Aufgabe 1 (Analysator) *20 Punkte*

Gegeben sei die Grammatik $G = (\{S, A, B\}, \{\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}\}, P, S)$ mit den Produktionen

$P = \{\ S \rightarrow A,$

$A \rightarrow \mathbf{x}B\mathbf{yx} \mid \mathbf{wz}A\mathbf{x},$

$B \rightarrow \mathbf{yzz}B \mid B\mathbf{wz} \mid A\mathbf{w}\}.$

Implementieren Sie für diese Grammatik einen vorgreifenden Analysator durch rekursiven Abstieg.

Aufgabe 2 (First- und Follow-Mengen) *30 Punkte*

Sei die Grammatik $G = (N, \Sigma, P, S)$ mit

$N = \{S, T, U, V, W, X, Y, Z\}$

$\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}\}$

$P = \{\ S \rightarrow \mathbf{a}T\mathbf{b}U\mathbf{c}U \mid \mathbf{a}T\mathbf{b}U,$

$T \rightarrow \mathbf{d}V\mathbf{d} \mid \mathbf{d}V\mathbf{d}W \mid \mathbf{d}V\mathbf{d}X,$

$U \rightarrow Y \mid Z,$

$V \rightarrow \mathbf{d},$

$W \rightarrow \mathbf{e}T,$

$X \rightarrow \mathbf{f}T,$

$Y \rightarrow \mathbf{h}Z,$

$Z \rightarrow \mathbf{g} \mid \varepsilon\}$

gegeben.

(a) Geben Sie eine äquivalente LL(1)-Grammatik G' an. *10 Punkte*

(b) Berechnen Sie die FIRST- und FOLLOW-Mengen und geben Sie die resultierenden Steuermengen für die Produktionen der Grammatik G' an. *15 Punkte*

(c) Erstellen Sie die Analysetabelle zu G'. *5 Punkte*

50 Punkte

Aufgabe 3 (myPS-Teil 2)

Hinweise zu flex und bison

Auf dem letzten Übungsblatt sollte eine Grammatik für die Sprache *myPS* entwikkelt werden. Ziel dieses Teils soll es sein, diese Grammatik in entsprechende lex / yacc (bzw. flex / bison) Spezifikationen zu überführen.

flex und bison sind kostenlos unter den folgenden Web-Adressen erhältlich:

- http://flex.sourceforge.net
- http://www.gnu.org/software/bison

In vielen Linux-Distributionen sind flex und bison bereits integriert. Achten Sie bei der Installation mittels des jeweiligen Installationstools darauf, die Entwicklerversion zu installieren, damit alle benötigten Bibliotheken verfügbar sind.

Unter Windows können Sie sich eine „Unix-Umgebung“ namens *mingw* (http://www.mingw.org) schaffen, in der die beiden Tools sowie der GNU-C-Compiler vorhanden sind.

Obwohl man einen Scanner für die Quellsprache auch unabhängig vom Parser entwickeln kann, empfiehlt es sich aus technischen Gründen, gleichzeitig einen minimalen Parser zu schreiben. Der Grund ist, dass sich bei einer Parallelentwicklung des Parsers die Token automatisch aus dessen Spezifikation generieren lassen. Eine minimale Scanner/Parser Spezifikation finden Sie im Anschluß an diese Aufgabe. Angenommen, die Spezifikationen stehen in den Dateien `myps.l` bzw. `myps.y`. Dann wird zunächst der Parser-C-Code erzeugt:

```
bison -d myps.y
```

Die Option ’-d’ sorgt dafür, daß zusätzlich zur C-Datei auch eine Datei myps.tab.h erzeugt wird, welche die Tokendefinitionen enthält. Diese wird in der Scannerspezifikation in einer `include` Direktive angegeben. Die Ergebnisse dieses Aufrufs sind die Dateien `myps.tab.c` sowie `myps.tab.h`. Nun läßt sich der Scanner mittels ’`flex myps.l`’ erzeugen. Die folgenden 3 Befehle erzeugen aus dem so generierten C-Code ein ausführbares Programm:

```
g++ -c -o parse.o myps.tab.c
g++ -c- -o scan.o lex.yy.c
g++ -o myps scan.o parse.o -lfl
```

Aufgabenbeschreibung

Erzeugen Sie aus der Grammatik des letzten Übungsblattes flex- und bison Spezifikationen, um syntaktisch korrekte *myps*-Programme zu erkennen. Sie können dabei wahlweise die von Ihnen erstellte Grammatik oder die Grammatik aus dem

Lösungsvorschlag der vorherigen Kurseinheit verwenden. Ist die Eingabe des Compilers ein syntaktisch korrektes myps-Programm, soll „akzeptiert“, anderenfalls eine geeignete Fehlermeldung ausgegeben werden. Damit ist das Ergebnis dieser Aufgabe ein Rahmen, der noch keinerlei Übersetzungsregeln enthält.

Hinweis Korrekturen

Bitte haben Sie Verständnis dafür, daß wir unseren Korrektoren nicht zumuten können, die vielen möglichen Lösungen auf Herz und Nieren zu prüfen. Stattdessen finden Sie in der Virtuellen Universität eine Reihe von Testdateien, mit denen Sie Ihren Compiler selbständig testen können. Sie sollten Ihre Lösung daher nicht zur Korrektur einsenden. Damit wir trotzdem wissen, wie Ihre Lösungen aussehen (und wieviele von Ihnen Spaß an dieser Aufgabe haben), bitten wir Sie, Ihre Lösungen direkt per E-Mail an die Kursbetreuer zu senden. Die Korrekturen zu den übrigen Aufgaben bleiben von dieser Regelung natürlich unberührt.

Beispielcode

Einen (fast) minimalen Rahmen einer flex/bison-Spezifikation bieten die folgenden Dateien `minimal.l` und `minimal.y`.

Die Datei `minimal.l` hat folgenden Inhalt:

```
%{
#include "minimal.tab.h"
#include <iostream>
#include <string>
using namespace std;
%}

letter    [a-zA-Z]
digit     [0-9]
int       [+-]?(([1-9][0-9]*)|0)
id        {letter}({letter}|{digit})*

%option yylineno
%option noyywrap

%%
[ \a\b\f\t\v\n\r]+            {}
{int}                         { return (ZZINT); }
{id}                          { return (ZZID); }
.                             { return ZZERROR; }
%%
```

Die Datei `minimal.y` sieht wie folgt aus:

```
%{
#include <iostream>
#include <string>
using namespace std;

extern int yylex();

void yyerror( char* s ) {
cerr << endl << s << endl << endl;
}
%}
%token ZZINT ZZID  ZZERROR
%%
simpletest  : simpletest ZZID { cout << " ID "; }
            | simpletest ZZINT {cout << " INT ";}
            | {}
            ;
%%
main(int argc, char* argv[]) {
  return yyparse();
}
```