

Übung zur Vorlesung Theoretische Informatik I

Abgabetermin: **Mittwoch**, den **07. Juli 2004** bis spätestens 12:00 Uhr.

Aufgabe 21 *kontextfreie Sprache* \rightarrow *DPDA* (5+3 = 8 Punkte)

Sei $L_1 = \{w \in \{a, b\}^+ \mid \#_a w = 2 \cdot \#_b w\}$, wobei $\#_a w$ ($\#_b w$) der Anzahl von a -Symbolen (b -Symbolen) in w entspricht.

- a) Geben Sie einen DPDA an, welcher L_1 akzeptiert.
- b) Gibt es einen DPDA, welcher L_1 mit leerem Keller akzeptiert? Geben Sie ggf. einen solchen DPDA an, und begründen Sie Ihre Antwort ausführlich.

Aufgabe 21 **(Lösungsvorschlag)** *kontextfreie Sprache* \rightarrow *DPDA*

- a) – Idee: Der DPDA berechnet auf dem Keller den Wert $\#_a w - 2\#_b w$
 - Falls aus der Eingabe ein a gelesen wird und
 - * das oberste Kellersymbol $\#$ oder A ist, so wird ein A auf den Keller gelegt (Regeln 1 und 3).
 - * das oberste Kellersymbol B ist, so wird dieses vom Keller gelöscht (Regel 4).
 - Falls aus der Eingabe ein b gelesen wird und
 - * das oberste Kellersymbol $\#$ oder B ist, so werden zwei B s auf den Keller gelegt (Regeln 2 und 6).
 - * die obersten beiden Kellersymbole A s sind, so werden diese vom Keller gelöscht (Regeln 7 und 8).
 - * nur ein A auf dem Keller liegt, so wird das A durch ein B ersetzt (Regeln 7 und 9).
 - z_{accept} ist einziger Start- und Endzustand.
 - Befindet sich der DPDA im Zustand z_{reject} und ist das oberste Kellersymbol $\#$, dann muß der DPDA akzeptieren (Regel 5).

- (1) $(z_{\text{accept}}, a, \#) \rightarrow (z_{\text{reject}}, A\#)$
- (2) $(z_{\text{accept}}, b, \#) \rightarrow (z_{\text{reject}}, BB\#)$
- (3) $(z_{\text{reject}}, a, A) \rightarrow (z_{\text{reject}}, AA)$
- (4) $(z_{\text{reject}}, a, B) \rightarrow (z_{\text{reject}}, \varepsilon)$
- (5) $(z_{\text{reject}}, \varepsilon, \#) \rightarrow (z_{\text{accept}}, \#)$
- (6) $(z_{\text{reject}}, b, B) \rightarrow (z_{\text{reject}}, BBB)$
- (7) $(z_{\text{reject}}, b, A) \rightarrow (z_{\text{del } A}, \varepsilon)$
- (8) $(z_{\text{del } A}, \varepsilon, A) \rightarrow (z_{\text{reject}}, \varepsilon)$
- (9) $(z_{\text{del } A}, \varepsilon, \#) \rightarrow (z_{\text{reject}}, B\#)$

- b) Es gibt keinen DPDA, der L_1 mit leerem Keller akzeptiert. Betrachten wir zum Beispiel das Wort $aabaab \in L_1$. Da bereits $aab \in L_1$ ist und somit vom DPDA akzeptiert werden muß, muß der Keller nach dem Lesen von aab bereits leer sein. Da nach Definition der Übergangsfunktion $\delta: Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Z \times \Gamma^*$ ein Stack-symbol ungleich ε gelesen werden muss, kann daher die Rechnung nicht fortgesetzt und die Eingabe nicht vollständig gelesen werden. Die Eingabe $aabaab$ kann daher nicht akzeptiert werden.

Aufgabe 22 PDA's (4+4 = 8 Punkte)

Zeigen Sie, dass es zu jedem PDA P_ε , welcher die Eingabe mit leerem Keller akzeptiert, einen äquivalenten PDA P_E gibt, welcher mit Endzustand akzeptiert, und umgekehrt.

Hinweis: Überlegen Sie sich dazu, wie P_E definiert werden muß, sodass er P_ε simuliert, und umgekehrt.

Aufgabe 22 (Lösungsvorschlag) PDA's

- (\Rightarrow) Die Idee besteht darin, dass P_E den P_ε simuliert und genau dann in einen Endzustand übergeht, wenn P_ε seinen Keller geleert hat. Sei $P_\varepsilon = (Z, \Sigma, \Gamma, \delta, q_0, \#, \emptyset)$. Dann definieren wir $P_E = (Z \cup \{q'_0, q_f\}, \Sigma, \Gamma \cup \{\perp\}, \delta', q'_0, \perp, \{q_f\})$, wobei δ' wie folgt definiert ist:

$$\begin{aligned} \delta' &= \delta \cup \\ &\quad \{(q'_0, \varepsilon, \perp) \rightarrow (q_0, \# \perp)\} \cup \\ &\quad \{(q, \varepsilon, \perp) \rightarrow (q_f, \varepsilon) \mid \forall q \in Z\} \end{aligned}$$

- (\Leftarrow) Die Übergangsrelation δ muß so angepasst werden, dass nach Erreichen eines Endzustandes der verbleibende Kellerinhalt gelöscht wird. Ein Problem tritt auf, falls der ursprüngliche PDA die Eingabe nicht akzeptiert, aber dennoch den Keller leert. Damit die Simulation in diesem Fall die Eingabe nicht fälschlicher Weise akzeptiert, muß ein zusätzliches, in dem ursprünglichen Automaten nicht vorhandenes Kellerzeichen vor Beginn der "Simulation" auf den Stack gelegt werden. Dieses kann durch die Simulation nicht gelöscht werden. Sei $P_E = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$. Dann definieren wir $P_\varepsilon = (Z \cup \{q'_0, q_{\text{erase}}\}, \Sigma, \Gamma \cup \{\perp\}, \delta', q'_0, \perp, \emptyset)$, wobei δ' wie folgt definiert ist:

$$\begin{aligned} \delta' &= \delta \cup \\ &\quad \{(q'_0, \varepsilon, \perp) \rightarrow (q_0, \# \perp)\} \cup \\ &\quad \{(q, \varepsilon, \gamma) \rightarrow (q_{\text{erase}}, \varepsilon) \mid \forall q \in E, \forall \gamma \in \Gamma \cup \{\perp\}\} \cup \\ &\quad \{(q_{\text{erase}}, \varepsilon, \gamma) \rightarrow (q_{\text{erase}}, \varepsilon) \mid \forall \gamma \in \Gamma \cup \{\perp\}\} \end{aligned}$$

Aufgabe 23 PDA \rightarrow kontextfreie Grammatik (3+5 = 8 Punkte)

Gegeben sei der PDA $P_\varepsilon = (\{q_0, q_1\}, \{a, b\}, \{a, b, \#, S, A\}, \delta, q_0, \#, \emptyset)$, wobei δ wie folgt definiert ist:

- | | |
|---|---|
| (1) $(q_0, \varepsilon, \#) \rightarrow (q_1, S\#)$ | |
| (2) $(q_1, a, S) \rightarrow (q_1, SA)$ | (4) $(q_1, a, S) \rightarrow (q_1, A)$ |
| (3) $(q_1, b, S) \rightarrow (q_1, SA)$ | (5) $(q_1, b, S) \rightarrow (q_1, A)$ |
| (6) $(q_1, a, A) \rightarrow (q_1, \varepsilon)$ | (7) $(q_1, \varepsilon, \#) \rightarrow (q_1, \varepsilon)$ |

- a) Geben Sie an, welche Sprache von P_ε akzeptiert wird. Begründen Sie Ihre Antwort ausführlich.
- b) Konstruieren Sie mit dem in der Vorlesung vorgestellten Verfahren eine kontextfreie Grammatik, die $L(P_\varepsilon)$ erzeugt.

Aufgabe 23 (Lösungsvorschlag) $PDA \rightarrow$ kontextfreie Grammatik

- a) Zunächst wird das Symbol S auf den Keller gelegt und in den Zustand q_1 gewechselt (Regel 1). Der Keller kann nur im Zustand q_1 geleert werden (Regel 7), aber vorher muß das Symbol S durch Lesen mindestens eines Zeichens aus der Eingabe verarbeitet worden sein. Daher kann das leere Wort nicht in der Sprache enthalten sein.

Mit den Regeln 2 und 3 können beliebig a - und b -Zeichen aus der Eingabe gelesen werden, und jedesmal wird der Keller um ein A erweitert. Das oberste Kellersymbol bleibt S , und der aktuelle Kellerinhalt gibt an, wieviel Zeichen aus der Eingabe bereits gelesen wurden. Durch Anwendung der Regeln 4 und 5 wird zusätzlich S vom Keller entfernt, was zur Folge hat, dass der Keller jetzt nur noch durch Lesen von a -Zeichen aus der Eingabe geleert werden kann (Regel 6). Zur Leerung des Kellers müssen also genauso viele a -Zeichen gelesen werden, wie zuvor a - und b -Zeichen zur Füllung des Kellers gelesen wurden.

P_ε akzeptiert somit die folgende Sprache:

$$L(P_\varepsilon) = \{w \circ a^n \mid w \in \{a, b\}^+ \wedge |w| = n\}$$

- b) – $\{S \rightarrow (q_0, \#, q) \mid q \in Z\}$
 $S \rightarrow (q_0, \#, q_0) \quad S \rightarrow (q_0, \#, q_1)$
– $\{(q, A, q') \rightarrow a \mid (q', \varepsilon) \in \delta(q, a, A)\}$
 $(q_1, A, q_1) \rightarrow a \quad (q_1, \#, q_1) \rightarrow \varepsilon$
– $\{(q, A, q') \rightarrow a(q_1, B, q') \mid (q_1, B) \in \delta(q, a, A), q' \in Z\}$
 $(q_1, S, q_0) \rightarrow b(q_1, A, q_0) \quad (q_1, S, q_0) \rightarrow a(q_1, A, q_0)$
 $(q_1, S, q_1) \rightarrow b(q_1, A, q_1) \quad (q_1, S, q_1) \rightarrow a(q_1, A, q_1)$
– $\{(q, A, q') \rightarrow a(q_1, B, q_2)(q_2, C, q') \mid (q_1, BC) \in \delta(q, a, A), q', q_2 \in Z\}$
 $(q_0, \#, q_0) \rightarrow (q_1, S, q_0)(q_0, \#, q_0)$
 $(q_0, \#, q_0) \rightarrow (q_1, S, q_1)(q_1, \#, q_0)$
 $(q_0, \#, q_1) \rightarrow (q_1, S, q_0)(q_0, \#, q_1)$
 $(q_0, \#, q_1) \rightarrow (q_1, S, q_1)(q_1, \#, q_1)$
 $(q_1, S, q_0) \rightarrow a(q_1, S, q_0)(q_0, A, q_0)$
 $(q_1, S, q_0) \rightarrow a(q_1, S, q_1)(q_1, A, q_0)$
 $(q_1, S, q_1) \rightarrow a(q_1, S, q_0)(q_0, A, q_1)$
 $(q_1, S, q_1) \rightarrow a(q_1, S, q_1)(q_1, A, q_1)$
 $(q_1, S, q_0) \rightarrow b(q_1, S, q_0)(q_0, A, q_0)$
 $(q_1, S, q_0) \rightarrow b(q_1, S, q_1)(q_1, A, q_0)$
 $(q_1, S, q_1) \rightarrow b(q_1, S, q_0)(q_0, A, q_1)$
 $(q_1, S, q_1) \rightarrow b(q_1, S, q_1)(q_1, A, q_1)$

Aufgabe 24 *kontextsensitive Sprachen und LBAs* (4+4+4 = 12 Punkte)

Geben Sie für die folgenden Sprachen jeweils einen LBA an:

(a) $L_2 = \{0^i 1^i 2^i \mid i > 0\}$

(b) $L_3 = \{ww \mid w \in \{0, 1\}^+\}$

(c) $L_4 = \{0^{i^2} \mid i > 0\}$

Hinweis: Raten sie zunächst einen Wert für i .

Für L_2 muss der LBA vollständig spezifiziert werden, d.h. es wird erwartet, dass Sie zunächst das Verhalten des Automaten beschreiben und anschließend den LBA samt Zustandsübergangsrelation formal angeben. Für die Sprachen L_3 und L_4 genügt es, das Verhalten des jeweiligen LBAs informell zu beschreiben.

Aufgabe 24 **(Lösungsvorschlag)** *kontextsensitive Sprachen und LBAs*

- a) Sei $M = (\{z_{start}^0, z_{start}^1, z_{start}^2, z_0, z_1, z_2, z_L, z_e\}, \{0, 1, 2, \}, \{0, 1, 2, \square, \#\}, \delta, z_{start}^0, \square, \{z_e\})$, wobei δ wie folgt definiert ist:

(1) $(z_{start}^0, 0) \rightarrow (z_{start}^0, 0, R)$

(2) $(z_{start}^0, 1) \rightarrow (z_{start}^1, 1, R)$

(3) $(z_{start}^1, 1) \rightarrow (z_{start}^1, 1, R)$

(4) $(z_{start}^1, 2) \rightarrow (z_{start}^2, 2, R)$

(5) $(z_{start}^2, 2) \rightarrow (z_{start}^2, 2, R)$

(6) $(z_{start}^2, \square) \rightarrow (z_L, \square, L)$

(7) $(z_L, 0) \rightarrow (z_L, 0, L)$

(8) $(z_L, 1) \rightarrow (z_L, 1, L)$

(9) $(z_L, 2) \rightarrow (z_L, 2, L)$

(10) $(z_L, \#) \rightarrow (z_L, \#, L)$

(11) $(z_L, \square) \rightarrow (z_0, \square, R)$

(12) $(z_0, \#) \rightarrow (z_0, \#, R)$

(13) $(z_0, 0) \rightarrow (z_1, \#, R)$

(14) $(z_0, \square) \rightarrow (z_e, \square, N)$

(15) $(z_1, \#) \rightarrow (z_1, \#, R)$

(16) $(z_1, 0) \rightarrow (z_1, 0, R)$

(17) $(z_1, 1) \rightarrow (z_2, \#, R)$

(18) $(z_2, \#) \rightarrow (z_2, \#, R)$

(19) $(z_2, 1) \rightarrow (z_2, 1, R)$

(20) $(z_2, 2) \rightarrow (z_L, \#, L)$

Ohne Beschränkung der Allgemeinheit wird davon ausgegangen, dass der Lese-/Schreibkopf zu Beginn der Berechnung links auf dem ersten Zeichen der Eingabe positioniert ist. Zunächst wird überprüft, ob die Eingabe die erwartete Form aufweist. Dazu wird das Wort beginnend im Zustand z_{start}^0 einmal von links nach rechts durchlaufen (Regeln 1 bis 6). Anschließend wird der Lese-/Schreibkopf wieder auf den linken Rand der Eingabe positioniert (Regeln 7 bis 11) und die Maschine in den Zustand z_0 überführt. Die Maschine läuft nun nach rechts, sucht nach der ersten 0 und ersetzt sie durch # (Regeln 12 und 13). Anschließend wechselt sie in

den Zustand z_1 und läuft solange nach rechts, bis die erste 1 gefunden wird. Die Maschine ersetzt die 1 durch # und wechselt in den Zustand z_2 (Regeln 15 bis 17). Im Zustand z_2 läuft die Maschine solange nach rechts, bis die erste 2 gefunden wird. Diese wird durch # ersetzt und die Maschine wechselt in den Zustand z_L , der angibt, dass sie nun wieder bis zum linken Rand der Eingabe zurücklaufen muß (Regeln 18 bis 20). Diese Schritte werden solange wiederholt, bis die Eingabe komplett durch #-Zeichen ersetzt wurde. Dieses wird überprüft, indem die Maschine die Eingabe im Zustand z_0 von links nach rechts durchläuft und nur noch #-Zeichen liest. Am rechten Rand ankommend kann dann akzeptiert werden (Regel 14).

- b) Zuerst wird die Eingabe in ein zweidimensionales Alphabet überführt, wobei die erste Komponente die Eingabe enthält. Die zweite Komponente wird z.B. mit ε initialisiert. Gleichzeitig wird modulo 2 gezählt. Ist die Wortlänge nicht gerade, so kann die Eingabe bereits abgelehnt werden.

Als nächstes muß die Wortmitte gefunden werden. Dies kann z.B. so geschehen, dass man sich vom linken und rechten Rand der Eingabe zur Mitte vortastet. Hierzu können entsprechende Markierungen in der zweiten Komponente des Bandalphabets gesetzt werden, so dass immer die letzte linke bzw. rechte markierte Stelle gesucht werden muß. Von Links kann z.B. mit l und von Rechts mit r markiert werden. Schließlich wird (das Wort hat nach dem ersten Schritt gerade Länge) die Wortmitte durch die einzige Stelle, an der sich ein l und ein r "berühren", gekennzeichnet.

In der dritten Phase wird nun immer das am weitesten links stehende, nicht markierte Eingabesymbol aus der ersten Komponente gelesen, markiert und im aktuellen Zustand gespeichert. Dann wird zunächst die Wortmitte, und ab dort das erste noch nicht markierte Zeichen der Eingabe gesucht. Stimmt dieses nicht mit dem im Zustand kodierten Zeichen überein, so wird die Eingabe abgelehnt, ansonsten wird das Zeichen als verarbeitet markiert.

Entweder fährt man nun wieder zu dem am weitesten links stehenden, unmarkierten Symbol zurück. Oder man liest sofort das nächste Zeichen des zweiten w-Blocks und vergleicht dieses mit dem ersten noch nicht verarbeiteten Zeichen des ersten w-Blocks.

c) **Nichtdeterministischer LBA:**

Die Idee besteht darin, den erlaubten Nichtdeterminismus zum Raten des Faktors $\tilde{i} = i$ auszunutzen.

Um zu testen, ob $\tilde{i} = i$ gilt, reicht es, \tilde{i} -mal \tilde{i} Zeichen aneinander zu schreiben, und die Wortlänge des so erhaltenen Wortes mit der Eingabelänge zu vergleichen. Genau dann, wenn diese übereinstimmen, gilt $\tilde{i} = i$.

Wir brauchen somit Bänder für:

- das geratene \tilde{i}
- den Zähler k , der angibt, wie oft bereits $0^{\tilde{i}}$ hintereinandergeschrieben wurde
- das aktuelle (Zwischenergebnis) $0^{\tilde{i}k}$

In der ersten Phase wird daher die gegebene Eingabe in ein vierdimensionales Alphabet umgeschrieben, wobei die erste Komponente die Eingabe enthält und die

restlichen Komponenten ohne Einschränkung der Allgemeinheit mit ε initialisiert werden. Anschließend werden nichtdeterministisch beliebig " $\tilde{i} \leq$ Eingabelänge" viele 0-Zeichen in die zweite Komponente des Alphabets geschrieben. (Der Nichtdeterminismus erlaubt es ja, an einer beliebigen Stelle mit dem Schreiben aufzuhören).

Ab dann wird $0^{\tilde{i}^2}$ "berechnet", indem \tilde{i} mal $0^{\tilde{i}}$ an die erste (von links) leere Stelle in der vierten Komponente geschrieben wird. Zu Beginn jedes Additionsschritts wird in der zweiten Komponente, welche \tilde{i} speichert, die erste (von links) 0 durch 1 ersetzt, um zu merken, dass dieses Zeichen kopiert wurde/wird. Existiert keine 0 mehr, so wurde \tilde{i} einmal erfolgreich kopiert. Die zweite Komponente wird daher von $1^{\tilde{i}}$ wieder auf $0^{\tilde{i}}$ umgeschrieben. Dann wird in der dritten Komponente der Zähler k um eins erhöht, d.h. hinten ein Zeichen angehängt, und verglichen, ob $k = \tilde{i}$ gilt. Falls ja, wird geprüft, ob die erste (Eingabe) und die vierte ($0^{\tilde{i}^2}$) Komponente übereinstimmen. Falls ja, so wird akzeptiert. Ansonsten wird abgelehnt. In beiden Fällen terminiert die Rechnung. Wurde jedoch eine noch nicht kopierte 0 in der zweiten Komponente gefunden, so wird diese in der vierten Komponente angehängt. Wird hierbei das Band verlassen, so folgt, dass $\tilde{i} > i$ gilt, und die Berechnung kann abgebrochen werden.

Der Nichtdeterminismus stellt sicher, dass für eine korrekte Eingabe eine akzeptierende existiert. Für eine nicht korrekte Eingabe kann jedoch auf keinem Berechnungspfad ein Faktor gefunden werden.

Deterministischer LBA:

Es gilt

$$\frac{n(n+1)}{2} = \sum_{i=1}^n i$$

und daher auch:

$$n^2 = 2 \frac{n(n+1)}{2} - n = 2 \left(\sum_{i=1}^n i \right) - n = \sum_{i=1}^n 2i - 1$$

Durch Summierung der ungeraden Zahlen ist das Aufzählen aller Quadratzahlen möglich. Der LBA muß einfach solange "hochzählen", d.h. für wachsendes i immer $2i - 1$ Zeichen der Eingabe löschen, bis alle Eingabezeichen gelöscht wurden. Nur falls dann keine Zeichen mehr gelöscht werden müssten, wird akzeptiert.

Aufgabe 25 *kontextfreie Sprachen und deterministische LBAs* (6 Punkte)

Zeigen Sie, dass jede kontextfreie Sprache durch einen deterministischen LBA erkannt werden kann. Zeigen Sie das Gleiche für das Komplement einer kontextfreien Sprache.

Es reicht aus, das Verhalten des LBAs zu beschreiben und zu begründen, dass jeder der angegebenen Schritte von einem LBA ausgeführt werden kann.

Hinweis: Stellen Sie dazu folgende Vorüberlegungen an:

- Zeigen Sie, dass die Satzformen, welche bei Ableitungen aus einer in Greibach Normalform vorliegenden Grammatik entstehen, nie länger als das abzuleitende Wort sein können.

- Überlegen Sie sich, wie alle relevanten Parsebäume mittels Tiefensuche traversiert werden können, wenn die Ableitungsregeln geeignet angeordnet sind. Dabei sollen die abgeleiteten Satzformen als Knoten eines Baumes verstanden werden, dessen Kanten mit den Ableitungsregeln beschriftet sind.
- Begründen Sie, warum es für die Traversierung eines solchen Parsebaumes ausreichend, sich die angewendeten Regeln und die aktuelle Satzform zu merken.
- Beschreiben Sie nun, wie mit diesen Vorüberlegungen die ursprüngliche Behauptung gezeigt werden kann.

Aufgabe 25 (Lösungsvorschlag) *kontextfreie Sprachen und deterministische LBAs*

1. Teilschritt:

Jede Ableitungsregel einer GNF-Grammatik G erzeugt genau ein Terminal, d.h. die Anzahl der Nichtterminale einer Satzform kann nie größer sein als die Länge des abgeleiteten Wortes. Verwendet man daher wieder ein vektorwertiges Alphabet, um mehrere Bänder zu simulieren, so reicht bereits ein zusätzliches Band zum Speichern der aktuellen Satzform aus.

2. Teilschritt:

Für jedes Nichtterminal A der Grammatik G werden die Regeln angeordnet: $A \xrightarrow{r_1^A} \alpha_1, A \xrightarrow{r_2^A} \alpha_2, \dots$. Um zu Testen, ob die Eingabe z in $L(G)$ liegt, werden mittels Tiefensuche entlang der Regeln und Satzformen beginnend bei S alle möglichen Syntaxbäume überprüft. Dabei werden beginnend bei S immer auf das forderste Nichtterminal der aktuellen Satzform der Reihe nach alle möglichen Regeln angewendet. Anschließend wird das neu hinzugekommene Terminal (GNF) mit der Eingabe verglichen. Stimmen die Zeichen überein, so wird rekursiv entsprechend abgestiegen (DFS), ansonsten wird zur ursprünglichen Satzform zurückgekehrt und die nächste Regel angewendet/ausprobiert. Wurden alle Regeln für das forderste Nichtterminal erfolglos ausprobiert, so wird wiederum zu der vorhergehenden Satzform zurückgekehrt.

3. Teilschritt:

Jede Regelanwendung bindet ein Zeichen der Eingabe. Da die Regeln durch die Grammatik vorgegeben sind, kann jeder Regel eindeutig ein Bandsymbol zugeordnet werden. Dabei müssen maximal soviele Regeln gespeichert werden wie die Eingabe Zeichen hat. Weiterhin reicht es aus, sich die jeweils aktuelle Satzform zu merken, denn unter Verwendung der gespeicherten, angewendeten Regeln kann die vorherige Satzform rekonstruiert werden. Seien z.B. die zuletzt angewendete Regel $r_2^A : A \rightarrow aSS$ und die aktuelle Satzform $baSSA$ bekannt. Dann muss nur die rechte Seite aSS der Regel in der aktuellen Satzform durch A ersetzt werden, um die vorherige Satzform zu erhalten.

4. Teilschritt:

Ein DLBA kann somit deterministisch alle möglichen Syntaxbäume für eine Eingabe z "aufzählen". Hierfür muß er nur die aktuelle Satzform und einen Keller mit den angewendeten Regeln verwalten. Kann eine Satzform nicht geschrieben werden, da sie zu lang ist, so kann dies kein Syntaxbaum zu z sein, und es kann zu der vorherigen Satzform zurückgegangen werden. Hierbei wird die oberste Regel

vom Keller entfernt. Entsprechendes gilt, falls das aktuelle Terminal nicht mit z in Einklang gebracht werden kann.

Somit findet der DLBA genau dann einen Syntaxbaum, falls $z \in L(G)$. Auch der Fall $\Sigma^* \setminus L(G)$ kann entschieden werden, da der DLBA in diesem Fall keinen Syntaxbaum findet und zuletzt wieder bei S ankommt.

Allgemeiner Hinweis zu den Aufgaben 24 und 25:

Zusätzliche Bänder (und somit mehr Platz) könnten hilfreich sein, welche jedoch nach Definition eines LBAs nicht gegeben sind. Jedoch könnten sie zum Beispiel durch Verwendung eines vektorwertigen Bandalphabets "simuliert" werden. "Komprimieren" Sie daher die Eingabe, indem Sie sie zunächst mit Hilfe eines geeigneten Bandalphabets Γ kodieren. Desweiteren könnten auch zusätzliche Informationen, wie zum Beispiel das zuletzt gelesene/geschriebene Zeichen, in die Zustände eines LBAs hineinkodiert werden.