases}$$

ist uniform berechenbar, d.h. $\{f_{x,0}, f_{x,1} : x \in \mathbb{N}\}$ ist eine berechenbare Numerierung der Klasse S. Man beachte, dass $f_{x,0} = f_{x,1}$ für alle $x \notin K$. Folgendes $[2,3]$-Team lernt dann jede Funktion in S:

M_1: Ausgabe von $f_{x,0}$ bei Eingabe $\sigma \succeq x$;
M_2: Ausgabe von $f_{x,1}$ bei Eingabe $\sigma \succeq x$;
M_3: Ausgabe von $f_{x,c}$ bei Eingabe $\sigma \succeq x0^{a(x)}c$.

Falls $x \notin K$ erkennen die beiden ersten Maschinen die Funktion $x0^\infty$ während die dritte nie eine Hypothese ausgibt. Falls $x \in K$ ist die Funktion von der Form $x0^{a(x)}c^\infty$ für ein $c \in \{0,1\}$. Eine der beiden ersten Maschinen gibt eine richtige Hypothese aus, ebenso ist M_3 richtig nachdem der Präfix $x0^{a(x)}c$ eingelesen ist. Also erfüllen stets zwei von drei Maschinen die Spezifikation. ∎

Satz 6.4 $[m,n]\mathrm{Fin} = \mathrm{Fin}$ *genau dann wenn* $\frac{m}{n} > \frac{2}{3}$.

Beweis Wenn $\frac{m}{n} \leq \frac{2}{3}$, dann kann man obiges $S \in [2,3]\mathrm{Fin} - \mathrm{Fin}$ auch als Beleg für $[m,n]\mathrm{Fin} \not\to \mathrm{Fin}$ verwenden. Nun arbeiten die Maschinen Nummer $1,4,7,\ldots$ wie M_1, die Maschinen Nummer $2,5,8,\ldots$ wie M_2 und die Maschinen Nummer $3,6,9,\ldots$ wie M_3. Höchstens $\lceil \frac{n}{3} \rceil$ Maschinen machen einen Fehler und da $m \leq \lfloor \frac{2}{3} \cdot n \rfloor$ ist, lernen jeweils m Maschinen jedes $f \in S$. Also ist $S \in [m,n]\mathrm{Fin}$; $S \notin \mathrm{Fin}$ wurde bereits gezeigt.

Umgekehrt sei nun $\frac{m}{n} > \frac{2}{3}$ und $f \in S$ für ein $S \in [m,n]\mathrm{Fin}$. Ein Fin-Lerner für S wartet bei einer Eingabe $f \in S$ so lange bis mindestens m der Maschinen aus dem (m,n)-Team ihre Hypothese ausgegeben haben; sei E die Menge dieser Hypothesen. Danach wird folgende Funktion ausgegeben:

> $\varphi_{s(E)}(x)\downarrow = y$ genau dann wenn $\varphi_e(x)\downarrow = y$ für alle bis auf $n-m$ Hypothesen $e \in E$.

Da nur $n-m$ Hypothesen inkorrekt sein können und alle anderen Funktionen mit f übereinstimmen, ist $\varphi_{s(E)}$ total. Ferner ist $m > 2(n-m)$ (wegen $\frac{m}{n} > \frac{2}{3}$) und wenn alle bis auf $n-m$ Funktionen φ_e bei Eingabe x den Wert y ausgeben, so ist unter diesen Funktionen mindestens eine korrekte und daher ist $\varphi_{s(E)} = f$. ∎

Satz 6.5 [54, Velauthapillai] $[2,4]\mathrm{Fin} \not\subseteq [1,2]\mathrm{Fin}$.

Beweis Diese Nichtinklusion wird von einer Klasse bewiesen, die derjenigen von der Nichtinklusion $[2,3]\text{Fin} \not\subseteq \text{Fin}$ ähnelt. Es bezeichne wieder $a(x)$ die Anzahl der Schritte, nach der ein Wert x nach K aufgezählt wird. Nun sei S die Familie der folgenden Funktionen (mit $x, y \in \mathbb{N}$ und $b, c \in \{0,1\}$):

$$f_{x,y,b,c} = \begin{cases} xy0^\infty & \text{falls } x \notin K; \\ xy0^{a(x)}b^\infty & \text{falls } x \in K \text{ und } y \notin K; \\ xy0^{a(x)}b^{a(y)}c^\infty & \text{sonst } (x, y \in K). \end{cases}$$

Der $[2,4]$Fin-Inferenzalgorithmus läuft wie folgt:

M_1: Ausgabe von $f_{x,y,0,1}$ bei Eingabe $\sigma \succeq xy$;
M_2: Ausgabe von $f_{x,y,1,0}$ bei Eingabe $\sigma \succeq xy$;
M_3: Ausgabe von $f_{x,y,b,b}$ bei Eingabe $\sigma \succeq xy0^{a(x)}b$;
M_4: Ausgabe von $f_{x,y,b,c}$ bei Eingabe $\sigma \succeq xy0^{a(x)}b^{a(y)}c$.

Wenn $x \notin K$, erkennen M_1 und M_2 die Funktion $f_{x,y,b,c}$ die dann von b und c unabhängig ist. Wenn $x \in K$ und $y \notin K$ erkennen im Fall $b = 0$ die Maschinen M_1 und M_3 sowie im Fall $b = 1$ die Maschinen M_2 und M_3 die gesuchte Funktion $f_{x,y,b,c}$ die von c unabhängig ist. Wenn $x, y \in K$ erkennen eine der Maschinen M_1, M_2, M_3 sowie M_4 die Funktion $f_{x,y,b,c}$. Die andere Richtung $S \notin [1,2]\text{Fin}$ ist etwas kompliziert und wird weggelassen. Sie kann mit Hilfe von Beigel's Nonspeedup Theorem gezeigt werden [9]. ▮

Im Folgenden werden noch die verschiedenen Ergebnisse über die Inklusionsstruktur des Team-Lernens zusammengefasst.

PEx
$[m,n]\text{PEx} = \text{PEx}$ für alle m, n, da PEx gegen Vereinigung abgeschlossen ist: $S \cup S' \in \text{PEx} \Leftrightarrow S \in \text{PEx} \wedge S' \in \text{PEx}$.

Ex mit Fehlern [47]
$[m,n]\text{Ex}^a \subseteq [h,k]\text{Ex}^b$ genau dann wenn $\lfloor \frac{n}{m} \rfloor \cdot \lfloor \frac{a+b+1}{b+1} \rfloor \leq \lfloor \frac{k}{h} \rfloor$.

Fin-Teams mit $\frac{m}{n} > \frac{1}{2}$ [18, 22, 23]
$[2,3]\text{Fin} \to [3,5]\text{Fin} \to [4,7]\text{Fin} \to \dots$
$[m,n]\text{Fin} = [k+1, 2k+1]\text{Fin} \quad \Leftrightarrow \quad \frac{k+2}{2k+3} < \frac{m}{n} \leq \frac{k+1}{2k+1}$.

Fin-Teams mit $\frac{m}{n} = \frac{1}{2}$ [34, 54]
$[1,2]\text{Fin} = [3,6]\text{Fin} = [5,10]\text{Fin} = \dots$,
$[2,4]\text{Fin} = [4,8]\text{Fin} = [6,12]\text{Fin} = \dots$ und
$[1,2]\text{Fin} \subset [2,4]\text{Fin}$.

Fin-Teams mit $\frac{m}{n} < \frac{1}{2}$ [16]

Wenn $p_{n+1} < \frac{m}{n} \leq \frac{h}{k} < p_n$ dann $[m,n]\text{Fin} = [h,k]\text{Fin}$
wobei $p_0 = \frac{1}{2}$, $p_1 = \frac{24}{49}$, $p_2 = \frac{20}{41}$, $p_3 = \frac{18}{37}$, $p_4 = \frac{17}{35}$ und $n < 4$.

Pfin-Teams

Bis zur Grenze $\frac{1}{2}$ ist die Inklusionsstruktur von Fin und PFin gleich. Alle in obigen Sätzen angegebenen Beispiele für Nichtinklusionen zeigen stets auch $[m,n]\text{Pfin} \not\subseteq [h,k]\text{Fin}$. Unterhalb der Schranke $\frac{1}{2}$ stimmen die Inklusionsstrukturen nicht mehr überein. So ist z.B. $[24,49]\text{Pfin} = [2,4]\text{PFin}$.

Kalyanasundaram und Kummer [39] fanden eine Formel, um das Inklusionsproblem für PFin-Teams mit $\frac{m}{n} \leq \frac{1}{2}$ und $\frac{h}{k} < \frac{1}{2}$ induktiv zu berechnen, in den anderen Fällen ist es ja schon bekannt.

$[m,n]\text{PFin} \not\subseteq [h,k]\text{PFin}$ genau dann wenn $[m,n-m]\text{PFin} \not\subseteq [h,k-h]\text{PFin}$ oder es $g > 1$ und $i_1,\ldots,i_g$ mit $1 \leq i_1 \leq i_2 \leq \ldots \leq i_g$ und $i_1+i_2+\ldots+i_g = m$ gibt mit $(\forall j_1,\ldots,j_g)\,[j_1 \leq j_2 \leq \ldots \leq j_g \wedge j_1+\ldots+j_g = m \;\Rightarrow (\exists l)\,[[m,n+i_l-m]\text{Pfin} \not\subseteq [h,k+j_l-h]\text{Pfin}]]$.

PFin-Breakpoints

Ein rationaler Punkt p heisst Breakpoint, wenn $[m,n]\text{PFin} \not\subseteq [h,k]\text{PFin}$ für alle m,n,h,k mit $\frac{m}{n} \leq p < \frac{h}{k}$. Ambainis zeigte [2]:
Zu jedem Breakpoint p gibt es m,n und einen Breakpoint $p' < p$ mit $\frac{m}{n} = p$ und $[m,n]\text{PFin} \subseteq [h,k]\text{PFin}$ für alle h,k mit $\frac{h}{k} > p'$.
Die Menge P aller Breakpoints ist umgekehrt wohlgeordnet (jede Teilmenge $Q \subseteq P$ hat ein Maximum).

Teams und Orakel

In Bezug auf Teams interessiert vor allem die Frage, welche Orakel nötig sind, um bestimmte Nichtinklusionen zu überwinden. Dabei gilt:

Ex: $[1,n+1]\text{Ex} \subseteq [1,n]\text{Ex}[A]$ nur wenn A high, d.h. allwissend ist. Wenn A nicht high ist, dann gilt $\{f : f =^n \varphi_{f(0)}\} \in [1,n+1]\text{Ex} - [1,n]\text{Ex}[A]$.

PFin: Es treten genau drei Fälle auf [39]:
$[m,n]\text{PFin} \subseteq [h,k]\text{PFin}[A]$ für kein Orakel A.
$[m,n]\text{PFin} \subseteq [h,k]\text{PFin}[A]$ genau dann wenn $A \geq_T K$.
$[m,n]\text{PFin} \subseteq [h,k]\text{PFin}[A]$ für alle Orakel A.
Beispielinklusionen für den ersten Fall ist $[1,3]\text{PFin} \not\subseteq [1,2]\text{PFin}$, für den zweiten Fall $[2,3]\text{PFin} \subseteq [1,1]\text{PFin}[K]$ und für den dritten Fall $[3,6]\text{PFin} \subseteq [1,2]\text{PFin}$.

Fin: Es treten vier Fälle auf [39], vermutlich gibt es keine weiteren:
$[m,n]\text{Fin} \subseteq [h,k]\text{Fin}[A]$ für kein Orakel A.

$[m,n]\mathrm{Fin} \subseteq [h,k]\mathrm{Fin}[A]$ genau dann wenn $A \geq_T K$.
$[m,n]\mathrm{Fin} \subseteq [h,k]\mathrm{Fin}[A]$ genau dann wenn A PA-vollständig ist.
$[m,n]\mathrm{Fin} \subseteq [h,k]\mathrm{Fin}[A]$ für alle Orakel A.
Beispielinklusionen für den zweiten Fall ist $[2,4]\mathrm{Fin} \subseteq [1,2]\mathrm{Fin}[K]$ und für den dritten Fall $[\,[24,49]\mathrm{Fin} \subseteq [2,4]\mathrm{Fin}[A] \;\Leftrightarrow\; A$ ist PA-vollständig].

Zum Team-Ex-Lernen ist noch anzumerken, dass $[m,n]\mathrm{Ex}[A]$ allwissend ist falls A high ist und trivial falls $A \leq B \leq K$ für eine 1-generic Orakel B gilt. Damit stimmen bei Ex die allwissenden und trivialen Orakel vom Team-Lernen mit denen im Normalfall überein.

Übung 6.6 Betrachten Sie die Klasse S, welche alle Funktionen

$$f_{x,b} = \begin{cases} x0^{a(x)}b^{\infty} & \text{falls } x \in K; \\ x0^{\infty} & \text{sonst}; \end{cases}$$

mit $x \in \mathbb{N}$ und $0 \leq b \leq k$ enthält. Zeigen Sie: $S \in [k+1, 2k+1]\mathrm{PFin} - [k, 2k-1]\mathrm{PFin}$.

7 Uniform rekursive Klassen und Monotonie

Angluin [3] betrachtete das Lernen von Text mit einer besonderen Restriktion: Die zu lernende Klasse von Mengen muss uniform rekursiv sein, d.h. $S = \{\{x : L(i,x) = 1\} : i \in \mathbb{N}\}$ für eine berechenbare und $\{0,1\}$-wertige Funktion L. Statt $\{x : L(i,x) = 1\}$ schreibt man kurz L_i. Eine Maschine M lernt S bezüglich einer Indizierung $H_0, H_1, \ldots$ wenn M auf jedem Text einer Menge L_i gegen ein j konvergiert mit $H_j = L_i$. Dabei müssen die H_y ebenfalls uniform berechenbar sein, d.h. die Funktion $H(j,x) = H_j(x)$ ist berechenbar in beiden Eingaben. In Abhängigkeit von der Familie H unterscheidet man drei Arten des Lernens:

- Exaktes Lernen: $H_i = L_i$ für alle i.
- Klassentreues Lernen: $\{H_0, H_1, \ldots\} = S$.
- Klassenumfassendes Lernen: $\{H_0, H_1, \ldots\} \supseteq S$.

Diese drei Begriffe werden dann mit Lernkriterien kombiniert, z.B. Fin und Ex. Zuerst wird das exakte Lernen betrachtet. Angluin [3, Theorem 1] zeigt folgendes Resultat:

Satz 7.1 *Eine Klasse $S = \{L_0, L_1, \ldots\}$ kann genau dann exakt im Limes gelernt werden wenn es zu jeder Menge L_i eine uniform in i aufzählbare endliche Menge $W_{f(i)}$ gibt so dass kein $L \in S - \{L_i\}$ die Beziehung $W_{f(i)} \subseteq L \subset L_i$ erfüllt.*

Beweis Angenommen zu jeder Menge L_i gibt es eine endliche Menge $W_{f(i)}$ so dass kein $L \in S - \{L_i\}$ die Beziehung $W_{f(i)} \subseteq L \subset L_i$ erfüllt. Dann lernt folgender Algorithmus die Klasse S exakt:

$$M(\sigma) = i \text{ für das kleinste } i \leq |\sigma| \text{ so dass } W_{f(i),|\sigma|} \subseteq range(\sigma) \subseteq L_i.$$

Es sei i nun der kleinste Index einer Menge $L \in S$ und T ein Text für L. Dann gilt für alle hinreichend langen $\sigma \preceq T$:

- $W_{f(i),|\sigma|} \subseteq range(\sigma) \subseteq L_i$
- $range(\sigma) \not\subseteq L_j$ falls $j < i$ und $L_i \not\subseteq L_j$
- $W_{f(j),|\sigma|} \not\subseteq L_i$ falls $j < i$ und $L_i \subset L_j$

Die erste Eigenschaft ist erfüllt, sobald alle Elemente von $W_{f(i)}$ im Text aufgetaucht sind – ab dann unabhängig davon, wann diese Elemente in $W_{f(i)}$ auftauchen. Für jedes feste j mit $L_i \not\subseteq L_j$ taucht irgendwann mal ein Zeuge $x \in L_i - L_j$ im Text auf und von da ab ist die Eigenschaft erfüllt. Ebenso gilt für jedes feste j mit $L_i \subset L_j$, dass $W_{f(j)}$ ein Element $x \in L_j - L_i$ enthält. Sobald ein solches x nach $W_{f(j)}$

aufgezählt ist, ist auch die dritte Bedingung für das betreffende j erfüllt. Durch die Bedingung $j < i$ bei der zweiten und dritten Eigenschaft wird erzwungen, dass nur endlich viele Requirements zu erfüllen sind und dass alle betrachteten Mengen L_j von L_i verschieden sind. Jede der endlich vielen Einzelbedingung gilt für fast alle $\sigma \preceq T$, daher ist auch die Gesamtbedingung für fast alle $\sigma \preceq T$ erfüllt. Sobald die Gesamtbedingung erfüllt ist, ergibt sich aus der ersten Eigenschaft $M(\sigma) \leq i$ und aus der zweiten bzw. dritten (je nachdem ob $L_i \not\subseteq L_j$ oder $L_i \subset L_J$) dass $M(\sigma) \neq j$ für jedes $j < i$. Es folgt $M(\sigma) = i$ für fast alle $\sigma \preceq T$. Also lernt M die Klasse S exakt von Text.

Umgekehrt sei nun M ein Algorithmus, der eine Klasse $S = \{L_0, L_1, \ldots\}$ von Text lernt. Die Eigenschaft, ob ein String σ Locking-Sequenz für L_i ist, ist in Π_1:

$$\sigma \text{ ist Locking-Sequenz für } L_i \ \Leftrightarrow\ (\forall \tau \in L_i^*)\,[M(\sigma\tau) = M(\sigma)]$$

Die Mengen L_i^* sind uniform aufzählbar, es sei $\sigma_0, \sigma_1, \ldots$ die Aufzählung aller Strings in L_i^*. σ_n sei die erste Locking-Sequenz für L_i. Die Menge

$$W_{f(i)} = range(\sigma_0) \cup range(\sigma_1) \cup \ldots \cup range(\sigma_n)$$

ist uniform aufzählbar: der Wertebreich von σ_m wird nach $W_{f(i)}$ aufgezählt, sobald sich herausstellt, dass $\sigma_0, \sigma_1, \ldots, \sigma_{m-1}$ alle keine Locking-Sequenzen sind. Auf diese Weise wird $range(\sigma_m)$ genau dann irgendwann mal nach $W_{f(i)}$ aufgezählt, wenn $m \leq n$ ist.

Wenn nun $L \supseteq W_{f(i)}$ ist, dann gibt es einen Text T für L, der mit der Locking-Sequenz σ_n für L_i beginnt. Wenn $L \subseteq L_i$ ist, dann macht M auf T nach σ_n keinen Hypothesenwechsel mehr und $M(\sigma_n)$ ist nicht nur ein Index für L_i sondern auch für L. Also ist $L = L_i$ und keine Menge $L \in S - \{L_i\}$ erfüllt die Beziehung $W_{f(i)} \subseteq L \subset L_i$. Somit ist Angluin's Satz bewiesen. ∎

Definition 7.2 Gegeben sei eine Klasse S und eine Maschine M, die S bezüglich der Numerierung $L_0, L_1, \ldots$ lernt. Im folgenden werden einige Eigenschaften des Inferenzvorgangs definiert, wobei M die genannte Eigenschaft erfüllt, wenn sie für jede Menge $L \in S$ und alle $\sigma, \tau \in L^*$ mit $\sigma \preceq \tau$ die genannte Bedingung erfüllt ist (sofern nicht $M(\sigma) = ?$ oder $M(\tau) = ?$):

endliches Lernen (Fin)
Zur Wiederholung: $M(\sigma) = M(\tau)$.

streng-monotones Lernen (SMon) [32]
$L_{M(\sigma)} \subseteq L_{M(\tau)}$.

monotones Lernen (Mon) [55]
$L \cap L_{M(\sigma)} \subseteq L \cap L_{M(\tau)}$.

konservatives Lernen (Conv) [3]
$M(\sigma) \neq M(\tau) \Rightarrow range(\tau) \not\subseteq L_{M(\sigma)}$.

Streng monotones und monotones Lernen sind zwei verschiedene Modellierungen von Lernern, die ihre Hypothesen vor allem durch das Neuhinzunehmen von Elementen verändern; im zweiten Fall dürfen versehentlich hinzugenommene $x \notin L$ wieder weggelassen werden. Konservatives Lernen modelliert einen Lerner, der stets so lange an einer Hypothese festhält bis sie erwiesenermassen falsch ist.

Die Begriffe wurden von Angluin, Jantke, Kapur, Lange, Wiehagen und Zeugmann in verschiedenen Veröffentlichungen [3, 32, 35, 36, 41, 42, 43, 55, 56, 57] eingeführt und weitergehend untersucht.

Satz 7.3 *Beim exakten Lernen von Text gilt:* Fin $\rightarrow$ SMon $\rightarrow$ Mon $\rightarrow$ Conv $\rightarrow$ Ex.

Beweis Von den Inklusionen folgt nur Mon $\rightarrow$ Conv nicht direkt aus der Definition. Zu einem gegeben einen monotonen Lerner M konstruiert man N induktiv wie folgt ($N(\lambda) = M(\lambda)$):

$$N(\sigma a) = \begin{cases} M(\sigma a) & \text{falls } N(\sigma) = ? \text{ oder } range(\sigma a) \not\subseteq L_{N(\sigma)}; \\ N(\sigma) & \text{sonst.} \end{cases}$$

Hypothesenwechsel werden also solange aufgeschoben bis sie gemäss der Definition des konservativen Lernens erlaubt sind. Von daher bleibt nur noch zu zeigen, dass N dieselben Sprachen lernt wie M:

Es sei $L \in S$, T ein Text für L und M konvergiert auf T gegen $M(\tau)$. N konvergiert ebenfalls auf T (da M alle Hypothesen von N übernimmt und insgesamt diese weniger oft wechselt) und zwar gegen einen Werte $M(\sigma)$ mit $\sigma \preceq \tau$.

$L_{M(\sigma)}$ ist keine echte Obermenge von $L_{M(\tau)}$ da sonst M gegen die Monotonie-Eigenschaft verstossen würde: Weil M ein exakter Lerner ist, ist $L_{M(\sigma)}$ eine der zu lernenden Sprachen und hätte einen Text, der mit τ beginnt. Auf diesem Text würde M dann erst $L_{M(\sigma)}$ vermuten, dann zu der Teilmenge $L_{M(\tau)}$ übergehen und schliesslich zu der richtigen Menge $L_{M(\sigma)}$ zurückkehren.

Des weiteren gilt für alle $\eta \preceq T$ die Eigenschaft $range(\eta) \subseteq L_{M(\sigma)}$ da sonst N nicht gegen $M(\sigma)$ konvergiert. Somit folgt dann $L = L_{M(\tau)} \subseteq L_{M(\sigma)}$ und insbesondere die Gleichheit $L_{M(\sigma)} = L_{M(\tau)}$.

Zu zeigen bleiben die Nichtinklusionen. Für die jeweiligen Fälle wird jeweils eine Klasse $S = \{L_0, L_1, \ldots\}$ angegeben.

SMon $\not\subseteq$ Fin: $L_n = \{0, 1, \ldots, n\}$.
Ein streng monotoner Inferenzalgorithmus gibt bei Eingabe σ stets den Index $\max(range(\sigma))$ der Menge $\{0, 1, \ldots, \max(range(\sigma))\}$ aus wobei $\max(\emptyset) = 0$. Umgekehrt kann die Klasse nicht endlich gelernt werden, da es einen String 0^k

geben muss mit $M(0^k) = 0$ (d.h. $L_{M(0^k)} = L_0 = \{0\}$) und somit z.B. die Menge $L_1 = \{0, 1\}$ nicht mehr von ihrem Text $0^{k+1}1^\infty$ erkannt wird.

Mon $\not\subseteq$ SMon: $L_0 = \{0, 2, 4, 6, \ldots\}$ und $L_{n+1} = \{0, 2, \ldots, 2n\} \cup \{2n + 1\}$.
Ein monotoner Lerner gibt solange den Index 0 von L_0 aus bis die erste ungerade Zahl $2n + 1$ in der Eingabe auftaucht. Dann kommt der einzige Hypothesenwechsel nach L_{n+1}. Es gibt jedoch keinen streng monotonen Lerner, weil dieser auf irgendeinem String σ aus geraden Zahlen die Hypothese 0 ausgibt, aber auch ein Text für $L_{\max(range(\sigma))+1}$ ebenfalls einen Text hat, der mit σ beginnt.

Conv $\not\subseteq$ Mon: $L_n = \mathbb{N} - \{n\}$
Ein konservativer Lerner gibt stets $\min(\mathbb{N} - range(\sigma))$ aus. Jeder Hypothesenwechsel von L_i nach L_j erfolgt nur, wenn i in der Eingabe erscheint und es ist $i \notin L_i$. Umgekehrt ist kein Lerner M von S monoton: Zuerst such man ein σ mit $M(\sigma) = i \neq ?$. Es gibt $j \notin range(\sigma) \cup \{i\}$. Also gibt es einen Text für L_j der mit σ beginnt. Daher gibt es $\tau \succeq \sigma$ mit $M(\tau) = j$. Nun kann man $k \notin range(\tau) \cup \{i, j\}$ wählten und einen Text $T \succeq \tau$ von L_k finden. Dann macht M auf T die Hypothesen i und j aber es gilt $L_k \cap L_i \not\subseteq L_k \cap L_j$ denn $j \in L_k \cap L_i - L_j$. Somit ist M nicht monoton.

Ex $\not\subseteq$ Conv: $\{n, n+1, \ldots\}$ für $n \in K$, $\{n, n+1, \ldots, n+m\}$ für $n \notin K \wedge m \in \mathbb{N}$.
Bei dieser Klasse muss man erst noch eine passende Numerierung $L_0, L_1, \ldots$ finden. Die Funktion

$$F(x, y) = \begin{cases} t+1 & \text{falls } x \text{ zum Zeitpunkt } t \text{ nach } K \\ & \text{aufgezählt wird und } t < y \text{ ist;} \\ 0 & \text{sonst;} \end{cases}$$

ist berechenbar. Es gilt stets $0 \leq F(n, m) \leq m$. Nun definiert man

$$L_{n,m} = \begin{cases} \{x \geq n : F(n, x) = 0\} & \text{falls } F(n, m) = 0; \\ \{n, n+1, \ldots, n+m-F(n, m)\} & \text{falls } F(n, m) > 0. \end{cases}$$

Ist $n \notin K$, so ist $L_{n,m} = \{n, n+1, \ldots\}$ für alle m. Ist $n \in K$, so ist jedes $L_{n,m}$ endlich und $\{n, n+1, \ldots, n+m\} = L_{n,m+k}$ für das erste k mit $F(n, k) > 0$. Mit Angluin's Bedingung kann man zeigen, dass die Klasse im Limes gelernt wird. Man setzt hierzu

$$W_{f(n,m)} = \{n\} \cup \{k \geq n : (\exists h \geq k)\, [h \in L_{n,m} \wedge h+1 \notin L_{n,m}]\}.$$

Falls $L_{n,m}$ unendlich ist, ist $W_{f(n,m)} = \{n\}$, falls $L_{n,m}$ endlich ist, ist $W_{f(n,m)} = L_{n,m}$. Somit ist Angluin's Bedingung im zweiten Fall erfüllt. Im ersten Fall ist

$L_{n,m} = \{n, n+1, \ldots\}$ und jede Menge $L \in S$ mit $\{n\} \subseteq L \subseteq \{n, n+1, \ldots\}$ ist gleich $L_{n,m}$. Damit ist Angluin's Bedingung erfüllt und S im Limes lernbar.

Umgekehrt sei nun M eine Maschine, die S konservativ lernt. Man zeigt nun, dass M nicht berechenbar ist, d.h. dass $K \leq_T M$ gilt. Dazu betrachtet man die Menge

$$A = \{n : (\exists \sigma, m)\, [range(\sigma) \subseteq \{n, n+1, \ldots, n+m\} \subset L_{M(\sigma)}]\}.$$

A ist aufzählbar in M. Wenn $n \notin K$, dann ist $L_{n,0} = \{n, n+1, \ldots\}$ und es gibt es ein $\sigma \in L_{n,0}^*$ so dass $M(\sigma)$ einen Index für $L_{n,0}$ ausgibt. $m = \max(range(\sigma))$ und σ erfüllen obige Bedingung und bezeugen $n \in A$.

Wenn umgekehrt $n \in A$ ist, dann gibt es m und σ mit $range(\sigma) \subseteq \{n, n+1, \ldots, n+m\} \subset L_{M(\sigma)}$. σ kann zu einem Text für $\{n, n+1, \ldots, n+m\}$ erweitert werden, aber M macht auf diesem Text keinen Hypothesenwechsel mehr und lernt die Menge $\{n, n+1, \ldots, n+m\}$ nicht. Also ist $n \notin K$.

Somit ist $A = \mathbb{N} - K$. K und $\mathbb{N} - K$ sind beide aufzählbar in M. Daraus folgt $K \leq_T M$. Die Definition für konservatives Lernen erfordert jedoch einen berechenbaren Lerner M und deshalb ist $S \notin$ Conv.

Diese Zeugen für die jeweiligen Nichtinklusionen sind übrigens in dem einen Kriterium exakt lernbar, aber in dem anderen nicht einmal klassenumfassend. ∎

Im folgenden soll noch gezeigt werden, dass es einen Unterschied zwischen exaktem, klassentreuen und klassenumfassenden streng-monotonen Lernen gibt.

Satz 7.4 *Es gibt eine Numerierung $L_0, L_1, \ldots$ der Klasse S_{fin} aller endlichen Mengen so dass S_{fin} nicht relativ zu dieser Numerierung exakt streng-monoton gelernt werden kann.*

Beweis Für jedes $x \in K$ sei $f(x)$ die Zeit welche eine gegebene Maschine braucht, um x nach K aufzuzählen. Ohne Beschränkung der Allgemeingültigkeit sei $f(x) > x$ für alle $x \in K$. Des weiteren sei $H_0, H_1, \ldots$ die kanonische Aufzählung der endlichen Mengen so dass $x \in H_e$ genau dann gilt wenn das Bit Nummer x (von hinten mit 0 beginnend gezählt) der Dualdarstellung von e eine 1 ist. Nun definiert man

$$L_e = \begin{cases} \{x, f(x)\} & \text{falls } H_e = \{x\} \text{ und } x \in K; \\ \{x\} & \text{falls } H_e = \{x, f(x)\} \text{ und } x \in K; \\ H_e & \text{sonst.} \end{cases}$$

Jede Menge L_e ist endlich und die Funktion $e, y \to L_e(y)$ ist rekursiv, da der Graph $G = \{(x, y) : f(x) \text{ is defniert und } f(x) = y\}$ der Funktion f rekursiv ist: Wenn

$x = \min(H_e)$ existiert, $(x, y) \in G$ und $\{x\} \subseteq H_e \subseteq \{x, y\}$, dann gilt $y \in L_e$ genau dann wenn $y \notin H_e$. Andernfalls gilt $y \in L_e$ genau dann wenn $y \in H_e$.

Angenommen, M sei ein streng-monotoner und exakter Lerner für $\{L_0, L_1, \ldots\}$. Dann gibt es eine Funktion g welche bei Eingabe x den Wert $M(x^k)$ berechnet für das erste k für welches $M(x^k) \neq ? \wedge x \in L_{M(x^k)}$ ist. Weil M streng-monoton und $\{x\} \in S_{fin}$ ist, muss $L_{g(x)} = \{x\}$ sein. Im Fall $x \in K$ ist $H_{g(x)} \neq L_{g(x)}$ und daher $H_{g(x)} = \{x, f(x)\}$. Im Fall $x \notin K$ gilt $L_{g(x)} = H_{g(x)} = \{x\}$. Bei den kanonischen Indizes kann man die Kardinalität berechnen und der Test $x \in K \Leftrightarrow |H_{g(x)}| = 2$ wäre berechenbar, im Widerspruch zur Unentscheidbarkeit von K. Also gibt es keinen exakten und streng-monotonen Lerner M für $\{L_0, L_1, \ldots\}$. ∎

Satz 7.5 *S enthalte für jedes $n \notin K$ die Menge $\{n\}$ sowie alle zweielementigen Mengen. S kann nicht klassentreu und streng monoton gelernt werden.*

Beweis Zuerst muss man zeigen, dass S eine Numerierung hat. Eine solche ist z.B. gegeben durch

$$\begin{aligned} L_{n,0} &= \begin{cases} \{n, n+m\} & \text{falls } n \text{ zum Zeitpunkt } m \text{ nach } K \text{ aufgezählt wird;} \\ \{n\} & \text{sonst, d.h., } n \notin K; \end{cases} \\ L_{n,m+1} &= \{n, n+m+1\}. \end{aligned}$$

Angenommen, M lernt gemäss einer Numerierung $H_0, H_1, \ldots$ die Klasse S streng-monoton und klassentreu. Die Menge

$$A = \{n : (\exists k)\, [M(n^k) \neq ? \wedge n \in H_{M(n^k)}]\, \}$$

ist aufzählbar und enthält jedes n mit $\{n\} \in S$ da diese Mengen $\{n\}$ vom Text n^∞ gelernt werden. Wenn M umgekehrt auf einer Eingabe n^k einen Index $M(n^k) \neq ?$ ausgibt und $\{n\} \notin S$ ist, dann muss die Menge $H_{M(n^k)}$ mindestens eine Zahl $m \neq n$ enthalten. Nun jedoch kann für kein $h \notin \{n, m\}$ die zweielementige Menge $\{n, h\}$ vom Text $n^{k+1}h^\infty$ streng monoton gelernt werden, also ist S nicht klassentreu lernbar. Wohl aber ist $S \subseteq S_{fin}$ und damit klassenumfassend lernbar. ∎

Übung 7.6 Zeigen Sie: Wenn M die Klasse S_{fin} monoton von Text lernt, so lernt M sie bereits streng monoton.

Übung 7.7 Unter welchen der Kriterien Fin, SMon, Mon, Conv und Ex ist die Klasse S gegeben durch $L_0 = \{1, 2, \ldots\}$ und $L_{n+1} = \{0, 1, \ldots, n\}$ exakt von Text lernbar?

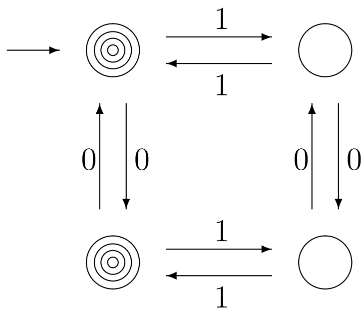
8 Polynomialzeitlernen mit einem Lehrer

In diesem Abschnitt wird das Lernen in polynomialer Zeit untersucht. Dazu muss man sich jedoch auf einfache Konzepte beschränken. Des weiteren wird das Lernen von einem Lehrer betrachtet, da zumindest exaktes Lernen nur von Beispielen in den meisten Modellbildungen nicht möglich ist.

Angluin [4] fand ein Modell, in dem die regulären Sprachen in polynomialer Zeit von einem Lehrer gelernt werden können. Dieser Abschnitt behandelt im wesentlichen das Lernen regulärer Sprachen und für die weitere Betrachtung werden zuerst die verschiedenen Darstellungen von regulären Sprachen an Hand eines Beispiels in Erinnerung gerufen:

Regulärer Ausdruck
$0^*(10^*10^*)^*$

Deterministischer Endlicher Automat



Linkslineare Grammatik
Startsymbol: A,
Weiteres Nonterminalzeichen: B,
Terminalzeichen: 0 und 1,
Produktionen: $A \to \lambda$, $A \to A0$, $A \to B1$, $B \to B0$, $B \to A1$.

Im weiteren wird bei Aufwandsberechnungen stets die zweite Darstellung zu Grunde gelegt. Aber es soll nicht vergessen werden, dass manchmal ein deterministischer endlicher Automat exponentiell grösser ist als ein regulärer Ausdruck derselben Sprache.

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

Elementanfragen
Ist $x \in L$? – Die Antwort ist entweder "Ja" oder "Nein".

Gleichheitsanfragen
: Erzeugt der Automat M die Sprache L? – Die Antwort ist entweder "Ja" oder ein Gegenbeispiel x auf dem M einen Fehler macht.

Teilmengenanfragen
: Erzeugt M eine Teilmenge von L? Ist L eine Teilmenge der von M erzeugten Sprache? – Die Antwort ist entweder "Ja" oder "Nein".

Die Länge der Antworten des Lehrers gehen in die Aufwandsabschätzung ein, da eine entsprechend lange Antwort die Rechenzeit des Algorithmus beliebig stark verlängern kann.

Beispiel 8.1 Erlaubt man im "Nein"-Fall nur Elementanfragen und Gleichheitsanfragen ohne Gegenbeispiel, dann benötigt man zum Lernen mindestens exponentiell viele Anfragen in der Grösse des kleinsten deterministischen endlichen Automaten.

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, d.h. der Lernalgorithmus muss fragen, ob $y \in L$ ist (oder ob $L = \{y\}$ ist). Dieser Aufwand ist exponentiell.

Umgekehrt reicht aber auch ein exponentieller Aufwand: Der Lernalgorithmus erzeugt in der Reihenfolge ihrer Grösse alle endlichen Automaten und stellt Gleichheitsanfragen. Den ersten Automaten, für welche die Anfrage bejaht wird, gibt der Algorithmus aus.

Satz 8.2 [4, Angluin] *Es gibt einen Algorithmus, der jeden unbekannten, endlichen Automaten M der Grösse n in Zeit $p(n, m)$ mittels Gleichheitsanfragen und Elementanfragen einen Automaten N lernt, der zu M äquivalent ist. Dabei ist p ein Polynom und m die grösste Länge eines Gegenbeispiels.*

Beweis Es wird nur der Algorithmus angegeben, für die Verifikation des Algorithmus wird auf Angluin's Artikel [4] verwiesen. Der Algorithmus braucht eine Datenstruktur, nämlich eine Tabelle aller bisher gesehenen Gegenbeispiele einschliesslich einiger zusätzlicher, durch Elementanfragen bestimmten Einträge. Diese Tabelle (S, E, T) ist gegeben wie folgt:

E ist eine endliche, unter Suffix abgeschlossene, Menge von Strings, d.h.:

$$(\forall \tau\eta \in E)\, [\eta \in E].$$

S ist eine endliche, unter Prefix abgeschlossene, Menge von Strings, d.h.:

$$(\forall \sigma\tau \in S)\, [\sigma \in S].$$

T ist eine endliche Funktion $T : S \cdot \{0,1,\lambda\} \cdot E \to \{0,1\}$ die für jedes $\sigma\tau\eta \in S \cdot \{0,1,\lambda\} \cdot E$ speichert, ob M die Eingabe $\sigma\tau\eta$ akzeptiert oder nicht.

Die Tabelle (S, E, T) heisst konsistent wenn für alle $\sigma_1, \sigma_2 \in S$ gilt: Sind die Zeilen $row(\sigma_1)$ und $row(\sigma_2)$ gleich, so gilt auch $row(\sigma_1 0) = row(\sigma_2 0)$ und $row(\sigma_1 1) = row(\sigma_2 1)$. Dabei ist $row(\sigma) = \{(\eta, T(\sigma\eta)) : \eta \in E\}$, also die Menge aller Einträge $(\sigma\eta)$ in der Tabelle mit $\eta \in E$.

Des weiteren heisst (S, E, T) geschlossen, wenn es zu jedem $\sigma_1 \in S$ und jedem $a \in \{0,1\}$ ein $\sigma_2 \in S$ gibt mit $row(\sigma_2) = row(\sigma_1 a)$. Zu jeder konsistenten und geschlossenen Tabelle (S, E, T) definiert man den Automaten $M(S, E, T)$ wie folgt:

Zustandsmenge ist S.
Startzustand ist λ.
Akzeptierende Zustände sind alle σ mit $T(\sigma) = 1$.
Übergangerelation: Der Zustand σ_1 geht bei Eingabe a nach σ_2 über wenn σ_2 das lexikographisch erste Element in S ist mit $row(\sigma_2) = row(\sigma_1 a)$.

Der Lernalgorithmus ist nun gegeben wie folgt:

Initialisiere $S = \{\lambda\}$ und $E = \{\lambda\}$.
(1) % Beginn Schleife 1 um M zu lernen.

(2) % Beginn Schleife 2 um (S, E, T) konsistent und geschlossen zu machen.
Solange T auf einem $\tau \in S \cdot \{0,1,\lambda\} \cdot E$ undefiniert ist,
bestimme $T(\tau)$ mittels einer *Elementanfrage* bei τ.
Wenn (S, E, T) nicht konsistent ist,
dann finde $\sigma_1, \sigma_2 \in S$, $a \in \{0,1\}$ und $\eta \in E$ so dass
$row(\sigma_1) = row(\sigma_2)$ und $T(\sigma_1 a\eta) \neq T(\sigma_2 a\eta)$,
addiere $a\eta$ zu E und gehe zu (2).
Wenn (S, E, T) nicht geschlossen ist,
dann finde $\sigma_1 \in S$ und $a \in \{0,1\}$ so dass
$row(\sigma_1 a) \neq row(\sigma_2)$ für alle $\sigma_2 \in S$,
addiere $\sigma_1 a$ zu S und gehe zu (2).

% Wenn dieser Punkt erreicht ist, ist (S, E, T) konsistent und geschlossen.
Mache eine *Gleichheitsanfrage* ob $M(S, E, T) = M$ ist.
Wenn "Ja" gib $N = M(S, E, T)$ aus.
Sonst addiere alle Präfixe $\tau \preceq \sigma$ des Gegenbeispiels σ zu S und geh zu (1).

Dieser Algorithmus lernt M mittels Element- und Gleichheitsanfragen. ∎

Es gibt noch einen weiteren Algorithmus, der sogar ohne Elementanfragen auskommt.

Satz 8.3 *Eine reguläre Sprache kann in Zeit $O(n^3 + m^3)$ mittels Gleichheitsanfragen gelernt werden, wobei n die Anzahl der Zustände des kleinsten Automaten für diese Sprache und m die maximale Länge eines Gegenbeispiels ist.*

Beweis Es sei $M_0, M_1, \ldots$ eine Liste aller endlicher Automaten geordnet nach der Anzahl ihrer Zustände. $L_0, L_1, \ldots$ seien die von ihnen erkannten Sprachen. Ferner bezeichnet $L \,\Delta\, L'$ die symmetrische Differenz der Sprachen L und L', d.h. die Sprache aller Wörter, die in genau einer der beiden Sprachen L und L' ist. Daher hat M_k höchstens $log(k)$ Zustände. Der Algorithmus läuft wie folgt:

Setze $k = 0$.
$(*)$ % Schleifenstart
    Produziere den Automaten N_k der die Sprache $L_k \,\Delta\, \{0^k\}$ erkennt.
    Frage ob N_k die gesuchte Sprache erzeugt.
    Bei Antwort "Ja" gib N_k aus und terminiere.
    Bei Antwort "Nein" mit Gegenbeispiel $x \neq 0^k$,
        setze $k = k + 1$ und geh nach $(*)$.
    Bei Antwort "Nein" mit Gegenbeispiel 0^k, frage ob M_k die Sprache erzeugt.
        Wenn diesmal "Ja", gib M_k aus und terminiere.
        Wenn wieder "Nein", setze $k = k + 1$ und geh nach $(*)$.

Die Grundidee des Algorithmus ist, die Automaten $M_0, M_1, \ldots$ in der Reihenfolge ihrer Grösse durchzuprobieren und gleichzeitig mit den Anfragen $N_0, N_1, \ldots$ das Gegenbeispiel so hochzutreiben, dass die polynomiale Schranke in n und m erhalten bleibt. $N_0, N_1, \ldots$ werden zuerst gefragt damit in dem Fall, dass M_k die gesuchte Sprache erzeugt, noch gerade eben vorher ein entsprechend langes Gegenbeispiel ausgegeben wird. Antwortet der Lehrer mit einem anderen Gegenbeispiel als 0^k, so weiss man, dass auch M_k falsch ist. Der Algorithmus terminiert nach endlich vielen Schritten gemäss einem der beiden folgenden Fälle:

- $L_k \,\Delta\, \{0^k\}$ ist die gesuchte Sprache, die von N_k erzeugt wird. M_k hat $\log(k)$ Zustände und $L_k \,\Delta\, \{0^k\}$ kann von einem Automaten mit n Zuständen erzeugt werden. Der Produktautomat akzeptiert $\{0^k\} = L_k \,\Delta\, (L_k \,\Delta\, \{0^k\})$ mit $n \cdot log(k)$ Zuständen. Es folgt $n \geq \frac{k+1}{\log(k)}$, da ein endlicher Automat, der $\{0^k\}$ 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 die Antwort 0^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. ∎

Dieser Algorithmus zeigt, dass das Gegenbeispiel irgendwie eine unnatürlichen Einfluss auf die Modellierung hat, denn der gegebene Algorithmus ist nicht sehr intelligent, sondern treibt den Parameter m systematisch hoch. Von daher versucht man, ohne Gegenbeispiele auszukommen. Äquivalenzanfragen sind dann nicht mehr stark genug, daher verwendet man Teilmengenanfragen. Der folgende Satz beruht vor allem darauf, dass bei Angluin's Algorithmus die Schranken noch eine zusätzliche Uniformitätsbedingung erfüllen: Bei der k-ten Anfrage ist die Rechenzeit bis zu dieser Anfrage sowie auch ihre Länge polynomial beschränkt ist in n und der maximalen Länge m' der Gegenbeispiele zu den Anfragen 1, 2, …, $k-1$.

Satz 8.4 *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.*

Beweis Der Algorithmus simuliert den von Angluin aus Satz 8.2 unter Berücksichtigung folgender Punkte:

- Er läuft mit Parametern $n' = 0, 1, \ldots$ und $m' = n'(n' + 1)$.
- Die Elementanfrage "Ist $x \in L$?" wird ersetzt durch "Ist $\{x\} \subseteq L$?"
- Gleichheitsanfragen werden durch zwei Teilmengenanfragen ersetzt: "Ist $L \subseteq H$?" und "Ist $H \subseteq L$?" Bei einer verneinten Gleichheitsanfrage wird durch Intervallschachtelung ein Gegenbeispiel bis zur Länge m' gesucht.

  Ist z.B. $H \not\subseteq L$, sucht man das erste $k \leq n'$ mit $H \cap \{0,1\}^k \not\subseteq L$. Danach bestimmt man induktiv ein $a_1 a_2 \ldots a_k \in H - L$, indem man für $h = 0, 1, \ldots, k{-}1$ fragt, ob $a_1 a_2 \ldots a_h 0 \{0,1\}^{k-h-1} \not\subseteq L$ oder $a_1 a_2 \ldots a_h 1 \{0,1\}^{k-h-1} \not\subseteq L$ ist.
- Wenn L von einem Automaten mit n' Zuständen erzeugt wird und wenn die von einem Automaten N erzeugte Sprache mit L auf allen Wörter mit Länge $\leq m' = n'(n' + 1)$ übereinstimmt, dann kann in polynomialer Zeit ein Automat $M(N, n')$ erzeugt werden, der L akzeptiert.

  Hierzu betrachtet man die Menge U derjenigen Zustände von N, die innerhalb von n Schritten von dem Startzustand erreicht werden. Zu $u \in U$ und $a \in \{0, 1\}$

und den Übergang $ua \to u'$ wählt man den kleinsten Knoten $u' \in U$ so dass für alle Wörter w der Länge $\leq n'^2$ gilt, dass N von u aus bei Eingabe w genau dann in einen akzeptierenden Zustand übergeht, wenn N von u' aus bei Eingabe aw in einen akzeptierenden Zustand übergeht. Ein Zustand $u \in U$ ist akzeptierend, wenn er in N akzeptierend ist. Danach entfernt man aus U alle Zustände, die nicht erreicht werden.

Der Algorithmus selbst läuft wie folgt:

Initialisiere $n' = 1$

($*$) Setze $m' = n'(n'+1)$ und simuliere Angluin's Algorithmus bis (a) oder (b).

(a) Der gesuchte Automat wurde gefunden:
Terminiere mit Ausgabe dieses Automaten.

(b) Bei einer verneinten Äquivalenzanfrage mit Automat N existiert
kein Gegenbeispiel der Länge $\leq m'$:
Teste, ob $M(N, n')$ existiert und die Sprache L erzeugt.
Wenn ja, terminiere mit Ausgabe $M(N, n')$.
Wenn nein, setze $n' = n' + 1$ und führe die Simulation ($*$) fort.

Da die Laufzeit von Angluin's Algorithmus zwischen zwei Fragen immer polynomial in n und der längsten, bisher gesehenen Antwort ist, ist der ganze Algorithmus polynomial in n und m. Wegen der Abbruchbedingung (b) ist das m aber durch $n(n+1)$ beschränkt und der Gesamtalgorithmus läuft polynomial in n. ∎

Einen anderen Weg gingen Ibarra und Jiang [31, Theorem 5]. Sie fixierten die Gegenbeispiele und zeigten:

Satz 8.5 *Falls der Lehrer auf eine Anfrage immer das lexikographisch erste der kürzesten Gegenbeispiele liefert, kann ein Algorithmus alle regulären Sprachen mittels Äquivalenzanfragen lernen wobei der zeitliche Aufwand polynomial in der Zustandszahl n eines minimalen endlichen Automaten ist.*

Des weiteren hat man das Hochtreiben unterbunden, in dem man verlangt, dass jede Anfrage polynomial in der Grösse der bisher gesehenen Gegenbeispielen sowie in n sein muss. Unter diesen Umständen zeigte Angluin [5, 8], dass es keinen polynomial beschränkten Lernalgorithmus gibt, der nur mit Äquivalenzanfragen und Gegenbeispielen auskommt.

Übung 8.6 Geben Sie die formale Definition eines Automaten an, der die Menge $a_0 a_1 \ldots a_k \cdot \{0,1\}^h$ akzeptiert. Des weiteren seien M und N Automaten, die L und H akzeptieren. Geben Sie die Produktautomaten an, die $L \cap H$, $L \cup H$ und $L \Delta H$ akzeptieren.

9 Probabilistische Approximation (PAC)

Valiant [53] führte ein Modell des statistischen Lernens ein, bei dem der Lerner von zufällig gezogenen Beispielen mit hoher Wahrscheinlichkeit eine fast überall richtige Funktion lernt. Diese beiden Einschränkungen sind notwendige Zugeständnisse an das Lernen von zufälligen Daten: Auf der Eingabe kann – wenn auch nur mit äusserst geringer Wahrscheinlichkeit – immer dasselbe nichtssagende Beispiel kommen, also darf der Algrotihmus in seltenen Ausnahmefällen was absolut falsches liefern und muss nur mit hoher Wahrscheinlichkeit ungefähr richtig sein. Einige Daten, auf der sich zwei der zu lernenden Funktionen unterscheiden, könnten so selten sein, dass man sie nicht unter vorgegebener Zeitschranke mit hinreichend grosser Wahrscheinlichkeit findet und so die beiden Funktionen unterscheiden kann. Daher wird nur ungefähre Richtigkeit und nicht exakte Übereinstimmung verlangt. Diese Idee wird nun für das Lernen Boole'scher Funktionen näher spezifiziert:

Definition 9.1 Es sei S eine Familie von Klassen Boole'scher Funktionen, die in Abhängigkeit von einem Parameter n die Eingaben $\{0,1\}^n$ nach $\{0,1\}$ abbilden. S heisst PAC-lernbar, wenn es einen Algorithmus M sowie Polynome p, q gibt mit folgenden Eigenschaften:

- M lernt in Abhängigkeit des Parameters n, δ, ϵ jede Funktion $f \in S$.
- M erhält als Eingabe die Parameter n, δ, ϵ sowie $m = \lceil p(n, \frac{1}{\delta}, \frac{1}{\epsilon}) \rceil$ Beispiele $(x_1, f(x_1)), \ldots, (x_m, f(x_m))$ wobei die Vektoren $x_m \in \{0,1\}^n$ zufällig gemäss einer unbekannten Verteilung D gewählt werden.
- M berechnet aus den Daten in Zeit $q(n, \delta, \epsilon)$ eine Formel g.
- Mit Wahrscheinlichkeit $1 - \delta$ ("probably") gilt, dass $D(\{x : f(x) \neq g(x)\}) \leq \epsilon$ ist ("approximately correct").

Der Zufall kommt dabei sowohl durch die zufällige Verteilung der Daten als auch ggf. durch zufällige Verzweigungen im Lernalgorithmus ins Spiel.

Beispiel 9.2 *Die Klasse* $S_{mon} = \{f_a : (\forall x)\, [f_a(x) = 0 \Leftrightarrow x \leq_{lex} a]\, \}$ *aller monotonen Funktionen ist PAC-lernbar.*

Beweis Der Lernalgorithmus bestimmt unter den Beispielen das lexikographisch grösste b mit $f(b) = 0$ und wählt die Funktion f_b. Falls kein solches Beispiel existiert, wird $b = 0^n$ gewählt. Zu bestimmen bleibt die notwendige Anzahl m der Beispiele, damit der Algorithmus seinen Spezifikationen genügt. Es sei c der lexikographisch erste Vektor, so dass $D(\{d : c <_{lex} x \leq_{lex} a\}) \leq \epsilon$ ist. Es gibt zwei Fälle:

- $c = 0^n$: Dann ist stets $c \leq_{lex} b \leq_{lex} a$ und mit Wahrscheinlichkeit 1 wird eine Formel gelernt, die der Bedingung $D(\{x : f(x) \neq g(x)\}) \leq \epsilon$ genügt.
- $c \neq 0^n$. Dann ist $D(\{x : c \leq_{lex} x \leq_{lex} a\}) > \epsilon$. Also ist mit Wahrscheinlichkeit $1 - (1-\epsilon)^m$ eins aus m Beispielen in diesem Intervall. m muss so gross gewählt werden, dass $1 - (1-\epsilon)^m \geq 1 - \delta$ ist.

Also ist die Bedingung $1 - (1-\epsilon)^m \geq 1 - \delta$ hinreichend. Sie lässt sich umformen:

$$\begin{aligned} 1 - \delta &\leq 1 - (1-\epsilon)^m \\ (1-\epsilon)^m &\leq \delta \\ m \cdot \log(1-\epsilon) &\leq \log(\delta) \\ m &\geq \tfrac{\log(\delta)}{\log(1-\epsilon)} \end{aligned}$$

Nun benötigt man am Ende nur ein hinreichend grosses m, dieses muss aber nicht optimal sein. Man kann den Term $|\log(1-\epsilon)| = \epsilon + \epsilon^2 + \epsilon^3 + \ldots$ nach unten durch ϵ abschätzen und $|\log(\delta)|$ nach oben durch $\frac{1}{\delta}$ abschätzen. Dadurch erhält man $m \geq \frac{1}{\delta\epsilon}$. Dieses m ist ein Polynom in n, $\frac{1}{\delta}$ und $\frac{1}{\epsilon}$. Die Rechenzeit ist ebenfalls polynomial in n und der Beispielanzahl m, also polynomial in den drei Parametern von p. Zusammenfassend gilt:

> Zu gegebenem n, δ und ϵ liesst M dann $\lceil \frac{2}{\delta\epsilon} \rceil$ Beispiele $(x, f(x))$ ein und bestimmt unter ihnen das lexikographisch grösste b mit $f(b) = 0$. Falls es ein solches nicht gibt, ist $b = 0^n$. Dann gilt mit Wahrscheinlichkeit $1 - q$, dass der Fehler von g bezüglich D durch ϵ begrenzt ist: $D(\{x : g(x) \neq f(x)\}) \leq \epsilon$.

Um z.B. mit 99% Wahrscheinlichkeit einen Fehler von nur 1% zu haben, benötigt man 10000 Beispiele. ▮

Im Anfrage-Lernmodell kann man S_{mon} übrigens mit $n + 1$ Element-Anfragen durch Halbierungssuche gelernt werden. Der nächste Satz nennt ein weiteres Beispiel:

Beispiel 9.3 [48] *Die Klasse S_{k-CNF} aller Formeln in konjunktiver Normalform aus Termen mit höchstens k Variablen ist PAC-lernbar.*

Beweis Man betrachte folgenden Lernalgorithmus:

g wird initialisiert als eine Konjunktion über alle Terme in bis zu k Variablen,
d.h. $g = t_1 \wedge t_2 \wedge \ldots \wedge t_h$ mit $t_1 = (x_1 \vee x_2 \vee \ldots \vee x_k)$, ...
$(*)$ Lies die nächsten $m = \lceil \frac{1}{\delta\epsilon} \rceil$ Beispiele ein.
Falls g alle richtig klassifiziert, gib g aus und terminiere.
Sonst ist $g(x) = 0$ für einige Beispiele $(x, 1)$.
Entferne alle Terme t von g mit $t(x) = 0$ für ein Beispiel $(x, 1)$.
Durchlaufe erneut die Schleife $(*)$.

g besteht zu Beginn aus $h = (2n)^k$ Termen. Bei jedem Durchlauf der Schleife wird mindestens ein Term gelöscht oder die Schleife verlassen. Daher kommt man mit $(2n)^k + 1$ Durchläufen der Schleife aus. Beim Verlassen der Schleife stimmt g mit der zu lernenden Funktion f auf $m = \lceil \frac{1}{\delta\epsilon} \rceil$ neu gezogenen Beispielen überein, die von denjeningen stochastisch unabhängig sind, aus welchen g berechnet wurde.

g enthält dann alle Terme, die nicht den Daten widersprechen, d.h. g ist die Konjunktion aller Terme t, die für jedes der Beispiele $(x, 1)$ die Bedingung $t(x) = 1$ erfüllen. f dagegen enthält nur einige dieser Terme, also gilt $g(x) \leq f(x)$ für alle Vektoren x. Zu zeigen bleibt, dass die Menge der Unterschiede klein ist, d.h. dass $D(\{x : g(x) = 0 \wedge f(x) = 1\}) \leq \epsilon$.

Wie im Beweis von Beispiel 9.2 kann man auch hier folgendes zeigen: wenn es genug Unterschiede gibt, d.h. wenn $D(\{x : f(x) \neq g(x)\}) > \epsilon$ ist, dann ist mit Wahrscheinlichkeit $1 - \delta$ mindestens eines der letzten m Beispiele in dieser Menge $\{x : f(x) \neq g(x)\}$. Da aber g und f auf den zuletzt gezogenen m Beispielen übereinstimmen, gilt im Umkehrschluss, dass g und f mit Wahrscheinlichkeit $1-\delta$ hinreichend ähnlich sind, d.h. mit Wahrscheinlichkeit $1 - \delta$ ist $D(\{x : f(x) \neq g(x)\}) \leq \epsilon$.

Also ist S_{k-CNF} PAC-lernbar mit der Schranke $p(n, \frac{1}{\delta}, \frac{1}{\epsilon}) = (2n)^{k+1} \cdot \frac{1}{\delta} \cdot \frac{1}{\epsilon}$ auf die Gesamtzahl der Beispiele. ∎

Das Lernverfahren gibt gleichzeitig eine obere Anzahl der notwendigen Beispiele an, und zwar $(2n)^{k+1} \cdot \frac{1}{\delta} \cdot \frac{1}{\epsilon}$. Im folgenden soll auch eine Abschätzung nach unten erfolgen. Diese Abschätzung benötigt den Begriff der Vapnik-Chervonenkis-Dimension:

Definition 9.4 S sei eine Klasse von Konzepten über einer Grundmenge X. Das grösste d mit

$$(\exists x_1, \ldots, x_d \in X)\,(\forall y_1, \ldots, y_d \in \{0, 1\})\,(\exists f \in S)\,[f(x_1) = y_1 \wedge \ldots \wedge f(x_d) = y_d]$$

heisst die Vapnik-Chervonenkis-Dimension $VC(S)$ von S. Falls es kein grösstes solches d gibt, d.h. zu jedem gibt es ein noch grösseres, ist $VC(S) = \infty$.

Satz 9.5 [13, Theorem 2.1] *Wenn eine Maschine M zu Beispielen $(x_1, y_1), (x_2, y_2)$, ..., (x_m, y_m) in polynomialer Zeit eine Funktion $g \in S$ mit $g(x_k) = y_k$ findet und* *wenn $1 < VC(S) = d < \infty$ ist, dann ist S PAC-lernbar von $\frac{16}{\epsilon} \cdot |\log(\delta) + d \cdot \log(\epsilon)|$ Beispielen. Umgekehrt kommt im Fall $\delta, \epsilon < \frac{1}{2}$ kein Lernverfahren mit weniger als $\frac{|\log(\delta)|}{4\epsilon} + \frac{d}{4}$ Beispielen aus.*

Der Lernalgorithmus ist dabei die vorgegebene Funktion M: M berechnet zu den gegebenen Beispielen ein konsistentes $g \in S$ und es gilt bereits mit Wahrscheinlichkeit $1-\delta$, dass $D(\{x : f(x) \neq g(x)\}) \leq \epsilon$ ist.

Man kann natürlich auch in obige Formeln das d auch in Abhänigkeit von n einsetzen, d.h. d_n ist die Vapnik-Chervonenkis-Dimension der Funktionen $f \in S$ von $\{0,1\}^n$ nach $\{0,1\}$. Ferner kann man erwarten, dass für jede Funktion $f \in S$ mit Eingabebereich $\{0,1\}^n$ eine Beschreibung der Länge $p(n)$ für ein Polynom p existiert – sonst könnte man gar nicht alle Funktionen in polynomialer Zeit spezifizieren. Es folgt:

Satz 9.6 *Wenn eine Maschine M zu Beispielen $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ in polynomialer Zeit eine Funktion $f \in S$ mit $f(x_k) = y_k$ findet und $d_n \leq p(n)$ ist für ein Polynom p, dann ist S PAC-lernbar von $\frac{16}{\epsilon} \cdot |\log(\delta) + p(n) \cdot \log(\epsilon)|$ Beispielen.*

Es gibt jedoch Klassen, wo ein solches M nicht existiert – M braucht nicht einmal mit einer gewissen Wahrscheinlichkeit berechenbar zu sein. Ein solches Beispiel ist die Klasse $S_{CNF} = \{t \wedge s : t$ und s sind $\vee$-Terme$\}$ wobei t und s alle Variablen in positiver oder negativer Form haben dürfen. Pitt und Smith [47] zeigten:

Satz 9.7 *S_{CNF} ist nicht PAC-lernbar unter der Hypothese $R \neq NP$.*

Die Hypothese $R = NP$ besagt, dass man bei jeder erfüllbaren Formel ϕ mit Wahrscheinlichkeit $> \frac{1}{2}$ eine erfüllende Belegung der Variablen finden könnte. Die Hypothese $R \neq NP$ besagt entsprechend, dass es keinen solchen Algorithmus gibt. Es ist unbekannt, welche der Hypothesen richtig ist.

Das Modell des PAC-Lernens ist natürlich nicht auf Boole'sche Funktionen beschränkt. Man kann z.B. auch Funktionen $f : \mathbb{N} \to \mathbb{N}$ lernen. Die obigen Definitionen lassen sich leicht anpassen; man gibt aber die Laufzeitschranke auf, sondern zählt nur noch die Anzahl der Beispiele.

Satz 9.8 *Die Klasse S_{lin} der linearen Funktionen $f(x) = ax + b$ ist PAC-lernbar.*

Beweis Zuerst der Lernalgorithmus für S_{lin}. OBdA sei $\epsilon < \frac{1}{2}$. f bezeichne die zu lernende Funktion.

Man analysiert die ersten $m = 1 + \lceil \frac{2}{\delta\epsilon} \rceil$ Beispiele.
Wenn sich zwei verschiedene Paare (a, b) und (c, d) finden, dann setzt man

$g(x) = \frac{b(c-x)+d(x-a)}{c-a}$; ansonsten gibt es nur ein Paar (a, b) und man setzt $g(x) = b$.

Man unterscheidet zwei Fälle:

(a) Es gibt ein a mit $D(\{a\}) \geq 1 - \epsilon$.
Mit Wahrscheinlichkeit $1 - (1 - \epsilon)^m \geq 1 - \delta$ kommt a unter den m Beispielen vor. Damit ist die Formel g bei der Eingabe a richtig. Also gilt die Inklusion $\{x : g(x) \neq f(x)\} \subseteq \{x : x \neq a\}$ und es ist $D(\{x : g(x) \neq f(x)\}) \leq \epsilon$.

(b) Es gibt kein solches a.
Es sei (a, b) das erste Beispiel. Da $D(\{x : x \neq a\}) \geq \epsilon$ taucht gemäss der Rechnung in Beispiel 9.2 unter den folgenden $\lceil \frac{2}{\delta\epsilon} \rceil$ Paaren mit Wahrscheinlichkeit $1 - \delta$ ein Beispiel (b, d) mit $b \neq a$ auf. Dann berechnet der Algorithmus die lineare Funktion an zwei Stützstellen und es ist $g = f$, insbesondere ist auch $D(\{x : g(x) \neq f(x)\}) = 0$.

In beiden Fällen wird die lineare Funktion gelernt. ∎

Übung 9.9 Ist die Klasse aller Polynome p mit $\text{Grad}(p) \leq k$ für jedes feste k gemäss dem Kriterium PAC lernbar?

10 Lernen mit Platzbeschränkung

Freivalds, Kinber und Smith [24, 37] führten die Unterscheidung zwischen Kurz- und Langzeitgedächtnis ein. Die Grundidee ist, dass die lernende Maschine nur in einem begrenzten Langzeitgedächtnis relevante Informationen speichert während das Kurzzeitgedächtnis vor dem Lesen jeder neuen Eingabe gelöscht wird. Es dient hauptsächlich dazu, Rechnungen auszuführen ohne das Langzeitgedächtnis überschreiben zu müssen. Man kann sowohl Funktionen als auch Sprachen nach diesem Modell lernen, aber im folgenden liegt das Augenmerk auf dem Sprachen-Lernen von Text, da hierfür die schönsten Ergebnisse erzielt wurden.

Definition 10.1 Eine Maschine mit beschränktem Langzeitspeicher lernt eine Sprache L von einem Text $T = a_0 a_1 \ldots$ wie folgt:

Daten bei Schritt n
Inhalt des Langzeitgedächtnisses vor Schritt n und a_n.

Ausgabe
Feststellen, ob eine Ausgabe gemacht werden soll.
Wenn ja, Berechnung und Ausgabe einer Hypothese e.
Wenn nein, Ausgabe von "?".

Neuer Langzeitspeicher
Aus altem Inhalt des Langzeitgedächtnisses und aus a_n wird der neue Inhalt des Langzeitgedächtnisses berechnet.

Schritt $n+1$
Nach Durchlauf der obigen drei Schritte wird das Kurzzeitgedächtnis gelöscht und dieselben Schritte mit für $n+1$ durchlaufen.

Hierbei ist zu beachten, dass der Wert von n der Maschine nicht unbedingt zur Verfügung stehen muss. Das Langzeitgedächtnis ist auf jedem Text T einer Sprache $L \in S$ stets in seiner Grösse beschränkt: Wenn $D_x = range(a_0 a_1 \ldots a_n)$ dann darf das Langzeitgedächtnis jede Zahl $y \leq f(x)$ enthalten wobei f die Beschränkung des Langzeitgedächtnisses näher spezifiziert. Der kanonische Index x einer Menge $D = \{b_1, b_2, \ldots, b_m\}$ ist gegeben durch $x = 2^{b_1} + 2^{b_2} + \ldots + 2^{b_m}$, z.B. $D_0 = \emptyset$ und $D_6 = \{1, 2\}$. Wenn jedoch $range(\sigma) \not\subseteq L$ für jede Menge $L \in S$, dann werden an das Verhalten von M ab dem Einlesen von σ keine Anforderungen mehr gestellt.

Die Maschine M konvergiert gegen den Index e falls M auf dem Text T nur endlich viele Ausgaben macht und die letzte davon e ist oder falls M unendlich viele Ausgaben auf dem Text T macht, die fast alle mit e übereinstimmen.

In der Literatur [24, 37, 38] wird anstatt mit Zahlen mit Strings gearbeitet, aber dies macht die Darstellung manchmal etwas komplizierter. Die Beweise und Belegmengen aus der zitierten Literatur wurden daher an die hier verwendete Notation angepasst.

Beispiel 10.2 Die Klasse $S_{single} = \{\{x\} : x \in \mathbb{N}\}$ aller einelementigen Mengen kann ohne jedes Langzeitgedächtnis gelernt werden – aber nur wenn die Ausgabe nicht unendlich oft wiederholt werden muss.

Unter dem gegebenen Konvergenzkriterium lernt eine Maschine, in dem sie bei augenblicklicher Eingabe x einen kanonischen Index für $\{x\}$ und bei augenblicklicher Eingabe # ein "?" ausgibt. Im Fall der Platzbeschränkung unterscheidet sich also das hier verwendete Konvergenz-Kriterium von dem üblichen. Letzteres funktioniert nicht, da der Text $x\#^\infty$ dann eine Speicherung des x in irgendeiner Form erfordert.

Satz 10.3 *Die Menge S_{fin} aller endlichen Mengen kann genau dann von Text mit der monotonen Schranke f (für das Langzeitgedächtnis) gelernt werden, wenn für alle x die Beziehung $f(x) \geq x$ gilt.*

Beweis Angenommen es sei $f(x) \geq x$ für alle x. Dann inferiert folgender Lernalgorithmus alle endlichen Mengen:

> Schritt n: Gegeben ist das Langzeitgedächtnis mit Inhalt x;
> D_x ist die Menge der bisher gesehenen Elemente $a_0, a_1, \ldots, a_{n-1}$.
> Eingabe: Nächstes Element a_n.
> Rechnung: Wähle y mit $D_y = D_x \cup \{a_n\}$.
> Ausgabe: Index $e(y)$ mit $W_{e(y)} = D_y$.
> Neuer Inhalt des Langzeitgedächtnis: y.

Der Algorithmus hält alle geforderten Platzschranken ein. Umgekehrt nehme man nun an, eine Maschine M lernt alle endlichen Mengen; zu zeigen bleibt, dass sie den Platz auch benötigt: $G(\sigma)$ bezeichne das Langzeitgedächtnis nachdem die Eingabe σ abgearbeitet wurde und nun die nächste Zahl erwartet wird.

Angenommen es sei $G(\sigma) = G(\tau)$ aber $range(\sigma) \neq range(\tau)$. Es gibt ein $x \notin range(\sigma) \cup range(\tau)$, so dass weder während dem Einlesen von σ noch während dem Einlesen von τ ein Index für $range(\sigma) \cup \{x\}$ oder $range(\tau) \cup \{x\}$ ausgegeben wurde. Daher muss M auf dem Text σx^∞ nach dem Lesen einiger x-e einen Index für $range(\sigma) \cup \{x\}$ und auf dem Text τx^∞ nach dem Lesen einiger x-e einen Index für $range(tau) \cup \{x\}$ ausgeben. Da aber das Konvergenzverhalten von M aus den Texten σx^∞ und τx^∞ gleich ist, kann M entweder $range(\sigma) \cup \{x\}$ oder $range(\tau) \cup \{x\}$ nicht von diesen Texten lernen.

Zu jedem x sei σ_x der String mit $range(\sigma_x) = D_x$. Die Strings σ_0, σ_1, …, σ_x

haben alle verschiedenen Wertebereich und daher sind die Werte $G(\sigma_0)$, $G(\sigma_1)$, $\ldots$, $G(\sigma_x)$ alle verschieden. Da $G(\sigma_y) \leq f(x)$ für $y \leq x$ ist, hat die Menge $\{0, 1, \ldots, f(x)\}$ mindestens $x + 1$ Elemente und ist $f(x) \geq x$ für alle x. ▮

Definition 10.4 [46, Schäfer] Eine Maschine M heisst set-driven, wenn für je zwei Eingaben σ, τ mit $range(\sigma) = range(\tau)$ die Beziehung $M(\sigma) = M(\tau)$ gilt.

Satz 10.5 *Wenn eine Klasse S mit einer Platzschranke f von Text gelernt werden kann, dann kann S auch von einer set-driven Maschine gelernt werden. Umgekehrt kann jede mit einer set-driven Maschine lernbare Klasse S auch mit der Platzschranke $f(x) = x$ gelernt werden.*

Beweis M sei eine mittels f platzbeschränkte Maschine, die S lernt. Gegeben sei nun eine Eingabe $\sigma \in L^*$ mit $range(\sigma) = D_x$ für ein $L \in S$. $G(\tau)$ bezeichne den Langzeitspeicher nach dem Einlesen von τ. Es ist $G(\tau) \leq f(x)$ für alle $\tau \in D_x^*$. Nun tabelliert man sukzessiv für jedes y die Werte τ_y, $e_{y,a}$ und $z_{y,a}$ bis es zu jedem $y = G(\tau)$ und jedem $a \in D_x \cup \{\#\}$ eine Zeile gibt. Hier die fünf Werte in der (y, a)-ten Zeile:

y	Alter Wert des Langzeitgedächtnisses.
a	Nächste Zahl oder # an der Eingabe.
τ_y	Erster String $y \in D_x^*$ mit $G(\tau_y) = y$.
$e_{y,a}$	Ausgabe bei Speicher y und Eingabe $a \in D_x \cup \{\#\}$.
$z_{y,a}$	Neuer Wert des Langzeitgedächtnisses.

Die Tabelle ist nicht nur endlich, sondern man kann auch feststellen, wann sie vollständig ausgefüllt ist: Wenn es zu jedem y mit $y = 0$ oder $y = z_{y',a'}$ sowie zu jedem $a \in D_x \cup \{\#\}$ eine Zeile in der Tabelle gibt. Anhand dieser Tabelle sucht man den ersten String in α_a (relativ einer Aufzählung $\alpha_0, \alpha_1, \ldots$ aller Strings in $\mathbb{N}^*$) so dass $\alpha_a \in D_x^*$, $M(\alpha_a) = e \neq ?$ und $M(\alpha_a \tau) \in \{?, e\}$ für alle $\tau \in D_x^*$, d.h. α_a ist die erste Locking-Sequenz für D_x. Nun setzt man

$$N(\sigma) = \begin{cases} M(\alpha_a) & \text{falls } \alpha_a \text{ gefunden wurde;} \\ ? & \text{sonst, d.h. falls es kein solches } \alpha_a \text{ gibt.} \end{cases}$$

Die neue Maschine N ist set-driven, denn die ganze Konstruktion hängt nur von $D_x = range(\sigma)$ ab. Des weiteren ist N immer definiert, wenn $range(\sigma) \subseteq L$ für ein $L \in S$ ist; ansonsten jedoch könnte N undefiniert sein (was erlaubt ist), da die Konstruktion der Tabelle nicht terminiert. Auf endlichen Mengen konvergiert N gegen einen korrekten Index, kritisch ist der Fall der unendlichen Mengen. Sei T nun der Text einer unendlichen Menge $L \in S$ und α_b die kürzeste Locking-Sequenz von L. Dann gilt für alle $c < b$ dass entweder $\alpha_c \notin L^*$ oder $M(\alpha_c) = ?$ oder $M(\alpha_c \tau) \notin \{?, M(\alpha_c)\}$ für ein $\tau \in L^*$ ist. Die Strings der beiden ersten Fällen werden nie ausgewählt, der String im dritten Fall wird nur so lange ausgewählt bis $range(\alpha_c \tau) \subseteq range(\sigma)$ gilt. Also werden die endlich vielen Strings α_c mit $c < b$ alle nur endlich oft ausgewählt und daher gilt für fast alle $\sigma \preceq T$, dass ein String α_a mit $a \geq b$ ausgewählt wird. Es ist jedoch auch $range(\alpha_b) \subseteq range(\sigma)$ für fast alle $\sigma \preceq T$; aus diesem Grund wird dann fast immer der String α_b ausgewählt und $M(\alpha_b)$ ausgegeben: also konvergiert N auf T gegen $M(\alpha_b)$ und nach Voraussetzung ist $M(\alpha_b)$ ein Index für L. Somit lernt die set-driven Maschine N jede Sprache $L \in S$.

Umgekehrt sei M eine set-driven Maschine, welche die Klasse S lernt. Zu M erzeugt man einen Lernalgorithmus, der jede Platzschranke f mit $(\forall x)\,[f(x) \geq x]$ beachtet:

Altes Langzeitgedächtnis: x mit $D_x = range(a_0 a_1 \ldots a_{n-1})$.
Neue Eingabe: a_n.
Berechnung: Es werden y mit $D_y = D_x \cup \{a_n\}$ sowie der erste String σ_y mit $range(\sigma_y) = D_y$ und $e_y = M(\sigma_y)$ berechnet.
Ausgabe: e_y.
Neues Langzeitgedächtnis: y.

Es ist leicht zu sehen, dass dieses Lernverfahren einerseits die Platzschranke einhält und andererseits das selbe Ein-Ausgabe-Verhalten hat wie M. ∎

Satz 10.6 *Es gibt eine Klasse S die im Limes aber nicht set-driven gelernt werden kann. Insbesondere ist sie mit keiner Platzschranke lernbar.*

Beweis Es gibt eine partiell-rekursive $\{0,1\}$-wertige Funktion ψ, die keine total-rekursive Fortsetzung hat. Nun enthalte S die Menge $H = \{\langle x, y\rangle : \psi(x)\!\downarrow = y\}$ sowie jede endliche Menge F die ein Element $\langle x, y\rangle$ mit $\psi(x)\!\downarrow \neq y$ enthält.

S kann im Limes gelernt werden: Man gibt zuerst einen Index für H aus und sobald man bemerkt, dass ein $\langle x, y\rangle \in range(\sigma)$ auch in der aufzählbaren Menge $\{\langle x, y\rangle : \psi(x)\!\downarrow \neq y\}$ ist, macht man einen Hypothesenwechsel zu $range(\sigma)$ und gibt auch weiterhin einen Index für die jeweilige Menge $range(\sigma)$ aus.

Wenn nun S von M set-driven gelernt wird, dann hat ψ (entgegen der Voraussetzung) eine berechenbare und totale Fortsetzung. σ sei eine Locking-Sequenz für H. Nun definiert man:

$$F(x) = \begin{cases} 1 & \text{falls } M(\sigma\langle x, 1\rangle) = M(\sigma); \\ 0 & \text{falls } M(\sigma\langle x, 1\rangle) \neq M(\sigma). \end{cases}$$

$M(\sigma\langle x, 1\rangle)$ ist für jedes x definiert, da die endliche Menge $range(\sigma\langle x, 1\rangle)$ eine Obermenge in S hat, z.B. die Menge $range(\sigma) \cup \{\langle x, 1\rangle, \langle 0, 0\rangle, \langle 0, 1\rangle\}$. Also ist F berechenbar und total. Wenn $\psi(x)\!\downarrow = 1$ ist, dann ist $\sigma\langle x, 1\rangle \in H^*$ und $M(\sigma) = M(\sigma\langle x, 1\rangle)$ weil σ eine Locking-Sequenz ist. Insbesondere ist $F(x) = 1$. Wenn $\psi(x)\!\downarrow = 0$ ist, dann ist die endliche Menge $range(\sigma\langle x, 1\rangle)$ in S und weil M set-driven ist, muss M auf dieser Eingabe einen Index für diese Menge ausgeben. Also macht M einen Hypothesenwechsel und es ist $F(x) = 0$. Entgegen zur Wahl von ψ setzt F diese Funktion fort und folglich kann es so ein N nicht geben. ∎

Ein weiterer Zusammenhang besteht zwischen dem zur Verfügung stehenden Langzeitgedächtnis und der Anzahl der Hypothesen, die während einem Lernvorgang ausgegeben werden können. Der folgende Satz ist allerdings nicht umkehrbar, da die Klasse aller zweielementigen Mengen ohne Hypothesenwechsel gelernt werden kann, aber kein Lernverfahren für diese Klasse kommt mit konstantem Langzeitspeicher aus.

Satz 10.7 *Wenn eine Klasse L mit einer konstanten Platzschranke c gelernt werden kann, dann kommt ein anderer Lerner beim Lernen mit insgesamt $2c+1$ Hypothesen aus.*

Übung 10.8 Zeigen Sie, dass die Sprache $S_{0\ldots 2c} = \{\{0, 1, \ldots, b\} : b \leq 2c\}$ mit Platzbeschränkung konstant c gelernt werden kann. Sie können dazu folgenden Algorithmus verwenden: Das Langzeitgedächtnis ist stets $\lceil \frac{a}{2} \rceil$ für das grösste bisher gesehene a. Bei Eingabe b und bisherigem Speicher d wird genau dann einen Index der Menge $\{0, 1, \ldots, b\}$ ausgegeben, wenn $b \geq 2d$ ist. Ansonsten wird "?" ausgegeben. Zeigen Sie auch mit Hilfe von Satz 10.7, dass $S_{0\ldots 2c+1} = \{\{0, 1, \ldots, b\} : b \leq 2c+1\}$ nicht mehr mit Platzschranke c gelernt werden kann.

11 Lernen von gestörten Daten

Gestörte Daten sind in der Wirklichkeit häufiger als in der Theorie. Eine lernende Maschine muss dann während des Lernvorganges die Störungen erkennen und ausgleichen. Dies führt im Allgemeinen zu einer Einschränkung der Lernfähigkeit, deshalb werden hier auch nur die Fälle des Lernens im Limes und des verhaltenskorrekten Lernens behandelt.

Baliga, Fulk, Jain und Sharma [10, 25, 33] haben das Konzept des Lernen von gestörten Daten in das Gebiet der induktiven Inferenz übertragen und verschiedene Modellierungen definiert. Die hier verwendete Modellierung geht davon aus, dass falsche Daten nur endlich oft auftauchen während korrekte unendlich oft anliegen. Die hier gewählte Modellierung hat den Vorteil, dass schon an Hand der Eingabe jeder Funktionswert eindeutig definiert ist und die Störungen es nur schwierig machen, diesen Wert zu finden. Dadurch fügt sich das Modell recht gut in andere Konzepte der Induktiven Inferenz ein.

Definition 11.1 Eine Folge $(x_0, y_0)(x_1, y_1)\ldots$ heisst Noisy Informant zu einer Funktion f wenn

- korrekte Daten unendlich oft auftauchen,
  d.h. für jedes x gibt es unendlich viele Paare (x_n, y_n) mit $x_n = x$ und $y_n = f(x)$,
- und inkorrekte Daten nur endlich oft auftauchen,
  d.h. für jedes x gibt es nur endlich viele Paare (x_n, y_n) mit $x_n = x$ und $y_n \neq f(x)$.

Eine Klasse S ist in NoisyEx, wenn es eine berechenbare Maschine M gibt, die jede Funktion $f \in S$ im Limes von jedem Noisy Informant lernt.

Beispiel 11.2 $\{f : f \text{ ist ein Polynom und } \mathrm{Grad}(f) \leq 2\} \in \mathrm{NoisyBC}$. Zuerst das Beispiel eines Informant

$$(0,0),\ (1,1),\ (2,4),\ (3,9),\ \ldots$$

und eines Noisy Informant

$$\begin{array}{l}
(0,5),\ (0,2),\ (0,0),\ (0,0),\ (0,0),\ (0,0),\ \ldots \\
(1,5),\ (1,2),\ (1,0),\ (1,1),\ (1,1),\ (1,1),\ \ldots \\
(2,2),\ (2,2),\ (2,2),\ (2,2),\ (2,4),\ (2,4),\ \ldots \\
(3,9),\ (3,9),\ (3,6),\ (3,9),\ (3,9),\ (3,9),\ \ldots \\
\ldots
\end{array}$$

für die Funktion f wobei der Noisy Informant als Tabelle dargestellt ist. Der folgende Algorithmus lernt die Klasse von Noisy Informant:

Initialisiere $c_a = 0$ für $a = 0, 1, 2$; c_a ist jeweils eine Hypothese über den Wert $f(a)$.
In jedem Schritt:

Lies nächstes Paar (a, b);
Falls $a \leq 2$ setze $c_a = b$;
Hypothese $f(x) = c_0 + \frac{-c_2+4c_1-3c_0}{2}\, x + \frac{c_2-2c_1+c_0}{2}\, x^2$.

Ab irgendeinem Zeitpunkt tauchen nur noch Paare $(0, f(0))$, $(1, f(1))$, $(2, f(2))$ und (x, y) mit $x > 2$ auf. Von da an haben c_0, c_1 und c_2 die richtigen Werte und von da an ist die Ausgabe korrekt.

Während man die Polynome eines festen Grades noch von gestörten Daten lernen kann, so gilt dass nicht mehr für die Klasse S_{pol} aller Polynome. Die Klasse S_{pol} kann nicht einmal unter dem Kriterium NoisyBC* gelernt werden obwohl man ganz REC unter BC* lernen kann wie in Satz 3.2 gezeigt wurde.

Satz 11.3 $S_{pol} \notin$ NoisyBC*.

Beweis Zu einem endlichem String $a_0 a_1 \ldots a_n$ beinhalte die Menge $H(b_0 b_1 \ldots b_n)$ alle jene Strings der Form $(x_0, y_0)\,(x_1, y_1)\,\ldots\,(x_m, y_m)$ in welchen für $a = 0, 1, \ldots, n$ das Paar (a, b_a) vorkommt, d.h. $(a, b_a) = (x_k, y_k)$ für ein $k \leq m$ aber kein Paar (a, c) mit $c \neq b_a$ vorkommt.

Angenommen, ein NoisyBC*-Lerner lernt jedes der Polynome $p_n(x) = x \cdot (x-1) \cdot \ldots \cdot (x-n)$. Dann konstruiert man induktiv beginnend mit $\sigma_0 = \lambda$ folgenden Noisy Informant $T = \lim_n \sigma_n$ für das Polynom $q(x) = 0$:

Zu gegebenem σ_n gibt es $\tau_n \in H(p_n(0)p_n(1)\ldots p_n(n))$ mit $\varphi_{M(\sigma_n\tau_n)} =^* p_n$: Dazu betrachtet man die Folge $T_n = (u_0, p_n(u_0))\,(u_1, p_n(u_1))\,\ldots$, wobei es zu jedem m unendlich viele k mit $m = u_k$ gibt. Die Folge σT_n ist ein Noisy Informant für p_n und M muss auf diesem Noisy Informant im Sinne BC* gegen p_n konvergieren. Wenn $\sigma\tau_n$ ein hinreichend langer Präfix von σT_n ist, dann sind alle Paare $(0, p_n(0)), (1, p_n(1)), \ldots, (n, p_n(n))$ in τ_n aufgetaucht und $M(\sigma\tau_n)$ ist ein Index, der bis auf endlich viele Fehler p_n berechnet. Also gibt es das gesuchte τ_n und man setzt $\sigma_{n+1} = \sigma_n\tau_n$.

M konvergiert auf der Folge T nicht gegen q, denn für jedes n gilt $\varphi_{M(\sigma_{n+1})} \neq^* q$. Andererseits ist T ein Noisy Informant für q. Für jedes x und $n \geq x$ ist $p_n(x) = 0 = q(x)$. Also kommt $(x, q(x))$ in jedem String τ_n mit $n \geq x$ mindestens einmal und in der Gesamtfolge T unendlich oft vor. Andererseits kommt ein Paar (x, y) mit $y > x$ in keinem τ_n mit $n \geq x$ vor und daher kann nur der endliche Präfix σ_x Paare (x, y) mit $y \neq q(x)$ enthalten. Also erkennt M das Polynom q nicht unter dem Kriterium BC* und es gibt keine Maschine, die ganz S_{pol} gemäss dem Kriterium NoisyBC* lernt. ∎

Somit ist dieses Modell der Störungen recht restriktiv. S_{pol} kann allerdings auch nicht endlich gelernt werden und aus der Sicht des folgenden Satzes, der einen Zusammenhang zwischem endlichem Lernen und dem Lernen von gestörten Daten herstellt, ist das Ergebnis naheliegend.

Satz 11.4 [52] NoisyEx = NoisyBC = Fin[K].

Beweis Die Implikation NoisyEx $\rightarrow$ NoisyBC folgt aus der Definition, die beiden anderen müssen explizit gezeigt werden.

NoisyBC $\rightarrow$ Fin[K]: Es sei M eine Maschine, die die Klasse S unter dem Kriterium NoisyBC lernt. Unter Zuhilfenahme der Menge H aus Satz 11.3 definiert man zu M folgenden Fin[K]-Lerner N:

> Wenn es zu $b_0b_1 \dots b_n$ einen String $\sigma = (x_0, y_0)(x_1, y_1) \dots (x_m, y_m)$ gibt so dass
>
> - $m + x_1 + y_1 + \dots + x_m + y_m \leq n$ und
> - $\varphi_{M(\sigma\tau)}(x)\downarrow \neq \varphi_{M(\sigma\eta)}(x)\downarrow$ für kein x und keine $\tau, \eta \in H(b_0b_1 \dots b_n)$,
>
> dann macht der Lerner die Hypothese $e(b_0b_1 \dots b_n)$ mit
>
> $$\varphi_{e(b_0b_1\dots b_n)}(x) = y \Leftrightarrow (\exists \tau \in H(b_0b_1 \dots b_n))\, [\varphi_{M(\sigma\tau)}(x)\downarrow = y].$$
>
> Sonst verzichtet der Lerner auf eine Hypothese und gibt ? aus. Nach der ersten Ausgabe eines solchen Index e erfolgt kein Hypothesenwechsel.

D.h. der Lerner wartet so lange bis das jeweilige $\varphi_{e(b_0b_1\dots b_n)}$ eindeutig definiert ist (was mit K-Orakel überprüft werden kann) und gibt dann diesen Index $e(b_0b_1 \dots b_n)$ aus. Zu zeigen bleibt, dass das Lernverfahren terminiert (d.h. irgendwann mal eine solche Ausgabe $e(b_0b_1 \dots b_n)$ tätigt) und das diese dann korrekt ist.

Angenommen, das erste wäre nicht der Fall. Dann kann man wie bei Satz 11.3 induktiv eine Folge $\sigma_0, \sigma_1, \dots$ mittels $\sigma_0 = \lambda$ und $\sigma_{m+1} = \sigma_m\tau_m$ definieren wobei

- $\tau_m \in H(f(0)f(1) \dots f(m))$;
- $\varphi_{M(\sigma_m\tau_m)}(x)\downarrow \neq f(x)$ für irgendein x.

Diese Folge existiert, da es zu σ_m eine hinreichend lange Eingabe $f(0)f(1) \dots f(n)$ gibt so dass $n \geq m$ und $\varphi_{M(\sigma\tau)}(x)\downarrow \neq \varphi_{M(\sigma\eta)}(x)\downarrow$ für ein x und $\tau, \eta \in H(f(0)f(1) \dots f(n))$; einer dieser beiden Werte weicht von $f(x)$ ab, oBdA der von τ. Dann ist $\tau_m = \tau$ die gesuchte Erweiterung: $\tau_m \in H(f(0)f(1) \dots f(m))$, da $m \leq n$ und $\varphi_{M(\sigma_m\tau_m)}(x)\downarrow \neq f(x)$ nach Auswahl von τ. Nun betrachte man $T = \lim_m \sigma_m = \tau_0\tau_1 \dots$. In T kommt jedes Paar $(x, f(x))$ unendlich oft vor, da $(x, f(x))$ in jedem τ_m mit $m \geq x$ vorkommt. Andererseits kommt jedes Paar (x, y) mit $y \neq f(x)$ nur in σ_x vor, danach aber nicht mehr. Hieraus folgt, dass T ein Noisy Informant für f ist und aus diesem Grunde muss

M auf T im Sinne BC konvergieren. Dies steht aber im Widerspruch zur Konstruktion von T welche $\varphi_{M(\sigma_m)} \neq f$ für jedes m garantiert. Aus diesem Widerspruch ergibt sich, dass N beim Lernen jeder Funktion $f \in S$ irgendwann einen Index $e(f(0)f(1)\dots f(n))$ ausgibt.

Nun bleibt zu zeigen, dass $f = \varphi_{e(f(0)f(1)\dots f(n))}$ ist. Hierzu betrachtet man noch einmal das zugehörige σ aus dem Lernalgorithmus. Es gibt einen Noisy Informant $T \succ \sigma$, dessen gestörte Daten (x, y) mit $y \neq f(x)$ nur im Anfangsstück σ vorkommen. Also konvergiert M auf T verhaltenskorrekt gegen f und es gibt τ mit $\sigma\tau \preceq T$ und $\varphi_{M(\sigma\tau)} = f$. Dieses τ enthält nur Paare der Form $(x, f(x))$ und ist daher in $H(f(0)f(1)\dots f(n))$. Daraus folgt, dass es zu jedem x ein $\tau \in H(f(0)f(1)\dots f(n))$ gibt mit $\varphi_{M(\sigma\tau)}(x)\downarrow = f(x)$. Da die Definition von $\varphi_{e(f(0)f(1)\dots f(n))}$ widerspruchsfrei ist, ist dann auch $\varphi_{e(f(0)f(1)\dots f(n))}(x) = f(x)$ für alle x.

Fin$[K]$ $\rightarrow$ NoisyEx: M sei eine K-rekursive Maschine. Sie hat eine Approximation M_s, d.h. $\lim_s M_s(\sigma) = M(\sigma)$ für alle σ. Diese Approximation habe zusätzlich die Eigenschaft

$$M_s(\sigma) \neq ? \Rightarrow (\forall \tau \succeq \sigma)\,[M_s(\tau) = M_s(\sigma)]$$

und gleiches gelte natürlich auch für M selbst – die Maschine macht nach der ersten Hypothese prinzipiell und unabhängig von der weiteren Eingabe nur noch die gleiche Ausgabe. Zu einem String $\tau = (x_0, y_0)(x_1, y_1)\dots(x_n, y_n)$ definiert man

$$\mathit{Last}(x, \tau) = \begin{cases} y & \text{falls es } m \text{ gibt mit } x = x_m,\ y = y_m \\ & \text{und } x_k \neq x \text{ für } k = m+1, m+2, \dots, n, \\ 0 & \text{sonst, d.h. } x \neq x_m \text{ für alle } m. \end{cases}$$

Die Funktion *Last* gibt also den letzten Wert y_m eines Paares (x_m, y_m) mit $x_m = x$ an sofern ein solches existiert. Da in einem Noisy Informant T fast alle solche Paare den richtingen Wert $f(x)$ angeben, ist $\mathit{Last}(x, \tau)$ für fast alle $\tau \preceq T$ gleich $f(x)$. Mit Hilfe von *Last* wird nun der neue Lerner N definiert:

$$N(\tau) = M_{|\tau|}(\mathit{Last}(0, \tau)\mathit{Last}(1, \tau)\dots\mathit{Last}(|\tau|, \tau)).$$

Wenn M die Funktion f endlich lernt, dann gibt es ein $\sigma \preceq f$ so dass $M(\sigma)$ einen Index e für f ausgibt. Nun gilt für jeden Noisy Informant T von f und alle hinreichend grossen $\tau \preceq T$:

- $\mathit{Last}(\tau, x) = \sigma(x)$ für alle $x \in \mathit{dom}(\sigma)$;
- $|\sigma| < |\tau|$;
- $M_{|\tau|}(\sigma) = e$.

Es ist dann auch $M_{|\tau|}(\mathit{Last}(0, \tau)\mathit{Last}(1, \tau)\dots\mathit{Last}(|\tau|, \tau)) = e$ da σ ein Präfix des Vektors $\mathit{Last}(0, \tau)\mathit{Last}(1, \tau)\dots\mathit{Last}(|\tau|, \tau)$ ist. Also konvergiert N auf jedem Noisy Informant T gegen den selben Index e wie M auf f. ∎

Die Klasse Fin$[K]$ ist etwas klein, z.B. kann die Klasse S_{pol} aller Polynome nicht unter diesem Kriterium gelernt werden. Daher ist es durchaus überlegenswert, beim Lernen von gestörten Daten allgemeinere Kriterien anzulegen als BC. Eines dieser Kriterien ist PId (partial Identification). Hier wird nicht mehr Konvergenz verlangt, sondern nur noch, dass genau ein Index e in der Ausgabe unendlich oft vorkommt und dass dieser dann die gesuchte Funktion berechnet. Gemäss diesem sehr allgemeine Kriterium können alle berechenbaren Funktionen gelernt werden und der folgende Satz 11.5 zeigt, dass dies sogar bei gestörten Daten möglich ist.

Satz 11.5 REC $\in$ NoisyPId.

Beweis Hierzu betrachtet man erst mal eine besondere Numerierung der berechenbaren Funktionen. Es gibt eine berechenbare Index-Funktion s so dass es zu jeder berechenbaren Funktion f genau ein e mit $\varphi_{s(e)} = f$ gibt (allerdings sind einige der $\varphi_{s(e)}$ an einigen Stellen undefiniert). Eine solche Numerierung heisst Friedbergnumerierung. Der Algorithmus lässt sich am einfachsten dadurch beschreiben, wie oft ein Index $s(e)$ auf einem Noisy Informant T ausgegeben wird. Mit Hilfe einer Warteschlange sowie des Symbols "?" für keine Ausgabe können dann die einzelnen Ausgaben auf die verschiedenen Anfangsabschnitte $\sigma \preceq T$ verteilt werden. Nun zum Algorithmus:

> Ein Index $s(e)$ wird beim Noisy Informant T genau dann mindestens n mal ausgegeben, wenn für jedes $x \leq n$ der Term $\varphi_{s(e)}(x)$ definiert ist und $(x, \varphi_{s(e)}(x))$ mindestens n mal in T vorkommt.

Man kann hieraus ableiten, dass $s(e)$ genau dann unendlich oft ausgegeben wird wenn für jedes x sowohl $\varphi_{s(e)}(x)$ definiert ist als auch $(x, \varphi_{s(e)}(x))$ unendlich oft in T vorkommt. Somit wird $s(e)$ genau dann unendlich oft ausgegeben wenn $\varphi_{s(e)} = f$ ist. Durch die Eigenschaft der Numerierung hat f nur einen Index, also lernt der Algorithmus ganz REC unter dem Kriterium PId. ∎

Übung 11.6 Welche der Klassen $S_0 = \{f : f = \varphi_{f(0)}\}$, $S_1 = \{f : (\forall^\infty x)\,[f(x) = 0]\}$ und $S_2 = \{f : f = \varphi_{\max(\{f(y):y \text{ ist gerade}\})}\}$ ist in NoisyEx und in NoisyEx*? Gilt FinEx$^* \to$ NoisyEx*?

Literaturverzeichnis

[1] **Adleman und Blum (1991):** Inductive Inference and Unsolvability. Journal of Symbolic Logic 56:891–900.

[2] **Ambainis (1996):** Probabilistic and Team PFIN-type Lerning: General Properties. Manuscript.

[3] **Angluin (1980):** Inductive Inference of Formal Languages from Positive Data. Information and Control 45:117–135.

[4] **Angluin (1987):** Learning Regular Sets From Queries and Counterexamples. Information and Computation 75:87–106.

[5] **Angluin (1990):** Negative Results for Equivalence Queries. Machine Learning 5:121–150.

[6] **Angluin und Smith (1983):** Inductive Inference: Theory and Methods. Computing Surveys 15(3):237–269.

[7] **Angluin und Smith (1987):** Inductive Inference. Encyclopedia of Artificial Intelligence, volume 1, Wiley-Intersicence Publication New York.

[8] **Balcázar, Díaz, Gavaldà und Watanabe (1994):** The Query complexity of Learning DFA. New Generation Computing 12:337–358.

[9] **Beigel, Gasarch, Gill und Owings (1993):** Terse, superterse and verbose sets. Information and Computation 103:68–85.

[10] **Baliga, Jain und Sharma (1992):** Learning from Multiple Sources of Inaccurate Data. Tagungsband des AII-Workshop 108–128.

[11] **Barzdins (1974):** Two Theorems on the Limiting Synthesis of Functions. Theory of Algorithm and Programs I:82–88. Herausgegeben von Barzdins, Latvian State University.

[12] **Blum und Blum (1975):** Towards a Mathematical Theory of Inductive Inference. Information and Control 28:125–155.

[13] **Blumer, Ehrenfeucht, Haussler und Warmuth (1989):** Learnability and Vapnik-Chervonenkis Dimension. Journal of the Association for Computing Machinery 36:929–965.

[14] **Case und Smith (1983):** Comparison of Identification Criteria for Machine Inductive Inference. Theoretical Computer Science 25(2):95–104.

[15] **Daley und Kalyasundaram (1993):** Use of Reduction Arguments in Determinating Popperian FIN-Type Learning Capacibilities. Tagungsband Vierter ALT-Workshop 173–186.

[16] **Daley, Kalyanasundaram und Velauthapillai (1992):** Breaking the Probability 1/2 Barrier in FIN-Type Learning. Tagungsband Fünfte COLT-Konferenz 203–217.

[17] **Daley, Kalyanasundaram und Velauthapillai (1992):** The Power of Probabilism in Popperian Finite Learning. Tagungsband Dritter AII-Workshop 151–169.

[18] **Daley, Pitt, Velauthapillai und Will (1991):** Relations Between Probabilistic and Team One-Shot Learners. Tagungsband Vierte COLT-Konferenz 228–239.

[19] **Ehrenfeucht, Haussler, Kearns und Valiant (1989):** A general lower bound on the number of examples needed for Learning. Information and Computation 82:247–261.

[20] **Fenner (1991):** Notions of Resource-Bounded Category and Genericity. Tagungsband Sechste Konferenz Structure in Complexity Theory 196–212.

[21] **Fortnow, Gasarch, Jain, Kinber, Kummer, Kurtz, Pleszkoch, Slaman, Solovay und Stephan (1994):** Extremes in the Degrees of Inferability. Annals of Pure and Applied Logic 66:231–276.

[22] **Freivalds (1975):** Functions computable in the limit by probalistic machines. Lecture Notes to Computer Science 28:77-87.

[23] **Freivalds (1979):** Finite Identification of General Recursive Functions by Probabilistic Strategies. Tagungsband Fundamentals of Computation Theory 1979 (FCT) 138–145.

[24] **Freivalds, Kinber und Smith (1993):** On the Impact of Forgetting on Learning Machines. Tagungsband Sechste COLT-Konferenz 165–174.

[25] **Fulk und Jain (1989):** Learning in Presence of Inaccurate Information. Tagungsband Zweite COLT-Konferenz 175–188.

[26] **Gasarch und Pleszkoch (1989):** Learning via Queries to an Oracle. Tagungsband Zweite COLT-Konferenz 175–188.

[27] **Gasarch, Pleszkoch und Solovay (1992):** Learning via Queries in $[+,<]$. Journal of Symbolic Logic 57(1):53–81.

[28] **Gasarch und Smith (1992):** Learning via Queries. Journal of the ACM 39(3) 649–675.

[29] **Gasarch und Velauthapillai (1993):** Asking Questions Versus Verifiability. Tagungsband Dritter AII-Workshop 197–213.

[30] **Gold (1967):** Language Identification in the Limit. Information and Control 10:447–474.

[31] **Ibarra und Jiang (1988):** Learning Regular Languages from Counterexamples. Tagungsband Erste COLT-Konferenz 371–385.

[32] **Jantke (1991):** Monotonic and Non-Monotonic Inductive Inference. New Generation Computing 8:349–360.

[33] **Jain (1994):** Program Synthesis in Presence of Infinite Number of Inaccuracies. Tagungsband Fünfter ALT-Workshop 333-348.

[34] **Jain und Sharma (1990):** Finite Learning by a Team. Tagungsband Dritte COLT-Konferenz 163–177.

[35] **Kapur (1992):** Monotonic Language Learning. Tagungsband Dritter ALT-Workshop 147–158.

[36] **Kapur (1994):** Language Learning under Various Types of Constraint Combinations. Tagungsband Fünfter ALT-Workshop 365–378.

[37] **Kinber (1994):** Monotonicity versus Efficiency for Learning Languages from Texts. Tagungsband Fünfter ALT-Workshop 395–406.

[38] **Kinber und Stephan (1995):** Language Learning from Text: Mind Changes, Limited Memory and Monotonicity. Information and Computation 123:224–241.

[39] **Kummer (1994):** The Strength of Noninclusions for Teams of Finite Learners. Tagungsband Siebte COLT-Konferenz 268–277.

[40] **Kummer und Stephan (1993):** On the Structure of Degrees of Inferability. Tagungsband Sechste COLT-Konferenz 117–126.

[41] **Lange und Zeugmann (1992):** Types of Monotonic Language Learning and Their Characterization. Tagungsband Fünfte COLT-Konferenz 377–390.

[42] **Lange und Zeugmann (1993):** Monotonic Versus Non-Monotonic Language Learning. Tagungsband Zweiter Workshop on Nonmonotonic and Inductive Logic 254–269.

[43] **Lange und Zeugmann (1993):** Language Learning in Dependence on the Space of Hypotheses. Tagungsband Sechste COLT-Konferenz 127–136.

[44] **Odifreddi (1989):** Classical Recursion Theory I. North-Holland.

[45] **Odifreddi:** Classical Recursion Theory II. Erscheint bei North-Holland.

[46] **Osherson, Stob und Weinstein (1986):** Systems that Learn. Bradford – The MIT Press, Cambrigde, Massachusetts.

[47] **Pitt und Smith (1988):** Probability and Plurality for Aggregations of Learning Machines. Information and Computation 77:77–92.

[48] **Pitt und Valiant (1988):** Computational Limitations on Learning from Examples. Journal of the ACM 35:965–984.

[49] **Slaman und Solovay (1991):** When Oracles Do Not Help. Tagungsband Vierte COLT-Konferenz 379–383.

[50] **Smith (1982):** The Power of Pluralism for Automatic Program Synthesis. Journal of the ACM 29(4)1144–1165.

[51] **Smith (1994):** Three Decades of Team Learning. Tagungsband Fünfter ALT-Workshop 211–228.

[52] **Stephan (1995):** Learning via Queries and Oracles. Annals of Pure and Applied Logic, 94:273–296.

[53] **Valiant (1984):** A Theory of the Learnable. Communications of the ACM 27(11):1134–1142.

[54] **Velauthapillai (1989):** Inductive Inference With a Bounded Number of Mind Changes. Tagungsband Dritte COLT-Konferenz 200–213.

[55] **Wiehagen (1990):** A Thesis in Inductive Inference. Tagungsband Erster Workshop on Nonmonotonic and Inductive Logic 184–207.

[56] **Zeugmann (1993):** Algorithmisches Lernen von Funktionen und Sprachen. Habilitationsschrift an der Technischen Hochschule Darmstadt.

[57] **Zeugmann, Lange und Kapur (1996):** Monotonic and Dual Monotonic Language Learning. Theoretical Computer Science 155:365–410