WS 2017/2018 22.01.2018

Übungen zur Vorlesung
Theoretische Informatik I
Blatt 7

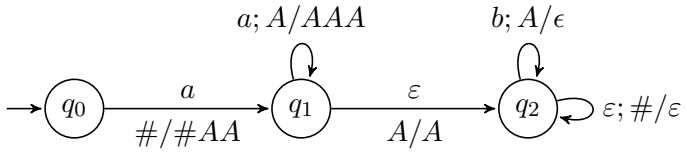
Prof. Dr. Roland Meyer,
M. Sc. Elisabeth Neumann Abgabe bis 29.01.2018 um 12 Uhr

Aufgabe 7.1 (Alternative PDA-Konstruktion)
Ein TPDA P ist ein Pushdown-Automat mit initialem Stack-Symbol #, also ein Tupel $P = (Q, \Sigma, \Gamma, q_0, \#, \delta, Q_F)$, der mit Endzuständen akzeptiert und dessen Transitionen

- immer genau ein Symbol vom Stack nehmen und
- immer höchstens zwei Elemente auf den Stack pushen.

Zeigen Sie, dass die Sprachen, die von TPDAs akzeptiert werden genau die Sprachen sind, die von PDAs akzeptiert werden.
Hinweis: Konstruieren Sie für jeden PDA (TPDA) einen TPDA (PDA), der die gleiche Sprache erzeugt.

Aufgabe 7.2 (Von PDA zu kontextfreier Grammatik)
Gegeben sei der folgenden PDA M, der mit leerem Stack akzeptiert.



Nutzen Sie das Verfahren aus der Vorlesung und konstruieren Sie eine kontextfreie Grammatik G mit $L(G) = L(M)$. Welche Sprache wird von M erzeugt?
Hinweis: Ein Nicht-Terminal der Form (q, A, q') erzeugt ein Terminalwort w genau dann, wenn M, gestartet auf q mit Input w und initialem Stacksymbol A, nach q' übergehen kann und dort leeren Stack hat.

Aufgabe 7.3 (Zwei-Stack-Pushdowns)
Ein *Zwei-Stack-Pushdown-Automat* (2PDA) ist ein Tupel $(Q, \Sigma, \Gamma, q_0, \delta, Q_F)$, mit einer endlichen Menge von Zuständen Q, einem Eingabealphabet Σ, einem Stackalphabet Γ, einem Startzustand $q_0 \in Q$, einer Menge von Endzuständen $Q_F \subseteq Q$ und einer Transitionsrelation, die es erlaubt, zwei Stacks zu manipulieren:

$$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \underbrace{((\Gamma \cup \{\varepsilon\}) \times \Gamma^*)}_{\text{Stack 1}} \times \underbrace{((\Gamma \cup \{\varepsilon\}) \times \Gamma^*)}_{\text{Stack 2}} \times Q$$

Das Ziel dieser Aufgabe ist es, zu beweisen, dass 2PDAs eine ausdrucksstärkere Sprachklasse bilden als PDAs.

a) Konstruieren Sie einen 2PDA für die Sprache $L = \{a^n.b^k.c^n.d^k \mid n, k \in \mathbb{N}\}$.

b) Beweisen Sie, dass es keinen PDA geben kann, der die Sprache L akzeptiert.

Aufgabe 7.4 (Modellierung rekursiver Programme)
Gegeben ist der folgende C-ähnliche Code mit der Funktion `int r()`, die zufällig `0` oder `1` zurückgibt.

```
0: void t()
{
1: if(r()==1) s()
2: if(r()==1) t()
3: print ''t fertig''
}
```

```
0: void s()
{
1: if(r()==1) s()
2: if(r()==1) t()
3: print ''s fertig''
}
```

```
0: void main()
{
1: t()
2: print ''main fertig''
}
```

Konstruieren Sie einen Pushdown-Automaten, der für einen gegebenen Abfolge von Befehlen prüft ob es sich um eine korrekte Ausführung der *main*-Methode handeln könnte. Eine Abfolge von Befehlen ist ein beliebiges Wort aus Σ^*, mit

$$\Sigma = \{t(), \text{if } (r() == 1)\ t(), \text{if } (r() == 1)\ s(),$$
$$\text{print "t fertig"}, \text{print "s fertig"}, \text{print "main fertig"}\}.$$

Die folgende Abfolge kann zum Beispiel einer korrekten Ausführung entsprechen:

```
t()
if (r()==1) s()
if (r()==1) s()
if (r()==1) t()
print ''s fertig''
if (r()==1) t()
print'' t fertig''
print ''main fertig''
```

Die folgende Abfolge enstpricht keiner korrekten Ausführung (Rücksprung an die falsche Stelle in Methode t):

```
t()
if (r()==1) s()
if (r()==1) s()
if (r()==1) t()
print ''s fertig''
print'' t fertig'' //Fehler
print ''main fertig''
```

Eine weitere inkorrekte Abfolge (Rücksprung in die falsche Funktion) ist:

```
t()
if (r()==1) s()
if (r()==1) t()
if (r()==1) s()
if (r()==1) t()
print ''t fertig''
print ''main fertig'' //Fehler
```

Geben Sie danach die Konfigurationsfolgen an die der PDA beim Abarbeiten der folgenden Abfolge durchläuft und zeigen Sie damit dass die Abfolge einer korrektem Ausführung von *main* entsprechen kann.

```
t()
if (r()==1) s()
if (r()==1) s()
if (r()==1) s()
if (r()==1) t()
if (r()==1) s()
if (r()==1) t()
print ''t fertig''
print'' s fertig''
if (r()==1) t()
print'' s fertig''
if (r()==1) t()
if (r()==1) s()
if (r()==1) t()
print ''t fertig''
print'' t fertig''
print ''main fertig''
```

Hinweis: Der PDA sollte den aktuellen Programmzähler speichern. Der Programmzähler gibt an, in welcher Zeile des Codes (und in welcher Funktion) sich das Program gerade befindet. Funktionenaufrufe werden im Stack verwaltet. Dabei ist immer wichtig, welche Funktion den Aufruf getätigt hat. Ist dieser Aufruf abgearbeitet, sollte das entsprechende Element vom Stack entfernt werden.

Abgabe bis 29.01.2018 um 12 Uhr im Kasten neben Raum 343.