Übungsblatt 1

(10. Januar 2008)

Aufgabe 1 Gegeben sei die Grammatik $\mathcal{G} = (\mathcal{T}, \mathcal{N}, S, \mathcal{R})$ mit:

- $\mathcal{T}$= {`ident`, `(`, `)`, `+`, `-`, `[`, `]`, `:=`, `eq`, `neq`}
- $\mathcal{N}$= {S,R,A,P,L}
- S =S
- R={

$$
\begin{array}{lcll}
S & \rightarrow & R \mid L \texttt{ := } R & (1)\\
R & \rightarrow & A \mid A \texttt{ eq } A \mid A \texttt{ neq } A & (2)\\
A & \rightarrow & P \mid P \texttt{ + } P \mid P \texttt{ - } P & (3)\\
P & \rightarrow & \texttt{ident} \mid \texttt{(} R \texttt{)} \mid P \texttt{[} A \texttt{]} & (4)\\
L & \rightarrow & \texttt{ident} \mid \texttt{(} L \texttt{)} \mid L \texttt{[} A \texttt{]} & (5)
\end{array}
$$

}

a) Ist die Grammatik LL(1)? Woran kann dies sofort erkannt werden?

b) Berechnen Sie für die fünf Nichtterminalsymbole die First-Mengen.

c) Warum spielen FOLLOW Mengen für diese Art der Grammatik überhaupt keine Rolle, auch wenn man mit Hilfe von FIRST und FOLLOW Mengen die LL(1) Eigenschaft überprüfen will.

d) Eleminieren Sie die Linksrekursion für die Regel für das Symbol P.

e) Linksfaktorisieren Sie die Regel für das Symbol R.

f) Nach Elimination der Linksrekursionen und Linksfaktorisierung der Regeln für R und A ist die Grammtik nicht LL(1). Geben Sie ein Beispiel an, um zu zeigen, daß durch anschauen allein des ersten Symbols nicht entschieden werden kann, mit welcher Regelalternative eindeutig geparst werden muß (ohne ein Backtracking durchführen zu müssen.)

g) Zeigen Sie schrittweise, wie ein Shift-Reduce-Parser den folgenden Satz parsen würde: `ident := ident[ident]`

Aufgabe 2 Gegeben seien folgende Klassen:

Node.java

```
class Node{
  public void welcome(NodeVisitor nv){
    throw new
      UnsupportedOperationException(this+" unknown visitor case");
  }
}
```

NodeVisitor.java

```
interface NodeVisitor{
    void visit(IntLit n);
    void visit(Var n);
    void visit(Assign n);
    void visit(Add n);
    void visit(Seq n);
}
```

IntLit.java

```
class IntLit extends Node{
    int i;
    IntLit(int i){this.i=i;}
    public void welcome(NodeVisitor nv){
        nv.visit(this);
    }
}
```

Var.java

```
class Var extends Node{
    String name;
    Var(String name){this.name=name;}
    public void welcome(NodeVisitor nv){
        nv.visit(this);
    }
}
```

Assign.java

```
class Assign extends Node{
    Var v;
    Node n;
    Assign(Var v,Node n){this.v=v;this.n=n;}
    public void welcome(NodeVisitor nv){
        nv.visit(this);
    }
}
```

Add.java

```
1 class Add extends Node{
2     Node l;
3     Node r;
4     Add(Node l,Node r){this.l=l;this.r=r;}
5     public void welcome(NodeVisitor nv){
6         nv.visit(this);
7     }
8 }
```

Seq.java

```
1 import java.util;
2 class Seq extends Node{
3     List<Node> xs;
4     Seq(List<Node> xs){this.xs=xs;}
5     public void welcome(NodeVisitor nv){
6         nv.visit(this);
7     }
8 }
```

Schreiben Sie einen Visitor, der das durch den Baum dargestellte Programm berechnet.

Aufgabe 3 Gegeben sei die Grammatik $\mathcal{G} = (\mathcal{T}, \mathcal{N}, S, \mathcal{R})$ mit:

- $\mathcal{T}$= {`ident, (, ), bar`}
- $\mathcal{N}$= {A,B}
- S =A
- R={

$$A \rightarrow \texttt{ident}\ B \mid \texttt{(}\ A\ \texttt{)}\ B \tag{6}$$
$$B \rightarrow \texttt{bar}\ A \mid A \mid \epsilon \tag{7}$$

}

a) Leiten Sie den Satz:
`ident ident bar ident`
mit der Grammatik ab.

b) Zeichnen Sie für Ihre Ableitung den Syntaxbaum.

c) Zeigen Sie anhand der First und Follow Mengen, ob die Grammatik LL(1) ist.

Aufgabe 4 Geben Sie eine Grammatik an, die sie Menge aller Palllindrome über die Menge $\{0, 1\}$ erzeugt.

Ein Pallindrom ist ein Satz, der vorwärts wie rückwärts identisch ist. Die Wörter `stets`, `rentner` oder `köblbök` sind z.B. Pallindrome.

Aufgabe 5 Gegen sei folgendes Programm in C-Syntax:

```
1 int wenn(bool b,int a1,int a2){
2   return b?a1:a2;
3 }
4
5 int square(int x){return x*x;}
```

Reduzieren Sie den Ausdruck `wenn(square(5)==25,square(2*2),square(17+4)` einmal mit der normalen und einmal mit der applikativen Auswertungsreihenfolge.