20 Vorlesung vom 20. Juni 2000

20.1 Odgens Lemma

Satz:

Sei $G=(V,\Sigma,P,S)$ eine kontextfreie Grammatik. Dann existiert eine Konstante $k\geq 1$ (welche im Allgemeinen recht groß ist), so dass für alle Wörter z der Sprache mit $z\in L(G)$ und $|z|\geq k$, in denen mindestens k Positionen markiert sind, gilt:

Das Wort z kann geschrieben werden, als die Zerlegung $z=uvwxy$ mit den folgenden 4 Bedingungen:

1. w enthält mindestens eine markierte Position,
2. u und v enthalten beide markierte Positionen, oder aber x und y enthalten beide markierte Positionen,
3. vwx enthält höchstens k markiere Positionen
4. Es existiert ein Nichtterminal A mit:

$$S\overset{+}{\underset{G}{\Rightarrow}}uAy\overset{+}{\underset{G}{\Rightarrow}}uvAxy\overset{+}{\underset{G}{\Rightarrow}}\ldots\overset{+}{\underset{G}{\Rightarrow}}uv^iAx^iy\overset{+}{\underset{G}{\Rightarrow}}uv^iwx^iy\quad\forall i\geq 0 \tag{20.1}$$

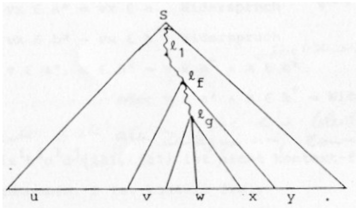

Abbildung 37

Das Bild zeigt, wie man sich dieses am besten vorstellen kann. Es gilt einen Weg vom Startsymbol S aus hin zu den markierten Blättern zu finden. Da nun der Weg sehr lang ist, existiert eine Schliefe. Man betrachtet dann den Teilbaum der letzten Wiederholung. Man sieht nun, dass w zumindest eine markierte Position enthalten muss.

Beweis des Satzes:

Es gelte:

$$\begin{aligned} m &=_{Df} \#(V-\Sigma) \\ l &=_{Df} \max.\left(n \middle| A\rightarrow\alpha\in P,\ |\alpha|=n\right) \\ k &=_{Df} l^{2\cdot m+3} \end{aligned} \tag{20.2}$$

Es sei dann $z\in L(G)$ mit $|z|\geq k$ und weiterhin mindestens k Positionen in z markiert. Schließlich sei T der Ableitungsbaum für z mit der Länge $\geq 2\cdot m+3$. Ein Knoten n heißt Verzweigungsknoten, wenn n mindestens zwei direkte Nachfolger hat. Ein Beispiel wäre, wenn der Knoten n zwei Nachfolger n_1 und n_2 hat, welche beide markierte Blätter unter ihren Nachfolgern haben.

Nun lässt sich ein Weg $n_1,\ldots,n_p$ durch den Ableitungsbaum T wie folgt konstruieren:

1. n_1 ist die Wurzel von T,

2. betrachtet man die Nachfolger eines Knoten n_i. Sollte nur einer von n_i's direkten Nachfolgern markierte Knoten haben, so wird dieser der Nachfolger n_{i+1}.
3. Wenn n_i ein Verzweigungsknoten ist, dann soll n_{i+1} derjenige direkte Nachfolger sein, der die größte Zahl von markierten Blättern als Nachfolger hat. Bei Gleichheit kann n_{i+1} beliebig gewählt werden.
4. Wenn n_i ein Blatt ist, endet die Konstruktion.

Nun ist $n_1, n_2, \ldots, n_p$ der Weg.

Behauptung:

Wenn der Weg $n_1, n_2, \ldots, n_i$ r Verzweigungsknoten enthält, dann hat $n_{i+1} \geq l^{2 \cdot m+3-r}$ markierte Blätter. Der Beweis folgt durch Induktion. Für $i=0, r=0$ gilt:

$$n_1 \text{ hat } \geq 1^{2m+r-0} = k \tag{20.3}$$

markierte Blätter. Angenommen, die Aussage ist richtig für $(i-1)$. Wenn n_i kein Verzweigungsknoten ist, dann haben n_i und n_{i+1} dieselbe Anzahl von markierten Blättern. Wenn n_i ein Verzweigungsknoten ist, dann hat n_{i+1} mindestens $\frac{1}{l} \cdot l^{2m+3-r} = l^{2m+3-(r+1)}$ viele markierte Blätter bei $(r+1)$ Verzweigungsknoten.

In $n_1, n_2, \ldots, n_p$ muss es mindestens $2 \cdot m + 3$ viele Verzweigungsknoten geben. Denn anderenfalls hätte $n_p \geq l^{2m+3-r} > 1 \cdot (r < 2m+3)$ viele markierte Blätter. Das wäre jedoch ein Widerspruch, denn n_p ist ein Blatt, und somit kein Verzweigungsknoten. Es gilt also $p > 2 \cdot m + 3$.

Seien nun $b_1, \ldots, b_{2m+3}$ die letzten $(2m+3)$ Verzweigungsknoten. b_i heißt „*linker Verzweigungsknoten*", wenn ein linker direkter Nachfolger von b_i, der nicht auf dem Weg liegt, ein markiertes Blatt links von n_p enthält. Analog dazu ist b_i der „*rechte Verzweigungsknoten*", wenn ein rechter direkter Nachfolger von b_i, der nicht auf dem Weg liegt, ein markiertes Blatt rechts von n_p enthält.

Ohne Beschränkung der Allgemeinheit gilt: Es gibt mindestens $(m+2)$ viele linke Verzweigungsknoten. Seien nun $l_1, \ldots, l_{m+2}$ die letzten linken Verzweigungsknoten in $b_1, \ldots, b_{2m+3}$. Unter $l_2, \ldots, l_{m+2}$ gibt es dann zwei Knoten l_f und l_g mit der selben Bezeichnung, da $\#(V - \Sigma) = m$. Für den Fall, dass f echt kleiner als g ist, sieht dies wie auf dem folgenden Bild aus:

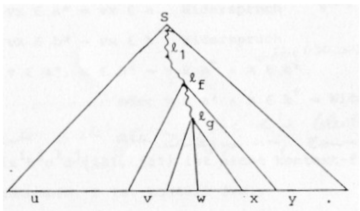

Abbildung 38

24 Index

A

B

F

L

M

P

S

T

23 Abbildungsverzeichnis

Man kann nun $l_f = l_g = A$ nennen. Es ergibt sich dann:

$$S \overset{+}{\Rightarrow} uAy \tag{20.4}$$

$$A \overset{+}{\Rightarrow} vAx \tag{20.5}$$

$$A \overset{+}{\Rightarrow} w \tag{20.6}$$

Und damit gilt dann:

$$S \overset{+}{\Rightarrow} uAy \overset{+}{\Rightarrow} uvAxy \overset{+}{\Rightarrow} uv^iAx^iy \overset{+}{\Rightarrow} uv^iwx^iy \quad \forall i \geq 0 \tag{20.7}$$

Da nun l_1 ein linker Verzweigungsknoten ist, hat u mindestens ein markiertes Blatt. Des weiteren ist l_f ein linker Verzweigungsknoten und damit hat v mindestens ein markiertes Blatt. w enthält als markiertes Blatt n_p.

Es gilt auch, dass b_1 der $(2m+3)$-te Verzweigungsknoten vom ende ist. Somit hat b_1 höchstens $l^{2m+3} = k$ viele markierte Blätter. Da l_f ein Nachfolger von b_1 ist, hat vwx höchstens k markierte Blätter.

Analog zu dem eben vorgeführten Beweis für linke Verzweigungsknoten, läuft auch der Fall für $m+2$ rechte Verzweigungsknoten. Hier enthalten dann x und y jeweils mindestens ein markiertes Blatt.

Korollar: (Pumping Lemma nach Bar-Hillel, Perles, Shamir)

Sei L eine kontextfreie Sprache. Dann existiert eine Konstante k in der Art, dass, wenn $|z| \geq k$ und $z \in L$ ist, dann gilt:

Es existiert für z eine Zerlegung in $z = uvwxy$ mit $u, v, w, x, y \in \Sigma^*$ und:

$$\begin{aligned} & vx \neq \varepsilon \\ & |vwx| \leq k \\ & \forall i \geq 0:\ uv^iwx^iy \in L \end{aligned} \tag{20.8}$$

Zum Beweis dieses Korollars, wählt man eine beliebige kontextfreie Grammatik G für L und markiert alle Positionen in z

20.1.1 Beispiele für Kontextfreiheit und Nichtkontextfreiheit

Behauptung:

Die Sprache $L = \{a^nb^nc^n \mid n \geq 1\}$ ist nicht kontextfrei.

Beweis:

Angenommen die Sprache L wäre kontextfrei. Es sei dann k die Konstante des Pumping Lemmas. Und es gilt:

$$\begin{aligned} & z =_{Df} a^kb^kc^k \\ & z \in L \end{aligned} \tag{20.9}$$

Es existiert dann eine Zerlegung für z mit:

$$\begin{aligned} \exists u,v,w,x,y:\ z &= uvwxy \\ |vwx| &\leq k \\ \Rightarrow vwx &\in a^*b^* \\ \vee vwx &\in b^*c^* \end{aligned} \tag{20.10}$$

Es soll hier nur der Fall $vwx \in a^*b^*$ betrachtet werden. Dafür gilt:

$$\begin{aligned} &v \in a^* \cup b^* \\ &x \in a^* \cup b^* \end{aligned} \tag{20.11}$$

Nun gibt es drei Fälle zu beachten:

1. $vx \in a^* \Rightarrow vx \in a^+$ dies stellt aber einen Widerspruch dar, denn die Anzahl der a's würde schneller wachsen.
2. $vx \in b^* \Rightarrow vx \in b^+$ und wieder haben wir einen Widerspruch, da hie die b's zu schnell wachsen, schließlich bleibt noch
3. $v \in a^*, x \in b^* \Rightarrow \begin{array}{ll} & v \in a^+ \wedge x \in b^* \\ oder & v \in a^* \wedge x \in b^+ \end{array}$ auch in diesem Fall bekommt man einen Widerspruch, denn die c's bleiben auf der Strecke.

Behauptung:

Die Sprache $L = \left\{ a^i b^j c^i d^j \,\middle|\, i \geq 1, j \geq 1 \right\}$ ist nicht kontextfrei.

Beweis:

Angenommen, die Sprache L wäre kontextfrei. Es sei nun n die Konstante des Pumping Lemmas. Dann gibt es eine Zerlegung von $z = a^n b^n c^n d^n$ in $z = uvwxy$ mit $u,v,w,x,y \in \Sigma^*$ und $|vwx| \leq n$.

Wie man sieht, gilt:

1. vx enthält höchstens zwei verschiedene Symbole,
2. Falls vx zwei verschiedene Symbole enthält, müssen diese „benachbart" sein. Dann erhält man durch „pumpen":

$$\left.\begin{aligned} \#a's &> \#c's \\ \#b's &> \#d's \\ \#c's &> \#a's \\ \#d's &> \#b's \end{aligned}\right\} \text{und dies stellt einen Widerspruch dar}$$

Man erhält also jedes Mal einen Widerspruch, da es nicht zu schaffen ist eine gleiche Anzahl von a's und c's oder b's und d's zu erreichen.

Behauptung:

Die Sprache $L = \left\{ a^i b^j c^k \,\middle|\, i \neq j, j \neq k, k \neq i \right\}$ ist nicht kontextfrei.

Beweis:

Angenommen die Sprache L wäre kontextfrei. Sei nun n die Odgens Konstante. Dann gibt es für ein Wort $z = a^n b^{n+n!} c^{nü2 \cdot n!}$ der Sprache L eine Zerlegung $z = uvwxy$. Es seien nun alle a's markiert.

22 Literaturverzeichnis

1. Uwe Schöning: "Theoretische Informatik - kurzgefaßt", 3. Auflage, 1997
2. Spektrum Akademischer Verlag GmbH Heidelberg, Berlin
3. Skript „Theoretische Informatik II", Prof. Dr. Wotschke Universität Frankfurt
4. Vorlesung „Theoretische Informatik II", Prof. Dr. Wotschke Universität Frankfurt
5. Skript „Theoretische Informatik II", Prof. Dr. Schnittger Universität Frankfurt
6. J. E. Hopcroft und J. D. Ullman: "Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie", Addison-Wesley, 3. Auflage, 1996
7. "Schülerduden: Informatik", Dudenverlag, 3. Auflage, 1997
8. "Schülerduden: Die Mathematik II", Dudenverlag, 3. Auflage, 1991

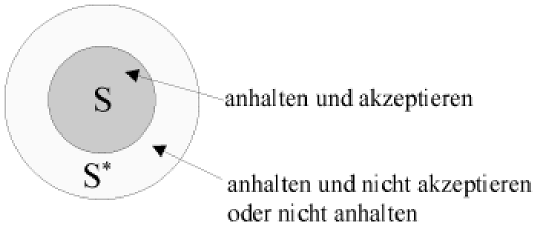

Abbildung 41 - Rekursiv aufzählbare Mengen

Beweis:

L ist rekursiv. Daraus folgt, dass es eine Turing-Maschine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ gibt mit

- $\forall x \in \Sigma^* : M(x)$ hält
- L = T(M)

Sei $M = (Q_1, \Sigma, \Gamma, \delta_1, q_0, B, F)$ eine neue Turingmaschine, die M simuliere. Wir haben $Q_1 = Q \cup \{q'\}$ mit $q' \notin Q$.

Es gelte nun

$$\begin{gathered}\forall q \in Q, a \in \Gamma: \text{ Wenn } \delta(q,a) \text{ für } q \in Q - F \text{ nicht definiert } : \delta_1(q,a) = q' \\ f_1 := \{q'\}\end{gathered} \tag{21.25}$$

Die Maschine M_1 hält immer!

Weiter gilt $x \in T(M) \Leftrightarrow x \notin T(M_1) \Rightarrow \overline{L} = T(M_1)$.

Damit ist $\overline{L}$ auch rekursiv. Die umgekehrte Richtung zeigt man analog.

Satz:

Es gibt eine rekursiv aufzählbare Menge, deren Komplement nicht rekursiv aufzählbar ist.

Beweis:

Es sei $L = \{x_i \mid x_i \in T(M)\}$. Weiter sei U die modifizierte universelle Turingmaschine[15]. Dann gilt:

$$\begin{gathered}L = T(U) \Rightarrow \text{L ist rekursiv aufzählbar} \\ L = \{x_i \mid x_i \notin T(M_i)\} \text{ist nicht rekursiv aufzählbar!}\end{gathered} \tag{21.26}$$

[15] Modifiziert heißt hier: die i-te Turingmaschine muss erst wie in Abschnitt 21.2.7 beschrieben konstruiert werden.

1. u und v enthalten a's.

   Dann gilt: $v \in a^*$ und damit gilt $v = a^i$ für $i \le n$. Es gibt dann die folgenden 3 Fälle:

   $x \in a^* \vee x \in b^* \vee x \in c^*$

   - $x \in a^*$

     Dann gilt: $x = a^j$ und somit $vx = a^{i+j}$ mit $(i+j) \le n$. Dann sollte das Wort:

     $uv^{1+\frac{n!}{i+j}} wx^{1+\frac{n!}{i+j}} y = a^{n+(i+j)\cdot\frac{n!}{(i+j)}} n^{n+n!} c^{n+2n!}$ in der Sprache sein, dies kann jedoch nicht sein, da die Anzahl der a's gleich der Anzahl der b's.

   - $x \in b^*$

     Dann ist $x = b^j$ mit $j \ge 0$ und somit müsste das Wort: $uv^{1+2\cdot\frac{n!}{i}} wx^{1+2\cdot\frac{n!}{i}} = a^{n+i\cdot 2\frac{n!}{i}} b^r c^{n+2n!}$ in der Sprache sein. Jedoch führt auch dies zu einem Widerspruch, da die Anzahl der a's gleich der Anzahl der c's ist. Bleibt noch ein Fall zu behandeln.

   - $x \in c^*$

     hier ist $x = c^j$ für $j \ge 0$. Dann müsste das Wort: $uv^{1+\frac{n!}{i}} wx^{1+\frac{n!}{i}} y = a^{n+i\cdot\frac{n!}{i}} b^{n+n!} c^r$ in der Sprache sein. Jedoch ist hier die Anzahl der a's so groß wie die Anzahl der b's, was einen Widerspruch darstellt.

2. Der Fall, dass x und y beide a's enthalten, funktioniert analog zum 1. Fall. Somit soll es hier nicht wiederholt werden.

Behauptung:

Die Sprache $L = \left\{a^i b^j c^k \,\middle|\, i = j \vee j = k\right\}$ ist eine kontextfreie Sprache, die inhärent mehrdeutig ist.

Beweis:

Zuerst soll noch mal wiederholt werden, was inhärent mehrdeutig bedeutet. Dies bedeutet, dass egal welche Ableitung gewählt wird, immer 2 mögliche Wege existieren.

Um dies zu zeigen, genügt es eine kontextfreie Grammatik aufzustellen, welche mehrdeutig ist. Eine solche kontextfreie Grammatik ist:

$$\begin{aligned}&S \to AB \mid DC \\ &A \to aA \mid \varepsilon \\ &B \to bBc \mid \varepsilon \\ &C \to cC \mid \varepsilon \\ &D \to aDb \mid \varepsilon\end{aligned} \tag{20.12}$$

Diese Grammatik ist mehrdeutig, wie man schnell an folgendem Beispiel erkennt:

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aabBc \Rightarrow aabbBcc \Rightarrow a^2b^2c^2 \tag{20.13}$$

eine andere Ableitung für dieses Wort ist:

$$S \Rightarrow DC \Rightarrow aDbC \Rightarrow aaDbbC \Rightarrow a^2b^2C \Rightarrow a^2b^2cC \Rightarrow a^2b^2c^2C \Rightarrow a^2b^2c^2 \tag{20.14}$$

wir behaupten, dass jede kontextfreie Grammatik für L mehrdeutig sein muss. Sei G eine beliebige Grammatik, die L erzeugt. Sei weiterhin k die Konstante des Odgens Lemma. Es sei nun ohne Einschränkung $k \ge 3$. Sei $z = a^k b^k c^{k+k!}$. Nun seien alle a's markiert und $z \in L$. Nach Odgens Lemma gibt es dann für z eine Zerlegung $z = uvwxy$ für $u, v, w, x, y \in \Sigma^*$. Dann gilt:

1. w enthält mindestens ein a.

2. $uv \in a^*$
3. $x \in a^* \cup b^* \cup c^*$. Hierbei gilt, dass x mit Sicherheit nicht zwei verschiedene Symbole enthalten kann, denn sonst würde durch das Pumpen „gemischte" Folgen erzeugt werden.
4. vwx enthält höchstens k viele a's.

Nun sind die folgenden Fälle zu betrachten: $x \in a^* \vee x \in b^* \vee x \in c^*$:

1. $x \in a^*$

   Dann gilt: $vx \in a^*$ und damit $v = a^i$ für $0 < i \le k$ und man erhält Wörter: $uv^2wx^2y = a^{k+i}b^kc^{k+k!}$ die auch aus der Sprache stammen müssten. Jedoch stellt dieses einen Widerspruch dar, da weder die Anzahl von a's und b's, noch die Anzahl der b's und c's gleich ist.
2. $x \in c^*$

   Es ist dann $v = a^i$ mit $0 < i \le k$ und es gilt $x = c^j$ für $j \ge 0$. Damit bekommt man Wörter: $uv^2wx^2y = a^{k+i}b^kc^{k+k!+j}$, die auch in der Sprache sein müssen. Doch wie schon bei a) bekommt man einen Widerspruch.
3. $x \in b^*$

   Es gilt: $v = a^i$ für $0 < i \le k$ und weiterhin $x = b^j$ für $j \ge 0$. Damit bekommt man die folgenden Wörter: $uv^2wx^2y = a^{k+i}b^{k+j}c^{k+k!}$, die Element der Sprache sein müssen. Es gilt also zwei weitere Fälle zu unterscheiden zum einen den Fall $i = j$ und zum anderen $(j = k! \wedge j \ne i)$:

   - $(j = k! \wedge j \ne i)$

     Damit würden Wörter: $uv^3wx^3y = a^{k+2i}b^{k+2k!}c^{k+k!}$ diese müssten in der Sprache sein, jedoch ist dies ein Widerspruch.
   - $i = j$

     Es gilt damit: $S \overset{+}{\Rightarrow} uAy \overset{+}{\Rightarrow} uv^mAx^my \overset{+}{\Rightarrow} uv^mwx^my$. Für $m = k!/i + 1$ gilt dann:

$$uv^mwx^my = a^{k+k!}b^{k+k!}c^{k+k!} \tag{20.15}$$

Eine ähnliche Argumentation liefert dann für den umgekehrten Fall: $a^{k+k!}b^kc^k$. Für $u',v',w',x',y' \in \Sigma^*$ mit $a^{k+k!}b^kc^k = u'v'w'x'y'$ und mit $v' \in b^*$. Schließlich kommt man zu der Ableitung:

$$S \overset{+}{\Rightarrow} u'By' \overset{+}{\Rightarrow} u'(v')^{m'}B(x')^{m'}y' \overset{+}{\Rightarrow} u'(v')^{m'}w'(x')^{m'}y' = a^{k+k!}b^{k+k!}c^{k+k!} \tag{20.16}$$

Behauptung:

Diese beiden Ableitungen repräsentieren verschiedene Ableitungsbäume.

Beweis:

Angenommen, diese beiden Ableitungen beschreiben denselben Ableitungsbaum.

A erzeuge a's und b's während, B b's und c's erzeugt. Somit ist A kein Nachfolger von B und B ist kein Nachfolger von A. Betrachtet man nun die Ableitung:

$$S \overset{+}{\Rightarrow} t_1At_2Bt_3 \text{ mit } t_1,t_2,t_3 \in \Sigma^*$$

Nun muss für alle i, j gelten:

$$t_1v^iwx^it_2(v')^jw'(x')^jt_3 \in L$$

Satz (Halteproblem):

Es gibt keinen Algorithmus (Turing-Maschine), der für ein beliebiges Paar (M,x), wobei M eine Turing-Maschine und x ein Wort sei, entscheidet, ob M auf x hält.

Beweis:

Wir nehmen an, dass ein solcher Algorithmus A existiert:

$$A(M,x) = \begin{cases} 0 \Leftrightarrow \text{M hält nicht auf x} \\ 1 \Leftrightarrow \text{M hält auf x} \end{cases} \tag{21.24}$$

Wir behaupten, dass $L_1 := \{x_i \mid x_i \notin T(M_i)\}$ von einer Turing-Maschine akzeptierbar ist. U sei die universelle Turing-Maschine.

Wir konstruieren jetzt eine Turing-Maschine M:

1. Eingabe ist x
2. Durch sukzessives Iterieren der Worte $x_1,x_2,...,x_n$ bestimmen wir das i mit $x_i = x$. (Wir bestimmen also die Nummer des Wortes x)
3. Durch sukzessives Iterieren erzeugen wir über dem Kodierungsalphabet die Turing-Maschine M_i.
4. Jetzt wenden wir A (M_i,x_i) an.
5. Wenn A $(M_i,x_i) = 0 \Rightarrow$ M akzeptiert x_i.
6. Wenn A $(M_i,x_i) = 1 \Rightarrow$ M akzeptiert nicht x_i
7. Mi akzeptiert nicht $x_i \Rightarrow$ M akzeptiert x_i

Daraus folgt: $T(M) = \{x_i \mid x_i \notin T(M_i)\}$. Das ist aber ein Widerspruch zum obigen Lemma. Damit folgt ein Widerspruch zur Annahme und die gewünschte Turing-Maschine kann nicht existieren.

21.2.8 Rekursive Mengen

Definition (rekursive Menge):

Eine Menge S heißt „*rekursiv*" $\Leftrightarrow$ Es existiert eine Turing-Maschine M, die auf *allen Eingaben* aus Σ^* hält, mit S = T(M). Wichtig ist hierbei die Aussage „auf allen Eingaben". Bildlich kann man sich die Definition wie in Abbildung 40 vorstellen.

Defintion (rekursiv aufzählbar):

Eine Menge heißt „*rekursiv aufzählbar*" :$\Leftrightarrow$ Es existiert eine Turing-Maschine M mit S = T(M).

Diese Situation ist in Abbildung 41 dargestellt.

Satz:

Eine Menge $L \subseteq \Sigma^*$ ist rekursiv $\Leftrightarrow \overline{L} = \Sigma^* - L$ ist rekursiv.

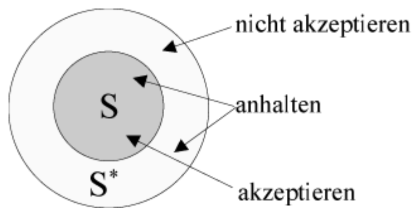

Abbildung 40 - Rekursive Mengen

Das Band sieht zu Beginn wie folgt aus (mit Markierung von m_1, m_2):

	m_1		m_2
ccc	...	ccc	Kod(x)

Die Vorgehensweise ist nun die folgende:

1. U sucht m2 und merkt sich das dort gefundene Symbol, z. B. A ∈ {B,0,1}
2. U sucht im aktuellen Zustandsblock (Merker: m1) die zugehörige Anweisung für A
3. U merkt sich, wodurch A zu ersetzen ist und ob nach rechts oder links gegangen wird
4. U versetzt m1 an den Anfang des Zustandsblocks, der als Nachfolgezustand angegeben wird (Subroutine)
5. U geht zu m2, ändert A und versetzt m2 nach links oder nach rechts
6. Gehe nach 1

Diese Vorgehensweise kann natürlich auch formalisiert werden.

Insgesamt folgt aber

$$\begin{aligned} &U \text{ hält auf } Kod(M,x) \Leftrightarrow M \text{ hält auf } x \\ &U \text{ akzeptiert } (M,x) \Leftrightarrow M \text{ akzeptiert } x \end{aligned} \tag{21.21}$$

In diesem Ablauf haben wir die akzeptierenden Zustände von M nicht betrachtet. Man kann sich aber leicht vorstellen, dass jede Turingmaschine mit mehreren akzeptierenden Zuständen in eine äquivalente Turingmaschine mit nur einem akzeptierenden Zustand umgewandelt werden kann (indem man einfach einen neuen Zustand einführt und alle akzeptierenden Zustände auf diesen verweist). Dann kann die universelle Turingmaschine durch eine geeignete Erweiterung der Kodierung von M leicht feststellen, ob diese in einem akzeptierenden Zustand ist.

21.2.7 Halteproblem für Turing-Maschinen

Wir formulieren das Halteproblem:

Gibt es einen Algorithmus (Turing-Maschine), der für beliebige Paare (M, x), wobei M eine Turing-Maschine und x ein Wort sei, entscheidet, ob M auf x hält?

Wir kodieren wieder, diesmal jedoch *alle* Turing-Maschinen in einem endlichen Alphabet und alle Eingabewerte in $\{0,1\}^*$. In beiden Kodierungen existiert eine lineare (lexikographische Ordnung).

Wir beweisen zunächst einen Hilfssatz:

Lemma:

Sei $L_1 := \{x_i \mid x_i \notin T(M_i)\}$ die Sprache der Wörter mit der Nummer i, die nicht von der i-ten Turing-Maschine akzeptiert werden. Zu dieser Sprache existiert keine Turingmaschine mit $T(M) = L_1$.

Beweis:

Angenommen, es gäbe eine solche Turingmaschine M mit $L_1 = T(M)$. Daraus folgt automatisch

$$\exists j : M = M_j \wedge L_1 = T(M_j) \tag{21.22}$$

(weil M ja selbst eine Turingmaschine ist und sie daher eine Nummer j besitzen muss). Dann aber gilt

$$\begin{aligned} &x_j \in L_1 \Rightarrow x_j \in T(M_j) \Rightarrow x_j \notin L_1 \\ &x_j \notin L_1 \Rightarrow x_j \notin T(M_j) \Rightarrow x_j \in L_1 \end{aligned} \tag{21.23}$$

Beide Fälle sind Widersprüche, so dass die Annahme falsch gewesen sein muss.

Nun ist $i = j$ hinreichend groß zu wählen.

Es gilt $|v| = |x|$ und $|v'| = |x'|$. Damit ist also $v \in a^+, x \in b^+ \wedge v' \in b^+, x' \in c^+$. Somit erhält man also $z' \in L$ mit $\#_b(z') > \#_a(z') \wedge \#_b(z') > \#_c(z')$. Dies ist jedoch ein Widerspruch, da wir entweder eine gleiche Anzahl von a's und b's wollten, oder aber eine gleiche Anzahl von b's und c's. Somit können die beiden Ableitungen nicht den selben Ableitungsbaum beschrieben haben.

21 Vorlesung vom 27. Juni 2000

21.1 Deterministische PDA

Wir betrachten nun eine Sprachklasse, die echt zwischen den regulären Mengen und den kontextfreien Sprachen liegen: die *deterministischen kontextfreien Sprachen* (DCFL). Diese sind interessant, da es sich herausstellt, dass die Syntax vieler Programmiersprachen mittels DCFLs beschrieben werden kann.

Definition:

Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ ein PDA. M heißt dann deterministisch (DPDA): $\Leftrightarrow$

$$\begin{aligned} &1.\ \forall q \in Q, a \in \Sigma \cup \{\varepsilon\}, A \in \Gamma : \left|\delta(q,a,A)\right| \leq 1 \\ &2.\ \delta(q,\varepsilon,A) \neq \emptyset \Rightarrow \forall a \in \Sigma : \delta(q,a,A) = \emptyset \end{aligned} \tag{21.1}$$

$$T(M) := \left(x \middle| x \in \Sigma^* \wedge (q_0, x, Z_0) \overset{*}{\mapsto} (f, \varepsilon, \alpha), f \in F, \alpha \in \Gamma^* \right) \tag{21.2}$$

Regel 1 verhindert Wahlmöglichkeiten für dieselbe Eingabe. Regel 2 verhindert die Wahl zwischen der Benutzung eines Eingabesymbols und einer ε-Bewegung.

Definition:

Eine Sprache heißt deterministisch kontextfrei (dkfS): $\Leftrightarrow$ Es existiert ein DPDA mit L = T(M).

21.1.1 Beispiele

- Die Sprache $\{a^n b^n \mid n \geq 1\}$ ist deterministisch.
- Die Sprache $\{wcw^R \mid w \in \{a,b\}^*\}$ ist deterministisch.
- Die Sprache $\{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$ ist nicht deterministisch.

Beim letzten Beispiel müsste der Automat raten, ob für ein a nun ein oder zwei b vorliegen müssen.

Dagegen gilt:

- $d_1\{a^n b^n \mid n \geq 1\} \cup d_2\{a^n b^{2n} \mid n \geq 1\}$ ist deterministisch, falls gilt $d_1 \neq d_2$.
- $\{a^n d_1 b^n \mid n \geq 1\} \cup \{a^n d_2 b^{2n} \mid n \geq 1\}$ ist deterministisch, falls gilt $d_1 \neq d_2$. Dabei wird so vorgegangen: zunächst werden für jedes a zwei Symbole auf den Keller gelegt. Wenn d_1 gelesen wird, so wird je ein Symbol wieder vom Keller gelöscht, ansonsten wird mit den je zweiten weitergearbeitet.
- $\{a^n b^n \mid n \geq 1\} d_1 \cup \{a^n b^{2n} \mid n \geq 1\} d_2$ ist nicht deterministisch.

21.1.2 kfS und dkfS

Satz:

Sei L eine kontextfreie Sprache (kfS). Dann gibt es eine deterministische kontextfreie Sprache L‘ (dkfS) und einen Homomorphismus h mit h(L‘) = L.

Beweis:

Sei L kontextfrei. Daraus folgt, dass es einen PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ geben muss mit L = T(M).

- c ein Trennzeichen für verschiedene Blöcke unserer Codierung
- 1 verwenden wir zur Darstellung der Zustände
- 0 verwenden wir, wenn δ an einer Stelle nicht definiert ist
- L, R sind die Kopfbewegungen

Wir verwenden folgendes Schema:

ccc	.	c	.	c	.	cc	.	c	.	c	.	cc	.	ccc
1)	Block für Eingabe „B“	2)	„0“	2)	„0“	3)	„B“	1)	„0“	1)	„1“	3)		1)

Dabei gilt folgendes:

1. ccc bezeichnet die Randmarkierung
2. c bezeichnet die Eingabesymbolblockmarkierung
3. cc bezeichnet die Zustandsblockmarkierung

Jeder Zustand erhält einen Zustandsblock. In jedem Zustandsblock gibt es je einen Eingabesymbolblock für die drei möglichen Eingabesymbole B, 0 , 1. Ein Eingabesymbolblock sieht beispielhaft so aus:

c111L1c

Dies bedeutet, dass in den Zustand 3 (111) gegangen, der Kopf nach links bewegt und an die aktuelle Position eine 1 geschrieben werden soll.

Beispiel:

Wir wollen die Kodierung folgender Übergangstabelle haben:

$q_1$1	$q_2$0R
q_2B	$q_3$1L
$q_2$0	$q_3$1L
$q_2$1	$q_2$1R
q_3B	$q_4$0R
$q_3$0	$q_4$0R
$q_3$1	$q_3$1L

Nach dem obigen Schema lautet die Kodierung:

ccc	0c0c11R0	cc	111L1c111L1c11R1	cc	1111R0c1111R0c111L1	cc	0c0c0	ccc
	q_1		q_2		q_3		q_4	

21.2.6 Arbeitsweise der universellen Turing-Maschine

Bei der Arbeit benutzt die virtuelle Turing-Maschine zwei zusätzliche Symbole m_1, m_2 zur Markierung. Dabei markiert m_1 den aktuellen Zustandsblock auf dem Eingabeband (auf dem „cc“) und m_2 das aktuelle Eingabezeichen.

Es ist

$$T(M) = \left\{ w \mid w \in \Sigma^* \wedge (q_0 w) \overset{*}{\mapsto} (\alpha_1 f \alpha_2), f \in F \right\} \quad (21.18)$$

die Menge der von M akzeptierten Eingabewörter.

Wir vereinbaren, dass M in einem akzeptierenden Zustand $f \in F$ stets anhält.

Beispiel

Wir betrachten eine Turing-Maschine für die Sprache $L = \left\{ 0^n 1^n \mid n \geq 1 \right\}$.

Es sind $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F), Q = \{q_0, q_1, \ldots, q_5\}, \Sigma = \{0,1\}, \Gamma = \{0,1,B,X,Y\}, F = \{q_5\}$.

Die Übergangsfunktion δ geben wir in folgender Tabelle an:

$q_0 0$	$q_1 XR$	erste 0 von links durch X ersetzen
$q_1 0$	$q_1 0R$	zur ersten 1 nach rechts gehen
$q_1 Y$	$q_1 YR$	durch Y ersetzen
$q_1 1$	$q_2 YL$	
$q_2 Y$	$q_2 YL$	nach links die erste 0 suchen
$q_2 X$	$q_3 XR$	falls keine mehr da: in q_3 gehen
$q_2 0$	$q_4 0L$	
$q_4 0$	$q_4 0L$	nach links das erste X suchen
$q_4 X$	$q_0 XR$	
$q_3 Y$	$q_3 YR$	keine Nullen mehr da ⇒
$q_3 B$	$q_5 YR$	prüfen, ob auch keine Einsen mehr da sind.
		Falls ja: in akzeptierenden Zustand gehen!

21.2.5 Universelle Turingmaschine

Wir wollen nun eine Turing-Maschine bauen, die andere Turingmaschinen simulieren kann – und zwar auf jedes Eingabewort hin. Wir wollen also eine Turing-Maschine U mit folgender Eigenschaft:

- M sei eine beliebige Turingmaschine
- x sei ein beliebiges Eingabewort
- Dann führt U bei Eingabe von (M,x) die Simulation von M auf x durch.

Jetzt ist aber die Frage, wie die Eingabe (M,x) aussehen soll. Hierfür müssen wir eine Kodierung auf dem Band vornehmen.

Wir stellen zunächst folgendes fest:

Zu jedem Alphabet Γ - {B} gibt es eine Kodierung in $\{0,1\}^*$, so dass gilt

$$Kod(M) \text{ akzeptiert } Kod(x) \Leftrightarrow \text{M akzeptiert x} \quad (21.19)$$

Wir nehmen also im weiteren an, dass alle Turing-Maschinen auf einem Alphabet {0,1,B} mit Σ = {0,1} arbeiten. Wir wollen nun die zu simulierende Turing-Maschine selbst kodieren. Wir wählen hierfür das Alphabet

$$\{c, 0, 1, L, R\} \quad (21.20)$$

Dabei ist

Es sei

$$m := \max\left\{ n \mid (q', \alpha) \in \delta(q, a, A), |\alpha| = n \right\} \quad (21.3)$$

Also ist m die maximale Länge von Symbolen, die auf den Keller gestellt werden. Es sei weiter

$$\Sigma' := \Sigma \times Q \times \Gamma^{\leq m} \quad (21.4)$$

mit $\Gamma^{\leq m} := \bigcup_{i=0}^{m} \Gamma^i$.

Wir definieren nun den deterministischen PDA M‘ wie folgt:

$$M' = (Q, \Sigma', \Gamma, \delta', q_0, Z_0, F) \quad (21.5)$$

mit

$$\delta'(q, [a, q', \alpha], A) := \left\{ (q', \alpha) \mid (q', \alpha) \in \delta(q, a, A) \right\} \quad (21.6)$$

Wir müssen durch Induktion den Determinismus zeigen:

$$\begin{gathered} (q_0, [a_1 \ldots a_n, Z_0]) \underset{M}{\mapsto} (q_{i_1}, a_2 \ldots a_n, \alpha_1) \\ \ldots \mapsto (f, \varepsilon, \gamma) \\ \Leftrightarrow \\ (q_0, [a_1, q_{i_1}, \alpha_1][a_2, q_{i_2}, \square] \ldots [a_n, f, \square], Z_0) \underset{M'}{\mapsto} (q_{i_1}, [a_2, q_{i_2}, \square] \ldots [a_n, f, \square]) \\ \ldots \mapsto (f, \varepsilon, \gamma) \end{gathered} \quad (21.7)$$

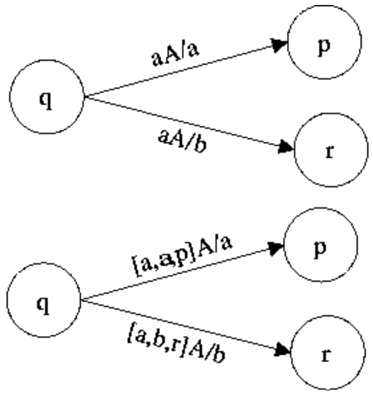

Abbildung 39 - Beispiel zum Beweis

In Abbildung 39 sieht man das Prinzip des Beweises. Oben sieht man einen nichtdeterministischen Kellerautomaten (da bei der gleichen Eingabe in zwei verschiedene Zustände p, r gewechselt werden kann).

Im unteren Teil der Abbildung sieht man nun den deterministischen Automaten mit den neuen Eingabesymbolen [a,α,p] und [a,β,r]. Es wird also eine Alphabetvergrößerung vorgenommen. Um diese Zeichen wieder auf die alten Zeichen zurückzumappen wird der folgende Homomorphismus definiert:

$$h:\Sigma' \to \Sigma \text{ mit } h\left(\left[a,q',\alpha\right]\right) := a \tag{21.8}$$

Somit gilt $h\left(T\left(M'\right)\right) = T\left(M\right) = L$.

21.2 Turing-Maschinen und Entscheidbarkeit

21.2.1 Problem, Algorithmus, Entscheidbarkeit

Definition (Problem):

Ein *Problem* P ist ein Paar $P = \langle V, \chi \rangle$ mit

$$\begin{aligned}&1.\ \text{V ist die Menge der Objekte}\\&2.\ \chi \text{ ist ein Prädikat } \chi : V \to \{0,1\}\end{aligned} \tag{21.9}$$

χ ist also letztendlich die Beschreibung des Problems, das wir lösen wollen.

Definition (Algorithmus):

Ein *Algorithmus A* für ein Problem $P = \langle V, \chi \rangle$ ist eine endliche Menge von Instruktionen, die für alle $x \in V$ hält mit

$$A\left(x\right) = \chi\left(x\right)\ \forall x \in V \tag{21.10}$$

Definition (Entscheidbarkeit):

Ein Problem $P = \langle V, \chi \rangle$ heißt entscheidbar $:\Leftrightarrow \exists$ ein Algorithmus A für $\langle V, \rho \rangle$

Beispiel 1

Das Problem $P = \langle V, \rho \rangle$ mit

$$\begin{aligned}&V = \left\{\left(G,x\right) \mid \text{G eine reguläre Grammatik, x ein Wort}\right\}\\&\rho : v \to \{0,1\}\\&\rho\left(G,x\right) = 1 \Leftrightarrow x \in L\left(G\right)\\&\rho\left(G,x\right) = 0 \Leftrightarrow x \notin L\left(G\right)\end{aligned} \tag{21.11}$$

Beispiel 2

Das Problem (Leerheitsproblem) $P = \langle V, \rho \rangle$ mit

$$\begin{aligned}&V = \left\{G \mid \text{G eine reguläre Grammatik}\right\}\\&\rho : v \to \{0,1\}\\&\rho\left(G,x\right) = 1 \Leftrightarrow L\left(G\right) = \emptyset\\&\rho\left(G,x\right) = 0 \Leftrightarrow L\left(G\right) \neq \emptyset\end{aligned} \tag{21.12}$$

21.2.2 Modelle für Algorithmen

Es gibt unter anderem die folgenden beiden Modelle für Algorithmen:

- Turing Maschinen (Alan Turing)
- Rekursive Funktionen (Kurt Gödel)

Diese beiden Ansätze sind jedoch äquivalent.

21.2.3 Churchsche These

Ende endliche Menge A von Instruktionen ist ein Algorithmus genau dann, wenn eine Turing-Maschine m existiert mit:

$$\begin{aligned}&1.\ m(x) \text{ hält für alle x}\\&2.\ m(x) \text{ erreicht einen Endzustand} \Leftrightarrow A\left(x\right) = 1.\end{aligned} \tag{21.13}$$

21.2.4 Turing-Maschine

Definition (Turing-Maschine):

Eine Turing-Maschine M ist ein 7-Tupel $M = \left(Q, \Sigma, \Gamma, \delta, q_0, B, F\right)$ mit

$$\begin{aligned}&1.\ \text{Q ist eine endliche Menge von Zuständen}\\&2.\ \Gamma \text{ ist eine endliche Menge von Symbolen (Bandalphabet)}\\&3.\ B \in \Gamma \text{ ist das "Blank"-Symbol}\\&4.\ \Sigma \text{ ist das Eingabealphabet mit } \Sigma \subset \Gamma, B \notin \Sigma\\&5.\ q_0 \text{ ist der Anfangszustand}\\&6.\ \text{F ist die Menge der akzeptierenden Zustände}\\&7.\ \delta : Q \times \Gamma \to Q \times \left(\Gamma - \{B\}\right) \times \{L, R\}\end{aligned} \tag{21.14}$$

Dabei ist zu beachten, dass {L, R} die Bewegungen des Schreib-/Lesekopfes darstellen sollen. δ ist in der Regel partiell, d. h. nicht überall definiert.

Eine Turingmaschine kann man sich also vorstellen als eine Maschine, die auf einem (Eingabe)band operiert. Sie kann dabei ein Zeichen lesen und abhängig von diesem und dem momentanen Zustand, in dem sie sich befindet, ein neues Zeichen an die Stelle des Gelesenen schreiben, in einen neuen Zustand wechseln und den Schreib-/Lesekopf nach recht oder links um eine Stelle bewegen.

Definition (Konfiguration):

Sei $M = \left(Q, \Sigma, \Gamma, \delta, q_0, B, F\right)$ eine Turing-Maschine. Eine *Konfiguration* von M wird wie folgt bezeichnet:

$$\alpha_1 q \alpha_2 \tag{21.15}$$

Dabei gilt $\alpha_1\alpha_2 \in \Gamma^*$, so dass α_1 das am weitesten links stehende Nicht-Blank-Symbol und α_2 das am weitesten rechts stehende Nicht-Blank-Symbol enthält. q ist der gegenwärtige Zustand der Maschine und kennzeichnet auch die Position des Schreib-/Lesekopfes.

Wir können die Bewegung einer Turing-Maschine wie folgt formal beschreiben. Zunächst die Linksbewegung:

$$\begin{aligned}&X_1X_2 \ldots X_{i-1}qX_i \ldots X_n \mapsto X_1X_2 \ldots X_{i-2}pX_{i-1}Y \ldots X_n\\&\Leftrightarrow \delta\left(q, X_i\right) = \left(p, Y, L\right)\end{aligned} \tag{21.16}$$

Hierbei wird vom Zustand q aus das aktuelle Zeichen (X_i) ersetzt durch Y, in den Zustand p verzweigt und der Kopf eine Stelle nach links bewegt.

Analog die Rechtsbewegung:

$$\begin{aligned}&X_1X_2 \ldots X_{i-1}qX_i \ldots X_n \mapsto X_1X_2 \ldots X_{i-1}YpX_{i+1} \ldots X_n\\&\Leftrightarrow \delta\left(q, X_i\right) = \left(p, Y, R\right)\end{aligned} \tag{21.17}$$

Wir bezeichnen mit $\overset{+}{\mapsto}$ und $\overset{*}{\mapsto}$ die üblichen Erweiterungen.