Provadis Hochschule

Theoretische Informatik
Übung 5

Dr. Florian Volk

Aufgabe 1 – Einfacher NPDA

Konstruieren Sie einen NPDA $(Q, \Sigma, \Gamma, \delta, \mathrm{q_o}, \mathrm{s_0})$ für die Sprache Σ^* über dem Alphabet $\Sigma = \{x, y, z\}$.

Aufgabe 2 – Turing-Maschine für bin+2

Gegeben ist die aus der Vorlesung bekannt Turing-Maschine, welche 1 zu einer auf dem Band gegebenen Zahl (in Binärschreibweise) addiert:

$\Sigma = \{0, 1, \#\}$, # als Zeichen für leere Bandstelle
$Q = \{z_1, z_2, z_3\}$

z_1 ist der Zustand, in welchem der Schreib-Lese-Kopf bis zum (rechten) Wortende bewegt wird.
z_2 ist der Zustand, in welchem 1 addiert wird, solange ein Übertrag vorliegt.
z_3 ist der Zustand, in welchem der Schreib-Lese-Kopf bis zum (linken) Wortanfang bewegt wird.

Turing-Tabelle für δ:

δ	#	0	1
z_1	$\#, z_2, L$	$0, z_1, R$	$1, z_1, R$
z_2	$1, z_3, L$	$1, z_3, L$	$0, z_2, L$
z_3		$0, z_3, L$	$1, z_3, L$

Konstruieren Sie nun eine Turing-Maschine, welche 2 zu einer auf dem Band gegebenen Zahl addiert.

Hausaufgabe 1 – NPDA für CF-Grammatik

Konstruieren Sie mittels Simulation der Produktionen einen NPDA, der die Sprache $\mathcal{L} = \{a^n b^n cc\}$ über dem Alphabet $\Sigma = \{a, b, c\}$ erkennt. $\mathcal{L}$ wird durch die folgende Grammatik $G = (V, T, P, S)$ beschrieben.

$V = \{S, X, C\}$
$T = \{a, b, c\}$

P:
$S \rightarrow cc \mid XC$
$C \rightarrow cc$
$X \rightarrow aXb \mid \epsilon$

(Diese Grammatik ist absichtlich unnötig kompliziert.)

Hausaufgabe 2 – Turing-Post-Programm für dez+1

Schreiben Sie ein TPP, welches 1 zu einer Zahl auf dem Band addiert. Die Zahl ist dabei in Dezimalschreibweise notiert, also über dem Alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \#\}$ – wobei $\#$ das Zeichen für eine leere Bandstelle ist.

Sie dürfen das `JUMP`-Makro nutzen. Verwenden Sie Label statt Zeilennummern.

Gehen Sie davon aus, dass beim Programmstart der Schreib-Lese-Kopf auf dem ersten (linksten) Zeichen der Zahl steht.

Lösung Aufgabe 1

Idee: Da die Sprache alle Worte aus Σ^* enthält, muss man nicht zuerst eine Grammatik konstruieren und dann durch Simulation der Produktionen einen NPDA konstruieren. Das wäre jedoch möglich.

Da jede Eingabe akzeptiert werden soll, reicht es aus, einen NPDA zu konstruieren, der das Wort vollständig liest und den Stack leert.
Ein (N)PDA akzeptiert genau dann, wenn der **Stack leer ist** und das **Wort vollständig gelesen** ist.

$Q = \{S\}$
$\Sigma = \{x, y, z\}$ wie vorgegeben
$\Gamma = \{S\}$ nur ein Symbol, da wir den Stack nicht verwenden werden
$q_0 = S$ als Startzustand, der zugleich der einzige Zustand ist
$s_0 = S$ als einziges Symbol des Stackalphabets

Weil es nur einen Zustand gibt, ignorieren wir Zustände bei der Definition der Transitionsfunktion δ.

Zuerst definieren wir Regeln, welche es erlauben, das Wort zeichenweise zu lesen. Dabei verändern wir den Stack nicht, um die Regeln einfach zu halten.

$\delta(x, S) = \{S\}$
$\delta(y, S) = \{S\}$
$\delta(z, S) = \{S\}$

Mit diesen drei Regeln ermöglichen wir es, das Wort zu durchlaufen.

Es bleibt das Symbol S auf dem Stack, welches wir nun abbauen müssen. Dazu verwenden wir folgende Regel:

$\delta(\epsilon, S) = \{\epsilon\}$

Lösung Aufgabe 2

Idee: Die gegebene Turing-Maschine für **bin+1** wird zweimal hintereinander ausgeführt. Die zweifache Addition von 1 entspricht der Addition von 2.

Dabei muss sichergestellt werden, dass die Ausführung nicht bei $\delta(z_3, \#)$ endet. Würde man hier direkt wieder in Zustand z_1 wechseln, z.B. mit $\delta(z_3, \#) = (\#, z_1, R)$, gäbe es keine Abbruchbedingung mehr: Die Maschine würde nicht mehr terminieren. Daher müssen die Zustände z_1 bis z_3 dupliziert werden. Die Duplikate übernehmen die zweite Addition und führen zur Terminierung.

Notiz: Eine generische Lösung wäre es, einen Zähler auf das Band schreiben, der angibt, in der wievielten Addition man sich befindet und damit die Abbruchbedingung steuert. Im vorliegenden Fall würde das die Auswertungslogik jedoch deutlich komplizierter gestalten als eine Verdopplung der Zustände.

Erweiterung der Zustände: $Q' = \{z_1, z_2, z_3, z_4, z_5, z_6\}$

Dabei ist z_4 ein Duplikat von z_1, z_5 eines von z_2 und z_6 ein Duplikat von z_6. Es muss lediglich der Übergang von z_3 zu z_4 (also $\delta(z_3, \#) = (\#, z_4, R)$) neu definiert werden.

δ	#	0	1
z_1	$\#, z_2, L$	$0, z_1, R$	$1, z_1, R$
z_2	$1, z_3, L$	$1, z_3, L$	$0, z_2, L$
z_3	$\#, z_4, R$	$0, z_3, L$	$1, z_3, L$
z_4	$\#, z_5, L$	$0, z_4, R$	$1, z_4, R$
z_5	$1, z_6, L$	$1, z_6, L$	$0, z_5, L$
z_6		$0, z_6, L$	$1, z_6, L$

Wie zuvor $\delta(z_3, \#)$ beendet nun $\delta(z_6, \#)$ die Ausführung nach der zweiten Addition.

Lösung Hausaufgabe 1

Der folgende NPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, s_0)$ erkennt die Sprache $\mathcal{L}$.

$$Q = \{S\}$$
$$\Sigma = T = \{a, b, c\}$$
$$\Gamma = T \cup V = \{a, b, c, S, X, C\}$$
$$q_0 = S$$
$$s_0 = S$$

Weil es nur einen Zustand gibt, ignorieren wir Zustände bei der Definition der Transitionsfunktion δ.

Regeln, welche Produktionen simulieren (wenn eine Variable auf dem Stack liegt):

$$\delta(\epsilon, S) = \{cc, XC\}$$
$$\delta(\epsilon, C) = \{cc\}$$
$$\delta(\epsilon, X) = \{aXb, \epsilon\}$$

Regeln, welche Terminale vom Stack entfernen:

$$\delta(a, a) = \{\epsilon\}$$
$$\delta(b, b) = \{\epsilon\}$$
$$\delta(c, c) = \{\epsilon\}$$

Lösung Hausaufgabe 2

Idee: Analog zum **bin+1**-Programm addiert man von rechts nach links jeweils 1 auf die aktuelle Stelle.

Zuerst bewegt man den Schreib-Lese-Kopf auf das rechteste Zeichen der Zahl.

Dort angekommen, gibt es zehn Fälle zu unterscheiden:

1. 0 wird zur 1
2. 1 wird zur 2
3. 2 wird zur 3
   …
10. 9 wird zur 0 und es entsteht ein Übertrag auf die Stelle links der aktuellen Position des Schreib-Lese-Kopfes.

In den Fällen 1.-9. endet das Programm. Im zehnten Fall muss der Übertrag berücksichtigt werden und 1 auf die jeweils links gelegene Stelle addiert werden. Dort sind erneut die zehn Fälle zu unterscheiden.

Kommt man am ersten Zeichen des Wortes an, endet das Programm.
Außer dort steht eine 9, dann kommt es erneut zum Übertrag. Dazu muss das #-Zeichen eine Stelle weiter links zu einer 1 werden. Es gibt also einen elften Fall:

11. # wird zur 1.

```
anfang:     RIGHT
            CASE # JUMP addieren      // ganz rechts angekommen
            JUMP anfang
addieren:   LEFT
            CASE # JUMP fall_#zu1
            CASE 0 JUMP fall_0zu1
            CASE 1 JUMP fall_1zu2
            CASE 2 JUMP fall_2zu3
            CASE 3 JUMP fall_3zu4
            CASE 4 JUMP fall_4zu5
            CASE 5 JUMP fall_5zu6
            CASE 6 JUMP fall_6zu7
            CASE 7 JUMP fall_7zu8
            CASE 8 JUMP fall_8zu9
            WRITE 0                   // Fall 10 (9 wird zu 0 mit Übertrag)
            JUMP addieren
fall_#zu1:  WRITE 1
            JUMP ende                 // eigentlich überflüssig
fall_0zu1:  WRITE 1
            JUMP ende
fall_1zu2:  WRITE 2
            JUMP ende
fall_2zu3:  WRITE 3
            JUMP ende
fall_3zu4:  WRITE 4
            JUMP ende
fall_4zu5:  WRITE 5
            JUMP ende
fall_5zu6:  WRITE 6
            JUMP ende
fall_6zu7:  WRITE 7
            JUMP ende
fall_7zu8:  WRITE 8
            JUMP ende
fall_8zu9:  WRITE 9
            JUMP ende                 // eigentlich überflüssig
ende:       HALT
```