Übungsblatt 09

Ausgabe: 21.12. Abgabeschluss: Mittw., 11.01., 9:45 Uhr, `eClaus.informatik.uni-stuttgart.de`

Abgabe erfolgt ausschließlich elektronisch über `eClaus.informatik.uni-stuttgart.de` – versuchen Sie nach Möglichkeit die Abgabe nicht in der letzten Minute zu machen!

Von jedem Aufgabenblatt werden maximal 20 Punkte auf den Schein angerechnet.

Auf diesem Aufgabenblatt sind deutlich mehr als 20 Punkte zu erreichen. Da maximal 20 davon auf den Schein angerechnet werden, suchen Sie sich einige Aufgaben aus den Bereichen aus, bei denen Sie noch die größten Schwächen haben.

---

1. (1+1+1+1+1 Punkte, leicht–mittel) **Grammatiken und Formale Sprachen:** Beim Arbeiten mit formalen Sprachen sind die in dieser Aufgabe vorkommenden Problemstellungen oft mit dabei.

   Gegeben sei die Grammatik $G_1 = (V_1, \Sigma_1, P_1, S_1)$ mit: $V_1 = \{S_1, A, E\}$, $\Sigma_1 = \{a, b, c\}$, $P_1 = \{S_1 \to AE, A \to aA, A \to \varepsilon, E \to bEc, E \to \varepsilon\}$. Die durch diese Grammatik beschriebene Sprache lässt sich auch als Menge charakterisieren: $L(G_1) = \{a^p b^q c^q | p \geq 0, q \geq 0\}$.

   Sei nun eine weitere Grammatik $G_2 = (V_2, \Sigma_2, P_2, S_2)$ gegeben mit: $V_2 = \{S_2, B, C\}$, $\Sigma_2 = \{a, b, c\}$, $P_2 = \{S_2 \to BC, B \to \varepsilon, B \to aBb, C \to \varepsilon, C \to cC\}$.

   Eine weitere Sprache L ist wie folgt gegeben: $L = \{ab^r c^s | r \geq 0, s \geq 0\}$.

   (a) Geben Sie die Mengencharakterisierung der Sprache $L(G_2)$ an.

   (b) Geben Sie eine kontextfreie Grammatik G_0 an, so dass gilt: $L(G_0) = L(G_1) \cup L(G_2)$. Geben Sie die Mengencharakterisierung der Sprache $L(G_0)$ an.

   (c) Geben Sie eine kontextfreie Grammatik G_3 an, so dass gilt: $L = L(G_3)$.

   (d) Geben Sie eine kontextfreie Grammatik G_4 an, so dass gilt $L(G_4) = L(G_1) \cap L(G_3)$. Geben Sie die Mengencharakterisierung der Sprache $L(G_4)$ an.

   (e) Geben Sie eine kontextfreie Grammatik G_5 an, so dass gilt $L(G_5) = L(G_3) \setminus L(G_1)$. Geben Sie die Mengencharakterisierung der Sprache $L(G_5)$ an.

2. (3+5 Punkte, schwer–sehr schwer) **Eindeutigkeit von Grammatiken:** Gegeben sei die Sprache $L = \{a^i b^j a^k b^l \mid i = j \ \vee \ k = l, \ i, j, k, l \geq 0\}$, bei der entweder zu Beginn eine gleich lange Folge von a und b auftritt (und danach beliebig viele a und danach beliebig viele b) oder zu Beginn beliebig viele a und dann b gefolgt von gleichvielen a wie b.

   Eine mögliche Grammatik für diese Sprache hat die Produktionsregeln
   $P = \{S \to AB, \ S \to BA, \ A \to aAb, \ A \to \varepsilon, \ B \to aB, \ B \to C, \ C \to bC, \ C \to \varepsilon\}$.

   (a) (3 Punkte) Zeigen Sie, dass diese Grammatik mehrdeutig ist (z.B. durch zwei wesentlich verschiedene Ableitung eines Wortes). Geben Sie eine Mengencharakterisierung der Wörter an, die bei dieser Grammatik mehrdeutig sind.

   (b) (5 Punkte) Geben Sie eine eindeutige Grammatik für die Sprache L an. Beschreiben Sie Ihre Idee und begründen Sie, warum die Grammatik eindeutig ist.

3. (5 Punkte, schwer) **Virenscanner:** Wir haben gesehen, dass es Probleme gibt, die algorithmisch nicht lösbar sind. In der heutigen Zeit wäre ein universeller Virenscanner V ein sehr hilfreiches Werkzeug. Dieser soll ein Programm P als Eingabe bekommen und ausgeben, ob P das System mit einem Virus infizieren wird oder nicht.

   Zeigen Sie, dass es solch einen universellen Virenscanner V nicht geben kann.

   (Hinweis: die Eingabe für V ist das zu überprüfende Programm P – dies ist vergleichbar mit dem Ada-95-Übersetzer, auch dieser bekommt als Eingabe ein Programm (den von Ihnen geschriebenen Ada-Sourcecode) und liefert eine Ausgabe (das in Maschinensprache übersetzte Programm).)

4. (10 Punkte, leicht–schwer) **Weihnachtliches Wunschprogrammieren:** Die folgenden Aufgaben sind eine Sammlung von mehr oder weniger schwierigen Aufgaben, zu deren Lösung Sie nur die bisher gelernten Ada-Sprachelemente benötigen. Suchen Sie sich zwei Aufgaben heraus, die Sie lösen möchten.

   - (leicht–mittel) **4 gewinnt:** Programmieren Sie das Spiel „4 gewinnt“. Die Regeln finden Sie unter `http://de.wikipedia.org/wiki/4_gewinnt`. (Ein Computerspieler braucht nicht programmiert zu werden.)
   - (leicht–mittel) **Memory:** Programmieren Sie ein Memory-Spiel. Das Spielprinzip ist z.B. unter `http://de.wikipedia.org/wiki/Memory` erklärt. (Ein Computerspieler braucht nicht programmiert zu werden.)
   - (mittel) **Die gesprochene Zahl:** Wir wollen zu den Zahlen das ausgeschriebene Wort von einem Programm berechnen lassen. Z.B.: 42 → zweiundvierzig, 425 → vierhunderfünfundzwanzig, 1300 → eintausenddreihundert, 24201 → vierundzwanzigtausendzweihunderteins, 101000 → einhunderteintausend, 400016 → vierhunderttausendsechzehn.
     Schreiben Sie ein Ada 95 Programm, das eine natürliche Zahl zwischen 1 und 999999 einliest und korrekt in Worten ausgibt. Erläutern Sie zunächst das Konstruktionsprinzip (fügen Sie dies als Kommentarzeilen in Ihrem Programm ein).
   - (mittel–schwer) **Türme von Hanoi:** Dieses klassische Beispiel für rekursive Programmierung haben wir bereits in der Einführung mit AdaLogo betrachtet. Es gibt jedoch auch einen einfachen iterativen Lösungsalgorithmus, dieser lautet umgangssprachlich: *Lege zu Beginn die kleinste Scheibe einen Stapel weiter nach rechts. Wiederhole (bis man fertig ist): Lege die zweitkleinste bewegbare Scheibe auf den Stapel, auf dem nicht die kleinste Scheibe liegt, und lege dann die kleinste Scheibe einen Stapel weiter nach rechts (lag die Scheibe bereits auf dem Stapel ganz rechts, so lege sie auf den Stapel ganz links).*
     Implementieren Sie diesen Algorithmus in Ada 95.
   - **Funktionale Programmierung:** Ada 95 ist eine imperative (befehls- oder anweisungsorientierte) Programmiersprache. Im dritten Semester werden Sie auch funktionale Programmiersprachen kennenlernen. Bei diesen kann man auf Zuweisungen und Schleifen verzichten – die einzigen Sprachelemente sind die Fallunterscheidung und die Rekursion. Bei den Beispielen zur Berechnung der Fakultät und der Fibonacci-Zahlen haben wir bereits ein wenig funktional programmiert. Prinzipiell lässt sich jede berechenbare Funktion auch funktional programmieren, d.h. nur mit Fallunterscheidung und Rekursion.
     - (mittel) Versuchen Sie zunächst `for`-Schleifen und `while`-Schleifen durch Rekursion zu simulieren (die Fallunterscheidung wird dabei nur zum Rekursionsabbruch benutzt).
     - (knifflig) Programmieren Sie eine `function istPrim(n:natural) return boolean`, die testet, ob der formale Parameter `n` eine Primzahl ist. Verwenden Sie auch hier ausschließlich Fallunterscheidung und Rekursion.
     - (noch etwas schwerer) Programmieren Sie eine `function ntePrimzahl(n:natural) return natural`, die zum formalen Parameter `n` die `n`-te Primzahl berechnet. Verwenden Sie auch hier ausschließlich Fallunterscheidung und Rekursion.
   - (mittel) **Durcheinandergewürfelt:** (Anmerkung für Studierende, die noch Sprachschwierigkeiten haben: diese Aufgabe ist nur bedingt für Sie geeignet.) Sind in einem Text manche Buchstaben vertauscht, so erkennen wir dies und verstehen den Inhalt trotzdem. Dieses Vertauschen kann sehr weit gehen: Solange in jedem Wort der erste und letzte Buchstabe an seiner Position bleibt, so ist der Mensch in der Lage, die Buchstaben instinktiv in die richtige Reihenfolge zu bringen. Versuchen Sie es selbst:
     Snid in eeinm Txet mnhcae Buhsecabtn vresauhtct, so erneknen wir dies und vserhteen den Ihnlat todtrzem. Deiess Vetrahuscen kann sher weit geehn: Slgaone in jdeem Wrot der estre und lttzee Buhabstce an senier Poiotsin bieblt, so ist der Mecsnh in der Lage, die Bcsshtuabn itit nnskiv in die rcitgihe Reeinlfhoge zu bienrgn.
     Programmieren Sie daraus ein Spiel, bei dem ein Spieler einen Satz eingibt, das Programm die Wörter des Satzes nach obigen Regeln durcheinanderwürfelt und den anderen Spieler nun raten lässt, wie der Satz im Original hieß.