RWTH Aachen **WS 2012/13**
Lehrgebiet Theoretische Informatik Übungsblatt 2
Reidl–Ries–Rossmanith–Sanchez–Tönnis 22.10.2012

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T3

Geben Sie die Gödelnummer $\langle M \rangle$ der unten angegebenen Turingmaschine an.

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, B, q_1, q_3, \delta)$$

δ	0	1	B
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_2, 0, L)$
q_2	$(q_2, 0, L)$	$(q_2, 1, L)$	(q_3, B, R)

Nutzen Sie dabei die Definition der Gödelnummer aus der Vorlesung.

Aufgabe T4

Wiederholen Sie kurz das Konzept der universellen Turingmaschine.
Es sei U die universelle Turingmaschine. Wie arbeitet U auf der Eingabe $\langle U \rangle \langle U \rangle \langle U \rangle \langle M \rangle w$?
Dabei sind M eine beliebige Turingmaschine und w ein beliebiges Eingabewort.

Aufgabe T5

Sind die folgenden Sprachen rekursiv? Begründen Sie Ihre Antwort.

a) $L_{100} = \{ \langle M \rangle \mid M$ hält auf der leeren Eingabe in höchstens 100 Schritten $\}$.

b) $L'_{100} = \{ \langle M \rangle \mid M$ besucht höchstens 100 Bandplätze auf der leeren Eingabe$\}$.

Aufgabe H3 (15 Punkte)

Es ist sehr nützlich über einen Turingmaschinensimulator zu verfügen, wenn man Turingmaschinen entwirft. Ziel dieser Aufgabe ist es, einen solchen Simulator zu implementieren.
Wir wollen genau solche Turingmaschinen simulieren, wie sie in der Vorlesung definiert wurden. Der Simulator soll als Eingabe einen Dateinamen einer Datei erhalten, welche eine Beschreibung der zu simulierenden Turingmaschine $M = \{Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta\}$ enthält, und ein Eingabewort $w \in \Sigma^*$.
Der Simulator soll dann M auf der Eingabe w simulieren und alle durchlaufenen Konfiguration in jeweils einer Zeile ausgeben.
Die Beschreibung einer Turingmaschine in einer Datei ist dabei folgendermaßen aufgebaut:

1. Die erste Zeile enthält die Anzahl der Zustände n, wobei wir $Q = \{q_1, q_2, \ldots, q_n\}$ vorraussetzen.

2. Die zweite Zeile enthält alle Zeichen aus Σ.

3. Die dritte Zeile enthält alle Zeichen aus Γ.
4. Die vierte Zeile enthält die Nummer i des Anfangszustands $q_i = q_0$.
5. Die fünfte Zeile enthält die Nummer j des Endzustands $q_j = \bar{q}$.
6. Die folgenden Zeilen enthalten je einen Übergang $\delta\colon (q, a) \mapsto (q', b, D)$ der Übergangsfunktion δ, wobei q, a, q', b und D in dieser Reihenfolge durch jeweils ein Leerzeichen getrennt auftreten.
7. Wir legen fest, daß das Blanksymbol `B` ist und nicht umdefiniert wird.

Folgende Vereinbarungen vereinfachen die Implementierung und Verwendung des Simulators etwas: Sie müssen nicht überprüfen, ob die Beschreibung und das Eingabewort syntaktisch und semantisch korrekt sind. Füttert man den Simulator mit falschen Daten, dann darf etwas beliebiges passieren. Der Simulator muß daher auch nicht alle Zeilen der Eingabe berücksichtigen. Es ist beispielsweise erlaubt, die Zeilen mit Σ und Γ zu ignorieren und nur die Beschreibung von δ zu verwenden. Es ist auch erlaubt, daß δ nicht vollständig angegeben wird und einige Übergänge fehlen. Das macht zum Beispiel dann Sinn, wenn Sie wissen, daß diese Übergänge sowieso nie verwendet werden. So läßt sich viel Schreibarbeit einsparen.

Sie müssen Ihr Programm nicht hocheffizient optimieren, aber es sollte auch nicht zu langsam sein. Längere Simulationen sollten mit nur kurzer Wartezeit durchgeführt werden können. Verwenden Sie eine vernünftige Programmiersprache, die halbwegs bekannt ist und deren Programme von den Tutoren verstanden werden (Java, C, C++, usw. sind alle sehr dafür geeignet. Javascript wäre wohl keine gute Idee).

Zum Schluß läßt sich das gewünschte Verhalten am besten an einem kleinen Beispiel erläutern:

```
$ head add.tm
7
01#
01#B
1
7
1 0 1 0 R
1 1 1 1 R
1 # 1 # R
1 B 2 B L
2 1 3 0 N
$ ./tm add.tm 10#1
```

```
[1]10#1
B1[1]0#1
B10[1]#1
B10#[1]1
B10#1[1]
B10#[2]1B
B10#[3]0B
B10[3]#0B
B1[4]0#0B
B1[1]1#0B
B11[1]#0B
B11#[1]0B
```

```
B11#0[1]B
B11#[2]0B
B11[2]#1B
B11#[5]1B
B11#B[5]B
B11#[5]BB
B11[5]#BB
B1[6]1BBB
B[6]11BBB
[6]B11BBB
BB[7]11BBB
$
```

Testen Sie Ihren Simulator an verschiedenen kleinen Turingmaschinen.

Erläutern Sie kurz, wie Ihr Simulator funktioniert und geben Sie Protokolle kleiner Beispielläufe und den Quelltext mit ab.

Aufgabe H4 (8 Punkte)

Entwerfen Sie eine Turingmaschine, welche zwei durch `#` getrennte, binärkodierte Zahlen entgegennimmt und als Ausgabe ihre binärkodierte Summe präsentiert.

Führen Sie mehrere Simulationen dieser Turingmaschine mithilfe des Simulators aus Aufgabe H3 an aussagekräftigen Beispieleingaben durch.

Natürlich addiert eine solche Turingmaschine relativ langsam, wenn es sich um sehr große Zahlen handelt:

```
$ time ./tm add.tm 1101101010110101l0#101001110110010 | tail -1
BB[7]11101111101000l000BBBBBBBBBBBBBBBB

real 0m0.451s
user 0m0.444s
sys 0m0.028s
$
```