Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Prof. Dr. Ernst W. Mayr
Moritz Maaß und Hanjo Täubig

Sommersemester 2005
Übungsblatt 7
03.06.2005

Informatik IV

Abgabetermin: 10.06.2005 **vor** *der Vorlesung*

Aufgabe 1 (10 Punkte)

Gibt es für kontextfreie Sprachen, die das leere Wort ϵ nicht enthalten, immer einer kontextfreie Grammatik, bei der alle Regeln die Form

$$\begin{aligned} A &\rightarrow a \\ A &\rightarrow a\alpha b \end{aligned}$$

haben, wobei $a, b \in \Sigma$, $A \in V$ und $\alpha \in V^*$ sind?

Aufgabe 2 (10 Punkte)

Zeigen Sie, dass die Sprache

$$L = \{a^n b^n \ ; \ n \in N_0\} \cup \{a^n b^{2n} \ ; \ n \in N_0\}$$

zwar kontextfrei, aber keine deterministisch kontextfreie Sprache ist.
Hinweis: Gehen Sie wie folgt vor:

- *Nehmen Sie an, dass es einen deterministischen Kellerautomaten M gibt, der L erkennt.*
- *Konstruieren Sie dann aus zwei modifizierten „Kopien“ M' und M'' von M einen deterministischen Kellerautomaten $\tilde{M}$, der die Sprache $\{a^n b^n c^n \ ; \ n \in N_0\}$ erkennt.*

Aufgabe 3 (10 Punkte)

Geben Sie für die Sprache

$$L = \{ww | w \in \Sigma^*\}$$

einen linear beschränkten Automaten (LBA) M an, der L akzeptiert.

Aufgabe 4 (10 Punkte)

Geben Sie eine Turingmaschine an, die zwei Binärzahlen addieren kann (gegeben als Startkonfiguration $(\epsilon, q_0, \{0,1\}^+ \Box \{0,1\}^+))$, von der resultierenden Zahl die Quersumme berechnet und akzeptiert, wenn diese gerade ist. Gehen Sie wie folgt vor:

1. Entwerfen Sie eine Turingmaschine, die eine Binärzahl inkrementiert (um eins erhöht).
2. Entwerfen Sie eine Turingmaschine, die eine Binärzahl dekrementiert (um eins verringert).
3. Entwerfen Sie eine Turingmaschine, die berechnet, ob die Quersumme (binär) einer Binärzahl gerade ist.
4. Kombinieren Sie Ihre Turingmaschinen geeignet.

Aufgabe 5 (10 Punkte)

In dieser Aufgabe soll ein einfacher Rechner Mithilfe einer kontextfreien Grammatik implementiert werden. Als Grammatik verwenden wir

$$G = (\{<\!zeile\!>, <\!addition\!>, <\!multiplikation\!>, <\!subausdruck\!>\}, \Sigma, P, <\!zeile\!>)$$

mit der Regelmenge

$$\begin{array}{rcl}
<\!zeile\!> & \to & <\!addition\!> \\
<\!addition\!> & \to & <\!multiplikation\!> \\
<\!addition\!> & \to & <\!addition\!> + <\!multiplikation\!> \\
<\!addition\!> & \to & <\!addition\!> - <\!multiplikation\!> \\
<\!multiplikation\!> & \to & <\!subausdruck\!> \\
<\!multiplikation\!> & \to & <\!multiplikation\!> * <\!subausdruck\!> \\
<\!multiplikation\!> & \to & <\!multiplikation\!> / <\!subausdruck\!> \\
<\!subausdruck\!> & \to & (<\!addition\!>) \\
<\!subausdruck\!> & \to & Z
\end{array}$$

1. Formen Sie die Grammatik ähnlich wie in der Vorlesung so um, dass sie nicht mehr linksrekursiv ist, so dass man eindeutige terminale Präfixe bestimmen kann. Bestimmen Sie für jede Produktion die Menge der terminalen Präfixe der Länge 1. Gehen Sie dabei davon aus, dass Z mit $0, 1, \ldots, 9$ beginnt. Ergänzen Sie anschließend das auf der Übungs-Website zu findende Java-Programm `Aufgabe7.java`, um die Grammatik top-down zu parsen.

2. Im zweiten Teil dieser Aufgabe benutzen wir die Standard-Werkzeuge *bison* und *flex*. Da die kontextfreien Sprachen unter Substitution abgeschlossen sind, geht man in der Regel so vor, dass man einen so genannten Scanner für reguläre Ausdrücke benutzt, um die Eingabe einfacher lesbar zu machen. Ein solcher Scanner ist von der Übungsseite herunterladbar. Er liefert dem Parser Tokens, die auch einen Wert haben können. Der Scanner kümmert sich um das Einlesen von Zahlen. Der Parser muss sich dann nur noch um die Struktur kümmern. In diesem Fall liefert der Scanner die Tokens `ZAHL`, `PLUS`, `MINUS`, `MULT`, `DIV`, `LKLAMMER`, `RKLAMMER` und `ZEILENENDE`, wobei `ZAHL` schon den Wert der gescannten Zahl enthält.

   Ein Gerüst für den Parsergenerator *bison* und eine Dokumentation für diesen sind auf der Übungs-Website zu finden. Bei den meisten Unix-Varianten (auch bei Linux) sind beide Werkzeuge schon installiert. Der Übersetzungsvorgang besteht aus den drei Aufrufen

   ```
   > bison -v parser7.y
   > flex -o scanner7.yy.c scanner7.flex
   > gcc -o aufgabe7 parser7.tab.c
   ```

   Ergänzen Sie das Gerüst so, dass es die durch die Grammatik G definierte Sprache $L(G)$ erkennt. Dabei steht Z für eine beliebige Zahl (die der Scanner vorverarbeitet hat). Für jede korrekt verarbeitete Zeile soll Ihr Programm den Wert des Ausdrucks ausgeben.