Komplexität in natürlichen Sprachen

Timm Lichte& Christian Wurm

July 17, 2017

Abstract

Alles noch sehr vorlaüfig; Kommentare willkommen!

Contents

1 Die Chomsky Hierarchie und Grundbegriffe formaler Sprachen

- Reguläre Sprachen, endliche Automaten, reguläre Grammatiken - Kontextfreie Sprachen, CFG, Kellerautomaten - Kontextsensitive Sprachen, CSG - Typ 0 Sprachen, TM, Typ 0 Grammatiken - Abschlusseigenschaften: Vereinigung, Schnitt mit Reg, Homomorphismus

2 Subreguläre Sprachen

Wenn wir phonotaktische Beschränkungen betrachten, fällt auf dass diese meist sehr einfach sind. Z.B. im Deutschen gilt: innerhalb einer Silbe kann es keine Abfolge

(1) ltV, V ein beliebiger Vokal

geben. Etwas allgemeiner spricht man von der sog. **Sonoritätshierarchie**: wir teilen Konsonanten in Klassen ein, die nach Sonorität linear geordnet sind. Z.B.

(2) {s,t,f,p} < {z,d,w,b} < {l,r}

Jede Silbe hat die Form

(3) $\mathbf{C}_1\mathrm{V}\mathbf{C}_2$

wobei $\mathbf{C}_1$, $\mathbf{C}_2$ *Konsonantencluster* sind. Phonotaktische Regeln besagen:

(4) a. Falls $\mathbf{C}_1$=$C_1^1...C_1^i$, dann ist $C_1^1 \leq ... \leq C_1^i$
b. Falls $\mathbf{C}_2$=$C_1^1...C_1^i$, dann ist $C_1^1 \geq ... \geq C_1^i$

Um eine solche Beschränkung auszudrücken, reicht es Beschränkungen zwischen adjazenten Buchstaben zu haben (eigentlich nicht: das gilt nur, weil die Länge von $\mathbf{C}_1$,$\mathbf{C}_2$ beschränkt ist). Hier mit regulären Sprachen zu arbeiten, wäre bereits übertrieben. Stattdessen kann man in diesem Fall streng lokale Sprachen nutzen.

Wir legen ein Alphabet Σ fest. Eine Sprache L ist streng n-lokal (in SL_k), falls gilt: es gibt $I \subseteq \Sigma^{\leq n-1}$, $F \subseteq \Sigma^{\leq n-1}$, $B \subseteq \Sigma^{\leq n}$, so dass gilt:

(5) $L = I\Sigma^* \cap \Sigma^* F \cap \overline{\Sigma^* B \Sigma^*}$

Intuitiv gilt: I sind die **initialen** Buchstaben, F die **finalen**, und B sind die **Blöcke**, die verboten sind. Wir setzen

(6) $SL := \bigcup_{n\in\mathbb{N}} SL_n$

Frage : warum ist die oben beschrieben Beschränkung nicht streng lokal? Unter welchen Bedingungen wird sie es?

Alternativ kann man streng lokale Sprachen auch mit Blockgrammatiken generieren. Eine solche Blockgrammatik ist einfach eine Menge

(7) $M \subseteq \Sigma^n \cup \rtimes\Sigma^{\leq n-1} \cup \Sigma^{\leq n-1}\ltimes$

wobei in der erzeugten streng lokalen Sprache (Parameter n) alle Faktoren der Länge n durch die Grammatik/Menge **lizensiert** sein müssen. $\rtimes$ steht hierbei für den Wortanfang, $\ltimes$ für das Wortende.

Intuitiv gilt in lokalen Sprachen: alle Beschränkungen gelten nur in einem konstant beschränkten Umkreis. Es gibt zwei wichtige Ergebnis zu lokalen Sprachen: Suffix-Substitution und der Satz von Chomsky/Schützenberger:

In manchen Sprachen gibt es phonotaktiche Beschränkungen, die über beliebig viele Segmente hinweg wirksam sind; z.B. falls ein Segment z sonor ist, muss jeder folgende palatale frikativ im Wort sonor sein. Diese Bedingung ist nicht mehr streng lokal. Dafür gibt es eine leicht stärkere Klasse, nämlich die schichtweisen streng lokalen Sprachen. Ein **Homorphismus** ist eine Abbildung $h : \Sigma^* \to T^*$, so dass

$$h(aw) = h(a)h(w) \text{ für alle } a \in \Sigma,\ w \in \Sigma^*$$

Ein **eliminierender** Homomorphismus ist ein Homomorphismus so dass entweder $h(a) = a$ oder $h(a) = \epsilon$. Jeder eliminerende Homomorphismus wird bestimmt durch eine Menge $T \subseteq \Sigma$, so dass wir definieren können:

$$E_T(a) = \begin{cases} a \text{ falls } a \in T \\ \epsilon \text{ andernfalls.} \end{cases}$$

Wir sagen nun: eine Sprache $L \subseteq \Sigma^*$ ist schichtweise streng lokal (SSL), falls gilt:

$$\text{es gibt ein } T \subseteq \Sigma,\ L' \subseteq T^*, \text{ so dass } L' \in SL \text{ und } L = (E_T)^{-1}(L')$$

Wir haben also Beschränkungen, die nur gewisse Buchstaben betreffen, alle anderen Buchstaben sind komplett frei. Damit können wir gewisse nicht-lokale Abhängigkeiten modellieren. Es ist klar dass $SL \subseteq SSL$: man setze einfach $T = \Sigma$.

Was etwas unpraktisch ist an dieser Definition ist folgendes: nehmen wir an, $\Sigma = \{a, b, c, d\}$. Nehmen wir an, wir möchten eine nichtlokale Abhängigkeit zwischen a und b modellieren, z.B.:

(8) Zwischen jeder Abfolge zweier as steht ein b.

Das ist lokal für $\Sigma = \{a, b\}$, also schichtweise lokal für $\Sigma = \{a, b, c, d\}$. Aber: wir können dann keine Beschränkungen mehr über c, d ausdrücken – das ist sehr unpraktisch und in der Praxis unrealistisch, es bleiben ja die lokalen phonotaktischen Beschränkungen. Deswegen ist die interessante Klasse die multi-SSL (MSSL), definiert als:

(9) $L \in MSSL$, falls $L = L_1 \cap ... \cap L_i$, wobei $L_1, ..., L_i \in SSL$.

Wir haben also einen endlichen Schnitt von SSL-Sprachen; in jeder dieser Sprachen können wir nicht-lokale Abhängigkeiten einzelner Buchstabengruppen beschreiben; am Ende müssen alle erfüllt werden. $MSSL$ ist ein sehr sinnvolles Modell phonotaktischer Beschränkungen.

Eine wichtige Frage noch: warum all die Mühe mit subregulären Sprachen, wo doch reguläre Sprachen bereits sehr viele positive Eigenschaften haben? Hier gibt es v.a. einen Grund, nämlich **Lernbarkeit**. Bereits die regulären Sprachen sind nicht mehr lernbar in einem vernünftigen Sinne, d.h. sie können nicht nach endlich vielen Schritten von einem Lerner identifiziert werden. Für viele subregulären Sprachen gilt das aber (dazu später mehr).

3 Rekursiv und rekursiv aufzählbar

Eine Sprache L ist rekursiv aufzählbar, wenn sie die Menge aller Worte ist, bei denen die TM in einem akzeptierenden Zustand hält. Eine Sprache L ist rekursiv, falls sie selbst und ihr Komplement rekursiv aufzählbar sind. D.h. also soviel: L ist rekursiv, wenn es eine TM gibt, die für jedes Wort $w \in \Sigma^*$ entscheidet, ob $w \in L$ oder $w \notin L$.

Eine wichtige Frage ist: ist $RA = R$? Das hat verschiedene äquivalente Formulierungen, nämlich:

- gibt es Sprachen, deren Komplement nicht rekursiv aufzählbar ist?
- also gibt es Sprachen, die nicht rekursiv sind?
- gibt es Sprachen, die r.a. sind, aber für die es keine TM gibt, die bei jeder Eingabe anhält?

Eine weitere wichtige Frage, die wiederum äquivalent ist, ist: ist das Halteproblem, also die Frage, ob eine TM T mit einer Eingabe W irgendwann anhält, entscheidbar?

Die Antwort auf die letzte Frage lautet nein, und als Folge davon können wir 1.-3. positive beantworten. So kommen wir also zu folgender Definition: eine Sprache $L \subseteq \Sigma^*$ ist rekursiv aufzählbar, falls es eine TM T gibt, so dass $L = \{w : T \text{ akzeptiert } w\}$. Eine Sprache L ist **rekursiv**, falls L rekursiv aufzählbar ist, und $\Sigma^* - L$ rekursiv aufzählbar ist. Rekursiv bedeutet in diesem Kontext also so viel wie **entscheidbar**: wir können für jede Eingabe entscheiden, ob sie zur Sprache gehört oder nicht (denn für alle Eingaben w gilt: entweder $w \in L$ oder $w \in \Sigma^* - L$).

4 Lernbarkeit

Wir besprechen hier Lernbarkeit im klassischen Sinne von Gold. Es gibt noch einige andere Begriffe von Lernbarkeit, die aber zu weit abführen würden. Wenn wir also hier von Lernbarkeit sprechen, dann meinen wir: Lernbarkeit im Sinne von Gold. Golds Begriff von Lernbarkeit basiert auf dem Konzept der

Identifizierbarkeit im Unendlichen

man sagt also auch, eine Sprache ist iiU. Zunächst eine wichtige Klärung:

> Lerntheorie befasst sich immer mit **Klassen von Sprachen**.

Man hört zwar oft (in der Linguistik) dass eine Sprache oder Regel lernbar wäre, mathematisch ist das aber Unsinn: denn jede Sprache/Regel ist trivial lernbar. Man nehme z.B. eine beliebige Sprache L. Der Algorithmus A, der L lernt, sieht so aus:

1 return L

Das heißt er gibt einfach L aus, unabhängig von allen möglichen Eingaben. Das ist nicht wirklich spannend. Was spannend ist ist die Lernbarkeit von **Klassen von Sprachen**. Das bedeutet, wir haben eine **Klasse** C, einen **Lerner** mit einer gewissen Strategie, und einen Strom von **Eingabedaten**. Der Lerner sieht den Strom der Eingabedaten und nutzt ihn, um die Sprache der Klasse zu identifizieren. Der Lerner kann seine Meinung ändern, aber irgendwann, nach endlich vielen Schritten, soll er seine Meinung nicht mehr ändern, und wenn er damit richtig liegt, hat er die Sprache **identifiziert**. Wir kommen nun zu den Definitionen. Wichtig ist:

> Wir behandeln hier Lernbarkeit von Text, d.h. der Lerner kann keine Fragen stellen, sondern sieht einfach Sätze (bei uns: Worte).

Präsentation einer Sprache Wir müssen nun bestimmen, welche Daten der Lerner zu sehen bekommt. Wir gehen davon aus, dass Lerner nach und nach alle Worte einer Sprache zu sehen bekommen. Sei L eine Sprache; dann nehmen wir eine beliebige surjektive Funktion

$$f : \mathbb{N} \to L$$

die jeder Zahl ein Wort zuordnet, und außerdem (Surjektivität) für jedes Wort $w \in L$ eine Zahl $n \in \mathbb{N}$ hat, so dass $f(n) = w$. Eine Präsentation von L ist eine unendliche Folge

$$\langle (1, f(1)), (2, f(2)), (3, f(3)), (4, f(4)), ... \rangle$$

wobei $f : \mathbb{N} \to L$ eine surjektive Funktion ist, also

$$f[\mathbb{N}] = L.$$

Wir denotieren die Menge aller Präsentationen von L mit

$$pres(L)$$

Endliche Eingaben Unsere Lerner sehen natürlich niemals ganze Präsentationen von Sprachen, sondern immer nur initiale Segmente davon (wer will schon unendlich lange warten, bevor er eine Hypothese bildet?). Wir definieren EF als die Menge aller endlichen Folgen

$$\langle (1, w_1), (2, w_2), (3, w_3), ..., (n, w_n) \rangle$$

für beliebige $n \in \mathbb{N}$. Sei $\alpha \in pres(L)$, $\beta \in EF$. Wir sagen dann: $\beta = pref_n(\alpha)$, falls gilt:

$\beta = \langle (1, w_1), (2, w_2), (3, w_3), ..., (n, w_n) \rangle$,
$\alpha = \langle (1, w_1), (2, w_2), (3, w_3), ..., (n, w_n), ... \rangle$,

Zu lernende Klasse/Hypothesenraum Eine Klasse von Sprachen ist zunächst eine beliebige Zusammenstellung von Sprachen C. Wichtig ist das die zu lernende Klasse zugleich den Hypothesenraum des Lerners ausmacht. Die beiden sind also dasselbe Objekt, nur je nach Kontext benennt man sie unterschiedlich.

Lernstrategie Sie H der Hypothesenraum. Eine Strategie ist eine Funktion

$$S : EF \to H.$$

Sie bestimmt also unsere Hypothesen gegeben endliche Eingaben.

Der Lerner ist eigentlich nur eine Metapher, um das mathematische Geschehen anschaulicher zu machen. Ein Lerner wird für uns vollständig charakterisiert über seine **Strategie**, dennoch ist er ganz praktisch als Konzept.

Nun kommen die entscheidenden Definitionen: wir sagen

Definition 1 *Eine Strategie S ist erfolgreich auf einer Präsentation $\alpha \in pres(L)$, falls gilt: es gibt eine $n \in \mathbb{N}$, so dass für alle $m > n$ gilt: falls $\beta \in pref_m(\alpha)$, dann ist $S(\beta) = L$.*

Wir *konvergieren* also an einem gewissen endlichen Punkt auf die richtige Hypothese. Daher kommt der Begriff identifizierbar im Unendlichen, der auf Konvergenz beruht.

Definition 2 *Eine Strategie S ist erfolgreich für eine Sprache LS, falls gilt: für alle $\alpha \in pres(L)$ ist S erfolgreich auf α.*

Wir generalisieren also dahingehend, dass die Präsentation keine Rolle spielt für den Erfolg. Zuletzt der Begriff der Lernbarkeit von Klassen:

Definition 3 *Eine Klasse von Sprachen C ist **lernbar**, falls gilt: es gibt eine Strategie S, so dass für alle $L \in C$ gilt: S ist erfolgreich auf L.*

Hier sehen wir also: während Erfolg auf Sprachen noch trivial, ist Lernbarkeit von Klassen bereits sehr problematisch: denn unsere Strategie muss für alle Sprachen und alle Präsentationen funktionieren.

Es gibt zwei Hauptkritiken am Gold Paradigma der Lernbarkeit:

1. Der Begriff ist zu streng, da unsere Strategie auf jeder Präsentation erfolgreich sein muss. Es kann aber Präsentationen geben, die sehr irreführend sind.

2. Der Begriff ist zu weit, da wir auch in diesem Paradigma nie wissen, wann wir etwas gelernt haben: es gibt keinen Punkt, an dem wir sagen können: jetzt werden wir unsere Hypothese nicht mehr ändern. Wir wissen nur es gibt diesen Punkt, aber wir wissen nicht wann er kommt.

4.1 Erste negative Ergebnisse

Die ersten Ergebnisse zu Gold-Lernbarkeit sind wenig ermutigend:

Lemma 4 *Sei C eine Klasse mit $L_1, L_2, ..., L_i, .. \in C$, wobei $L_1 \subset L_2 \subset ... \subset L_i \subset ...$ und $\bigcup_{n\in\mathbb{N}} L_n \in C$; wir haben also eine unendliche Kette von Sprachen, die immer größer werden, mit einer Sprache die alle umfasst. Dann ist C nicht lernbar.*

Der Beweis ist ganz einfach: Nimm nun an, wir haben eine beliebige Präsentation $\alpha \in pres(L_\infty)$. Es gibt ein n, so dass für alle $m > n$ gilt: $S(pref_m(\alpha)) = L_\infty$. Allerdings ist $\beta = pref_m(\alpha)$ nach Annahme endlich, also $\beta \in pres(L_i)$; das gilt für alle $m > n$! Also kann S nicht L_i, L_{i+1} etc. korrekt identifizieren.

Das hat einige Folgen:

1. Die kontext-sensitiven Sprachen sind nicht lernbar.
2. Die kontextfreien Sprachen sind nicht lernbar.
3. Die regulären Sprachen sind nicht lernbar.
4. Jede Klasse, die *alle* endlichen und *mindestens eine* unendliche Sprache enthält, ist nicht lernbar.

Aber Lemma 4 ist noch allgemeiner und lässt sich auch auf Klassen von Sprachen anwenden, die nur unendliche Sprachen enthalten.

4.2 Erste positive Ergebnisse

Der entscheidende Begriff zum Lernbarkeit ist der der **Elastizität** (warum auch immer).

Definition 5 *Eine Klasse C von Sprachen hat **unendliche Elastizität**, wenn es eine unendliche Folge von Worten $w_1, w_2, w_3, ...$ gibt, eine unendliche Folge von Sprachen $L_1, L_2, L_3, ...$, so dass gilt:*

1. $w_n \notin L_n$, *und*
2. $\{w_1, ..., w_n\} \subseteq L_{n+1}$.

Man beachte, dass eine Kette wie in Lemma 4 eine unendliche Elastizität impliziert. Endliche Elastizität ist das Gegenteil:

Definition 6 *Eine Klasse C von Sprachen hat **endliche Elastizität**, wenn für jede unendliche Folge von (verschiedenen) Worten $w_0, w_1, ...$ und jede unendliche Folge von (verschiedenen) Sprachen $L_1, L_2, ...$ es ein $n \in \mathbb{N}$ gibt, so dass gilt: falls $w_n \notin L_n$, dann ist $\{w_1, ..., w_{n-1}\} \not\subseteq L_n$.*

Lemma 7 *Wenn eine Klasse endliche Elastizität hat, dann ist sie lernbar.*

Eine weitere Bedingung, unter der wir lernen können ist wie folgt charakterisiert:

Definition 8 *Eine Klasse C hat **endliche Dichte**, wenn für jede nichtleere Menge $M \subseteq \Sigma^*$ gilt: es gibt nur endlich viele $L \subseteq \Sigma^*$, $L \in C$, so dass $M \subseteq L$ gilt.*

Lemma 9 *Wenn eine Klasse endliche Dichte hat, ist sie lernbar.*

Der Beweis funktioniert wie folgt: da es an jedem Zeitpunkt nur endlich viele Kandidaten gibt, können wir jeden falschen an einem gewissen Zeitpunkt ausschließen und das ganze terminiert – das funktioniert aber nicht, wenn $L \subseteq L'$. In diesem Fall nehmen wir immer die kleinste konsistente Hypothese.

4.3 Weitere Ergebnisse

Es lassen sich noch einige weitere Ergebnisse leicht zeigen:

Lemma 10 *1. Die endlichen Sprachen sind lernbar.*

2. SL_k ist lernbar für jedes k, aber:

3. SL ist nicht lernbar.

4. reguläre Sprachen mit nur n nicht-terminalen sind lernbar (d.h. ihre Sprachen)

Beweis 1. Die Strategie für die endlichen Sprachen ist einfach: nimm immer die kleinste Sprache, die mit den Eingaben soweit konsistent ist. Das konvergiert zwangsläufig. Man beachte dass diese Sprache unendliche Elastizität hat!

2. SL_k ist lernbar, denn: es gibt nur endlich viele SL_k Sprachen über ein bestimmtes Alphabet.

3. Die SL-Sprachen als ganzes unfassen die endlichen Sprachen, aber ebenso unendliche Sprachen. Daher das Ergebnis.

4. Ebenso wie 3. ⊣

Subreguläre Sprachen werden also interessant, wenn es um Lernbarkeit geht.

Weitere interessante Sprachklassen haben folgende Eigenschaften:

Definition 11 *L ist **distributionell**, falls gilt: falls es $x, y \in \Sigma^*$ gibt so dass $xwy \in L$, $xvy \in L$, dann gilt: für alle $z, z' \in \Sigma^*$, falls $zwz' \in L$, dann $zvz' \in L$.*

Klassen wie die distributionellen Sprachen sind normalerweise lernbar, auch wenn sie nicht mehr regulär sind.

5 Algorithmische Komplexität

5.1 Einleitung

Zur Erinnerung: eine Turingmaschine (TM) ist ein Automat $(\Sigma, \square, Q, q_0, A)$, wobei Σ ein Alphabet ist, welches das Eingabealphabet umfasst, aber durchaus noch zusätzliche Symbole umfassen kann; $\square$ ist das Leerzeichen, Q die Zustände, q_0 der Startzustand, $A \subseteq Q \times \Sigma \times \Sigma \times \{-1, 0, 1\} \times Q$ sind die Anweisungen: die Maschine ist in Zustand $q \in Q$, liest Buchstaben $a \in \Sigma$, schreibt dafür $a' \in \Sigma$, bewegt den Kopf nach links/rechts oder bleibt in derselben Zelle, und geht in Zustand $q' \in Q$. Wir haben ein einseitig unendliches Band, auf dem zu Anfang das Eingabewort steht; der Kopf der Maschine steht auf der Zelle ganz links. Wir akzeptieren normalerweise durch das leere Band, wir machen das aber einfacher: wir nehmen eine Menge $F \subseteq Q$ von akzeptierenden Zuständen, eine Menge $R \subseteq Q$ von zurückweisenden Zustände. Natürlich gilt $R \cap F = \emptyset$. Eine TM, wie wir sie benutzen, hat also die Form $(\Sigma, \square, Q, q_0, A, R, F)$

Algorithmische Komplexität ist ein zentrales Kapitel der theoretischen Informatik; hier möchten wir nur einen groben Überblick geben. Alles basiert auf zwei Fragen:

1. Gegeben eine rekursive Sprache L, wie lange braucht eine Turing Maschine um zu entscheiden, ob $w \in L$?

2. Gegeben eine rekursive Sprache L, wie viel Platz (auf dem Band) braucht eine Turing Maschine um zu entscheiden, ob $w \in L$?

Beide Fragen können natürlich nicht allgemein beantwortet werden (der Platz/die Zeit sind nicht beschränkt); die Antwort muss eine Funktion der Eingabe sein. Also: sei

$$f : \mathbb{N} \to \mathbb{N}$$

eine beliebige Funktion. Dann können wir manchmal sagen:

> unsere TM braucht, um zu entscheiden ob $w \in L$, höchstens $f(|w|)$ Rechenschritte/Speicherzellen. Das erste nennt man die **Zeitkomplexität** der Sprache, das zweite deren **Raumkomplexität**.

Wichtig sind hierbei einige fundamentale Beobachtungen:

1. Wir schreiben $f \leq g$ wenn f.a. $n \in \mathbb{N}$ gilt: $f(n) \leq g(n)$. Nun gilt offensichtlich: falls eine TM Zeit/Raumkomplexität f hat, dann hat sie auch g. Wir geben also immer nur **Obergrenzen** an. Die sollten natürlich möglichst eng sein, aber in der Praxis niemals ganz präzise.

2. Wenn eine TM die Zeitkomplexität f hat, dann hat sie auch Raumkomplexität f. Denn sie kann ja nicht mehr Raum ausfüllen, als sie Rechenschritte hat. Umgekehrt gilt das aber nicht: eine TM kann auf begrenztem Raum beliebig viele Rechenschritte ausführen.

3. Allerdings gibt es natürlich auch hier Grenzen, da die Anzahl der möglichen Konfigurationen auf einem begrenzten Raum begrenzt sind. Die Grenze liegt allerdings wesentlich höher.

Hier gilt es noch etwas zu beachten: normalerweise interessieren uns bei algorithmischer Komplexität nicht einzelne Funktionen, sondern **Klasssen von Funktionen**. Die wichtigsten Klassen sind folgende:

- **Lineare Funktionen** sind Funktionen der Form

$$f(x) = ax + b, \text{ wobei } a, b \in \mathbb{N}.$$

  Das b kann man dabei auch weglassen.

- **Polynomiale Funktionen** sind Funktionen der Form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0.$$

  Hierbei kann man alles bis auf $a_n x_n^b$ weglassen, und eigentlich relevant ist nur b_n als Parameter der Funktion.

- **Exponentielle Funktionen** sind Funktionen der Form

$$f(x) = a^x + p,$$

  wobei p eine polynomielle Funktion ist (die allerdings keine Rolle spielt).

Mit diesen 3 Klassen kann man schon das allermeiste abfrühstücken.

5.2 Lineare Funktionen und Kontextsensitive Sprachen

Mit linearen Funktionen können wir 4 Klassen charakterisieren:

1. LINTIME, die Klasse der Sprachen L, die von deterministischen Turing Maschinen in linearer Zeit erkannt werden. Vorsicht: das bedeutet immer, dass es eine TM gibt, die L in linearer Zeit erkennt – das muss nicht für alle gelten.
2. NLINTIME – dasselbe mit *nicht-deterministischen* Turing Maschinen.
3. LINSPACE
4. NLINSPACE

Ein wichtiges Ergebnis der theoretischen Informatik ist folgendes:

Theorem 12 *NLINSPACE ist die Klasse der Kontextsensitiven Sprachen (CSL).*

Man nennt die Klasse der nicht-deterministischen, linear raumbegrenzten TM auch LBA (linear bounded automata). Der Beweis hierfür sieht wie folgt aus:

CSL $\subseteq$ NLINSPACE Wir lassen die TM einfach (nichtdeterministisch) die möglichen Ableitungen rückwärts durchführen. Wenn es eine Ableitung gibt, kommen wir zum Startsymbol. Da außerdem alle Ableitungen das Wort vergrößern, brauchen wir keinen zusätzlichen Platz.

NLINSPACE $\subseteq$ CSL Nimm an, $L \in$ NLINSPACE. Dann gibt es also den Parameter a. Falls $a = 1$, dann können wir jede Operation der TM als eine Kontextsensitive Regel auffassen (mit etwas Kodierungsarbeit). Nun nimm an dass $a > 1$. Da aber a konstant ist, können wir auch diesen Fall simulieren (mit etwas Arbeit), in dem wir jede Abfolge von a Symbolen auf dem Band als ein *einziges* Nicht-terminal auffassen, und jede Veränderung als eine einfache Regel.

Es gibt weitere wichtige Ergebnisse. Für eine Klasse X nennt man co-X die Klasse aller Komplemente von Sprachen in X. Z.B.: co-NLINSPACE ist die Klasse aller Komplemente von CSL-Sprachen. Das bedeutet, dass es eine (nicht-deterministische) TM gibt, so dass sie entscheidet, ob $w \notin L$ in linearer Zeit.

> Für jede Klasse basierend auf deterministischen TM ist XTIME=co-XTIME, XSPACE=co-XSPACE. Für deterministische Maschinen ist Akzeptanz und Nicht-Akzeptanz austauschbar durch die Vertauschung zweier Zustände.
>
> Aber: bei nicht-deterministischen Maschinen ist das nicht trivial, den es geht um die Existenz einer Laufes; und die Vertauschung von Zuständen wird nicht den gewünschten Effekt haben!

Lemma 13 *NLINSPACE=co-NLINSPACE*

Das bedeutet soviel wie: CSL ist geschlossen unter Komplement. Dieses Problem war 20 Jahre ein prominentes offenes Problem (gestellt von Kuroda), bevor es gelöst wurde.

Es gibt ein anderes Problem von Kuroda das bis heute offen ist, nämlich:

Ist NLINSPACE=LINSPACE?

Wenn Sie beruühmt werden möchten (in der theoretischen Informatik), dann müssen sie nur dieses Problem lösen.

LINTIME und NLINTIME werden normalerweise als nicht sehr interessant betrachtet, da sie doch recht eingeschränkt sind. Die Klassen können aber durchaus linguistische Relevanz haben, wie wir noch sehen werden.

5.3 P und NP

Hier handelt es sich mit Abstand um die berühmtesten und wichtigesten Klassen. Wir fangen an mit PTIME, die insgesamt als die wichtigste Komplexitätsklasse bezeichnet werden kann. PTIME ist die Klasse aller Sprachen, die von einer deterministischen TM in polynomialer Zeit erkannt werden. Der Grund für die Wichtigkeit dieser Klasse ist folgender:

> PTIME gilt als die größte Klasse von Problemen, die im Normalfall noch gehandhabt werden kann.

Das ist natürlich mit Vorsicht zu genießen: wenn wir eine TM haben, die für jede Eingabe w in $|w|^{100}$ Schritten entscheidet, ob $w \in L$, dann ist das natürlich überhaupt nicht handhabbar, bereits für sehr kurze Worte. Insbesondere folgt daraus aber:

> Natürliche Sprachen sind in PTIME enthalten.

Erstmal eine Reihe wichtiger Ergebnisse:

Lemma 14 *1. $CFL \subseteq PTIME$*

2. PTIME ist geschlossen unter Komplement, Schnitt und Vereinigung.

1. folgt aus Standard Parsing-Algorithmen (z.B. CYK).
2. Komplement ist klar. Schnitt: nimm an, $L_1, L_2 \in$PTIME. Dann gibt es die entsprechenden TM_1, TM_2. Wir konstruieren nun TM_3, die zuerst TM_1 die Eingabe kopiert, dann auf dieser Kopie TM_1 ausführt; wenn diese in einen akzeptierenden Zustand kommt, dann führt TM_3 auf dem verbliebenen Original TM_2 aus; wenn diese ebenfalls akzeptiert, dann akzeptiert TM_3. TM_3 erkennt also $L_1 \cap L_2$.
3. Vereinigung folgt.

Eine weitere wichtige Klasse ist LOGSPACE, die wie folgt charakterisiert wird: wir nutzen eine multi-tape TM, die also mehrere Bänder hat, und bei der jede Anweisung für ein gewissen Band spezifiziert ist. Diese hat $k + 2$ Bänder, wobei $k \in \mathbb{N}_0$ beliebig ist, und es gilt: Band 1 ist *read-only*, wir können also nur die Eingabe lesen; Band k+2 ist *write-only*, wir können also nur schreiben, nichts lesen; und die übrigen Bänder sind begrenzt durch $log_2(|w|)$, also durch den Logarithmus der Länge des Eingabewortes. LOGSPACE ist die Klasse aller Sprachen, die man mit solchen Maschinen erkennen kann.

Lemma 15 $LOGSPACE \subseteq PTIME \subseteq NLINSPACE$

PSPACE ist tatsächlich eine Klasse von Sprachen, die gewisse kombinatorische Muster einfach nicht zulässt, und eine Klasse, die in gewissem Sinne natürlich ist. Das sieht man auch daran, dass PTIME eine von wenigen (fast die einzige) Klasse ist, die auch durch eine Klasse von Grammatiken charakterisiert wird, die sog. **Literal Movement Grammars**.

Eine LMG ist eine Grammatik mit Regeln

1. $P(a)$, wobei $a \in \Sigma$;
2. $P(\overline{x}) \leftarrow R_1(\overline{y_1}), ..., R_i(\overline{y_i})$, wobei gilt:
3. $\overline{y_1}, ..., \overline{y_i}$ sind Worte über Variablen (die nicht notwendig verschieden sind!), und
4. $\overline{x}$ ist eine Wort über die Variablen, in dem jede Variable der rechten Seite $\overline{y_n}$. genau einmal vorkommt.

Wir leiten ein Wort w ab in unserer Grammatik, falls wir $S(w)$ ableiten können mit unseren Regeln. Es ist nicht schwer zu sehen dass ohne Bedingung 4 wir eine Typ 0 Grammatik haben. Mit Bedingung 4 haben wir aber eine effektive Charakterisierung von PTIME:

Lemma 16 *L wird erzeugt von einer LMG, genau dann wenn $L \in$ PTIME.*

Es gibt noch zwei unabhängige Charakterisierungen von PTIME, eine mittels read-only multi-tape TM, und eine mittels Fixpunkt Logiken. Das führt aber zu weit ab.

NPTIME ist die Klasse der nicht-deterministisch polynomiellen Sprachen. Dementsprechend denkt man, dass wenn eine Sprache/ein Problem in NPTIME ist aber nicht mehr in PTIME, dann kann es nicht mehr effektiv gehandhabt weden. Das größte und bekannteste Problem der theoretischen Informatik ist die Frage:

$$\text{Ist PTIME} = \text{NPTIME?}$$

Das ist das berühmte P=NP Problem; wenn Sie das lösen, werden Sie nicht nur berühmt, sondern auch reich. Der Grund, warum das so extrem wichtig ist, ist folgender: viele angewandte Kodierungsmechanismen basieren auf Problemen in NP (NP-vollständigen Problemen), insbesondere Folgendes: gegeben eine Zahl n (gesehen als eine m-stellige Ziffer ist das einfach ein Wort), finde die Primfaktoren von n. Dieses Problem ist essentiell ein Kodierungsproblem, denn die Information der Primfaktoren ist in n vorhanden, aber sie zu entschlüsseln wäre zu zeitaufwändig.

Wäre nun aber P=NP, dann würde das bedeuten, dass alle diese Probleme praktisch lösbar wären – also sämtliche zur Zeit genutzte Kodierung von Information wirkungslos ist. Das hätte also eine verheerende Wirkung auf sämtliche Anwendungen, die auf Kodierung von Information beruhen!

5.4 P und NP: Funktionen und Reduktionen

Wir haben hier nur Sprachklassen betrachtet. Allgemeiner kann man sagen, dass Turingmaschinen Funktionen berechnen, also eine TM berechnet eine Abbildung

$$f_{TM} : \Sigma^* \to T^*$$

Falls eine TM eine Sprache erkennt L, dann sagen wir einfach: sie berechnet eine Funktion

$$f_{TM} : \Sigma^* \to \{0, 1\},$$

wobei

$$(1) \quad f_{TM}(w) = \begin{cases} 1, \text{ falls } w \in L \\ 0 \text{ andernfalls} \end{cases}$$

Das nennt man auch die **charakteristische Funktion** einer Sprache, oder allgemeiner einer Menge.

Das bedeutet: wir können also die gesamte obige Diskussion auf Funktionen erweitern, und in diesem Sinne sagt man auch: ein Problem ist in PTIME, NPTIME etc., mit der Bedeutung: es gibt eine Funktion in PTIME, die das Problem löst, d.h. eine TM mit polynieller Zeit etc.

Funktionen sind auch aus einem anderen Grund wichtig: man kann sie nämlich **komponieren**:

$$f \circ g(x) = f(g(x))$$

Nun gibt es folgende Ergebnisse:

Lemma 17 *Falls $f, g \in$ PTIME, dann ist auch $f \circ g \in$ PTIME.*

Das ist relativ klar, denn wenn wir $f(w)$ in $|w|^a$ Schritten berechnen, $g(v) \in |v|^b$, dann berechnen wir $f \circ g(v)$ in $|v|^b + |g(v)|^a$, was natürlich polynomiell ist. Mit demselben Argument kann man natürlich folgendes zeigen:

Lemma 18 *Falls $f, g \in$ NPTIME, dann ist auch $f \circ g \in$ NPTIME.*

Diese Methode nutzt man für sogenannte **Reduktionen**: nehmen wir an, wir haben eine Funktion f; wir möchten zeigen dass $f \in$ PTIME; wir wissen, dass g in PTIME ist. Dann gilt: wir brauchen Funktionen $h_1, h_2 \in$ PTIME, so dass gilt:

$$(2) \quad f = h_2 \circ g \circ h_1$$

Das bedeutet: wir können (mittels h) die Funktion f auf g **reduzieren**, indem wir erst die Eingabe entsprechend kodieren (h_1), dann g applizieren, und dann die Ausgabe nachbearbeiten (h_2). In solchen Fällen lässt man normalerweise g den schweren Teil erledigen, und dadurch spart man sich eine Menge Arbeit.

Das führt uns zu einem weiteren zentralen Konzept, nämlich der polynomiellen Reduktion und **NP-Vollständigkeit**:

Eine Sprache $L \subseteq \Sigma^*$ ist polynomiell auf L' reduzierbar, falls es eine Funktion $f \in$ PTIME gibt, so dass für jedes $w \in \Sigma^*$ gilt: $w \in L$ genau dann wenn $f(w) \in L'$.

Eine Sprache L ist NP-vollständig, falls gilt:

1. $L \in$ NPTIME, und
2. Jedes L' in NPTIME kann polynomiell auf L reduziert werden.

Das bedeutet also: es reicht zu zeigen dass $L \in$ PTIME, um zu beweisen das PTIME = NPTIME.

Es gibt eine lange Liste von NP-vollständigen Problemen, z.B. *travelling salesman* und Erfüllbarkeit von Aussagenlogik.

5.5 Jenseits von NP

PSPACE ist natürlich größer als NPTIME, also NPTIME $\subseteq$ PSPACE. Das ist nicht trivial, lässt sich aber zeigen, denn die nicht-deterministische TM kann modifiziert werden, so dass sie alle möglichen Berechnungen nacheinander auf demselben Platz ausführt, bis sie eine akzeptierende findet. Es ist meines Wissens nach nicht bekannt ob

$$\text{NPTIME} = \text{PSPACE}?$$

Da NPTIME bereits nicht mehr handhabbar ist, ist das Interesse an PSPACE begrenzt. Es gibt aber ein interessantes Problem, das PSPACE-vollständig ist, nämlich:

> Gegeben zwei reguläre Ausdrücke r_1, r_2, denotieren diese dieselbe Sprache?

Das Problem scheint sehr einfach, ist aber in der Tat hochkompliziert. Natürlich gilt PSPACE $\subseteq$ EXPTIME, also Probleme mit exponentieller Zeit. Solche Probleme findet man oft in der Prädikatenlogik (bestimmten Fragmenten, insgesamt ist Prädikatenlogik nicht mal rekursiv, nur rekursiv aufzählbar). Es gibt aber auch Probleme die sind nicht einmal in EXPTIME, z.B. Erfüllbarkeit von Formeln in Presburger Arithmetik (hyperexponentiell, zweiten Grades).

Es geht also immer noch komplexer!

6 Deskriptive Komplexität

6.1 Motivation 1: Ziffern und Zahlen

Man kann deskriptive Komplexität auf verschiedene Arten motivieren; wir machen das mit zwei Beispielen. Nehmen wir die **Ziffer**

$$1.000.000$$

Welche Zahl wird damit repräsentiert? Man wäre versucht zu sagen: "eine Million", aber das stimmt natürlich nur unter Annahme einer bestimmten Kodierung, nämlich der dezimalen. Es könnte aber genausogut die Zahl

$$3^6 = 729$$

sein – nämlich wenn wir ein Tertiärsystem annehmen (oder $2^6 = 64$ in binärer Darstellung). Hier muss man also vorsichtig sein: wir können keine Zahlen im engeren Sinne schreiben, was wir schreiben sind immer nur *Ziffern*, und Ziffern sind Worte über einem Alphabet $\{0, .., n\}$, die nach gewissen Regeln als Zahlen interpretiert werden.

Nun kann man folgende Beobachtung machen: manche Zahlen sind komplexer als andere im einfachen Sinne dass ich mehr Platz brauche, um sie aufzuschreiben. Das ist *nicht* wegen ihrer Größe, denn die Zahl "eine Milliarde" kann ich schreiben als

$$10^9$$

– also sehr kompakt. Was ist mit der Zahl "1.162.261.467" (dezimal)? Auch diese Zahl kann ich relativ kompakt schreiben, nämlich als

$$3^{19}$$

Nimm aber nun z.B. eine beliebige Primzahl p mit 10 Stellen. Diese Zahl kann ich – normalerweise – nicht akürzen; ich brauche 10 Stellen im Dezimalsystem um diese Zahl mitzuteilen.

Da Ziffern als Worte syntaktische Objekte sind, lässt sich das auch mittels syntaktischer Regelmäßigkeiten ausdrücken. Für "1.162.261.467" (dezimal) gibt es eine Repäsentation in einem System, die relativ einfach ist, nämlich

$$100.000.000.000.000.000.000 \text{ (Tertiärsystem)}$$

Für p gibt es keine solche redundante Darstellung, die einzige Möglichkeit, die Zahl kompakter darzustellen, ist das Alphabet zu vergrößern. Ein weiterer wichtiger Punkt ist die **Informativität**: je komplexer die Zahl, desto größer ist in gewissen Sinne ihr Informationsgehalt; dazu werden wir später kommen, das bedeutet aber soviel wie: Sie mögen 10^9 oder 3^{19} relativ einfach erraten; dass sie p finden, ist praktisch ausgeschlossen. Es gibt also folgende Korrelation:

1. Wie einfach kann ich etwas beschreiben (wieviel Platz brauche ich)?
2. Wie regelmäßig ist die Beschreibung in einem festgelegten System?
3. Wie einfach ist es, etwas zu erraten?

Hier gibt es noch eine Sache zu beachten. Nehmen Sie an, ich sage Ihnen:

> p ist die drittgrößte Primzahl kleiner als 10^{1000} (dezimal).

Damit habe ich p relativ kompakt charakterisiert, denn wir erwarten dass p 999 Ziffern hat in Dezimalform. Wo ist das Problem? Das Problem ist, dass Sie keine Ahnung haben, wie p aussieht. Sie könnten es zwar theoretisch rekonstruieren, aber praktisch ist das so gut wie unmöglich. Wir haben also noch eine wichtige Beobachtung:

> Eine Beschreibung ist nur dann relevant, wenn Sie mir erlaubt, das beschriebene Objekt eindeutig zu rekonstruieren. Außerdem: ich kann die Beschreibung manchmal verkürzen, aber nur um den Preis, die Berechnung zu verlängern.

Das sollte also auch in Rechnung gestellt werden.

6.2 Motivation 2: Sprachen und ihre Beschreibung

Nehmen Sie an, wir wissen über ein Sprache L folgendes:

- Jedes Wort a^{2n}, mit $n \leq 2000$, ist in L.
- Kein Wort a^{2n+1}, mit $n \leq 1999$, ist in L.
- Sonst wissen wir nichts.

Dieses Wissen ist durchaus vergleichbar mit dem Wissen, das ein Linguist hat, denn wir beobachten ja keine unendlichen Sprachen, sondern nur endliche Datenmengen von Worten (Sätzen) in der Sprache, sowie die Tatsache dass man gewisse Dinge nicht sagen kann. Wir stellen uns nun die Frage: Was wäre die *einfachste Sprache* L' so dass

1. Jedes Wort, von dem wir wissen dass es in L ist, auch in L' ist, und
2. kein Wort, von dem wir wissen dass es nicht in L ist, in L' ist.

Die Antwort, nach unserer Komplexitätstheorie soweit, wäre immer:

$$L' = (aa)^+ \cap \Sigma^{\leq n}$$

Denn die endlichen Sprachen stehen am Boden jeder Hierarchie, die wir bislang betrachtet haben. Auch die Darstellung, die wir gewählt haben, ist relativ kompakt.

Was passiert jetzt aber, wenn wir diese Sprache *effektiv* darstellen wollen, also einen Automaten, der die Sprache erkennt? Dieser Automat wird 2000 Zustände brauchen, von denen 1000 akzeptierend sind! Das bedeutet:

1. Unsere "einfachste Beschreibung" als *Klasse* resultiert in einer sehr komplexen Beschreibung als *Objekt*.

2. Das fällt zusammen mit der Tatsache, dass wir keinerlei **Generalisierung** getroffen haben.

Eine weitaus einfacherere Sprache (als Objekt) wäre natürlich

$$L' = (aa)^+$$

– aber das liegt ein einer deutlich höheren Komplexitätsklasse!

Betrachten wir ein zweites Beispiel, diesmal mit Generalisierung. Wir haben die Sprache L_1, über die wir folgendes wissen:

- Falls $|w| \leq 1000$, dann ist $w \in L_1$ genau dann wenn $w = a^n b^n$, $n \in \mathbb{N}$.
- Sonst wissen wir nichts.

Was wäre die *einfachste Sprache* L_1' so dass

1. Jedes Wort, von dem wir wissen dass es in L ist, auch in L_1' ist, und
2. kein Wort, von dem wir wissen dass es nicht in L ist, in L_1' ist,
3. und L_1' ist unendlich.

Nach unserer bisherigen Komplexitätstheorie gibt es eine einfache Antwort:

$$L_1' = L_1 \cup \{a^m b^o : m + o > 1000\}$$

Das ist eine reguläre Sprache (genau genommen ist das sogar streng lokal!) und dementsprechend einfach. Allerdings: der endliche Automat braucht wiederum mindestens 1000 Zustände! Andererseits gibt es einen sehr einfachen Kellerautomaten mit 3 Zuständen und 2 Stacksymbolen, der eine andere Generalisierung der Sprache erkennt, nämlich

$$L_1'' = \{a^n b^n : n \in \mathbb{N}\}$$

Wir haben also folgende Beobachtung:

1. In Szenarien wie dem unseren ist Komplexität als Klassenbegriff ist oft entgegengesetzt zur Komplexität eines einzelnen Objektes (je höher die Klasse, desto einfacher die Beschreibung – tendenziell)
2. Die Einfachheit der *effektiven* Beschreibung fällt oft zusammen mit unserem Begriff der *guten Generalisierung*.

Der Begriff der deskriptiven Komplexität ist also von enormer Wichtigkeit in der Linguistik, auch wenn er dort selten explizit benutzt wird, denn er gibt ein Maß dafür, was eine gute Beschreibung ist. Und implizit gibt er damit auch ein Kriterium dafür, wie wir uns vorstellen dass Sprache wirklich aussieht!

6.3 Definitionen

Um deskriptive Komplexität zu definieren, braucht man zunächst Turingmaschinen. Der Grund hierfür ist offensichtlich: TMs sind die mächstigste Automatenklasse, und jede schwächere Klasse würde *apriori* gewisse Beschränkungen an mögliche Beschreibungen stellen, was wir natürlich nicht wollen. Wir bekommen dann zwei natürliche Korrespondenzen:

deskriptive Komplexität eines Wortes, einer Zahl etc.
$\cong$
Länge der einfachsten eindeutigen effektiven Beschreibung

Ebenso:

deskriptive Komplexität eine Sprache, Menge von Zahlen (egal ob endlich oder unendlich) etc.
$\cong$
Beschreibungsgröße der einfachsten TM, die diese Menge charakterisiert (bzw. deren Kode bei Zahlen).

Die erste Frage ist: was ist die Beschreibungsgröße einer TM? Hierfür gibt es eine einfache Antwort: jede TM kann als ein Wort w kodiert werden, das dann wiederum die Eingabe für eine universelle TM ist (siehe unten). Die beiden Korrespondenzen sind also im Prinzip ein und dieselbe, dank der universellen Turingmaschine. Das müssen wir erstmal sorgfältig definieren.

- Im folgenden, nimm ein Eingabealphabet Σ als fest und gegeben an.
- Es gibt eine konstante c, so dass jede TM kann umgeschrieben werden als eine TM mit Arbeitsalphabet der Größe $|\Sigma| + c$ ($c = 4$, soweit ich weiß).

Sei T ein beliebiges Alphabet mit $|T| \geq 4$. Eine **universelle Turingmaschine** U für T, Σ, ist eine TM so dass es eine injektive Abbildung

$$\phi : \mathbf{TM} \to T^*$$

gibt, und für alle Worte $w \in T^*$, $v \in \Sigma^*$ gilt:

U akzeptiert $w\#v$ genau dann wenn die TM $\phi^{-1}(w)$ v akzeptiert.

Man sagt dann auch dass U eine universelle TM bezüglich der Kodierung ϕ ist.

Wichtig ist: es gibt nicht *die* universelle TM, sondern es gibt unendlich viele. Deswegen kann man auch $\phi(TM)$ nicht einfach als Maß für die Komplexität einer TM nehmen: ich kann z.B. für eine beliebige Maschine M eine universelle TM U_1 mit Kodierung ϕ_1 bauen, so dass

$$\phi(M) = 0$$

– wir haben also eine Länge der Kodierung von 1, denn Kodierungen sind immer *willkürlich*. Das ist ein grundlegendes Problem der deskriptiven Komplexität, und nur das Invarianztheorem löst dieses Problem zumindest teilweise.

6.4 Gödelisierung von TMs

Wir machen das mittels der sog. **Gödelisierung**, einem extrem wichtigen Verfahren aus der Theorie der Berechenbarkeit und Entscheidbarkeit. Es handelt sich hier um eine Kodierung, die auf der **Eindeutigkeit der Primfaktorzerlegung** basiert.

- Zunächst bilden wir jeden Buchstaben eines Alphabets auf eine Zahl ab, mittels einer Bijektion $i : \Sigma \to \{1, ..., |\Sigma|\}$.
- Wir lassen $p_1, p_2, ...$ die unendliche Folge der Primzahlen sein, also $p_1 = 2, p_2 = 3, p_3 = 5$ etc.
- Dann kodieren wir Wörter wie folgt: $göd(a_1a_2...a_n) = (P_1)^{i(a_1)} \cdot p_2^{i(a_2)} \cdot ... \cdot p_n^{i(a_n)}$

Wir haben also die Gödelfunktion $göd : \Sigma^* \to \mathbb{N}$.

Das Ergebnis der Gödelisierung einer TM wird also eine Zahl sein, und diese Zahl wird wiederum als eine Ziffer kodiert, die wiederum eine gewisse Länge hat. Das Schöne an der Gödelisierung ist: sowohl die Länge eines Wortes als auch die Anzahl der Buchstaben spielt eine Rolle!

Wenn wir TMs statt Worte gödelisieren, ist das noch etwas komplizierter: wir gödelisieren im Prinzip nur die Anweisungen, und jedes einzelne Symbol wird übersetzt, also auch $(,), ,$, etc.

Die Gödelkodierung einer TM ist ein Wort $\langle M \rangle \in \{0, 1\}^*$, so dass

$$\langle M \rangle_1 0 = göd(TM)$$

Also nichts weiter als die binäre Repräsentation der Zahl.

6.5 Kodierungen von Worten

Definition 19 *Eine Kodierung von w ist ein Wort $\langle M \rangle x$, wobei M eine TM ist, $\langle M \rangle$ die Gödelkodierung von M, und M mit Eingabe x die Ausgabe w gibt.*

Das würde bedeutetn: die Kodierung einer Sprache ist die Kodierung einer TM M $\langle M' \rangle w$, so dass M' für Eingabe w $\langle M \rangle$ als Ausgabe gibt. Wir werden uns – motiviert durch das Invarianztheorem – diesen Zwischenschritt schenken, denn $\langle M' \rangle x$ ist ja auch nur eine Kodierung genauso wie $\langle M \rangle$.

Definition 20 *Eine Kodierung einer Sprache $L \subseteq \Sigma^*$ ist ein Wort $\langle M \rangle$, wobei M eine TM ist, $\langle M \rangle$ die Gödelkodierung von M, und $L(M) = L$.*

6.6 Optimale Kodierung und Invarianz

Wir können nun definieren:

Definition 21 *Die Kolmogorov-Komplexität eines Wortes w ist die Länge der optimalen Kodierung $\langle M\rangle x$ von w, also:*

$$kk(w) = min_{M(x)=w}|\langle M\rangle x|$$

Das ist aber erstmal *immer noch nicht* wirklich informativ, denn es gibt viele andere Arten M zu kodieren, und wir haben die Gödelkodierung willkürlich herausgegriffen. Das wird aber informativ durch das **Invarianztheorem**:

Theorem 22 *Für alle Beschreibungsspezifikationen ϕ eines Wortes w gibt eine TM M und eine Konstante $c > 0$, so dass $\langle M\rangle x$ eine Kodierung von w ist, und*

$$|\langle M\rangle x| \leq \phi(w) + c$$

Hier bedeutet "Beschreibungsspezifikation" ϕ soviel wie: eine Eingabe $w_M x$ für eine beliebige universelle TM U, wobei U nach Eingabe w_M die Funktion der Maschine M berechnet, und $M(x) = w$. Invarianz bedeutet soviel wie: bis auf eine Konstante macht die Art der Kodierung keinen Unterschied, die Länge des Kodes ist also in gewissem Sinn invariant unter der Art der Kodierung. Das bedeutet, bis auf einen konstanten Wert ist unsere Gödelkodierung optimal. Das IT zeigt also, dass die Kolmogorov-Komplexität ein allgemein sinnvolles Maß für Beschreibungskomplexität ist. NB: das IT widerspricht nicht der Tatsache, dass es für jedes Wort/jede Maschine ein Kodierung gibt der Länge 1, und genau deshalb ist es wichtig, sich auf eine Kodierung festzulegen! Wenn wir das Minimum über jede Kodierung nehmen würden, dann käme immer 1 heraus!

Die Kolmogorov Komplexität ist sinnvoll sowohl auf Worten, als auch auf Mengen mittels deren (endlicher) Repräsentation (über Turingmaschinen); wir definieren dann:

Definition 23 *Die Kolmogorov-Komplexität einer Sprache ist die Länge der optimalen Kodierung $\langle M\rangle$ für $L(M) = L$, also*

$$kk(L) = min_{L(M)=L}|\langle M\rangle|$$

Wir können auch sagen, dass Mengen ohne endliche Repräsentation (nicht rekursiv aufzählbare Sprachen) eine unendliche Kolmogorov-Komplexität haben.

6.7 (Un)entscheidbarkeit

Hier geht es um ein Ergebnis, das die Nützlichkeit der Beschreibungskomplexität wieder relativiert:

Theorem 24 *Die Funktion $kk : \Sigma^* \to \mathbb{N}$ ist nicht berechenbar, d.h. es ist allgemein nicht entscheidbar, welche Kolmogorovkomplexität ein Wort/eine Sprache/eine Menge hat.*

Das beruht auf der Unentscheidbarkeit des Halteproblems: wir können nicht entscheiden, ob eine TM mit einer Eingabe x jemals hält. Nehmen wir an, wir wissen, es gibt eine TM M_1, ein Wort x_1 mit $M_1(x_1) = w$. Wir möchten wissen: gibt es M_2, x_2 mit

1. $M_2(x_2) = w$ und
2. $|\langle M_2 \rangle w| < |\langle M_2 \rangle w|$?

Das ist zunächst ja kein Problem, es gibt ja nur endlich viele Kandidaten die Bedingung 2 erfüllen. Wenn wir aber nur Bedingung 1 für nur *eine einzige* TM prüfen sollen, kann es bereits unendlich lange dauern, da es sein kann, dass die TM niemals hält, und wir gleichzeitig nie wissen ob sie nicht doch noch halten wird. Die Unentscheidbarkeit der kk hängt also an der Unentscheidbarkeit des Halteproblems.

6.8 Algorithmische Zufälligkeit

Was hat das mit Zufall und Wahrscheinlichkeit zu tun? Folgendes: wir sagen, ein Wort w ist algorithmisch zufällig, falls $kk(w) > |w|$, also die kürzeste Repräsentation von sich selber ist. Man sagt auch: es ist nicht komprimierbar (dann gilt $kk(w) > |w|$ weil wir trotzdem Platz für die TM brauchen).

Es gilt: die meisten Worte sind algorithmisch zufällig; sei $AZ_{\leq n}(\Sigma)$ die algorithmisch zufälligen Worte $w \in \Sigma^*$ mit $|w| \leq n$; es gilt also:

$$(3) \quad lim_{n\to\infty} \frac{AZ_{\leq n}(\Sigma)}{|\Sigma^{\leq n}|} = 1$$

Der Beweis ist einfach über Kardinalitäten: Die Anzahl der Worte einer gewissen Länge wächst in etwa gleich wie die Anzahl der möglichen Kodierungen einer gewissen Länge. Da einige Worte sehr kurz kodiert werden können, heißt das, die meisten können nicht verkürzt werden: es gibt 2^n Kodewörter der Länge n, und $|\Sigma|^m$ Wörter der Länge m. Wenn nun $n < m$, dann ist klar dass die meisten Wörter der Länge $\leq m$ kein Kodewort haben können, das kürzer ist.

Dennoch ist die praktische Wahrscheinlichkeit, einem algorithmisch zufälligen Wort zu begegnen, eher gering, wie man am Beispiel von Zahlen sehen kann: Sie haben alle schon die Ziffer

1.000.000.000

gesehen – aber es gibt wohl keine 7stellige, algorithmisch zufällige Zahl die Sie alle bereits gesehehn haben. Das hängt auch mit dem Informationsgehalt zusammen.

Übrigens ist es ebenfalls unentscheidbar, ob ein gewisses Wort oder eine Zahl algorithmisch zufällig sind, aus dem Grund, das bereits die Kolmogorov-Komplexität unentscheidbar ist – es ist unentscheidbar, ob ein Wort x ein Wort w kodiert, daher auch unentscheidbar, ob es ein kürzeres Wort gibt. Wir wissen nur, dass jedes Wort *wahrscheinlich* algorithmisch zufällig ist!

Zufällig bedeutet hier also soviel wie: Es gibt kein zugrundeliegendes Muster, oder zumindest keines, dass einfacher wäre als das Wort selber. Übrigens motiviert sich dieser Begriff daraus, dass ein zufällig ausgewähltes/konstruiertes Wort höchstwahrscheinlich genau eine solche Form hätte: eben weil die meisten Worte (aus der Gesamtheit aller Worte) algorithmisch zufällig sind!

6.9 Fazit

Insgesamt sollte man sich folgende Punkte merken:

- Unser Begriff der deskriptiven (Kolmogorov) Komplexität ist sehr wichtig, da er einem intuitiven Begriff von Einfachheit entspricht.
- Insbesondere für gute Generalisierungen und Lernen von Sprachen bietet die KK ein gutes Kriterium.
- In der Praxis allerdings ist es fast immer unmöglich, $kk(w)$ oder $kk(L)$ auszurechnen; selbst unsere Definitionen lassen viele Details implizit (die genaue Gödelisierung z.B.).
- Und selbst in der Theorie lässt sich $kk(w)$ oft nicht ausrechnen, d.h. es ist keine berechenbare Funktion.
- Oft genug gibt es aber klare Fälle, in denen man eindeutig sagen kann dass $kk(w) > kk(w')$ bzw. $kk(L) > kk(L')$, ohne die Details auszurechnen. Hierfür ist der Begriff also trotzdem sinnvoll!

7 Informationstheoretische Komplexität

7.1 Informativität und Entropie

Wir haben bereits besprochen das deskriptive Komplexität mit Information zu tun hat: Je höher die deskriptive Komplexität, desto schwieriger ist das Wort zu erraten, desto mehr Information enthält das Wort in einem gewissen, intuitiven Sinn. Wir werden das jetzt formal machen. Um den Informationsbegriff zu definieren, braucht man zunächst den Begriff der Wahrscheinlichkeit.

Sei P die Wahrscheinlichkeitsverteilung über einer Menge Ω. Wir definieren den Informationsgehalt von $x \in \Omega$ als

$$I(x) = \log(\frac{1}{P(x)}) = -\log(P(x)) \quad (4)$$

NB: Es ist letztlich egal, welche Basis man für die logarithmische Umformung benutzt.

Warum ist I so definiert? Das ist bedingt durch folgende Erwägungen:

1. I soll antiton ("gegenläufig") sein zu P: Je kleiner $P(x)$, desto größer $I(x)$. Intuitiv bedeutet das: Je unwahrscheinlicher ein Ereignis, desto mehr Information bekommen wir, wenn wir erfahren, dass es stattgefunden hat.
2. I soll nicht-negativ sein, also $I(x) \geq 0$
3. $I(1) = 0$, das heißt: Sichere Ereignisse haben keine Information.
4. Additivität: falls $P(x, y) = P(x)P(y)$ (Ereignisse sind unabhängig), dann ist $I(x, y) = I(x) + (y)$

Diese Desidarate lassen tatsächlich nur eine Lösung zu, nämlich (4). Mittels dessen lässt sich die Entropie eines Ereignisses berechnen:

$$H_P(x) = P(x) \cdot I(x) = P(x) \cdot -log(P(x)) \quad (5)$$

Die Entropie von Ereignissen ist – entgegen weit verbreiteter Annahmen – keine wirklich relevante Größe; was viel interessanter ist, ist die Entropie von Wahrscheinlichkeitsverteilungen:

$$H(P) = -\sum_{x \in \Omega} P(x) \cdot log(P(x)) \quad (6)$$

also die Summe der Entropien der Ereignisse. Hier gilt, für endliches Ω, folgende Regel: Je näher eine Verteilung an der **uniformen Verteilung** P_U mit

$$P_U(x) = \frac{1}{|\Omega|} \quad (7)$$

liegt, desto größer ist die Entropie. Die Entropie ist also ein Maß für die Gleichmäßigkeit und damit auch für die "Unsicherheit" einer Verteilung; je weniger wir in der Lage sind, ein Ergebnis vorherzusagen, je größer also unsere Unsicherheit, desto größer die Entropie.

7.2 Bedingte Entropie

Die bedingte Entropie von zwei Variablen (über demselben Wahrscheinlichkeitsraum) ist wie folgt definiert (hier bedeutet $y \in Y$ soviel wie: y ist ein Wert, den Y annehmen kann):

$$H(X|Y) = \sum_{y \in Y} P(Y = y) H(X|Y = y) \tag{8}$$

Wenn wir diese Definition auflösen, bekommen wir:

$$H(X|Y) = \sum_{x \in X, y \in Y} P(X^{-1}(x) \cap Y^{-1}(y)) \log \left(\frac{P(X^{-1}(x) \cap Y^{-1}(y))}{P(Y^{-1}(y))} \right) \tag{9}$$

Die bedingte Entropie ist also ein Maß dafür, wie stark die Werte einer Zufallsvariable Y die Werte einer Zufallsvariable X festlegen. Wenn der Wert von X durch den Wert von Y – egal wie er ist – immer festgelegt ist, dann ist

$$H(X|Y) = 0 \tag{10}$$

insbesondere also:

$$H(X|X) = 0 \tag{11}$$

Umgekehrt, falls der Wert von Y keinerlei Einfluss hat auf die Wahrscheinlichkeitsverteilung des Wertes von X, dann haben wir

$$H(X|Y) = H(X) \tag{12}$$

Es ist klar dass das hier nur für diskrete Wahrscheinlichkeitsräume funktionieren kann; in kontinuierlichen Räumen funktionieren diese Dinge etwas anders.

Es gibt auch eine Kettenregel für bedingte Entropie:

$$H(X|Y) = H(\langle X, Y \rangle) - H(Y) \tag{13}$$

wobei $\langle X, Y \rangle$ eine neue Variable ist, mit

$$P(\langle X, Y \rangle = (x, y)) = P(X = x, Y = y) = P(X^{-1}(x) \cap Y^{-1}(y)) \tag{14}$$

Wir nehmen also die Entropie der **Verbundverteilung**, und ziehen die Entropie von $H(Y)$ ab.

7.3 Entropie und formale Sprachen

Wir können Sprachen als unendliche Reihen von Zufallsexperimenten sehen: ein Wort wird dabei aufgefasst als eine (endliche oder unendliche) Folge. Unsere Sprache dementsprechend eine Menge von solchen Folgen. Damit können wir natürlich eine zugrundeliegende Wahrscheinlichkeitsverteilung **schätzen**, und

Begriffe wie Entropie einer Sprache basieren auf dieser geschätzten Wahrscheinlichkeitsverteilung. Schätzungen folgen hierbei der Einfachheit halber der Maximumlikelihood Methode, d.h. wir nehmen **relative Häufigkeiten** als Wahrscheinlichkeiten an.

Hier gibt es aber einiges zu beachten: wir haben eine unendliche Folge von Zufallsvariablen $X_1, ...X_n, ...$ Weiterhin gilt natürlich: wir arbeiten oft mit unendlichen Mengen, d.h. um sinnvolle Zahlen herauszubekommen, brauchen wir eine Grenzwertkonstruktion. Dementsprechend definieren wir: gegeben ein $L \subseteq \Sigma^*$ ist

$$L_n = \{w \in L : |w| \leq n\} \tag{15}$$

Entscheidend ist, welche Art von Wahrscheinlichkeit wir schätzen, z.B.:

1. Die Wahrscheinlichkeit, dass irgendein X_i irgendeinen Wert annimmt.
2. Die Wahrscheinlichkeit, dass X_i einen Wert annimmt, gegeben die Vorgängerwerte.

Die Wahrscheinlichkeit, dass irgendein X_i irgendeinen Wert annimmt

Das bezeichnen wir als

$$P_L(X = a)$$

schätzen wir mittels

$$\hat{P}_L(X = a) = lim_{n\to\infty} \sum_{w \in L_n} \frac{|w|_a}{|w|} \tag{16}$$

Also einfach Anzahl der as in Worten geteilt durch Gesamtzahl der Buchstaben. Damit bekommen wir Beispielsweise für

1. $L_1 = a^*$
2. $L_2 = ba^*$
3. $L_3 = (ab)^*$

folgende Ergebnisse:

$$\hat{P}_{L_1}(X = a) = 1 \tag{17}$$

$$\hat{P}_{L_2}(X = a) = 1 \tag{18}$$

$$\hat{P}_{L_3}(X = a) = \frac{1}{2} \tag{19}$$

Das hat aber einige Probleme, z.B.: Es ist nicht gesagt, dass der Wert in (16) überhaupt existiert; z.B. in

$$L_4 = \{a^{2^{2n}} : n \in \mathbb{N}\} \cup \{b^{2^{2n-1}} : n \in \mathbb{N}\}$$

In diesem Fall gibt es zwei Grenzwerte, nämlich $\frac{1}{3}$ und $\frac{2}{3}$, und die relative Häufigkeit oszielliert unendlich oft zwischen den beiden. Das bedeutet, der Wert ist noch nicht einmal definiert. Es gibt aber noch weitere Probleme: Nimm z.B. L_{prim-b} als die Sprache

$$L_{prim-b} = \{a_1...a_n : n \in \mathbb{N}, a_i = b \text{ falls } i \text{ eine Primzahl, } a_i = a \text{ andernfalls}\}$$

Natürlich gibt es hier einen Grenzwert, denn die Primzahlen sind *dünn* in den natürlichen Zahlen, und es gilt

$$\hat{P}_{L_{prim-b}}(X = a) = 1 \quad (20)$$

Wir sehen also hieran auch, dass unser Maß gewisse Arten von Komplexität komplett außer Acht lässt, denn die Entropie für L_3 wäre maximal, obwohl wir eine sehr einfache Sprache haben. Das liegt an einer gewissen *apriori*-Annahme, die wir unsere Schätzung sozusagen eingebaut haben, nämlich:

> Wir nehmen an, alle Ergebnisse sind voneineander unabhängig.

Diese Annahme ist aber bei formalen Sprachen nicht wirklich gerechtfertigt! Das bringt uns zum nächsten Punkt.

Die Wahrscheinlichkeit, dass X_i einen Wert annimmt, gegeben die Vorgängerwerte

Diesen Fall bezeichnen wir mit

$$P_L(X = a|w), \text{ für ein } w \in \Sigma^*$$

Wir schätzen das mittels

$$\hat{P}_L(X = a|w) = lim_{n\to\infty} \frac{|\{wax : x \in \Sigma^*\} \cap L_n|}{|\{wy : y \in \Sigma^*\} \cap L_n|} \quad (21)$$

Das ist also die Schätzung: Wie vorhersehbar ist die Sprache? Wenn wir alle Vorgänger kennen, können wir das nächste Ereignis vorhersagen? In diesem Fall haben wir folgende einfachen Ergebnisse:

$$\hat{P}_{L_1}(X = a|w) = 1 \quad (22)$$

$$\hat{P}_{L_2}(X = a|w) = 1 \text{ oder } 0 \quad (23)$$

$$\hat{P}_{L_3}(X = a|w) = 1 \text{ oder } 0 \quad (24)$$

$$\hat{P}_{prim-b}(X = a|w) = 1 \text{ oder } 0 \quad (25)$$

(26)

Das bedeutet: alle unsere Sprachen sind komplett vorhersehbar und haben also eine Entropie von 0 – es gibt keine Unsicherheit. Auch das ist allerdings nicht

wirklich informativ, denn wir können mit Sicherheit sagen dass L_3 weniger komplex ist als L_{prim-b}. Auf der anderen Seite gilt: die Sprache $\{a, b\}^*$ hat eine maximale Entropie, denn es gilt:

$$\hat{P}_{\{a,b\}^*}(X = a|w) = \frac{1}{2} \quad (27)$$

In diesem Fall liefert uns ein Präfix keinerlei Information, um den nächsten Buchstaben vorherzusagen, das ist also die Sprache mit der maximalen Entropie über $\{a, b\}$ – vorausgesetzt eben unsere Art und Weise, Wahrscheinlichkeiten zu schätzen. Wir sehen auch, wie die Art, die Wahrscheinlichkeit zu schätzen, die Entropie beeinflusst, z.B. bei $(ab)^*$.

7.4 Entropie und Kompression

Es gibt einen engen Zusammenhang von Entropie und Kompression, wenn wir Sprachen/Alphabete buchstabenweise kodieren möchten. Der Zusammenhang von Entropie und Kodierung (z.B. im Alphabet $\{0, 1\}$) beruht darauf, dass wir ein Symbol, was häufig ist ($\cong$ hohe Wahrscheinlichkeit) möglichst kurz kodieren, während relativ seltene Symbole eher lange Kodes bekommen. Auf diese Art sind unsere Kodes von Texten im Normalfall möglichst kurz. NB: Kodierung heißt in diesem Fall: wir kodieren jeden Buchstaben einzeln und unabhängig, das ganze ist also viel simpler als bei der Kolmogorov-Komplexität.

Seien Σ, T zwei Alphabete. Ein Buchstaben-Kode (von Σ in T) ist ein Paar

$$(\phi, X),$$

wobei

$$X \subseteq T^*, \text{ und } \phi : \Sigma \to X$$

eine Bijektion ist, so dass die homomorphe Erweiterung von

$$\phi : \Sigma^* \to X^*$$

eine Injektion ist. Mit Buchstaben-Kodes kodiert man Buchstabenweise, z.B. ein größeres Alphabet in einem kleineren (üblicherweise $\{0, 1\}$). Man kann aber auch allgemeinere Kodes betrachten, z.B. einen Kode

$$\phi : \Sigma \times (\Sigma \cup \{\ltimes\})^n \to T^*$$

definieren, wobei das erste Σ den Buchstaben gibt, der kodiert wird, und $(\Sigma \cup \{\ltimes\})^n$ die **Vorgängerbuchstaben**. Wir bekommen also eine Erweiterung auf Σ^*, die wie folgt aussieht:

$$\phi(a_1...a_i) = \phi(a_1, \ltimes^n)\phi(a_2, a_1\ltimes^{n-1})...\phi(a_i, a_{i-n}...a_{i-1}) \quad (28)$$

Wir nennen einen solchen Kode einen **Markov-Kode**. Das lässt sich noch weiter verallgemeinern. Sei

$$\phi : Q \times \Sigma \to T^*$$

eine Funktion wie gehabt, Q eine endliche Menge, $q_0 \in Q$, $\delta : Q \times \Sigma \times Q$ eine Funktion. Wir definieren weiterhin

$$\hat{\delta}(q, a_1...a_n) = \delta(...\delta(\delta(q_0, a_1), a_2)...), a_n) \quad (29)$$

Dann können wir eine Kodierungsfunktion $\psi : \Sigma^*$ definieren mittels:

$$\psi(a_1...a_n) = \phi(q_0, a_1)\phi(\hat{\delta}(q_0, a_1), a_2)....\phi(\hat{\delta}(q_0, a_{n-1}), a_n) \quad (30)$$

D.h. wie kodieren mittels eines endlichen Automaten; nennen wir einen solchen Kode **finite-state**.

Binäre Buchstabekodes und Entropie einer Sprache Ein Kode ist **präfixfrei**, falls es kein $x, y \in X$ gibt so dass

$$xz = y, \text{ wobei } z \in T^+.$$

Wir sind in der Informatik meist in Kodes über $\{0, 1\}^*$ interessiert, und wir möchten üblicherweise Alphabete kodieren, die mehr als zwei Buchstaben enthalten. Es stellt sich die Frage, wie man das am besten macht. Intuitiv ist unser Ziel: wir möchten, dass jede Kodierung eines Textes möglichst kurz wird. Das ist natürlich trivial, sofern wir nur die Buchstaben Σ haben. Aber nehmen wir an, wir haben eine Wahrscheinlichkeitsverteilung über Σ, und weiterhin, dass die Wahrscheinlichkeiten der Worte unabhänging voneinander sind. Das bedeutet:

- wenn ein Buchstabe sehr wahrscheinlich ist, dann wollen wir ihn kürzer kodieren,
- wenn er unwahrscheinlich ist, dann länger.

Sei $w \in \Sigma^*$. Wir bauen uns eine Zufallsvariable X, so dass $X(a) = |\phi(a)|$ (die Länge des Wortes). Was wir möchten ist: wir möchten den Erwartungswert E von X möglichst klein machen.Wir haben

$$\mathsf{E}(X) = \sum_{a \in \Sigma^*} |\phi(a)| \cdot P(a) \quad (31)$$

Jeder Buchstabe im Ausgangsalphabet Σ hat Länge 1; er wird – nach Erwartungswert – in der Kodierung im Schnitt mit $\mathsf{E}(X)$ Symbolen ersetzt. Deswegen nennen wir die *Inversion*

$$\frac{1}{\mathsf{E}(X)} \text{ den } \textbf{Kompressionsfaktor}$$

der Kodierung. Ein wichtiger Punkt ist nun:

$\mathsf{E}(X)$ kann niemals kleiner sein als die Entropie $H(P)$.

Das bedeutet wir müssen jedes Symbol im Schnitt mit mindestens $H(P)$ Zeichen kodieren.

Wir möchten im Allgemeinen den Erwartungswert minimieren, d.h. den Kompressionsfaktor maximieren. Es gibt einen einfachen Algorithmus, den sogenannten **Huffman code**, der folgendes liefert:

- Eingabe: ein beliebiges Alphabet Σ mit einer zugehörigen Wahrscheinlichkeitsfunktion $P : \Sigma \to [0, 1]$
- Ausgabe: eine Kodierung von Σ in $\{0, 1\}^*$ in einem Präfix-freien Kode mit maximalen Kompressionsfaktor (es gibt aber immer mehrere solcher Kodierungen).

Auch wenn das Thema nicht wirklich relevant ist, ist der Algorithmus ein Modell im Kleinen für das, was viele Lernalgorithmen machen.

Ein Beispiel Nehmen wir an, $\Sigma = \{a, b, c, d\}$, mit folgenden Wahrscheinlichkeiten (bzw. relativen Häufigkeiten):

- $P(a) = 0.1$
- $P(b) = 0.2$
- $P(c) = 0.3$
- $P(d) = 0.4$

Wir fangen damit an, das Buchstabenpaar zu nehmen, das am seltensten vorkommt. Das ist natürlich

$$\{a, b\} \text{ mit } P(\{a, b\}) = 0.3.$$

Wir ersetzen nun

$$\{a, b\} \mapsto \{x_1\},$$

so dass unser neues Alphabet ist

$$\{x_1, c, d\}, \text{ wobei } P(x_1) = 0.3.$$

Nun machen wir ebenso weiter: im neuen Alphabet ist das Buchstabenpaar mit der geringsten Wahrscheinlichkeit $\{x_1, c\}$, also ersetzen wir

$$\{x_1, c\} \mapsto \{x_2\}$$

mit dem resultierenden Alphabet

$$\{x_2, d\}, \text{ wobei } P(x_2) = P(\{x_1, c\}) = 0.6.$$

Nun machen wir den Schritt ein letztes Mal: das resultierende Alphabet ist

$$\{x_3\} \text{ mit } P(x_3) = 1.$$

Nun "entpacken" wir das ganze wieder. Wir nehmen an, x_3 wird kodiert durch das leere Wort ϵ. ϵ steht dann aber eigentlich für 2 Buchstaben: x_2 und d. Das erste ist wahrscheinlicher, also kodieren wir x_2, indem wir eine 0 an unser Kodewort hängen, d mit einer 1. Nun steht x_2 (bzw. 1) wiederum für zwei Buchstaben, und wir bekommen $x_1 = 00, c = 01$ (in diesem Fall ist es egal, die Wahrscheinlichkeiten sind gleich). Nun dasselbe mit x_1 (bzw. 00); in diesem Fall bekommen wir 000 für b, 001 für a. Wir bekommen also:

- $\phi(a) = 001$
- $\phi(b) = 000$
- $\phi(c) = 01$
- $\phi(d) = 1$

Wir nehmen nun X wie oben, und bekommen:

(32) $\mathsf{E}(X_\phi) = 0.1 \cdot 3 + 0.2 \cdot 3 + 0.3 \cdot 2 + 0.4 \cdot 1 = 1.9$

Der Kompressionsfaktor ist also $\frac{1}{1.9}$. Natürlich kommt dasselbe raus, wenn wir im Kode einfach 0 und 1 vertauschen. Wenn wir das vergleichen mit dem folgenden Block-Kode

- $\chi(a) = 00$
- $\chi(b) = 01$
- $\chi(c) = 10$
- $\chi(d) = 11$

(der auch Präfix-frei ist), dann bekommen wir

(33) $\mathsf{E}(X_\chi) = 0.1 \cdot 2 + 0.2 \cdot 2 + 0.3 \cdot 2 + 0.4 \cdot 2 = (0.1 + 0.2 + 0.3 + 0.4)2 = 2$

Der Kompressionsfaktor beträgt also nur $\frac{1}{2}$.

Wie ist die Entropie für P?

(34)
$$H(P) = -(0.1 \log_2(0.1) + 0.2 \log_2(0.2) + 0.3 \log_2(0.3) + 0.4 \log_2(0.4)) = 1.846439$$

Nehmen wir dagegen an, dass

$$P'(a) = \ldots = P'(d) = 0.25$$

ändert sich die Lage: in diesem Fall ist natürlich χ die optimale Kodierung. Die Entropie ändert sich wie folgt:

(35) $H(P') = -(4 \cdot (0.25 \log_2(0.25)) = 2$

Die Entropie ist größer, daher wird auch die Kompressionsrate schlechter sein. Am Ende gilt:

> Sei P eine Wahrscheinlichkeitsverteilung über Σ, ϕ ein Kode über $\{0, 1\}$. Dann kann der Kompressionsfaktor von ϕ niemals grösser sein als $\frac{1}{H(P)}$, berechnet zur Basis 2.

Markov-Kodes und bedingte Entropie Hier dieselbe Korrespondenz wie oben, nur dass wir eine bedingte Wahrscheinlichkeitsfunktion (Markov-Kette) und bedingte Entropie nehmen! Für finite-state Kodes funktioniert das nicht mehr ohne weiteres, denn hier gibt es nichts, was den Zuständen entsprechen würde. Das führt uns auf den nächsten Punkt:

7.5 Entropie und Kolmogorov-Komplexität

Was interessanter ist, ist die Frage: Gibt es so etwas wie die Wahrscheinlichkeit/Entropie/Informativität einer Sprache an und *an sich*? Um so etwas zu bekommen, bräuchte man eine *apriori*-Verteilung über Sprachen (also nicht über Worte). Das ist problematisch, denn es gibt überabzählbar unendlich viele Sprachen (über einem Alphabet), d.h. es gibt zuviele Sprachen, als dass wir Wahrscheinlichkeiten zuweisen könnten. Wir lösen dieses erste Problem wie folgt:

> Eine Sprache hat nur dann eine Wahrscheinlichkeit > 0, falls sie rekursiv aufzählbar ist.

Die Menge der rekursiv aufzählbaren Sprachen über einem Alphabet ist abzählbar, also wäre dieses Problem gelöst.

Die Wahrscheinlichkeit einer Sprache ist dann allerdings kein sinnvoller Begriff, genausowenig wie *die kürzeste Kodierung* einer Sprach/eines Wortes ein sinnvoller Begriff ist. Allerdings kann man, ebenso wie bei der Kolmogorov-Komplexität, einige plausible Annahmen treffen, die dieses Konzept sinnvoll werden lassen, und begleitet mit passenden Invarianztheoremen bekommen wir sinnvolle Ergebnisse. Dem liegen folgende Beobachtungen zugrunde:

1. (Rekursiv aufzählbare) Sprachen werden repräsentiert mit Ketten in $\{0,1\}^*$, nämlich der Gödelkodierung der TM, die diese Sprache erkennen.
2. Die Repräsentation hat – trotz vieler willkürlicher Entscheidungen, wie Gödelisierung – etwas allgemeingültiges, qua Invarianztheoremen.
3. Es gibt eine Familie von Wahrscheinlichkeitsverteilungen über $\{0,1\}^*$, die wir als *apriori maximal plausibel* bezeichnen können, nämlich da sie die maximale Entropie haben.

Punkt 3 müssen wir noch elaborieren: Wir nennen Funktionen $P : \mathbb{M} \to [0,1]$, die die Form haben

$$P_r(n) = (1-r)r^{n-1} \tag{36}$$

mit $r \in [0,1)$, **geometrische Verteilungen**. Jede geometrische Verteilung P_r hat den Erwartungswert $r/(1-r)$, denn es gilt unabhängig von r dass

$$\sum_{i=1}^{\infty}(1-r)r^{n-1} = r/(1-r) \tag{37}$$

Ein Spezialfall hiervon ist die Funktion

$$P_{0.5}(n) = \frac{1}{2^n} \tag{38}$$

die wir bereit kennengelernt haben, und die nach der letzten Gleichung also den Erwartungswert 1 besitzt. Die Wichtigkeit der geometrischen Verteilungen wird durch folgendes Ergebnis belegt:

Lemma 25 *Für jeden Wert $r/(1-r)$ ist die die geometrische Verteilung P_r die eindeutige Wahrscheinlichkeitsverteilung über $\mathbb{N}$ mit 1. diesem Erwartungswert und 2. der maximalen Entropie.*

Wir haben also eine Familie von Funktionen, die für ihren jeweiligen Erwartungswert die maximale Entropie haben. Wir können nun diese Verteilungen nutzen, um eine *apriori* Verteilung

$$P_0 : \{0,1\}^* \to [0,1]$$

zu definieren; wir sagen, für $w \in \{0,1\}^*$, $\|w\|_2$ ist der numerische Wert von w, gelesen als Binärzahl. Dann bekommen wir:

$$P_0(w) = \frac{1}{2^{\|w\|_2}} \tag{39}$$

Hiermit können wir die arpriori Wahrscheinlichkeit einer Sprache (gegeben ein bestimmtes Alphabet) bereits definieren; wir nennen die Funktion Q_0, und setzen:

$$Q_0(L) = \sum_{w=\langle M\rangle, L=L(M)} Q_0(w) \tag{40}$$

Wir bekommen die Wahrscheinlichkeit einer Sprache also als die Wahrscheinlichkeit, dass ein zufällig ausgewähltes Wort über $\{0,1\}$ ebendiese Sprache kodiert. Auf diese Art und Weise korrespondiert die Wahrscheinlichkeit mit der KK einer Sprache. Hier ergibt sich noch ein interessanter Zusammenhang: wir haben

$$\log_2(\frac{1}{2^n}) = n \tag{41}$$

Das bedeutet soviel wie: die Informativität eine Sprache, auf Basis unserer Verteilungen P_0 und Q_0, fallen wiederum zusammen mit ihrer Kolmogorov-Komplexität (modulo logarithmische Transformation).

8 Sprachen und Monoide

Was ist eine (formale) Sprache? Wir sind gewohnt, formale Sprachen aufzufassen im Kontext formaler Grammatiken; aber es ist wichtig, sich darüber im klaren zu sein dass Sprachen unabhängig von Grammatiken oder Automaten existieren, und in der Tat gibt es Sprachen, die von keiner Grammatik erzeugt, und von keinem Automaten erkannt werden. Eine erste Antwort ist: eine Sprache ist eine Menge, nämlich eine Menge von Zeichenketten. Aber nicht jede Menge ist eine Sprache; wenn wir die Menge $\{\{1\}, \emptyset\}$ bertrachten, würden wir nicht auf die Idee kommen sie als Sprache zu bezeichnen. Um das Konzept der Sprache *wirklich* zu verstehen, müssen wir zunächst das Konzept des **freien Monoids** verstehen.

Definition 26 *Ein Monoid ist ist eine Struktur $(M, \cdot, 1)$, wobei M eine Menge ist (die sog. Trägermenge), so dass*

1. *M abgeschlossen ist unter $\cdot$, d.h. falls $m, n \in M$, dann ist $(m \cdot n) \in M$;*
2. *$\cdot$ ist assoziativ; d.h. $m \cdot (n \cdot o) = (m \cdot n) \cdot o$ f.a. (für alle) $m, n, o \in M$;*
3. *$1 \in M$ ist das neutrale Element von $\cdot$, d.h. f.a. $m \in M$, $1 \cdot m = m \cdot 1 = m$.*

Monoide sind geradezu allgegenwaïtig in unserem Alltag; Beispiele sind $(\mathbb{N}_0, +, 0)$, die natürlichen Zahlen mit 0 und der Operation der Addition; $(\mathbb{N}, \cdot, 1)$, die natürlichen Zahlen (mit oder ohne 0) und der Operation der Multiplikation.

Frage: $(\mathbb{N}_0, -, 0)$, $-$ die Subtraktion, ist *kein* Monoid; warum? $\mathbb{Z}$ denotiert die Menge der ganzen (positiven, negativen, 0) Zahlen. $(\mathbb{Z}, -, 0)$ ist ebenfalls kein Monoid. Warum?

Objekte der Form $m \cdot n$ bezeichnet man auch als *Produkte*, oder etwas allgemeiner als Terme. Wir kommen nun zum freien Monoid. Die algebraische Definition ist etwas abstrakt, aber das Konzept lässt sich sehr gut intuitiv erklären.

Definition 27 *Ein Monoid ist frei, wenn es für jedes $m \in M$ genau ein endliches Produkt $m_1 \cdot ... \cdot m_n$ gibt, so dass $m_1 \cdot ... \cdot m_n = m$.*

Was diese Definition besagt ist soviel wie: zwei Objekte in M sind genau dann gleich, wenn sie syntaktisch gleich sind (gleich aussehen). Wenn wir etwa den Monoid $(\mathbb{N}_0, +, 0)$ betrachten, haben wir $3 = 2 + 1 = 2 + 1$. Es gibt also verschiedene Terme, die dasselbe Objekt denotieren (NB: wir fassen die Gleichheit $'='$ hier als etwas semantisches auf!), also ist der Monoid nicht frei.

Im freien Monoid ist syntaktische Gleichheit (Terme sehen gleich aus) äquivalent mit semantischer Gleichheit (Terme denotieren dasselbe Objekt). Deswegen lässt man die das $'\cdot'$ kurzerhand weg; die resultierenden Terme haben also die Form $m_1 m_2 ... m_n$, und wir bezeichnen diese Terme kurzerhand als **Wörter**. Das neutrale Element des freien Monoids ist ϵ, und ist in diesem Fall der *leere Term*, oder das *leere Wort* (es muss leer sein, sonst verlieren wir die Eindeutigkeit!). Für den freien Monoid $(M, \cdot, \epsilon)$ ist M also eine Menge von *Wörtern*; die atomaren Terme $N \subseteq M$ bezeichnet man dementsprechend als **Buchstaben**, und

man sagt auch dass M der freie Monoid ist der von N **generiert** wird; man kann schreiben: $M = N^*$; das bedeutet: M ist der Abschluss von N unter Konkatenation, der Operation des freien Monoids.

Definition 28 *Sei L eine Menge. L ist ein **Sprache**, falls $L \subseteq N^*$, wobei N^* der freie Monoid ist der von einer (endlichen) Menge N generiert wird.*

Sprachen sind also Teilmengen eines freien Monoids.

9 Monoide und Relationen

Dieser Abschnitt ist für interessierte und wird eigentlich erst später relevant (wenn überhaupt). Der freie Monoid ist der wichtigste Monoid für die Theorie der formalen Sprachen. Für die Theorie der Automaten gibt aber es noch einen Monoid, der mindestens ebenso wichtig ist, nämlich der Monoid der Relationen. Wichtig ist, dass dieser Monoid sich unterscheidet von den Relationen, die wir später als Erweiterung von Sprachen betrachten. Zunächst kommen wir zur Definition von Relationen.

Definition 29 *Seien M, N zwei beliebige Mengen, $M \times N := \{\langle m, n\rangle : m \in M, n \in N\}$ deren kartesisches Produkt. Eine Menge R ist eine **Relation**, falls es Mengen M, N gibt, so dass $R \subseteq M \times N$.*

Relationen sind also Mengen von geordneten Paaren. Die **Basis** einer Relation $R \subseteq M \times N$ ist $M \cup N$. Gegeben zwei Relationen R, Q, definieren wir deren **Komposition** als $R \circ Q := \{\langle m, o\rangle : \langle m, n\rangle \in R, \langle n, o\rangle \in Q\}$. Komposition von Relationen ist assoziativ; das neutrale Element für Komposition von Relationen ist die Identitätsrelation (genauer gesagt: Funktion), $id_M = \{\langle m, m\rangle : m \in M\}$.

Wir können nun einen **Relationen-Monoid** definieren:

Definition 30 *Ein Relationen-Monoid ist eine Struktur $(\mathcal{R}, \circ, id_M)$, wobei*

1. *$\mathcal{R}$ eine Menge von Relationen ist mit Basis M,*
2. *falls $R_1, R_2 \in RA$, dann $R_1 \circ R_2 \in \mathcal{R}$,*
3. *$\circ$ ist Komposition von Relationen.*

Nach allem was wir gesagt haben, ist klar dass Relationen-Monoide Monoide sind.

Aufgabe: Konstruieren Sie einen beliebigen endlichen und einen beliebigen unendlichen Relationen-Monoid.

Im nächsten Abschnitt werden wir verstehen, warum Relationen-Monoide derart wichtig sind für die Theorie der Automaten.

10 Homomorphismen

Auch dieses Kapitel ist eher für Interessierte und kann zunächst übersprungen werden. Ein (Monoid-)Homomorphismus ist eine Abbildung von einem Monoid auf einen Monoid, der *die Struktur erhält.* Da wir nur Monoid-Homomorphismen betrachten werden, lassen wir das Präfix weg.

Definition 31 *Seien $(M, \cdot, 1_M), (N, \cdot, 1_N)$ zwei Monoide. Ein **Homomorphismus** ist eine Abbildung $\phi : M \to N$, für die gilt: $\phi(m_1 \cdot ... \cdot m_n) = \phi(m_1) \cdot ... \cdot \phi(m_n)$.*

Aus der Definition folgt bereits dass $\phi(1_M) = 1_N$, denn f.a. $m \in M$ gilt: $\phi(m) = \phi(m \cdot 1_M) = \phi(m) \cdot \phi(1_N)$. Freie Monoide zeichnen sich durch übrigens durch folgende Eigenschaft aus:

Satz 32 *Sei N eine beliebige Menge, und M die Trägermenge eines Monoids. Ein Monoid $(M, \cdot, \epsilon)$ ist ein freier Monoid genau dann wenn es es für jede Abbildung $f : M \to N$ einen Homomorphismus ϕ gibt, so dass f.a. $m \in M$, $\phi(m) = f(m)$.*

Jede Abbildung (Funktion) aus dem freien Monoid lässt sich also als Homomorphismus auffassen. Diese Eigenschaft ist derart charakteristisch, dass sie auch manchmal als Definition benutzt wird.

11 Semi-Automaten als Homomorphismen

Der Grund warum wir diese Dinge besprechen ist folgender: sei ein (endlicher) Semi-Automat ein Tupel $\mathfrak{A} = (\Sigma, Q, \delta)$, Σ ein Eingabealphabet, Q ein Zustandsmenge, und $\delta \subseteq Q \times \Sigma \times Q$ die Übergangsrelation (ein Semiautomat ist also ein endlicher Automat ohne Startzustand und Endzustände). Wenn wir δ kanonisch erweitern auf Worte zu δ^* (siehe unten), dann ist ein Semiautomat nichts anderes als ein Homomorphismus $\phi_{\mathfrak{A}} : \Sigma^* \to Q \times Q$, der den freien Monoid in einen (endlichen) Relationen-Monoid abbildet, wobei $\phi_{\mathfrak{A}}(w) = \{q_i, q_j : (q_i, w, q_j) \in \delta^*$. In diesem abstrakten Sinne ist tatsächlich die gesamte Automatentheorie die Theorie der Abbildungen aus dem freien Monoid in einen Relationen-Monoid.

12 Endliche Automaten: ein Überblick

12.1 Definition

Wir schauen uns zunächst einige der grundlegendsten Ergebnisse über endliche Automaten an. Da einerseits die Beweise größtenteils recht einfach sind, und andererseits an vielen Orten nachgelesen werden können, lassen wir sie meistens aus.

Ein endlicher Automat ist ein Tupel $(\delta, Q, q_0, F, \Sigma)$; Q ist eine endliche Menge von Zuständen; $q_0 \in Q$ ist der Startzustand; $F \subseteq Q$ ist die Menge der akzeptierenden Zustände; Σ ist ein endliches Eingabealphabet, und $\delta \subseteq Q \times \Sigma \times Q$

ist die Übergangsrelation. Intuitiv können wir sagen, dass der Automat in einem Zustand ist, und wenn er einen Buchstaben liest geht er in einen anderen Zustand; in Symbolen: $\delta(q_i, a) = Q_j$, wobei q_i ein Zustand ist, $a \in \Sigma$, und Q_j eine *Menge* von Zuständen ist. Wir haben eine Menge, denn δ ist nicht notwendig eine Funktion, kann also für eine Eingabe mehrere Ausgaben haben. Wir können δ einfach von Buchstaben auf Worte erweitern; wir schreiben für diese Erweiterung δ^*, und $\vec{w}$ für Zeichenketten in Σ^*, also beliebige Reihen von Zeichen aus Σ. Die Erweiterung ist wie folgt definiert: $\delta^*(q_i, a) = \delta(q_i, a)$, und $\delta^*(q_i, a\vec{w}) = \delta^*(q_j, \vec{w})$, für $q_j \in \delta(q_i, a)$. NB: wir werden das erweiterte δ nicht immer gesondert notieren. Wir sagen, dass ein Automat $\mathfrak{A}$ ein Wort $\vec{w}$ akzeptiert, wenn $\delta^*(q_0, \vec{w}) \cap F \neq \emptyset$; das heißt, es gibt eine Berechnung des Automaten von q_0 nach F, bei der die Eingabe $\vec{w}$ gelesen wird. Wir definieren $L(\mathfrak{A}) := \{\vec{w} : \delta^*(q_0, \vec{w}) \cap F \neq \emptyset\}$.

12.2 Reguläre Sprachen/Grammatiken

Eine **reguläre Grammatik** ist ein Tupel $\langle S, \mathcal{N}, \Sigma, R\rangle$; $S \in N$, N eine endliche Menge von Nichtterminalen, Σ ein endliches Alphabet, und R eine endliche Menge von Regeln der Form: $N \to aM$, für $N, M \in \mathcal{N}$, $a \in \Sigma$. Eine Ableitung ist eine Folge von Regeln (man sagt auch: Regelanwendungen), $\langle R_i\rangle : i \leq k$, wobei die linke Seite der Regel R_0 S ist, das Nichtterminal der rechten Seite jeder Regel R_i, $0 \leq i < k$, ist gleich der linken Seite der Regel R_{i+1}; und die rechte Seite der Regel R_k ist in Σ. Eine Grammatik $\mathfrak{G}$ leitet ein Wort $a_1...a_n$ ab, falls es eine Ableitung A gibt, so dass für $1 \leq i \leq n$, das Terminal auf der rechten Seite von R_i jeweils a_i ist. Mit $L(\mathfrak{G})$ meinen wir die Menge aller ableitbaren Wörter, oder einfach die Sprache von $\mathfrak{G}$.

Satz 33 *Sei $L \subseteq \Sigma^*$ eine Sprache. Es gibt einen endlichen Automaten $\mathfrak{A}$, für den gilt $L(\mathfrak{A}) = L$, genau dann wenn es eine reguläre Grammatik gibt, für die gilt: $L(\mathfrak{G}) = L$.*

Aus diesem Grund nennt man die Sprachen, die von endlichen Automaten erkannt werden, **reguläre Sprachen**.

Ein Automat $\mathfrak{A}$ heißt **deterministisch**, falls $\delta_{\mathfrak{A}}$ eine Funktion ist, das heißt: für einen Zustand und eine Eingabe geht der Automat in genau einen Zustand. Determinismus ist eine vorteilhafte Eigenschaft; sie impliziert unter anderem, dass die Berechnungen des endlichen Automaten in linearer Zeit stattfinden.

Satz 34 *Für jeden endlichen Automaten $\mathfrak{A}$ gibt es einen deterministischen endlichen Automaten $\mathfrak{A}'$, so dass $L(\mathfrak{A}) = L(\mathfrak{A}')$.*

Der Beweis ist relativ einfach; wir haben bereits oben gesehen dass ein nichtdeterministischer Automat immer in eine Menge von Zuständen geht. Da es nur endlich viele Zustände gibt, sind auch diese Mengen immer endlich. Wir können den Automaten determinisieren, indem wir als Zustandsmege die *Potenzmenge* der Zustände $\wp(Q)$ nehmen; d.h. die Menge aller Teilmengen von Q. Da alle

Teilmengen darin enthalten sind, können wir sicher sein, dass der Automat deterministisch ist. Eine Kehrseite dieser Prozedur ist die folgende: wie man weißist der Betrag der Potenzmenge exponentiell größer: $|\wp(Q)| = 2^{|Q|}$. Wir haben also eine sehr hohe Obergrenze für die Zustandsanzahl des resultierenden Automaten. Etwas überraschend ist folgendes Ergebnis: diese Obergrenzen kann im Allgemeinen nicht verbessert werden. D.h., auch wenn im Einzelfall der resultierende Automat wesentlich kleiner sein kann, ist er im schlimmsten Fall so großwie es die Obergrenze angibt.

Das bedeutet beispielsweise, dass ein nichtdeterministischer Automat (NA) mit der moderaten Menge von 100 Zuständen, wenn man ihn determinisieren würde, in einem deterministischen Automaten (DA) mit nicht weniger als 2^{100} Zuständen resultieren würde - weit mehr als es Teilchen im Universum gibt ($\sim 2^{84}$). Das erklärt, warum auch NA nach wie vor interessante Forschungsgegenstände sind, auch wenn sie im Vergleich zu DA nachteilhafte Eigenschaften haben: die Determinisierung ist oft einfach nicht praktikabel. Das gilt übrigens auch, wenn der minimale deterministische Automat womöglich praktikabel wäre: das Problem ist dass wir einen Zwischenschritt haben (nach der Determinisierung, vor der Minimisierung), in dem der Automatat zu groß ist.

12.3 Reguläre Ausdrücke

Reguläre Ausdrücke über ein Alphabet Σ sind wie folgt definiert:

1. wenn $a \in \Sigma$, dann ist a ein regulärer Ausdruck;
2. wenn A, B reguläre Ausdrücke sind, dann ist $(A \cdot B)$ ein regulärer Ausdruck;
3. wenn A, B reguläre Ausdrücke sind, dann ist $(A + B)$ ein regulärer Ausdruck; und
4. wenn A ein regulärer Ausdruck ist, dann ist (A^*) ein regulärer Ausdruck.

Reguläre Ausdrücke werden *interpretiert* als Mengen von Zeichenketten; sie sind also nur Beschreibungen von Sprachen und sollten nicht verwechselt werden mit den Sprachen selbst, die sie denotieren. Die Bedeutung eines regulären Ausdrucks ist induktiv wie folgt definiert:

1. $\|a\| := \{a\}$
2. $\|A \cdot B\| := \{\vec{w}\vec{v} : \vec{w} \in \|A\| \text{ und } \vec{v} \in \|B\|\}$;
3. $\|A + B\| := \|A\| \cup \|B\|$;
4. $\|(A)^*\| := \bigcup_{i \geq 0} \|A^i\|$.

Meist wird die Verknüpfung $\cdot$ weggelassen, d.h. $a \cdot b \equiv ab$; wir schreiben sie hier aus, um uns klarzumachen dass es sich um eine Verknüpfung handelt, die verschieden ist von der Konkatenation in der denotierten Sprache. $+$ wird oft auch als $|$ geschrieben. Ein wichtiges Theorem von Kleene besagt:

Satz 35 *(Kleene) Es genau dann einen regulären Ausdruck R, so dass $\|R\| = L$, wenn es einen endlichen Automaten $\mathfrak{A}$ gibt, so dass $L(\mathfrak{A}) = L$.*

12.4 Satz von Myhill-Nerode, Minimisierung

Ein Satz, der vielleicht wie kein anderer das Wesen der endlichen Automaten/regulären Sprachen beschreibt, ist der Satz von Myhill-Nerode. Sei $L \subseteq \Sigma^*$ eine Sprache, und $\vec{w}, \vec{v} \in \Sigma^*$.

Definition 36 *Wir schreiben $\vec{x} \sim_L \vec{y}$, wenn gilt: für alle $\vec{z} \in \Sigma^*$, $\vec{x}\vec{z} \in L$ genau dann wenn $\vec{y}\vec{z} \in L$.*

Man nennt $\sim_L$ auch die Nerode-Äquivalenz; sie ist eine Äquivalenzrelation, und sie partitioniert Σ^* in eine Menge von Äquivalenzklassen. Für die Partitionierung von einer Menge in Klassen schreiben wir $[A]_\sim$. Die Elemente von $[\Sigma]_{\sim_L}$ sind die Mengen M_i von Zeichenketten, für die gilt: f.a. $\vec{w}, \vec{v} \in M_i, \vec{w} \sim_L \vec{v}$. Wir schreiben $|A|$ um die Anzahl der Elemente einer Menge A zu bezeichnen. Der Satz von Myhill Nerode besagt nun:

Satz 37 *Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn $|[\Sigma^*]_{\sim_L}|$ endlich ist.*

D.h. eine Sprache ist genau dann regulär, wenn es nur endlich viele nichtäquivalente Präfixe gibt. Es gibt noch eine Verfeinerung, die die Enge Relation von Automaten und Äquivalenzklassen noch deutlicher zeigt:

Satz 38 *Sei L regulär. Dann ist $|[\Sigma^*]_{\sim_L}|$ die Anzahl der Zustände des kleinsten deterministischen endlichen Automaten $\mathfrak{A}$, für den gilt: $L(\mathfrak{A}) = L$.*

.

Ein Satz, den wir leicht daraus ableiten können, ist der folgende:

Satz 39 *Für jeden endlichen Automaten $\mathfrak{A}$ können wir einen Automaten $\mathfrak{A}'$ konstruieren, so dass $L(\mathfrak{A}) = L = (\mathfrak{A}')$, und $\mathfrak{A}'$ ist der kleinste DA, der diese Sprache erkennt.*

Wir sollten noch zeigen, wie diese **Minimisierung** effektiv funktioniert. Wir werden ohne weiteres davon ausgehen, dass unser Automat bereits deterministisch ist (wenn nicht, kann er vorher determinisiert werden). Wenn man einen Automaten minimisieren möchte, muss man zunächst herausfinden, ob es Zustände gibt, die äquivalent sind. Der Satz von Myhill-Nerode und die Korrespondenz von Äquivalenzklassen sagt uns, dass zwei Zustände q_i, q_j äquivalent sind, wenn wir von ihnen aus die gleiche Sprache erkennen können; d.h. wenn man den Ausgangsautomaten dahingehend ändert, dass einmal q_i, einmal q_j der *Startzustand* ist, dann sind die beiden resultierenden Automaten äquivalent.

Es gibt einige sehr einfache Algorithmen, um das zu prüfen. Wir führen hier den folgenden an:

- Stelle eine Tabelle auf mit allen Paaren von Zuständen q_i, q_j, für $q_i \neq q_j$.
- Markiere alle Paare, bei denen genau ein Zustand akzeptierend ist.
- Für jedes unmarkierte Paar q_i, q_j, prüfe für alle $a \in \Sigma$, ob das Paar der Nachfolgezustände $\delta(q_i, a), \delta(q_j, a)$ markiert ist. Falls ja, dann markiere auch q_i, q_j.
- Wiederhole diesen Schritt, bis sich nichts mehr in der Tabelle ändert in einem kompletten Durchlauf.
- Alle Paare, die nicht markiert sind, sind äquivalent und werden zu einem Zustand zusammengefasst (NB: Äquivalenz ist transitiv, es werden also möglicherweise mehr als 2 Zustände in einen zusammengefasst!)

Ein besonderer Fall, der dadurch ebenfalls abgedeckt wird, sind **absorbierende Zustände**: Zustände die man nicht mehr verlassen kann. In einem Automaten braucht man also jeweils maximal zwei solche Zustände: evtl. einen akzeptierenden, und einen nicht akzeptierenden.

12.5 ϵ-Übergänge

Wir sagen, dass ein Automat ϵ-Übergänge hat, wenn wir in δ Übergänge der Form (q_i, ϵ, q_j) haben. D.h. der Automat kann den Zustand wechseln, ohne einen Buchstabe zu lesen. Ebenfalls leicht zu zeigen ist der folgende Satz:

Satz 40 *Für jeden endlichen Automaten $\mathfrak{A}$ können wir einen endlichen Automaten $\mathfrak{A}'$ ohne ϵ-Übergänge konstruieren.*

Wir können zwei Zustände, die durch einen ϵ-Übergang verbunden sind, kurzerhand zusammenlegen.

12.6 Boolesche Hülle

Satz 41 *Seien $\mathfrak{A}_1$, $\mathfrak{A}_2$ zwei endliche Automaten, $L(\mathfrak{A}_1) = L_1$, $L(\mathfrak{A}_2) = L_2$. Wir können endliche Automaten $\mathfrak{A}_3$, $\mathfrak{A}_4$, $\mathfrak{A}_5$ konstruieren, für die gilt: $L(\mathfrak{A}_3) = L_1 \cup L_2$, $L(\mathfrak{A}_4) = L_1 \cap L_2$, $L(\mathfrak{A}_5) = \Sigma^*/L_1$.*

Das bedeutet, dass reguläre Sprachen abgeschlossen sind unter mengentheoretischer Vereinigung, Schnitt und Komplement. Da dies die drei Operationen der Booleschen Algebra sind, spricht man von Boolescher Hülle; die Klasse der regulären Sprachen bildet also eine Boolesche Algebra.

12.7 Das Pumplemma für reguläre Sprachen

Es gibt ein sogenanntes Pumplemma der regulären Sprachen. Dieses Lemma ist von großer Wichtigkeit, wenn wir zeigen wollen dass eine gegebene Sprache nicht regulär ist. Die übliche Formulierung ist folgende:

Lemma 42 *Sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Dann gibt es ein $k \in \mathbb{N}$, so dass für alle $w : |w| \geq k$ gilt: es gibt $x, z \in \Sigma^*$, $y \in \Sigma^+$, so dass $xyz = w$, und f.a. $n \in \mathbb{N}$ gilt: $xy^n z \in L$.*

Die Teilkette y kann also "aufgepumpt" werden, daher der Name. Beachten sie, dass das Lemma nur für Ketten gilt, die eine gewisse Länge haben. Der Beweis läuft wie folgt: wenn L regulär ist, dann gibt es einen Automaten $\mathfrak{A}$ so dass $L = L(\mathfrak{A})$. Da $\mathfrak{A}$ endlich ist, gibt es ein $k \in \mathbb{N}$ so dass $\mathfrak{A}$ k-viele Zustände hat. Wenn nun $\mathfrak{A}$ ein Wort w akzeptiert mit $|w| \geq k + 1$, dann muss $\mathfrak{A}$ in der Berechnung zweimal im selben Zustand gewesen sein. Es gibt also ein Teilwort y von w, dass $\mathfrak{A}$ von q_i nach q_i führt. Daraus folgt: ich kann dieses Teilwort weglassen oder beliebig iterieren, ohne dass es etwas an der Akzeptanz des Wortes verändert.

Das Pumplemma ist sehr wichtig, um zu zeigen dass eine Sprache nicht regulär ist; denn dafür reicht es nun zu zeigen, dass sie das Pumplemma nicht erfüllt. Nehmen wir etwa die Sprache $L_1 = \{a^n b^n : n \in \mathbb{N}\}$. Nehmen wir an, sie ist regulär. Dann erfüllt sie dass Pumplemma. Wir nehmen ein beliebiges k. Wir wissen natürlich, dass es $w \in L$ gibt, so dass $|w| \geq k$. Nun nehmen wir eine Teilkette. Es gibt 3 Möglichkeiten: 1. $w \in a^+$. In dem Fall, wenn wir die Kette iterieren, dann haben wir nicht mehr gleichviele as und bs in der resultierenden Kette, also kann sie nicht in L_1 sein. 2. $w \in b^+$: dasselbe Argument. 3. $w \in a^+b^+$. In diesem Fall führt die Iteration dazu, dass wir eine Abfolge $a...b...a...b..$ bekommen. Das ist in L nicht möglich, also kann die resultierende Kette nicht in L_1 sein. Wir haben also die Annahme, dass L_1 regulär ist, auf einen Widerspruch geführt; also ist L_1 nicht regulär.

NB: das Pumplemma ist eine notwendige Bedingung für reguläre Sprachen, keine hinreichende. Wir bezeichen mit $|w|_a$ die Anzahl von as in w. Wir definieren $L_2 = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$. L_2 erfüllt das Pumplemma. *Aufgabe*: zeigen Sie warum! Aber L_2 ist damit noch nicht regulär! Wir zeigen dass wie folgt: wir wissen dass reguläre Sprachen abgeschlossen sind unter Schnitt, und dass a^*b^* regulär ist. Daher bilden wir die Sprache $L_2 \cap a^*b^*$; nach Annahme ist diese Sprache regulär. Wir haben aber $L_2 \cap a^*b^* = L_1$, und L_1 ist nicht regulär. Also ein Widerspruch, daher ist die Annahme, dass L_2 regulär ist, falsch.

12.8 Anmerkungen und Literatur

Als Anmerkung kann man sagen, dass diese Ergebnisse zeigen, dass die regulären Sprachen sehr vorteilhafte Eigenschaften haben, und dass es sehr viele Arten gibt, sie zu charakterisieren (wir haben nur die geläufigsten erwähnt; es gibt noch einige weitere). Man muss dazusagen, dass die regulären Sprachen die mächtigste Klasse ist, die derart wohlgesonnen ist; für alle mächtigeren Klassen lassen sich keine vergleichbaren Ergebnisse mehr erziehlen. Die regulären Sprachen werden oft als die grundlegendste aller Sprachklassen vorgestellt, die in allen interessanten Klassen enthalten ist. Das ist aber vollkommen falsch; es gibt eine Reihe interessanter Klassen, die echt in den regulären Sprachen enthalten ist;

wir werden einige dieser Klassen später besprechen.

Alle hier vorgestellten Ergebnisse (mit Beweisen) lassen sich im Internet (Wikipedia) oder in den meisten Büchern über Automatentheorie nachlesen. Endliche Automaten sind ein extrem gut erschlossenes Forschungsgebiet, im Gegensatz zu Transduktoren.

13 Produkt-Alphabete und Relationen

Wir haben immer verlangt, dass unser Eingabealphabet endlich ist. Wir können aber natürlich ein endliches Alphabet Σ nehmen, und definieren: $\Sigma' := \Sigma \times \Sigma$. D.h., wir haben mit $\Sigma' := \{(a,b) : a,b \in \Sigma\}$ eine Menge von Paaren, die wiederum endlich ist; und dementsprechend sind alle unsere Ergebnisse auch gültig wenn wir Automaten über Σ' definieren. Unsere Worte über Σ'^* sind dann Ketten von geordneten Paaren $(a_1,b_1)(a_2,b_2)(a_3,b_3)...$ (NB: die Subskripte sagen nur etwas über die Position aus, nicht über die Identität des Buchstaben selber). Wir definieren die Verknüpfung $'\cdot'$ des freien Monoids auf eine etwas andere Art und Weise, wenn wir ein Alphabet der Form $\Sigma \times \Sigma$ haben: für $(a_1,b_1),(a_2,b_2) \in \Sigma \times \Sigma$, $(a_1,b_1) \cdot (a_2,b_2) = (a_1a_2, b_1b_2)$. Streng genommen haben wir damit eine andere Verknüpfung, aber wir schreiben sie gleich; das bedeutet, wann immer wir ein Produkt-Alphabet haben, dann benutzen wir $\cdot$ in dem Sinne wie es hier definiert wurde.

Das bedeutet, dass wir nun einen neuen Monoid haben, dessen Elemente nicht *Worte*, sondern *Relationen* sind: $((\Sigma \times \Sigma)^*, \cdot, (\epsilon,\epsilon))$ (das ist nur eindeutig definiert wenn wir ein gegebenes Σ haben). Aus der Tatsache, dass $(\epsilon,\epsilon) \in \Sigma \times \Sigma$ (wir brauchen ein neutrales Element) können wir folgern, dass $\epsilon \in \Sigma$. Daraus folgt wiederum, dass auch für $a \in \Sigma$, $a \neq \epsilon$, $(a,\epsilon),(\epsilon,a) \in \Sigma \times \Sigma$. Aus dieser Tatsache lassen sich folgende Ergebnisse ableiten:

Lemma 43 *Falls $\epsilon \in \Sigma$, dann ist $(\Sigma \times \Sigma)^* = \Sigma^* \times \Sigma^*$*

Beweis. Nach Annahme haben wir $\epsilon \in \Sigma$. Nehmen wir ein beliebiges paar $(w,v) \in \Sigma^* \times \Sigma^*$. Sei $w = a_1...a_n$. Dann bilden wir $(w,\epsilon) = (a_1,\epsilon) \cdot ... \cdot (a_n,\epsilon)$. Auf die gleiche Art bilden (ϵ,v); also sind $(w,\epsilon),(\epsilon,v) \in (\Sigma \times \Sigma)^*$. Wir haben aber $(w,\epsilon)\cdot(\epsilon,v) = (w,v)$, also gilt $(w,v) \in (\Sigma \times \Sigma)^*$. Das zeigt dass $(\Sigma \times \Sigma)^* \supseteq \Sigma^* \times \Sigma^*$.

Die Richtung $(\Sigma \times \Sigma)^* \subseteq \Sigma^* \times \Sigma^*$ ist trivial. ⊣

Eine weitere Folge ist aber:

Lemma 44 *$\mathcal{R} = ((\Sigma \times \Sigma)^*, \cdot, (\epsilon,\epsilon))$ ist kein freier Monoid.*

Proof. Erinnern wir uns an die Definition des freien Monoids: damit $\mathcal{R}$ frei ist, müsste jeder Term eindeutig als Produkt von Elementen in $\Sigma^* \times \Sigma$ darstellbar sein. Wir haben aber nach Annahme $\epsilon \in \Sigma$. Damit können wir sehr leicht ein Gegenbeispiel konstruieren: wir haben $(a,b) = (\epsilon,b)\cdot(a,\epsilon) = (a,\epsilon)\cdot(\epsilon,b)$, wobei $(\epsilon,b),(a,\epsilon),(a,b) \in \Sigma \times \Sigma$. Es gibt also drei verschiedene Produkte, die denselben Term repräsentieren. ⊣

Wohlgemerkt: beide Lemmas beruhen auf der Tatsache dass $\epsilon \in \Sigma$; andernfalls ist die Gleichung in Lemma 17 falsch. (Übungsaufgabe: warum? Und gilt noch eine Inklusion?) Auch Lemma 18 beruht auf dieser Annahme. In diesem Fall ist die Annahme aber in gewissem Sinne notwendig: denn wenn $\epsilon \notin \Sigma$, dann ist $(\epsilon \times \epsilon) \notin \Sigma \times \Sigma$, und der Monoid hat kein neutrales Element.

Die Annahme dass $\epsilon \in \Sigma$ folgt also letztenendes aus der komplizierteren Definition von $\cdot$. Allerdings sind die Dinge etwas komplizierter, und wir können diese Annahme umgehen. Per Definition ist die Kleene Hülle $\Sigma^* := \bigcup_{n \in \mathbb{N}_0} a_1 ... a_n$: für beliebige $a_i \in \Sigma$. Wir haben also stets dass leere Wort in einer Menge dieser Form. Nehmen wir also an, dass $\epsilon \notin \Sigma$, und definieren $1_{\Sigma\times\Sigma}$ als $(\Sigma \times \Sigma)^0$, als leeres Paar. Das ist ein neutrales Element für unsere Konkatenation, und so haben wir den Monoid: $((\Sigma \times \Sigma)^*, \cdot, 1_{\Sigma\times\Sigma})$. Dieser Monoid wird generiert von Tupeln $(a, b) \in \Sigma \times \Sigma$, $a \neq \epsilon \neq b$. Er enthält alle $(w, v) \in \Sigma^* \times \Sigma^*$, so dass $|w| = |v|$ (zur Erinnerung: $|-| : \Sigma^* \to \mathbb{N}$ gibt die Länge einer Kette an). Dieser Monoid der **synchronen** Relationen ist in der Tat frei und hat einige positive Eigenschaften; wir werden ihn einer leicht verallgemeinerten Form später kennenlernen.

14 Zylinder und Projektionen

Wir definieren die *erste Projektion* $\pi_1((a_1, a_1')(a_2, a_2')(a_3, a_3')...) = a_1 a_2 a_3 ...$, die uns die Kette aller linken Komponenten der Kette von Paaren gibt, und parallel dazu *zweite Projektion*, $\pi_2((a_1, a_1')(a_2, a_2')(a_3, a_3')...) = a_1' a_2' a_3' ...$, die die Kette aller rechten Komponenten liefert. (NB: die Nummerierung bezieht sich auf die Position in der Kette, die Symbole sind also möglicherweise, aber nicht notwendig verschieden).

Wir gehen zurück zu Σ; sei $L \subseteq \Sigma^*$ eine reguläre Sprache. Wir definieren den Σ-**Zylinder** von L als die Menge aller Ketten $Z_1(L) \subseteq \Sigma \times \Sigma$ wie folgt: $Z_1(L) := \{w \in \Sigma \times \Sigma : \pi_1(w) \in L, \pi_2 \in \Sigma^*\}$. Das bedeutet, wir haben die Menge aller Ketten von Paaren, deren erste Projektion in L ist und deren zweite Projektion in Σ^* ist. Parallel dazu definieren wir $Z_2(L)$. Wir schreiben ein Produkt $(a_1, a_1')(a_2, a_2')(a_3, a_3')...$ als Tupel $(a_1 a_2 a_3 ..., a_1' a_2' a_3' ...)$. Die beiden folgenden Sätze werden uns begleiten, wenn wir von einfachen Automaten zu Transduktoren kommen.

Satz 45 *Sei $\mathfrak{A}$ ein endlicher Automat und $L = L(\mathfrak{A})$. Wir können einen Automaten $\mathfrak{A}'$ konstruieren, so dass $Z_1(L) = L(\mathfrak{A}')$ (ebenso für $Z_2(L)$).*

Satz 46 *Sei $\mathfrak{A}$ ein endlicher Automat und $L(\mathfrak{A}) \subseteq (\Sigma \times \Sigma)^*$. Wir können einen endlichen Automaten $\mathfrak{A}'$ konstruieren, für den gilt: $L(\mathfrak{A}') = \pi_1(L)$ (ebenso für $\pi_2(L)$).*

Die regulären Sprachen sind also abgeschlossen unter Projektion und Zylindrifizierung. Wir kommen nun bereits Transduktoren sehr nahe; denn eine Sprache $L \subseteq \Sigma \times \Sigma$ kann bereits aufgefasst werden als eine Relation, d.h. eine Menge von Paaren (w, w'); und damit kann der Automat, der diese Sprache

erkennt als ein Transduktor bezeichnet werden. Leider gibt es noch einige weitere Dinge zu beachten, wie wir gleich sehen werden.

15 Transduktoren, Funktionen, Relationen

15.1 Vorgeplänkel

Wir haben soeben gesehen, das wir Transduktoren bekommen, wenn wir Automaten über 2-Tupel Alphabete konstruieren. Der Schritt ist fast nur noch eine Änderung der Perspektive: wir denken uns eben nicht mehr eine einfache Sprache über Tupel, sondern wir denken uns den Automaten als eine Relation/Funktion, die uns für jede Eingabe eine Ausgabe liefert. Sei also $\mathfrak{A}$ ein Automat über ein Tupelalphabet, so dass $L(\mathfrak{A}) \subseteq \Sigma \times \Sigma$. Dieser Automat ist in der Tat isomorph (d.h. praktisch gleich) zu einem Transduktor $\mathfrak{T}$, und wir haben: $(w, w') \in L(\mathfrak{A})$ genau dann wenn $\mathfrak{T}(w) = w'$, bzw. $w' \in \mathfrak{T}(w)$. Wir haben also das Tupelwort getrennt in ein Eingabewort und ein Ausgabewort. Der Transduktor liest die Eingabe, und errechnet die Ausgabe.

Wenn wir Transduktoren statt Automaten betrachten, dann wechseln wir also in gewissem Sinne nur die Perspektive; und natürlich können wir auch wieder die ursprüngliche Perspektive einnehmen, wenn wir Transduktoren betrachen. Das ist vorteilhaft, denn wie wir gleich sehen werden, kommt die perspektivische Veränderung mit einigen Komplikationen einher.

Ein Transduktor (als solcher betrachtet) stellt eine Generalisierung eines endlichen Automaten dar. Ein Automat liest eine Eingabe, und am Ende akzeptiert er, oder er akzeptiert nicht (das deckt übrigens auch die Möglichkeit ab, wo er für die Eingabe nicht definiert ist, was leicht passieren kann). Wir können also schreiben: $\mathfrak{A}(w) = 1$ genau dann wenn $w \in L(\mathfrak{A})$; und $\mathfrak{A}(w) = 0$ andernfalls. Ein Automat ist also in gewissem Sinne eine Funktion, die nur zwei Ausgaben hat, 0 und 1. Wir haben den Automaten hier mit der **charakteristischen Funktion** seiner Sprache identifiziert. Eine charakteristische Funktion einer Menge M ist eine Funktion χ nach $\{0, 1\}$, so dass $\chi_M(x) = 1$, falls $x \in M$, und $\chi_M(x) = 0$ andernfalls

Ein Transduktor ist also ein Automat, der beliebige Ausgaben für seine *Eingabesprache* ausgeben kann; so er sie denn berechnen kann mit endlich vielen Zuständen. Mit der Eingabesprache bezeichnen wir die Menge der Eingabeketten, für die er eine Ausgabe gibt; ebenso bezeichnen wir mit der Ausgabesprache die Menge der möglichen Ausgabewörter.

Unser letzter Satz oben versichert uns folgendes:

Lemma 47 *Sei $\mathfrak{T}$ ein Transduktor. Dann ist sowohl seine Eingabesprache wie auch seine Ausgabesprache eine reguläre Sprache.*

Das folgt aus der Abgeschlossenheit unter Projektion; wenn wir den Transduktor als Tupel-Automaten sehen, dann ist die Eingabesprache nichts als die erste Projektion, und die Ausgabesprache die zweite.

15.2 ϵ-Übergänge

Aus der Tatsache, dass Transduktoren nicht mehr über den freien Monoid definiert sind, folgen aber einige Probleme. Womöglich möchten wir, dass Eingabewort und Ausgabewort unterschiedliche Länge haben; wenn wir beispielsweise die Ausgabe Männer für die Eingabe Mann haben wollen. In diesem Fall ist der Unterschied der Länge beschränkt. Wir können uns aber auch folgendes vorstellen: $\mathfrak{T}$ akzeptiert als Eingabesprache a^*; als Ausgabe gibt er die gleich Anzahl as zurück, schreibt aber hinter jedes dritte a ein b. In diesem Fall ist der Längenunterschied unbegrenzt: die Differenz zwischen Eingabe und Ausgabe wächst mit der Eingabe, und wird größer als jedes $k \in \mathbb{N}$.

Um einen Automaten zu konstruieren der diese Funktion errechnet, brauchen wir selbstverständlich einen ϵ-Übergang in der Eingabe. Das bedeutet, der Automat hat einen Übergang, wo er auf der Eingabeseite ϵ liest, auf der Ausgabeseite aber einen Buchstaben in Σ ausgibt. In der Tat, der minimale Automat, der die gewünschte Funktion errechnet, sieht wie folgt aus:

Wie passt das zu dem Satz dass wir ϵ-Übergänge eliminieren können? Wenn wir diesen Transduktor wieder als Automaten lesen, dann sehen wir, dass er gar keinen ϵ-Übergang hat; denn $(\epsilon, b) \neq (\epsilon, \epsilon)$. Den ϵ-Übergang haben wir nur, wenn wir die Eingabesseite von der Ausgabeseite trennen. Wir können für Transduktoren also (ϵ, ϵ)-Übergänge eliminieren (das folgt aus dem obigen Satz), nicht aber Übergänge wie wir sie hier beobachten. Der ϵ-Eliminierungssatz bezieht sich also immer auf das neutrale Element des Monoids.

Dassselbe wie oben geht übrigens auch umgekehrt: der folgende Transduktor hat als Eingabe wie als Ausgabesprache a^*; er verkürzt aber jede Eingabe um ein Drittel.

Diese Tatsache hat weitere Konsequenzen für die Hülleneigenschaften und Entscheidbarkeitsprobleme.

15.3 Determinismus

Wie wir an den obigen Beispielen bereits ablesen können, gilt auch der Determinisierungssatz nicht mehr für Transduktoren. Ein einfaches Beispiel dafür ist: wir möchten einen Transduktor, dessen Eingabesprache a^* ist, und dessen Berechnung nur darin besteht, dass er das *vorletzte* a in der Eingabe in ein b umschreibt. Ein Transduktor, der das macht, sieht wie folgt aus:

Es liegt aber auf der Hand, dass man diesen Transduktor nicht determinisieren kann; denn woher soll er wissen, welches a das vorletzte ist, wenn er die Eingabe von links nach rechts abarbeitet? NB, auch dieses Ergebnis steht nicht im Wiederspruch zu dem obigen Ergebnis, denn aus der *Automatenperspektive* ist der Automat deterministisch: (a, b) ist ja durchaus verschieden von (a, a).

Aus der Transduktorperspektive sieht man, dass die Berechnung, die wir machen, davon abhängt was später als Eingabe kommt. D.h. wir machen Berechnungen parallel, und je nachdem was wir später als Eingabe bekommen, verwerfen wir sie wieder. Das macht die Dinge komplizierter: denn es gibt keine allgemeine Obergrenze für die Berechnungen, die wir parallel machen: verändere einfach das Beispiel oben zu einem Transduktor, der den k-letzten Buchstaben umschreibt, für ein beliebiges k. Wir können eine ungültige Berechnung erst nach k Schritten verwerfen; da wir aber in jedem Schritt eine neue Berechnung bekommen, haben wir k potentielle Ausgaben, die parallel laufen, von denen aber alle bis auf eine verworfen werden.

Wir haben gesagt, es gibt keine allgemeine Obergrenze. "Allgemein" bezieht sich dabei auf Transduktoren im allgemeinen. Für einen bestimmten Transduktor gibt es jedoch immer eine Obergrenze, wie wir gleich sehen werden. Dieses Ergebnis wird uns in gewissem Sinne "retten": wir haben oben gesagt, dass Determinisierung die Verarbeitung in linearer Zeit garantiert. Wir möchten hier zeigen, dass wir dieses Ergebnis nicht verlieren, auch wenn wir nicht mehr determinisieren können.

Wir haben gezeigt, dass die "Vergangenheit" von regulären Sprachen und rationalen Relationen, modulo Äquivalenz, endlich ist. Dasselbe gilt auch für die Zukunft. Um das zu zeigen, zeigen wir einfach dass rationale Relationen abgeschlossen sind unter Spiegelung (oder Inversion). Dazu führen wir aber zunächst einmal das Konzept des rationalen Ausdrucks ein.

16 Rationale Ausdrücke und Relationen

Wir nennen die Relationen, die durch Transduktoren berechnet werden können, **rational**. Eine äquivalente Charakterisierung rationaler Relationen sind **rationale Ausdrücke**: rationale Ausdrücke sind eine Verallgemeinerung von regulären Ausdrücken, die dieselben Konstruktoren $\cdot$, $|$ und $[-]^*$ haben, aber statt einfachen Buchstaben auch Paare (oder allgemeiner: Tupel) erlauben, wobei $(a,b)\cdot(c,d)$ als (ac,bd) interpretiert wird.

Wir benutzen diese Notation um einfache Beweise für den Abschluss unter Inversion und Reversion zu liefern. Gegeben eine Relation R, bezeichnen wir mit R^i die Inversion $R^i := \{(a,b) : (b,a) \in R$. Wir vertauschen also immer die linke und rechte Komponente. Die Reversion (oder Spiegelung) ist zunächst auf Worten definiert, und zwar wie folgt: $(a_1a_2...a_n)^r = a_n....a_2a_1$. Wir spiegeln also einfach das Wort. Wir erweitern das Konzept auf Sprachen mit $L^r := \{w^r : w \in L\}$, und auf Relationen durch $R^r := \{(w^r, v^r) : (w,v) \in R\}$. Wir haben folgende Sätze:

Satz 48 *Rationale Relationen sind abgeschlossen unter Inversion und Reversion.*

Der Beweis ist einfach und instruktiv, da er zeigt wie man den induktiven Aufbau von rationalen Ausdrücken verwenden kann. Wir definieren eine Abbildung $[-]^{\mathtt{i}}$ von rationalen Ausdrücken auf rationale Ausdrücke wie folgt:

1. $(A \cdot B)^{\mathtt{i}} := (A^{\mathtt{i}} \cdot B^{\mathtt{i}})$;
2. $(A + B)^{\mathtt{i}} := (A^{\mathtt{i}} + B^{\mathtt{i}})$;
3. $(A^*)^{\mathtt{i}} := (A^{\mathtt{i}})^*$;
4. $(a, b)^{\mathtt{i}} := (b, a)$

Es ist eine einfache Übung zu prüfen dass für jeden rationalen Ausdruck A gilt: 1. $A^{\mathtt{i}}$ ist ein rationaler Ausdruck, und 2. $\|A^{\mathtt{i}}\| = \|A\|^i$. Das beendet den Beweis für Inversion; wir brauchen noch den Beweis für Reversion (Spiegelung). Wir definieren die Abbildung wie folgt: $[-]^{\mathtt{r}}$ ist eine Abbildung von rationalen Ausdrücken auf rationale Ausdrücke, und definiert durch:

1. $(A \cdot B)^{\mathtt{r}} := (B^{\mathtt{r}} \cdot A^{\mathtt{r}})$;
2. $(A + B)^{\mathtt{r}} := (A^{\mathtt{r}} + B^{\mathtt{r}})$;
3. $(A^*)^{\mathtt{i}} := (A^{\mathtt{r}})^*$;
4. $(a, b)^{\mathtt{i}} := (a, b)$

Während vorher die Bedingung 4 die entscheidende war, ist es nur 1. Wieder gilt für jeden rationalen Ausdruck A: 1. $A^{\mathtt{r}}$ ist ein rationaler Ausdruck, und 2. $\|A^{\mathtt{r}}\| = \|A\|^r$.

17 Abschluss unter Komposition

Wir haben oben bereits definiert was die Komposition zweier Relationen ist. Wir werden in der Folge eine Reihe negativer Ergebnisse für rationale Relationen präsentieren; aber zunächst gibt es folgendes Ergebnis, das entscheidend ist für die Theorie der Transduktoren, vor allem aber für ihre Anwendung.

Satz 49 *Seien R_1, R_2 rationale Relationen. Dann ist $R_1 \circ R_2$ eine rationale Relation.*

Wir haben also einen Abschluss unter Komposition. Der Beweis ist normalerweise eine automatentheoretische Konstruktion und kann an vielen Stellen nachgeschlagen werden.

18 Nicht-abgeschlossenheit unter Schnitt

Transduktoren können ohne weiteres die Relation $R_1 := (a^n, b^n c^*)$ errechnen, ebenso die Relation $R_2 := (a^n, b^* c^n)$; für beide brauchen wir einen Transduktor mit nur je zwei Zuständen. Der rationale Ausdruck für R_1 ist $(a, b)^* \cdot (\epsilon, c)^*$, der für R_2 ist $(\epsilon, b)^* \cdot (a, b)^*$.

Wir nehmen nun $R_3 := R_1 \cap R_2$. Der Schnitt von Relationen ist gleich definiert wie der Schnitt von Sprachen, mengentheoretisch über die beiden Mengen von Paaren. Jetzt sehen wir: R_3 ist die Relation $(a^n, b^n c^n)$. Nehmen wir an es gibt einen Transduktor, der R_3 errechnet. Da wir wissen, dass für Relationen, die von Transduktoren definiert werden, beide Komponenenten reguläre Sprachen sind, muss also auch $\pi_2(R_3) = a^n b^n$ eine reguläre Sprache sein. Das ist aber bekanntermaßen falsch - diese Sprache erfüllt nicht das Pumplemma für reguläre Sprachen. Wir haben also gezeigt:

Satz 50 *Die Relationen, die durch Transduktoren definiert werden, sind nicht abgeschlossen unter Schnitt.*

Außerdem, da wir rein mengentheoretisch den Schnitt zweier Sprachen (oder Relationen) aus Vereinigung und Komplement erzeugen können, bekommen wir folgendes Korollar:

Korollar 51 *Rationale Relationen sind nicht abgeschlossen unter Komplement.*

Um das zu beweisen, benutzen wir einfache Kontraposition oder den Widerspruchsbeweis: nehmen wir an rationale Relationen wären abgeschlossen unter Komplement. Da rationale Relationen abgeschlossen sind unter Vereinigung, folgt daraus dass sie auch abgeschlossen sind unter Schnitt. Das steht im Widerspruch zu obigem Ergebnis, also ist unsere Annahme falsch.

19 Unentscheidbare Probleme

Es gibt eine ganze Reihe weiterer negativer Resultate für Transduktoren. Ein wichtiges ist die sogenannte *Inklusion*. Das Problem ist folgendes: nehmen wir zwei endliche Automaten $\mathfrak{A}_1, \mathfrak{A}_2$ als gegeben. Wir möchten nun wissen: ist es wahr dass $L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2)$? Wir sagen das Problem ist entscheidbar, wenn es einen Algorithmus gibt, der uns für beliebige Automaten nach endlich vielen Rechenschritten immer die richtige Antwort gibt. Für endliche Automaten gibt es das folgende positive Ergebnis:

Satz 52 *Inklusion für endliche Automaten ist entscheidbar.*

Daraus folgt natürlich unmittelbar: Äquivalenz für endliche Automaten ist entscheidbar, denn $L(\mathfrak{A}_1) = L(\mathfrak{A}_2)$ genau dann wenn $L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2)$ und $L(\mathfrak{A}_2) \subseteq L(\mathfrak{A}_1)$. Der Beweis ist denkbar einfach: gegeben $\mathfrak{A}_1, \mathfrak{A}_2$ können wir $\mathfrak{A}_3$ konstruieren, so dass $L(\mathfrak{A}_3) = \overline{L(\mathfrak{A}_2)}$, das Komplement von $L(\mathfrak{A}_2)$. Ebenso können wir $\mathfrak{A}_4$ konstruieren, so dass $L(\mathfrak{A}_4) = L(\mathfrak{A}_3) \cap L(\mathfrak{A}_1)$. Es lässt sich leicht prüfen, dass $L(\mathfrak{A}_4) = \emptyset$ genau dann wenn $L(\mathfrak{A}_1) \subseteq L(\mathfrak{A}_2)$. Unser Inklusionsproblem lässt sich also reduzieren auf das Problem: gegeben ein Automat $\mathfrak{A}$, ist $L(\mathfrak{A}) = \emptyset$? Dieses Problem lässt sich leicht entscheiden für endliche Automaten: wir müssen nur prüfen ob es einen Pfad vom Startzustand zu einem akzeptierenden Zustand gibt.

Die Äquivalenz zweier Automaten kann man auch direkt beweisen, denn jeder endliche Automat hat einen deterministischen, minimalen Automaten der äquivalent ist, und dieser Automat ist bis auf Isomorphie (Umbennung der Zustände) eindeutig.

Letzteres gilt für Transduktoren *nicht*: wir haben keinen eindeutigen minimalen, deterministischen Transduktor, auch nicht wenn wir Relationen als Eingaben betrachten. Der Grund ist ganz einfach: wir haben oben festgestellt dass der Monoid, auf dem Transduktoren operieren, nicht frei ist. Daraus folgt dass Terme keine eindeutige Darstellung haben, und dasselbe gilt daher für die Transduktoren die diese Terme verarbeiten/erkennen. Auch die Reduktion auf die Leere einer Sprache, die wir oben gemacht haben, funktioniert nicht: denn wir haben keinen Abschluss unter Schnitt und Komplement. Und in der Tat: für endliche Transduktoren ist das Problem untenscheidbar:

Satz 53 *Inklusion für endliche Transduktoren ist unentscheidbar.*

Der Beweis geht über eine Reduktion, würde uns aber hier zu weit abführen vom Weg. Übrigens folgt auch in diesem Fall unmittelbar: Äquivalenz von endlichen Transduktoren ist unentscheidbar. Denn rationale Relationen sind abgeschlossen unter Vereinigung. Nun nehmen wir an, Äquivalenz ist entscheidbar. Dann nehmen wir $L(\mathfrak{T}_1), L(\mathfrak{T}_2)$, und konstruieren $\mathfrak{T}_3$ so dass $L(\mathfrak{T}_3) = L(\mathfrak{T}_1) \cup L(\mathfrak{T}_2)$; als nächstes prüfen wir ob $L(\mathfrak{T}_3) = L(\mathfrak{T})_2$. Es ist leicht zu sehen dass das äquivalent ist zum Inklusionsproblem; als folgt: wenn Äquivelenz entscheidbar ist, ist Inklusion entscheidbar. Inklusion ist unentscheidbar, also ist Äquivalenz unentscheidbar.

20 Unterhalb der rationalen Relationen

20.1 Korrespondenz von Sprachen und Relationen

Wir haben also gesehen, dass rationale Relationen in vielen Hinsichten "mächtiger" sind als wir es gerne hätten: wir haben zwar noch einige gute Eigenschafte, v.a. den Abschluss unter Komposition, aber wir haben keinen Abschluss unter Schnitt und Komplement; außerdem sind viele Eigenschaften von Transduktoren im allgemeinen Fall unentscheidbar. Wir werden daher zunächst einige interessante *Teilklassen* der rationalen Relationen betrachten, also Klassen von Relationen, die echt kleiner sind. Ein besonderes Augenmerk wird dabei auf dem Verhältnis von Sprachen und Relationen liegen. Dieses Verhältnis ist wie folgt: wir finden üblicherweise eine Klasse von Relationen $\mathcal{R}$, und eine Klasse von Sprachen $\mathcal{L}$, so dass für jede Relation $R \in \mathcal{R}$, $i \in \{1, 2\}$, es ein L gibt, so dass $\pi_i[R] = L$, und für jedes $L \in \mathcal{L}, i \in \{1, 2\}$ gibt es ein $R \in \mathcal{R}$ so dass $L = \pi_i[R]$.

Dies ist unser Kriterium für die Korrespondenz einer Klasse von Sprachen und einer Klasse von Relationen. In diesem Sinne korrespondieren Beispielsweise die rationalen Relationen mit den regulären Sprachen. Wie wir aber gleich sehen werden, ist dieses Kriterium keinesfalls eindeutig. Nehmen wir beispielsweise

die Klasse aller Relationen R, für die gilt: 1. R ist rational, und 2. für alle $(w, v) \in R$ gilt: $|w| = |v|$. Das bedeutet, in dieser Klasse haben wir nur rationale Relationen von Paaren von Worten die gleich lang sind. Das ist offensichtlich eine relativ restriktive Teilklasse der rationalen Relationen; dennoch lässt sich leicht prüfen, dass sie nach obigen Kriterien in Korrespondenz mit den regulären Sprachen steht. Dennoch ist das Kriterium der Korrespondenz sehr nützlich, um Klassen von Relationen zu ordnen und zu verorten.

Die rationalen Relationen haben wir bereits ausführlich besprochen; die Klasse, die wir zuletzt besprochen haben, ist zu restriktiv um für Anwendungen interessant zu sein. Dennoch hat diese Klasse von Relationen interessante Eigenschaften: da diese Relationen Teil eines *freien* Monoids sind, können wir jede solche Relation auf eine reguläre Sprache *reduzieren*; wir können also so tun als wäre die Relation eine reguläre Sprache. Das wiederum bedeutet: wir haben Abschluss unter Schnitt und unter Komplement; letzteres allerdings nur, wenn wir uns für das Komplement auf Relationen beschränken, in denen alle Paare von Worten ebenfalls gleiche Länge haben; sonst natürlich nicht. Diese Reduktion zeigt gleichfalls, dass Äquivalenz und Inklusion entscheidbar sind. Wir werden diese Klasse weiter unten *erweitern*, so dass wir alle positiven Eigenschaften behalten, aber unsere Ausdrucksmächtigkeit so vergößern, dass wir fast alle interessanten Relationen abdecken. Wir werden diese Klasse die synchronen regulären Relationen nennen. Zunächst wenden wir uns einer anderen Klasse zu.

20.2 Rationale Funktionen

Rationale Funktionen sind rationale Relationen, die Funktionen sind; das heißt: $R \subseteq \Sigma^* \times \Sigma^*$ ist eine rationale Funktion, falls R rational ist, und für alle $w \in \Sigma^*$ gibt es genau *ein* v, so dass $(w, v) \in R$. Wir benutzen hier die mengentheoretische Auffassung von Funktionen: eine Funktion ist nur eine Relation, die für jede Eingabe genau eine Ausgabe gibt. Ein wichtiges Problem ist folgendes:

(42) Ist es entscheidbar ob eine rationale Relation eine rationale Funktion ist?

Mir ist momentan nicht klar ob dieses Problem entscheidbar ist; ich weiß leider auch nicht, ob diese Frage überhaupt schon eine Antwort gefunden hat in der Literatur.

20.3 Synchrone reguläre Relationen

Wir kommen nun zur Klasse der synchronen regulären Relationen. Einfachheit halber werden wir diese Klasse fortan einfach als reguläre Relationen bezeichnen. Diese Bezeichnung ist teilweise üblich, führt aber leicht zu Missverständnissen, daher ist etwas Vorsicht geboten bei diesem Begriff. Wie wir gesagt haben sind die regulären Relationen eine Erweiterung der rationalen Relationen gleicher Länge. Für eine reguläre Relation R gilt: 1. R ist rational, und 2. es gibt einen

Automaten, der die Relation erkennt, der nur dann ein ϵ in der Komponente $i : i \in \{1, 2\}$ liest, wenn er daraufhin *nur noch* ϵs in der Komponente i liest. Wir erlauben also Übergänge der Form (ϵ, a) (oder (a, ϵ), aber wenn wir so einen Übergang haben, dann haben wir nur noch Übergänge dieser Form. Eine etwas formalere Definition ist die folgende:

Definition 54 *Sei $\Sigma_\perp := \Sigma \cup \{\perp\}$, wobei $\perp \notin \Sigma$, und $\epsilon \notin \Sigma$. Die **Konvolution** eines Tupels von Ketten $(w, v) \in \Sigma^* \times \Sigma^*$, in Symbolen $\otimes(w, v)$ mit Länge $max(\{|w|, |v|\})$ ist wie folgt definiert: das k-te Teiltupel von $\otimes(w, v)$ ist (a_k, b_k), mit $a_i, b_i \in \Sigma$, vorausgesetzt dass $k \leq |\vec{x}_i|$, und $\perp$ andernfalls.*
*Die **Konvolution einer Relation** $R \subseteq \Sigma^* \times \Sigma^*$, in Symbolen $\otimes R$, ist die Menge $\{\otimes(w, v)$: $(w, v) \in R\}$.*

Für die Konvolution fügen wir als, falls $|w| < |v|$, genausoviele $\perp$ ans Ende von w an, dass $w \perp^i$ gleichlang sind, oder umgekehrt.

Definition 55 *Eine Relation $R \in \Sigma^* \times \Sigma^*$ ist (synchron) **regulär**, wenn es einen endlichen, ϵ-freien Transduktor über $\Sigma_\perp \times \Sigma_\perp$ gibt, der $\otimes R$ erkennt.*

Wir nennen die Klasse von Transduktoren, die reguläre Relationen erkennen, **synchrone Transduktoren**. Reguläre Relationen werden also von endlichen Transduktoren erkannt, die keine ϵ-Übergänge haben, außer in einer Projektion $\pi_i(R)$, und diese Übergänge liegen alle zusammen am Ende der Transduktion. Diese Bedingung ist entscheidend: wir wissen dass (ϵ, ϵ)-Übergänge können, wohingegen Übergänge mit einem Tupel $\bar{t}$, wobei $\pi_i(\bar{t}) = \epsilon, \pi_j(\bar{t}) \neq \epsilon$, $i \neq j$, im Allgemeinen nicht eliminiert werden können und unsere Ausdrucksstärke erweitern. Wenn wir diese Übergänge beschränken, dann beschränken wir auch die Klasse von Relationen die wir berechnen können. Allerdings ist unsere Beschränkung nicht so streng, wie man auf den ersten Blick meinen könnte: beispielsweise jeder Transduktor mit einer oberen Grenze für die asymmetrischen ϵ-Übergänge in einer Transduktion kann in unserem Sinne *synchronisiert* werden, so dass alle diese Übergänge final sind.

Reguläre Relationen sind eine relativ große Teilklasse der rationalen Relationen, die die meisten interessanten Fälle umfasst, wenn auch nicht alle. Beispielsweise die Relation $(a, b)^* \cdot (\epsilon, c)^* = (a^n, b^n c^*)$ ist regulär; die Relation $(\epsilon, b)^* \cdot (a, c)^* = (a^n, b^* c^n)$ nicht. Das ist wichtig für die Abgeschlossenheit unter Schnitt! Eine weitere typische, nichtreguläre Relation ist $((a, a) \cdot (a, \epsilon))^* = (a^{2n}, a^n)$.

Die Frage ist nun: wird der Verlust an Ausdrucksstärke ausreichend kompensiert durch entsprechende positive Eigenschaften? Die Antwort ist natürlich nicht objektiv, aber es gibt wichtige Eigenschaften, die wir für reguläre Relationen zeigen können:

Satz 56 *Die Klasse der regulären Relationen ist abgeschlossen unter Schnitt und Komplement.*

Ich kenne keinen einfachen Beweis für diesen Satz; man erhält ihn aber als Korollar von einem wichtigen Satz von Eilenberg, Elgot und Shepherdson, der

zeigt dass es eine logische Charakterisierung der regulären Relationen gibt in erststufiger Prädikatenlogik. Der Beweis dafür ist allerdings ebenfalls sehr lang und kompliziert.

Aus der Abgeschlossenheit unter Schnitt und Komplement folgt auch unmittelbar folgendes:

Satz 57 *Inklusion und Äquivalenz für reguläre Relationen sind entscheidbar.*

Der Beweis ist nun genau parallel zum Beweis für einfache endliche Automaten. Die nächste Frage ist: behalten wir alle positiven Eigenschaften der rationalen Relationen? Wir haben folgende Antworten:

Satz 58 *Reguläre Relationen sind abgeschlossen unter Komposition.*

Auch hierfür ist der Beweis offensichtlich, wenn man reguläre Relationen logisch charakterisiert, ansonsten eher mühsam.

Einen Wermutstropfen gibt es dennoch:

Satz 59 *Seien L_1, L_2 reguläre Sprachen. Dann ist $L_1 \times L_2$ im Allgemeinen keine reguläre Relation.*

Aufgabe: finden sie zwei Sprachen, die diese Aussage beweisen.

Das bedeutet also, in einem etwas allgemeineren Relationenbegriff haben wir keinen Abschluss unter kartesischem Produkt. Für die rationalen Relationen hingegen lässt sich einfach zeigen: falls L_1, L_2 regulär sind, dann ist $L_1 \times L_2$ rational. Wir können also im Allgemeinen keine regulären Relationen einfach aus dem Produkt zweier Sprachen konstruieren.

Die beiden folgenden Ergebnisse lassen sich aber relativ einfach zeigen:

Satz 60 *Reguläre Relationen sind abgeschlossen unter Inversion und Reversion.*

Das Argument hierfür ist: beide Operationen ändern nicht die Längendifferenz zwischen den beiden Komponenten eines Paares, wir bleiben also "synchron".

20.4 Subsequentielle rationale Relationen

Es ist klar dass die regulären Relationen mit den regulären Sprachen korrespondieren. Es gibt noch eine wichtige Klasse, die nicht mit den regulären Sprachen korrespondiert, nämlich die subsequentiellen rationalen Relationen, oder einfach subsequentielle Relationen. Subsequentielle Relationen werden am einfachsten über subsequentielle Transduktoren erklärt. Ein subsequentieller Transduktor ist wie ein Transduktor $(\Sigma, Q, q_0, \delta, F)$, wobei $Q = F$, das bedeutet, jeder Zustand ist akzeptierend. Daher können wir F auch einfach weglassen und einen subsequentiellen Transduktor definieren als ein Tupel (Σ, Q, q_0, δ). Es gibt aber noch weitere Dinge zu berücksichtigen: wir möchten evtl. verhindern, dass wir bestimmte Eingaben qua Partialität ablehnen; also verlangen wir, dass für jede Eingabe mindestens eine Ausgabe definiert wird. Damit wiederum haben wir nicht mehr die Möglichkeit, Eingabe- und Ausgabealphabet ohne Beschränkung der Allgemeinheit gleich zu setzen; wir müssen die beiden also unterscheiden.

Definition 61 *Ein subsequentieller Transduktor ist ein Tupel* $(\Sigma, T, Q, q_0, \delta)$*, wobei* Σ *das Eingabealphabet ist,* T *das Ausgabealphabet, und* $\delta \subseteq Q \times \Sigma \times T \times Q$ *die Übergangsrelation. Ein Transduktor ist total, wenn f.a.* $q \in Q, a \in \Sigma$ *es ein* $q' \in Q, a' \in \Sigma$ *gibt so dass* $(q, a, a'q') \in \delta$*; andernfalls partiell.*

Die Akzeptanz ist wie folgt definiert: sei $\mathfrak{T}$ ein subsequentieller Transduktor. Dann ist $(a_1...a_n, b_1...b_n) \in L(\mathfrak{T})$, wobei für $1 \leq i \leq n$, $a_i \in \Sigma \cup \epsilon$, $b_i \in \Sigma \cup \epsilon$, genau dann wenn $(q_0, a_1, b_1, q_1) \in \delta$, und für alle $1 \leq i < n$, $(q_i, a_{i+1}, b_{i+1}, q^{i+1}) \in \delta$. NB: q_0 bezeichnet den Startzustand, aber $q_i : i \neq 0$ bezeichnet nicht einen bestimmten Zustand, sondern allgemein den i-ten Zustand der Transduktion. Wie wir gesehen haben haben wir hier also keine Referenz auf akzeptierende Zustände.

Definition 62 *Eine subsequentielle Relation ist eine Relation, die von einem subsequentiellen Transduktor erkannt wird.*

Eine wichtige Eigenschaft von subsequentiellen Relationen ist folgende: die "Vergangenheit", d.h. die Übergänge, die bereits gemacht wurden, können die Übergänge die wir machen beeinflussen wie in Transduktoren. Die "Zukunft", also zukünftige Eingaben, haben keinen Einfluss auf aktuelle Ausgaben. Nehmen wir beispielsweise folgende Relation:

(43) $R_0 := \{(a^n c, a^n c) : n \text{ gerade}\} \cup \{(a^n c, a^n b) : n \text{ ungerade}\}$

Diese Relation ist partiell subsequentiell, da wir nicht für jede Eingabe eine Ausgabe haben. Wir können sie aber auf einfache Art und Weise "vervollständigen:

(44) $R_1 := (id_\Sigma - \pi_1(R_0)) \cup R_0.$

R_1 ist eine vollständige subsequentielle Relation Allerdings ist $(R_0)^r = \{\langle ca^n, ca^n : n \text{ gerade}\} \cup \{\langle ca^n, ba^n : n \text{ ungerade}\}$ *nicht* subsequentiell, weil hier die erste Teilabbildung von dem abhängt, was nachfolgt. Daraus können wir bereits dass erste Ergebnis ableiten:

Lemma 63 *Subsequentielle Relationen sind nicht abgeschlossen unter Reversion.*

Im engeren Sinne korrespondieren subsequentielle Relationen *nicht* mit den regulären Sprachen. Um das zu sehen beachten wir folgendes: für $L \subseteq \Sigma^*$ denotieren wir mit $pref(L) := \{v : \exists w \in L : w = vx\}$ die Menge der Präfixe von L. Wir sagen L ist abgeschlossen unter Präfixen, falls $L = pref(L)$. Offensichtlich sind die regulären Sprachen nicht abgeschlossen unter Präfixen.

Aufgabe: finden Sie ein Beispiel!

Für jede subsequentielle Relationen gilt jedoch:

Lemma 64 *Sei R subsequentiell. Dann gilt für $i \in \{1,2\}$: $pref(\pi_i(R)) = \pi_i(R)$.*

Beweis: Sei $(a_1...a_n, b_1...b_n) \in R$. Dann gilt für jedes $i \leq n$, $(a_1...a_i, b_1...b_i) \in R$. Daraus folgt *a fortiori*: $a_1...a_n \in \pi_1(R) \Rightarrow a_1...a_i \in \pi_1(R)$; dasselbe für π_2. $\dashv$

Das ist erstmal ein negatives Ergebnis, da wir hier damit *unter* die Ausdrucksstärke der regulären Sprachen zurückfallen. Betrachten wir noch einige weitere Eigenschaften.

Lemma 65 *Subsequentielle Relationen sind nicht abgeschlossen unter Schnitt. Außerdem gilt: jede Klasse von Relationen, die die subsequentiellen Relationen enthält und in den rationalen Relationen enthalten ist, ist nicht abgeschlossen unter Schnitt.*

Um dieses Lemma zu beweisen müssen wir nur den Beweis für die rationalen Relationen bemühen; alle Relationen, die dort benutzt werden, sind subsequentiell. Daraus folgt mit dem selben Argument:

Korollar 66 *Subsequentielle Relationen sind nicht abgeschlossen unter Komplement. Außerdem gilt: jede Klasse von Relationen, die die subsequentiellen Relationen enthält und in den rationalen Relationen enthalten ist, ist nicht abgeschlossen unter Komplement.*

Der Grund, warum subsequentielle Relationen ein besonderes Interesse in der Literatur gefunden haben, ist womöglich folgender: wir definieren subsequentielle Funktionen auf die übliche, mengentheoretische Art und Weise.

Lemma 67 *Eine Funktion ist subsequentiell genau dann wenn es einen subsequentiellen Transduktor $\mathfrak{T}$ gibt, so dass es nach jeder Eingabe w genau einen Zustand gibt in dem $\mathfrak{T}$ sich befindet.*

Man kann also subsequentielle Funktionen als deterministische Automaten auffassen, die für jede Eingabe in höchstens einen Zustand geht. Diese Eigenschaft ist sehr nützlich und erlaubt eine sehr elegante mathematische Theorie (siehe Trakhtenbrodt/Barzdin: Finite Automata). Darüberhinaus hat jede sequentielle Funktion einen eindeutigen, deterministischen minimalen sequentiellen Transduktor, der sie berechnet.

Das erlaubt uns, folgende Eigenschaften zu verifizieren:

Lemma 68 *Subsequentielle Funktionen sind abgeschlossen unter Schnitt und Komplement.*

Der Beweis funktioniert wie folgt: wir haben für jede Eingabe (q, a) genau ein a' und q' als Ausgabe. Wir können also den Transduktor als normalen Automaten betrachten und die Ausgabe in dem Nachfolgezustand integrieren; denn sie ist eindeutig durch den Zustand bestimmt. Außerdem haben wir folgendes Ergebnis:

Satz 69 *Äquivalenz und Inklusion für subsequentielle Funktionen ist entscheidbar.*

Das folgt, ähnlich wie für endliche Automaten, aus den Ergebnissen dieses Abschnitts. Zuletzt noch folgendes Ergebnis über subsequentielle und synchron reguläre Relationen:

Lemma 70 *Es gibt Relationen R die nicht synchron regulär, aber subsequentiell sind, und es gibt Relationen R' die synchron regulär, aber nicht subsequentiell sind.*

Aufgabe: liefern Sie Beispiele, die dieses Lemma beweisen!

21 Sternfreie Sprachen

Bislang haben wir Sprachen und Relationen betrachtet, die mit den regulären Sprachen korrespondieren (mit Ausnahme der sequentiellen Relationen, die aber auch grob in diese Gruppe gehören). Wir werden jetzt eine neue Klasse von Sprachen einführen, die echt in den regulären Sprachen enthalten ist, nämlich die sog. **sternfreien** Sprachen. Diese Klasse ist eine relativ große Teilklasse der regulären Sprachen, und meines Wissens nach die größte die allgemein geläufig und "natürlich" ist. Was "natürlich" in der Theorie der formalen Sprachen bedeutet ist: es gibt mehrere Charakterisierungen (Automaten, algebraische Ausdrücke, Grammatiken, Logiken), die nicht offensichtlich äquivalent sind, und alle unabhängig voneinander diese Klasse charakterisieren. In diesem Sinne sind die regulären Sprachen eine sehr natürliche Klasse: wir haben eine ganze Reihe von Charakterisierungen besprochen und dabei sogar noch einige wichtige ausgelassen (algebraische und logische). Auch die sternfreien Sprachen haben eine ganze Reihe von Charakterisierungen, von denen wir nur zwei relativ naheliegende besprechen werden.

21.1 Sternfreie Ausdrücke

Der Ausdruck "sternfrei" kommt daher, dass alle regulären Ausdrücke, die keinen Kleene-Stern enthalten, sternfrei sind. Das Gegenteil ist aber nicht richtig: wir brauchen noch zusätzliche Konstruktoren - sonst könnten wir ja nur die endlichen Sprachen denotieren!

Aufgabe: zeigen sie, dass man mit regulären Ausdrücken ohne Stern genau alle endlichen Sprachen charakterisiert, d.h. keine unendliche! *Tip*: wie immer gibt es zwei Richtungen. Dass man jede endliche Sprache charakterisieren kann ist leicht zu zeigen, indem man die Konstruktion des Ausdrucks für die Sprache angibt. Um zu zeigen, dass jeder solche Ausdruck eine endliche Sprache denotiert macht man am besten eine Induktion über die (induktiv definierte) Struktur des Ausdrucks.

Wir haben zwei neue Konstruktoren für sternfreie Ausdrücke: eine Konstante $\perp$, deren Syntax gleich der von Buchstaben $a \in \Sigma$ ist, und deren Semantik

definiert ist durch $\| \perp \| := \emptyset$, und einen unären Operator$\overline{[-]}$, dessen Semantik gegeben ist durch $\|\overline{\alpha}\| := \Sigma^* - \|\alpha\|$. Wir können also die leere Sprache explizit denotieren, und das Komplement bilden. Wir zeigen einige Beispiele.

Die Sprache Σ^* wird denotiert von $\overline{\perp}$. Die Sprache $(ab)^*$ hat eine etwas kompliziertere sternfreie Charakterisierung:

$$\overline{(\overline{\perp}aa\overline{\perp}) + (\overline{\perp}bb\overline{\perp}) + (b\overline{\perp}) + (\overline{\perp}a)} \tag{45}$$

Wenn wir diesen Ausdruck in Worte fassen, dann bekommen wir: die Sprache über $\{a, b\}$, in der niemals aa, bb vorkommt, kein Wort mit b anfängt oder mit a aufhört.

Aufgabe: verifizieren Sie, dass der Ausdruck (4) tatsächlich dieselbe Sprache wie $(ab)^*$ denotiert. *Tip*: Äquivalenz zweier Mengen zeigt man über zwei Inklusionen; Inklusion zeigt man, indem man zeigt, dass jedes Element der einen Menge ein Element der anderen Menge sein muss.

Das Beispiel zeigt bereits warum sich die Popularität der sternfreien Ausdrücke in Grenzen hält: sie sind meist wesentlich kompliziert als die regulären.

Aufgabe: schreiben Sie folgende regulären Ausdrücke als sternfreie Ausdrücke: 1. $(a(b + c))^*$, 2. $(bab)^*$, 3. a^*b, 4. $((ab)^*c) + ((ab)^*ad)$.

Die algebraische Charakterisierung macht einige Eigenschaften sternfreier Sprachen ziemlich offensichtlich:

Satz 71 *Sternfreie Sprachen sind abgeschlossen unter Vereinigung, Komplement und Schnitt.*

Vereinigung und Komplement qua definition sternfreier Ausdrücke, Schnitt folgt aus mengentheoretischen Gründen. Ein Kommentar noch: wem nicht sofort klar ist, dass die sternfreien Sprachen in den regulären Sprachen enthalten sind, der sollte sich klar machen dass aus der Tatsache, dass reguläre Sprachen unter Komplement abgeschlossen sind, folgt, dass ein Komplementoperator keine zusätzliche Ausdrucksstärke gibt für reguläre Ausdrücke. Die leere Menge kann ebenfalls regulär denotiert werden mit dem leeren Ausdruck. Der Grund warum wir diese konstante brauchen für sternfreie Ausdrücke ist: wir haben sonst keine Möglichkeit, Σ^* zu denotieren, denn wir haben keinen Stern, und können ja nicht das Komplement des leeren Ausdrucks bilden!

21.2 Zählerfreie Automaten

Für andere Eigenschaften sternfreier Sprachen sind sternfreie Ausdrücke eher wenig informativ. Z.B. ist $(ab)^*$ sternfrei, aber $(aa)^*$ nicht. Warum ist das so? Diese etwas merkwürdige Eigenschaft wird verständlicher, wenn wir uns eine weitere wichtige Charakterisierung sternfreier Sprachen anschauen:

Definition 72 *Sei $\mathfrak{A} = (\Sigma, Q, q_0, F, \delta)$ ein endlicher Automat, wobei δ^* die Generalisierung von δ auf Ketten ist. $\mathfrak{A}$ ist **zählerfrei**, falls gilt: für alle $w \in \Sigma^*, q \in Q, n \geq 1$, falls $\delta^*(q, w^n) = q$, dann ist $\delta(q, w) = q$.*

Was besagt diese Bedingung? In Worten: wenn wir in Zustand q sind, ein Wort w n-mal lesen, und damit nach q zurückkehren, dann kehren wir auch nach q zurück nachdem wir es *einmal* gelesen haben. Was wir also beschränken sind die *Zyklen* in unserem Automaten. Beispielsweise: nehmen wir an $\mathfrak{A}$ ist ein zählerfreier Automat, der $(aa)^*$ erkennt. Wir werden dann (falls er minimal ist) finden dass er nach der Eingabe aa im selben Zustand ist, wie am Anfang, nämlich q_0. Da er aber zählerfrei ist, folgt daraus: er ist auch nachdem er a gelesen hat in q_0! Das bedeutet aber wiederum: entweder er akzeptiert aa *nicht*, oder er akzeptiert auch a - in jedem Fall erkennt er nicht die Sprache $(aa)^*$!

Dieses Argument ist leider noch unsauber: denn es kann ja einen anderen, viel größeren Automaten geben, der $(aa)^*$ erkennt und zählerfrei ist. Allerdings können wir das Argument, dass wir oben für $n = 2$ geführt haben, für jedes n führen: irgendwann (nach a^k) haben wir sicher einen Zyklus, und dann gilt: wir akzeptieren auch a^{k+1}. Ein berühmter Satz von Marcel Paul Schützenberger sagt nun:

Satz 73 *Eine Sprache L ist genau dann sternfrei, wenn es einen zählerfreien Automaten $\mathfrak{A}$ gibt, so dass $L = L(\mathfrak{A})$.*

Das, zusammen mit unseren vorigen Überlegungen, führt uns direkt zum **Pumplemma der sternfreien Sprachen**:

Lemma 74 *Eine Sprache $L \subseteq \Sigma^*$ ist genau dann sternfrei, wenn für alle $x \in \Sigma^+$ es ein $k \in \mathbb{N}$ gibt, so dass f.a. $y, z \in \Sigma^*$ gilt: falls $xy^{k'}z \in L$ für ein $k' \geq k$, dann ist $xy^n z \in L$ für alle $n \in \mathbb{N}$.*

Man beachte: auf den ersten Blick gibt es eine Ähnlichkeit zu den regulären Sprachen. Der Unterschied ist: bei den regulären Sprachen sagen wir: *es gibt* eine Zerlegung; hier sagen wir: *für alle Zerlegungen.* Der Unterschied liegt also in der Quantifikation. Man beweist das Pumplemma der sternfreien Sprachen recht einfach aus dem Satz von Schützenberger: gegeben einen zählerfreien Automaten für L, wählen wir k so, dass es den kleinsten Zyklus beschreibt (wir kommen zum zweiten Mal in denselben Zustand). Da der Automat endlich ist, gibt es so ein k.

Das können wir benutzen, um zu zeigen dass gewisse Sprachen nicht sternfrei sind: nehmen wir $L = (aa)^*$, und k eine beliebige Zahl. Dann ist entweder (i) $aa^k \in L$ oder (ii) $a^k \in L$ (je nachdem ob k gerade oder ungerade ist). Daraus folgt qua Pumplemma, dass im Fall (i) $aa^2 \in L$ - Widerspruch; oder im Fall (ii) $a^1 \in L$ - Widerspruch. Ein zweites Beispiel ist folgendes: sei $L_1 \subseteq \{a, b\}^*$, $L_1 := \{w : |w|_a \text{ gerade}\}$. Sei k beliebig. (i) k ist gerade. Dann ist $a^k \in L_1$; also nach Pumplemma auch $a \in L$ - Widerspruch. (ii) k ist ungerade. Dann ist $aa^k \in L_1$, also nach Pumplemma auch aa^2 - Widerspruch. Also ist das Pumplemma nicht erfüllt. Ein Nachtrag noch: während das Pumplemma für reguläre Sprachen notwendig ist als Kriterium, aber nicht hinreichend (es gibt nicht-reguläre Sprachen die das Lemma erfüllen, wie etwa $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$, ist das sternfreie Lemma notwendig und hinreichend: wenn eine Sprache

es erfüllt, dann ist sie sternfrei. Wir können es also auch benutzen um zu zeigen dass Sprachen sternfrei sind.

Die obigen Beispiele machen klar, wie zählerfreie Automaten zu ihrem Namen gekommen sind: was endliche Automaten können ist: zählen modulo irgendeine Zahl; d.h. sie können berechnen ob die Länge eines Wortes oder die Anzahl eines gewissen Buchstaben in einem Wort durch zwei/drei/vier... teilbar ist. Zählerfreie Automaten können genau dass nicht, oder nur in sehr beschränkten Fällen.

21.3 Eine weitere Eigenschaft

Sternfreie Sprachen haben auch eine logische Charakterisierung in $FO[<]$, erststufiger Prädikatenlogik in der Signatur $[<]$ (lineare Ordnung). Wir können hier nicht auf diese Charakterisierung eingehen, allerdings lässt sich aus einem wichtigen Ergebnis der endlichen Modelltheorie (den sog. 0-1 Gesetzen) folgendes interessante Ergebnis ableiten. Wir definieren $\Sigma^{\leq n} := \{w \in \Sigma^* : |w| \leq n\}$, und für $L \subseteq \Sigma^*$, $L^{\leq n} := L \cap \Sigma^{\leq n}$. Wir sagen eine Sprache ist **dicht**, falls

$$lim_{n\to\infty} \frac{L^{\leq n}}{\Sigma^{\leq n}} = 1; \tag{46}$$

eine Sprache ist **dünn**, falls

$$lim_{n\to\infty} \frac{L^{\leq n}}{\Sigma^{\leq n}} = 0. \tag{47}$$

Satz 75 *Für jede sternfreie Sprache L gilt: L ist dünn, oder L ist dicht.*

Der Beweis würde uns viel zu weit abführen. Wir können aber zeigen dass dieser Satz für die regulären Sprachen bereits nicht mehr richtig ist: wir nehmen die reguläre Sprache $L_1 := \{w : |w|_a \text{ gerade}\}$; eine einfache Induktion zeigt uns:

$$lim_{n\to\infty} \frac{(L_1)^{\leq n}}{\Sigma^{\leq n}} = \frac{1}{2}. \tag{48}$$

Auch dieses Argument kann also benutzt werden, um zu zeigen dass eine Sprache wie $(aa)^*$ nicht sternfrei ist. *Aufgabe*: warum? Benutzen Sie Satz 50 um zu zeigen, dass $(aa)^*$ nicht sternfrei ist! Aber wohlgemerkt: das ist kein hinreichendes Kriterium um zu zeigen dass eine Sprache sternfrei ist: $\{a^n b^n : n \in \mathbb{N}\}$ ist zwar dünn, aber noch nicht einmal regulär.

22 Lokale Sprachen

22.1 Streng Lokale Sprachen

Streng lokale Sprachen kann man wie folgt charakterisieren: gegeben ein Alphabet Σ, sowie zwei Symbole $\rtimes, \ltimes \notin \Sigma$, haben wir eine Menge $\mathcal{R}$ von Regeln der

Form $a \bullet b : a \in \Sigma_{\cup}\{ltimes\}$, $b \in \Sigma_{\cup}\{rtimes\}$. Eine solche Regel besagt: die Abfolge ab ist wohlgeformt. Die Bedeutung der Symbole $\ltimes, \rtimes$ ist der Wortanfang bzw. das Wortende, so dass die Menge aller Worte über die streng lokale Grammatik $(\Sigma, \mathcal{R})$ ist: $L(\Sigma, \mathcal{R}) := \{a_1 a_2 ... a_n : \ltimes \bullet a_1 \in \mathcal{R}, a_n \bullet \rtimes \in \mathcal{R}$, und f.a. $i, i+1$ so dass $1 \leq i \leq n$ gilt $a_i \bullet a_{i+1} \in \mathcal{R}\}$. Eine Regel der Form $a \bullet b$ besagt also soviel wie: nach einem a darf ein b stehen. Streng lokale Sprachen haben also die Eigenschaft, dass wir keine verborgenen Strukturen haben wie Nichtterminale oder Zustände; alle relevante syntaktische Information ist soz. "overt".

Die Klasse der streng lokalen Sprachen ist stark begrenzt, und umfasst noch nicht einmal die Klasse aller endlichen Sprachen; das bedeutet, es gibt endliche Sprachen, die nicht lokal sind. Ein einfaches Beispiel ist die Sprache $\{aa\}$, die nur aus einem Wort besteht. (*Aufgabe*: Zeigen Sie dass diese Sprache nicht streng lokal ist!)

22.2 k-lokale Sprachen

Die streng lokalen Sprachen werden auch manchmal 2-lokal genannt. Der Grund dafür ist folgender: es gibt eine unendliche Hierarchie, die auf einer Erweiterung der lokalen Regeln $a \bullet b$ beruht. Diese Regeln machen immer nur Aussagen über 2 Symbole, die einander folgen dürfen. Wir können die Regeln aber wie folgt erweitern: statt nur 2 Symbolen verwenden wir 3 (oder allgemeiner: k) Symbole, so dass unsere Regeln die Form haben $a \bullet b \bullet c$ (allgemeiner: $a_1 \bullet ... \bullet a_k$, und $L(\Sigma, \mathcal{R}) := \{a_1 a_2 ... a_n : \ltimes \bullet a_1 \bullet ... \bullet a_k \in \mathcal{R}, a_{n-k} \bullet ... \bullet \rtimes \in \mathcal{R}$, und f.a. j so dass $1 \leq j, j+k \leq n$ gilt: $a_j \bullet a_{j+1} \bullet ... \bullet a_{j+k} \in \mathcal{R}$. Wir müssen natürlich annehmen dass auch ϵ in den Regeln erlaubt ist, sonst müsste jedes Wort mindestens k Buchstaben haben. Unter dieser Annahme ist leicht zu zeigen dass für jedes $k \in \mathbb{N}$ eine k-lokale Sprache auch eine $k+1$-lokale Sprache ist; aber es für jedes k eine $k+1$ lokale Sprache gibt, die keine k-lokale Sprache ist (z.B. die endliche Sprache a^{k-1} - *Aufgabe*: zeigen Sie warum das so ist!).

22.3 Lokale Sprachen

Wenn es für jede k eine Klasse k-lokalen Sprachen gibt, so liefert die Definition von lokalen Sprachen eine Verallgemeinerung dieser Hierarchie:

Definition 76 *Eine Sprache L ist lokal, wenn es ein k gibt, so dass L k-lokal ist.*

Es ist wichtig zu verstehen dass die Klasse der lokalen Sprachen echt größer ist als jede Klasse von k-lokalen Sprachen. Das folgt aus der Tatsache, dass die k-lokalen Sprachen eine echte, unendliche Hierarchie bilden: nehmen wir an, es gibt ein $i \in \mathbb{N}$, so dass die i-lokalen Sprachen gleich den lokalen Sprachen sind. Da aber die lokalen Sprachen auch die $i+1$-lokalen Sprachen beinhalten, folgt daraus, dass die $i+1$-lokalen Sprachen gleich den i-lokalen sind - Widerspruch zu unserer vorigen Beobachtung der echten, unendlichen Hierarchie der k-lokalen Sprachen.

Ein Beispiel für die größere Ausdruckstärke der lokalen Sprachen ist folgendes: wir wissen dass es für alle $k \in \mathbb{N}$ eine endliche Sprache L gibt, so dass L nicht k-lokal ist. Es gilt hingegen:

Lemma 77 *Jede endliche Sprache ist lokal.*

Der Beweis ist einfach: wir schreiben einfach alle Worte explizit als Regeln auf, und erlauben sonst nichts.

23 Endlich, Koendlich, Nilpotent, Geordnet

Eine wichtige, oft vernachlässigte Familie von Sprachen sind die **endlichen Sprachen**, also die Klasse von Sprachen, die nur aus endlich vielen Worten bestehen. Eine nahe verwandte Klasse von Sprachen sind die **ko-endlichen Sprachen**: eine Sprache $L \subseteq \Sigma^*$ ist ko-endlich, wenn ihr Komplement $\Sigma^* - L$ endlich ist. Sei beispielsweise $\Sigma = \{a, b\}$, und $X = \{a, b, aa, ab, ba, bb\}$. Dann ist $\Sigma^* - X$ koendlich. Die Klasse der **nilpotenten Sprachen** ist die kleinste Klasse, die alle endlichen und koendlichen Sprachen umfasst. Folgendes Resultat bekommen wir mit den üblichen mengentheoretischen Methoden:

Lemma 78 *Nilpotente Sprachen sind abgeschlossen unter Vereinigung, Schnitt und Komplement.*

Aufgabe: liefern Sie den einfachen Beweis!

Eine lineare Ordnung $\leq$ auf einer Menge M ist eine Relation, so dass f.a. $m_1, m_2, m_3 \in M$ gilt:

1. $m_1 \leq m_2, m_2 \leq m_3 \Rightarrow m_1 \leq m_3$ (transitiv),
2. $m_1 \leq m_2, m_2 \leq m_1 \Rightarrow m_1 = m_2$ (antisymmetrisch),
3. $m_1 \leq m_1$ (reflexiv),
4. $m_1 \leq m_2$ oder $m_2 \leq m_1$ (total).

Definition 79 *Sei $\mathfrak{A} = (\Sigma, Q, \delta, q_0, F)$ ein deterministischer endlicher Automat, und $\leq$ eine lineare Ordnung auf Q. $(\mathfrak{A}, \leq)$ ist ein **geordneter Automat**, falls gilt: f.a. $a \in \Sigma$, falls $q \leq q'$, dann ist $\delta(q, a) \leq \delta(q', a)$.*

Definition 80 *Eine Sprache ist **geordnet**, wenn sie von einem geordneten Automaten erkannt wird.*

Die geordneten Sprachen sind aus folgendem Grund interessant: es gibt streng lokale Sprachen, die nicht geordnet sind, wie etwa $(ab)^*$ (*Aufgabe*: zeigen Sie warum!); aber es gilt:

Lemma 81 *Jede Nilpotente Sprache ist geordnet.*

Das lässt sich wie folgt zeigen: zunächst ist jede endliche Sprache geordnet; wir brauchen nur mehr Zustände als das längste Wort der Sprache lang ist (es gibt nämlich ein längstes Wort für jede endliche Sprache!). Für ko-endliche Sprachen L gilt: es gibt ein n, nämlich die Länge des längsten Wortes das nicht in der Sprache ist, so dass gilt: *jedes* Wort $w : |w| \geq n$ ist in L. Das heißt: ab einer gewissen Länge akzeptieren wir einfach jedes Wort, und dafür brauchen wir nur einen Zustand (der trivialerweise die Ordnungsbedingung respektiert).

Es gibt aber geordnete Sprachen, die nicht nilpotent sind: beispielsweise ba^* ist geordnet (*Aufgabe*: schreiben Sie einen Automaten für diese Sprache und zeigen Sie dass er geordnet ist!), aber weder endlich noch koendlich.

24 Punktweise testbare Sprachen

Eine letzte wichtige Klasse von subregulären Sprachen sind die punktweisen testbaren Sprachen (in Zukunft einfach: punktweise Sprachen). Die Definition ist etwas komplizierter. Wir defineren die Relation $\sqsubseteq$ wie folgt: gegeben ein Alphabet Σ schreiben wir für $w, v \in \Sigma^*$, $w \sqsubseteq v$ genau dann wenn es ein i gibt, so dass $w = w_1 w_2 ... w_i$, und $v = x_1 w_1 x_2 w_2 x_. ... x_i w_i x_{i+1}$ für beliebige $x_1, ..., x_{i+1} \in \Sigma^*$. $\sqsubseteq$ ist also eine verstreute Teilwortrelation. Gegeben ein Wort w bezeichnen wir mit $\uparrow w := \{v : w \sqsubseteq v\}$ die Menge aller Worte von denen w ein (verstreutes) Teilwort ist, und mit $\downarrow w := \{v : v \sqsubseteq w\}$ die Menge aller (verstreuten) Teilworte von w.

Definition 82 *Eine Sprache $L \subseteq \Sigma^*$ ist punktweise, wenn sie sich darstellen lässt als Vereinigung, Schnitt oder Komplement einer endlichen Menge von Sprachen der Form $\uparrow w$, für $w \in \Sigma^*$.*

Ein Beispiel wäre die Sprache $(\uparrow abcd) \cap (\uparrow aaa)$, die aus allen Worten w besteht, so dass $abcd \sqsubseteq w$ und $aaa \sqsubseteq w$.

Es gibt eine andere, äquivalente Charakterisierung der punktweisen Sprachen, die an den Satz von Myhill-Nerode erinnert. Wir definieren $\downarrow_k w := \{v : v \sqsubseteq w \& |v| \leq k\}$, die Menge der (verstreuten) Teilwörter von w die kürzer als k sind, für $k \in \mathbb{N}$. Wir schreiben nun $w \sim_k v$, genau dann wenn $\downarrow_k w = \downarrow_k v$. $\sim_k$ ist eine Äquivalenzrelation, wir können also wieder (s.o.) den Quotienten Σ^* modulo $\sim_k$ ziehen, geschrieben $[\Sigma^*]_{\sim_k}$, nämlich die Menge aller $\sim_k$-Äquivalenzklassen, die Teilmengen von Σ^* sind. Wir führen nun das Konzept der k-punktweisen Sprachen ein.

Definition 83 *Eine Sprache L ist k-punktweise, genau dann wenn es ein k gibt, so dass für $X_1, X_2, ..., X_i \in [\Sigma^*]_{\sim_k}$, $L = \bigcup_{j=1}^{i} X_j$.*

In Worten: L ist genau dann punktweise, wenn es ein k gibt, so dass L eine endliche Vereinigung von $\sim_k$-Äquivalenzklassen ist. Wichtig ist zu bedenken, dass $\sim_k$ keinerlei Bezug auf irgendeine Sprache nimmt (im Gegensatz zu $\sim_L$, der Nerode Äquivalenz). Außerdem gilt: für jedes $k \in \mathbb{N}$ ist $[\Sigma^*]_{\sim_k}$ endlich, da

es ja nur endlich viele Wörter mit Länge $\leq k$ über ein gegebenes Alphabet Σ gibt.

Entscheidend für die punktweisen Sprachen ist folgender Satz:

Satz 84 *Eine Sprache L ist punktweise (im Sinne von Definition 57), wenn es ein k gibt, so dass L k-punktweise ist.*

Es gilt hier dasselbe wie für die lokalen Sprachen: f.a. $k \in \mathbb{N}$ sind die k-punktweisen Sprachen echt enthalten in den $k+1$-punktweisen Sprachen, und folglich gilt: f.a. $k \in \mathbb{N}$ sind die k-punktweisen Sprachen echt enthalten in den punktweisen Sprachen.