Dr. Frank Drewes
Studiengang Informatik
MZH 3270, Tel.: 4335
email: drewes@informatik.uni-bremen.de

21. Oktober 1998
Semesteraufgabe

Theoretische Informatik 2

Semesteraufgabe

Grundlage dieser Aufgabe ist ein von jeder Arbeitsgruppe zu entwickelnder und zu implementierender Algorithmus, der für eine gegebene kontextfreie Grammatik eine Aufzählung $w_0, w_1, w_2, \ldots$ der erzeugten Sprache liefert. Die Grammatik sollte von einer Datei eingelesen werden können; bzgl. der dabei zu verwendenden Syntax gibt es keine Vorgaben. Es reicht dabei, Grammatiken zu betrachten, deren terminale Zeichen aus dem Alphabet $\{a, \ldots, z, A, \ldots, Z, 0, \ldots, 9, ;, :, =, >, (,)\}$ stammen und deren nichtterminale Zeichen die Form <x> haben, wobei x ein Buchstabe oder eine Ziffer ist. Weitere Einschränkungen bzgl. der Form der Grammatiken sind erlaubt, wenn sie inhaltlich sinnvoll und in Hinblick auf die Mächtigkeit unwesentlich sind (wie z.B. die Einschränkung auf λ-freie Produktionen).

Nach Einlesen der Grammatik soll der Algorithmus die Wörter $w_0, w_1, w_2, \ldots$ der Aufzählung der Reihe nach ausgeben, wobei nach jedem w_i auf eine Eingabe gewartet wird: Bei Eingabe von <return> wird mit w_{i+1} fortgefahren, bei <esc> wird terminiert. Außerdem soll an dieser Stelle die Möglichkeit bestehen, ein beliebiges Wort w der Sprache einzugeben. In dem Fall soll das Programm testen, ob w tatsächlich der Sprache angehört. Wenn ja, so soll die Aufzählung bei einem Index i mit $w_i = w$ weitergehen. Ansonsten soll w zurückgewiesen werden.

Ein möglicher Lösungsweg ist am Ende grob skizziert. Dieser Algorithmus, im Folgenden *Standardlösung* genannt, kann entsprechend ausgearbeitet verwendet werden. Es soll sich aber jede Gruppe ermutigt fühlen, nach einer besseren Alternative zu suchen. In jedem Fall sollte die Implementierung sich auf das Wesentliche beschränken, mit normaler Textausgabe ohne grafisches Brimborium etc. Für die Standardlösung sollte sich z.B. ein Programm von schätzungsweise zwei, drei Seiten ergeben.

Die eigentliche Hauptaufgabe besteht in einer schriftlichen Ausarbeitung zu dem implementierten Algorithmus von nicht mehr als 20 Seiten – weniger ist mehr. Dabei geht es um eine klare, nachvollziehbare und korrekte Darstellung bzw. Argumentation. Für eine Eins kann es nicht schaden, das eine oder andere auch zu beweisen.

In der Ausarbeitung sollten insbesondere die folgenden Fragen beantwortet werden (wobei die Ausführlichkeit der eigenen Einschätzung überlassen bleibt):

- Wie arbeitet der Algorithmus?
  (Hier ist kein Programmausdruck gefragt, sondern eine von der konkreten Realisierung abstahierende Darstellung der Arbeitsweise des Algorithmus'. Auch bei Verwendung der Standardlösung sollte eine Darstellung in eigenen Worten geliefert werden.)

- Falls die erlaubten Grammatiken Einschränkungen unterliegen, die über die o.g. Einschränkung der Alphabete hinausgehen: Welches sind diese Einschränkungen, aus welchen Gründen wurden sie vorgenommen und was müsste man ändern, um ohne sie auszukommen?
- Warum ist der Algorithmus korrekt?
- Terminiert der Algorithmus bei endlichen Aufzählungen (d.h., erkennt er ein mögliches Ende der Aufzählung)? Wenn ja, wie wurde das erreicht und warum funktioniert es? Wenn nein, wie könnte man das ändern?
- Bei Verwendung einer Nicht-Standardlösung: Was sind die Vor- und Nachteile im Vergleich zu dieser?
- Wie aufwendig ist der Algorithmus?
  (Hier sollte es um den Aufwand gehen, den es verursacht, von w_i zu w_{i+1} zu gelangen.)
- Bei welcher Sorte von Grammatik ist der Algorithmus besonders aufwendig, bei welcher besonders gut? Warum?
- Ergeben sich bezüglich dieser Fragen für rechtslineare Grammatiken irgendwelche Besonderheiten? Wie sähe ein effizienterer Algorithmus für den Spezialfall regulärer Grammatiken aus?
- Ist es auf relativ einfache Weise möglich, den Algorithmus so zu verändern, dass er auch für monotone Grammatiken funktioniert? Wie würde sich dies auf die Aufwandsabschätzung auswirken?
- Kann man einen (eventuell ganz anders funktionierenden) Algorithmus wie den hier verlangten auch für Grammatiken des Typs 0 konstruieren?

Konstruiert schließlich als Beispiele Grammatiken für die folgenden drei Sprachen und dokumentiert die ersten Elemente der resultierenden Aufzählungen:

1. die Menge aller Palindrome über $\{a, b\}$,
2. das Komplement der Menge aller Palindrome über $\{a, b\}$ und
3. die Menge aller PASCALchen-Programme. (Dabei brauchen als Variablen nur X und Y zugelassen zu werden, *begin*, *end*, *while*, *do*, *succ* und *pred* können durch b, e, w, d, s und p abgekürzt werden, und statt $\neq$ kann $>$ verwendet werden.)

Standardlösung

Wenn T das terminale Alphabet ist, erzeuge intern eine Aufzählung $u_0, u_1, \ldots$ *aller* Wörter über T. Bei jedem Wort u_i teste unter Zuhilfenahme des Algorithmus' von Cocke, Younger und Kasami, ob es ein Element der Sprache ist. Wenn ja, gebe u_i als nächstes Element der Aufzählung aus, sonst fahre mit u_{i+1} fort.

Der Cocke-Younger-Kasami-Algorithmus kann dann auch verwendet werden, um ein vom Benutzer eingegebenes Wort w auf Enthaltensein in der Sprache zu testen und gegebenenfalls bei diesem Wort in der Aufzählung fortzufahren.