

Übersetzerbau: Übung 03

von

Naja v. Schmude (4127652), Lisa Dohrmann (4130066)

Aufgabe 1

- a) Es ist ein Recursive-Descent-Parser zu der Grammatik $S \rightarrow 0S1 \mid 01$ gesucht.

```
void S() {
    if(lookahead == '0') {
        match('0');
        S();
        match('1');
    }
}
```

- b) Es ist ein entsprechender Parser für die Grammatik $S \rightarrow +SS \mid -SS \mid a$ zu entwerfen.

```
void S() {
    switch( lookahead ) {
        case '+':
            match('+');
            S();
            S();
            break;
        case '-':
            match('-');
            S();
            S();
            break;
        case default:
            match('a');
    }
}
```

- c) Nun ist ein Parser für $S \rightarrow S(S)S \mid \epsilon$ gesucht. Für diese Grammatik gibt es keinen Parser, da sie linksrekursiv ist und der Parser dadurch nicht wissen kann, wie tief der Baum wird. Die Linksrekursion müssen wir daher zunächst eliminieren. Das geht bei der gegebenen Grammatik aber gar nicht. Sie hat nämlich nicht die Form $A \rightarrow A\alpha \mid \beta$, sodass man sie zu $A \rightarrow \beta R, R \rightarrow \alpha R \mid \epsilon$ umwandeln könnte.

Wir können uns aber überlegen, dass die Grammatik

$$G' = S \rightarrow (S)S \mid \epsilon$$

äquivalent zu unserer gegebenen Grammatik ist. Für diese können wir direkt den Parser-Code erzeugen:

```

void S() {
    if (lookahead == '(')
        match('('); S(); match(')'); S();
}

```

Aufgabe 2

Wir benutzen eine einfache Stackarchitektur und legen jede gelesene öffnende Klammer auf dem Stack und nehmen sie wieder runter, sobald eine schließende Klammer gelesen wird. Ist der Stack am Ende leer, haben wir einen korrekten Klammersausdruck gelesen.

Produktion	semantische Regel
$S \rightarrow S_1(S_2)S_3$	$S.k = S_1.k \parallel \text{Push } '(' \parallel S_2.k \parallel \text{Pop } '(' \parallel S_3.k$
$S \rightarrow \epsilon$	

Aufgabe 3

- a) Hinweis: Terminalsymbole sind unterstrichen.

$$\begin{aligned}
 S &\rightarrow T \\
 T &\rightarrow \underline{MT} \mid H \\
 H &\rightarrow \underline{CZ} \mid \underline{CCZ} \mid \underline{CCCZ} \mid \underline{CDZ} \mid \underline{DZ} \mid \underline{DCZ} \mid \underline{DCCZ} \mid \underline{DCCCZ} \mid \underline{CMZ} \mid Z \\
 Z &\rightarrow \underline{Xd} \mid \underline{XXd} \mid \underline{XXXd} \mid \underline{XLd} \mid \underline{Ld} \mid \underline{LXd} \mid \underline{LXXd} \mid \underline{LXXXd} \mid \underline{XCd} \mid d \\
 d &\rightarrow \underline{I} \mid \underline{II} \mid \underline{III} \mid \underline{IV} \mid \underline{V} \mid \underline{VI} \mid \underline{VII} \mid \underline{VIII} \mid \underline{IX} \mid \epsilon
 \end{aligned}$$

- b) Ja, es kann für die Grammatik ein prädiktiver Parser erstellt werden, da über das erste Terminalsymbol eindeutig die anzuwendende Regel bestimmt ist.

Aufgabe 4

- a) Es soll ein Übersetzungsschema für ganze Zahlen in römische Zahlen erstellt werden.

$$\begin{aligned}
 t &\rightarrow 1\{\text{print}('M')\}h \mid 2\{\text{print}('MM')\}h \mid 3\{\text{print}('MMM')\}h \\
 &\mid 4\{\text{print}('MMMM')\}h \mid 5\{\text{print}('MMMMM')\}h \mid 6\{\text{print}('MMMMMM')\}h \\
 &\mid 7\{\text{print}('MMMMMMM')\}h \mid 8\{\text{print}('MMMMMMMM')\}h \\
 &\mid 9\{\text{print}('MMMMMMMMM')\}h \mid h \\
 h &\rightarrow 1\{\text{print}('C')\}z \mid 2\{\text{print}('CC')\}z \mid 3\{\text{print}('CCC')\}z \mid 4\{\text{print}('CD')\}z \\
 &\mid 5\{\text{print}('D')\}z \mid 6\{\text{print}('DC')\}z \mid 7\{\text{print}('DCC')\}z \\
 &\mid 8\{\text{print}('DCCC')\} \mid 9\{\text{print}('CM')\}z \mid 0z \mid z \\
 z &\rightarrow 1\{\text{print}('X')\}d \mid 2\{\text{print}('XX')\}d \mid 3\{\text{print}('XXX')\}d \mid 4\{\text{print}('XL')\}d \\
 &\mid 5\{\text{print}('L')\}d \mid 6\{\text{print}('LX')\}d \mid 7\{\text{print}('LXX')\}d \\
 &\mid 8\{\text{print}('LXXX')\}d \mid 9\{\text{print}('XC')\}d \mid 0z \mid d \\
 d &\rightarrow 1\{\text{print}('I')\} \mid 2\{\text{print}('II')\} \mid 3\{\text{print}('III')\} \mid 4\{\text{print}('IV')\} \\
 &\mid 5\{\text{print}('V')\} \mid 6\{\text{print}('VI')\} \mid 7\{\text{print}('VII')\} \mid 8\{\text{print}('VIII')\} \\
 &\mid 9\{\text{print}('IX')\} \mid 0
 \end{aligned}$$

- b) Es soll ein Übersetzungsschema von römischen Zahlen in ganze Zahlen erstellt werden.

$$\begin{aligned}
T &\rightarrow \underline{M}\{\text{print}('1')\}H \mid \underline{MM}\{\text{print}('2')\}H \mid \underline{MMM}\{\text{print}('3')\}H \\
&\mid \underline{MMMM}\{\text{print}('4')\}H \mid \underline{MMMMM}\{\text{print}('5')\}H \mid \underline{MMMMMM}\{\text{print}('6')\}H \\
&\mid \underline{MMMMMMM}\{\text{print}('7')\}H \mid \underline{MMMMMMMM}\{\text{print}('8')\}H \\
&\mid \underline{MMMMMMMMM}\{\text{print}('9')\}H \mid H \\
H &\rightarrow \underline{C}\{\text{print}('1')\}Z \mid \underline{CC}\{\text{print}('2')\}Z \mid \underline{CCC}\{\text{print}('3')\}Z \mid \underline{CD}\{\text{print}('4')\}Z \\
&\mid \underline{D}\{\text{print}('5')\}Z \mid \underline{DC}\{\text{print}('6')\}Z \mid \underline{DCC}\{\text{print}('7')\}Z \mid \underline{DCCC}\{\text{print}('8')\}Z \\
&\mid \underline{CM}\{\text{print}('9')\}Z \mid \{\text{print}('0')\}Z \\
Z &\rightarrow \underline{X}\{\text{print}('1')\}d \mid \underline{XX}\{\text{print}('2')\}d \mid \underline{XXX}\{\text{print}('3')\}d \mid \underline{XL}\{\text{print}('4')\}d \\
&\mid \underline{L}\{\text{print}('5')\}d \mid \underline{LX}\{\text{print}('6')\}d \mid \underline{LXX}\{\text{print}('7')\}d \mid \underline{LXXX}\{\text{print}('8')\}d \\
&\mid \underline{XC}\{\text{print}('9')\}d \mid \{\text{print}('0')\}d \\
d &\rightarrow \underline{I}\{\text{print}('1')\} \mid \underline{II}\{\text{print}('2')\} \mid \underline{III}\{\text{print}('3')\} \mid \underline{IV}\{\text{print}('4')\} \\
&\mid \underline{V}\{\text{print}('5')\} \mid \underline{VI}\{\text{print}('6')\} \mid \underline{VII}\{\text{print}('7')\} \mid \underline{VIII}\{\text{print}('8')\} \\
&\mid \underline{IX}\{\text{print}('9')\} \mid \{\text{print}('0')\}
\end{aligned}$$