Aufgabe 1 (LL(k) Grammatiken) *25 Punkte*

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

$N = \{S, A\}$,
$\Sigma = \{$ **(**, **a**, **b**, **)** $\}$,
$P = \{$
$S \rightarrow A$**b**,
$A \rightarrow$ **(b**A**b)**,
$A \rightarrow$ **(**A**b)**,
$A \rightarrow$ **a**
$\}$.

(a) Geben Sie zu G äquivalente LL(1)-Grammatik an. *8 Punkte*

(b) Berechnen Sie die FIRST- und FOLLOW-Mengen und geben Sie die resultierende Steuermengen für die Produktionen der Grammatik G' an. *10 Punkte*

(c) Geben Sie die Analysetabelle zu G an. *7 Punkte*

Aufgabe 2 (Analysator) *25 Punkte*

Sei die Grammatik $G = (\{S, A, B\}, \{$**x**, **y**, **z** $\}, P, S)$ mit den Produktionen

$P = \{ S \rightarrow A$,
$A \rightarrow$ **x**B**x**|**y**,
$B \rightarrow B$**z**$A|A$ $\}$

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

Aufgabe 3 (myPS – Teil 1)

Mit dieser Aufgabe wollen wir ein größeres Projekt vorstellen, das in den folgenden Übungen fortgesetzt wird. Wir hoffen, Ihnen ein interessantes Übersetzungsproblem anbieten zu können, das für Sie zudem einen hohen programmierpraktischen Lehr- bzw. Erfahrungswert hat. Auf diese Weise können Sie sich vertieftes Wissen über die Scanner- und Übersetzungsgeneratoren Lex und Yacc aneignen und Ihren eigenen Compiler implementieren.

Ziel des Projektes ist es, eine simple Bildbeschreibungssprache - wir nennen sie *myPS* - zu implementieren, deren Programme in eine Postscript-Datei übersetzt werden. Postscript ist eine Sprache, die von verschiedenen Druckern und Grafikprogrammen unterstützt wird.

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 Moodle Umgebung 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.

In dieser ersten Aufgabe wollen wir zunächst sehr detailliert die Quellsprache vorstellen. Auch wenn nicht gleich alle Eigenschaften bekannt sein müssten, um die Aufgabe zu lösen, so spricht dennoch vieles dafür, die Sprache zunächst einmal im Ganzen zu betrachten

Ein *myPS*-Programm hat die Struktur

```
picture <name>
        <Deklarationen>
start
        <Kommandos>
end
```

und kennt die folgenden Typen

Int	Positive bzw. negative ganze Zahlen.
Num	Ein Fließkomma- oder Integer-Wert.
String	Eine Zeichenkette, die am Anfang und Ende durch doppelte Gänsefüßchen oben (“) begrenzt wird.
Point	Ein Paar *(x, y)* aus Num-Werten
Path	Stellt einen beliebigen Pfad dar. Zur Erstellung von Pfaden stehen mehrere Operationen zur Verfügung.
Term	Eine beliebige Folge von Operationen. Darin Können auch Variablen vorkommen.

Variablen müssen zunächst deklariert werden, bevor ihnen ein Wert zugewiesen werden kann. Variablen können nicht - wie beispielsweise in C++ oder Java – an beliebigen Stellen deklariert werden. Dies ist ausschließlich im Deklarationsteil möglich, z.B.:

```
picture “example“
var kreis : Term;
var dreieck : Path;
var p : Point;
start
p := (10,10);
kreis <- { setcolor(1,0,0); fill(arc(p, 50, 0, 360)); };
dreieck := <<p, (10,100), (100,10), p>>;
end
```

Variablen können bei einer Zuweisung direkt (:=) bzw. später (<-) an Werte gebunden werden. Eine Besonderheit stellen Variablen vom Typ Term (Folgen von Anweisungen) dar, denn diese können ausschließlich mit später Bindung definiert werden (Makros).

Das Zuweisungssymbol „<-“ definiert Variablen mit einer späten Bindung, d. h. Variablen, die auf der rechten Seite der Zuweisung stehen, werden erst bei einer späteren Benutzung der Variablen ausgewertet und nicht bereits bei deren Definition. Sie können auf der rechten Seite also andere Variablen enthalten, die noch nicht definiert sind. Umgekehrt folgt daraus, dass auf der rechten Seite nicht die gerade

definierte Variable selbst wieder verwendet werden kann (z.B. `x <- x + 1`), da dies zu einer unendlichen Ersetzung führen würde. Eine Beispielauswertung der verschiedenen Zuweisungen wird weiter unten anhand eines größeren Programms erläutert. Terme, die aus mehr als einer Anweisung bestehen, müssen in einem Block, der durch geschweifte Klammern eingefasst wird, stehen. Gleiches gilt für die nachfolgend aufgeführten Zeichenoptionen:

`setcolor(`*r,g,b*`)`	Definiert die aktuelle Zeichenfarbe. Parameter: Ein Tripel (*r,g,b*) mit `Num`-Werten zwischen 0-1, welches das Mischungsverhältnis der Grundfarben Rot, Grün und Blau darstellt.
`setdrawstyle(`*s,e*`)`	Definiert den aktuellen Linienstil, mit dem Punkte eines Pfades verbunden werden. Die Parameter *s*, *e* sind `Int`-Werte, welche den Wechsel zwischen gezeichneten und nicht gezeichneten Punkten einer Linie angeben. Bsp: *(s,e) = (1,1)* zeichnet eine gestrichelte Linie, in der jeweils aufeinanderfolgend ein Teilstrich in der Länge einer Einheit gezeichnet wird und danach eine Einheit frei bleibt.
`setfont(`*font,s*`)`	Definiert den aktuellen Schrifttyp gegeben durch den `String` *font* und die Größe *s*.
`setlinewidth(`*w*`)`	Ändert die Linienstärke auf den in den Klammern angegebenen Wert.

Das wichtigste Zeichenelement von *myPS* ist ein Pfad. Es gibt eine Reihe von Operationen, um Pfade mit bestimmten Eigenschaften zu erzeugen. Weiterhin stehen Operationen zur Verfügung, um Pfade miteinander zu verbinden. Die Pfadoperationen sind:

`<<`p_1`,`p_2`,…>>`	Diese Anweisung definiert einen Linienzug durch die angegebenen Punkte. Jeder Punkt ist entweder als konstanter Punkt als '(*x,y*)' angegeben oder stellt einen Ausdruck vom Typ `Point` dar.
`arc(`*p,r,alpha,beta*`)`	Definiert einen Bogenabschnitt des Kreises mit Radius *r* um den Punkt *p* zwischen den Winkeln *alpha* und *beta*.
`ellipse(`*p,*r_1*,*r_2*,alpha,beta*`)`	Wie oben, aber für eine Ellipse mit Mittelpunkt *p* und den Radien r_1 und r_2.
`plot(`*x,y,n,min,max,(v,f(v))*`)`	Diese Anweisung erzeugt den Pfad der Kurve der Funktion *f*. Die Funktion selbst ist ein mathematischer Ausdruck, der die Variable *v* enthält. Für *v* werden *n* gleichverteilte Werte zwischen *min* und *max* eingesetzt und die daraus entstehenden Interpolationspunkte mit einfachen Linien verbunden. Der so entstandene Pfad wird noch um *x* Einheiten in X-Richtung und *y* Einheiten in Y-Richtung verschoben.
`string2path(`*p,s*`)`	Die Eckpunkte jedes Zeichens aus dem String *s* definieren einen geschlossenen Pfad. Das

	Ergebnis ist ein Pfad, der die Vereinigung aller Buchstabenpfade darstellt und im Punkt p beginnt.
`concat(`*p*`,`*q*`)`	Ergibt die Vereinigung der beiden Pfade *p* und *q*, wobei der Endpunkt von *q* mit dem Anfangspunkt von *p* verbunden wird.
`union(`*p*`,`*q*`)`	Vereinigt die Pfade *p* und *q*, ohne die Anfangs- und Endpunkte miteinander zu verbinden.
`scaletobox(`x_w`,`y_w`,`*p*`)`	Strecke (bzw. stauche) den Pfad *p* derart, dass er in ein Rechteck mit den Kantenlängen x_w und y_w passt.

Für die Ausgabe von Pfaddefinitionen stehen die folgenden Operationen zur Verfügung:

`draw(`*p*`)`	Zeichne die Kanten des Pfades p mit dem aktuellen Linienstil.
`fill(`*p*`)`	Schließe die Wege in Pfad *p* und fülle die Flächen mit der aktuellen Farbe aus.

Zum Erzeugen und Zeichnen von Strings gibt es die unten aufgeführten Operationen:

`num2string(`*n*`)`	Wandelt einen numerischen Wert *n* in einen `String` um.
`write(`*p*`,`*s*`)`	Zeichnet den `String` *s* mit den aktuellen Schrifteinstellungen mit Startpunkt *p*.
`write(`*s*`)`	Zeichnet den `String` *s* mit den aktuellen Schrifteinstellungen am gerade aktuellen Punkt (i.a. der Endpunkt des letzten definierten Pfades).

Weiterhin gibt es noch Operationen, die das Ergebnis von Termen manipulieren:

`rotate(`*alpha*`,`*t*`)`	Drehe das Resultat von `Term` *t* um *alpha* Grad im Uhrzeigersinn. Die Rotationsachse liegt im Nullpunkt des Koordinatensystems.
`scale(`x_s`,`y_s`,`*t*`)`	Strecke bzw. stauche die Anweisungen aus `Term` *t* in x-Richtung um Faktor x_s und in y-Richtung um Faktor y_s.
`translate(`x_t`,`y_t`,`*t*`)`	Verschiebe das Ergebnis von `Term` *t* in x-Richtung um x_t und y-Richtung um y_t Einheiten.
`clip(`*p*`,`*t*`)`	Zeichne nur den Teil des Terms *t* der innerhalb der Fläche, die durch den Pfad *p* definiert wird, liegt.

Für alle genannten Operationen gilt, dass die Argumente beliebige Wertausdrücke oder Variablen des entsprechenden Typs darstellen können.

Arithmetische Ausdrücke sind recht komplex. Sie bestehen aus Konstanten, numerischen Variablen, Klammern und den folgenden Operationen bzw. Funktionen, die den üblichen mathematischen Prioritäten gehorchen:

`+,-,*,/` : die üblichen Grundrechenoperatoren
`mod`: modulo Funktion

`sin, cos`: Winkelfunktionen
`ln`: natürlicher Logarithmus zur Basis *e*
`exp(b,e)`: berechnet den Wert b^e
`abs`: Absolutbetrag
`random(min,max)`: eine Zufallszahl aus dem angegebenen Zahlenbereich

Schließlich gibt es noch die folgenden Regelungen:

1. Ein Prozentzeichen leitet einen Programmkommentar ein, der mit dem Zeilenende abschließt.
2. Jede Anweisung wird mit einem Semikolon abgeschlossen
3. Variablen dürfen nur aus alphanumerischen Zeichen bestehen und das erste Zeichen muss ein Buchstabe sein.
4. Als (vorerst) einzige Kontrollstruktur bietet die Sprache einen Schleifendurchlauf an:
   ```
   for x := 10 to 100 step 5 do
       arc((0,0), x, 0, 45);
   done;
   ```
   Die Schleifenvariable x wird dadurch implizit definiert. Innerhalb der Schleife können keine Zuweisungen an die Schleifenvariable durchgeführt werden, sie kann lediglich ausgewertet werden. Eine Verschachtelung von Schleifen soll selbstverständlich möglich sein.

Aufgabenstellung: Legen Sie eine kontextfreie Grammatik für die Sprache *myPS* fest. Denken Sie daran, dass die Grammatik lediglich die Syntax der Sprache beschreibt. Typprüfungen usw. werden erst in einer der späteren Kurseinheiten im Projekt realisiert.

Zum Schluss folgt nun noch ein Beispielprogramm:

```
picture “myPS-Beispiel“
% Deklaration einiger Variablen
var x : Num;
var y : Num;
var z : Num;
var p : Path;
var black : Term;

start % Beginne mit Zeichenanweisungen


    x := 3;
    y <- x; % Variable y besitzt späte Bindung!
    z := x;
    x := 5;

    black <- { setcolor(0,0,0); };
    setfont(“Times“,20); % Schriftart Times, 20pt

    setcolor(1,0,0);
    translate( 150, 400,
        scale( 10, 10,
```

```
            rotate(45,
                draw( string2path((0,0),”Myps”))
            )
        )
    );

    black; % Setze die Farbe auf Schwarz
    write( (100, 700), “Spaete Bindung: y =“);
    write( num2string(y));   % y hat den Wert 5
    write( (100, 680), “Direkte Bindung: z =“);
    write( num2string(z));   % z hat den Wert 3

    % Zeichne einige Kreisbögen in unterschiedlichen Farben
    for i := 0 to 340 step 20 do
        for r := 20 to 100 step 5 do
            p := arc((300,300),r*i*0.01,i+5,360);
            setcolor(i/340,r/100,1);
            draw(p);
        done;
    done;

    setlinewidth(10);
    black;

    % Berechne 200 Punkte einer Sinuskurve und speichere den
    % resultierenden Pfad in Variable p.
    p := plot( 0, 0, 200, 0, 3*360, (t, sin(t)) );

    % Zeichne p in eine Bounding-Box der Größe 200x400 und verschiebe
    % diese in den Punkt (100, 100).
    translate( 100,100, { draw( scaletobox(400,200,9)); } );


end
```

Das Programm definiert das folgende Bild: