Compilerbau
Übungsblatt 1

Yangzi Zhang, Michael Gottschalk, Felix J. Oppermann

28. April 2006

Aufgabe 1 (1+1+1+2 Punkte)

a) **Lemma 1**
Zu zeigen:
Zu jeder negierten Zeichenklasse $Z = [\hat{}s]$ gibt es eine nicht negierte Zeichenklasse $Z' = [s']$ welche die gleichen Zeichen repräsentiert.

Seien $x_1, \dots x_n \in \Sigma$ alle Zeichen aus s. Dann gibt es $y_1, \dots y_n \in \Sigma$ für die gilt:
$(y_1, \dots y_n) \cup (x_1, \dots x_n) = \Sigma$
und $(y_1, \dots y_n) \cap (x_1, \dots x_n) = \varnothing$ (da Σ endlich ist)

Dann gilt:
$[\hat{}s] \Leftrightarrow [\hat{}x_1 \dots x_n]$
$\Leftrightarrow [y_1 \dots y_n]$
$\Leftrightarrow [s']$

Es gibt also eine äquvalente Zeichenklasse ohne Negierung.

□

Zu zeigen:
Ein regulärer Ausdruck der negierte Zeichenklassen enthält lässt sich in einen äquivalenten Ausdruck umformen der keine negierten Zeichenklassen enthält.

Sei R ein regulären Ausdruck mit negierten Zeichenklassen.
Für jede negierte Zeichenklasse lässt sich nach Lemma 1 eine äquivalente nicht negierte Zeichenklasse finden. Wir können also R durchlaufen und jedes Vorkommen einer negierten Zeichenklasse durch eine äquivalente nicht negierte Zeichenklasse ersetzen. Es ergibt sich ein regulärer Ausdruck R' der keine negierten Zeichenklassen enthält und für den gilt $R \equiv R'$

□

b) **Zu zeigen:**
Der Ausdruck $r\{m, n\}$ lässt sich durch einen äquivalenten Ausdruck ohne Wiederholungsoperatoren ersetzen.

Sei $r^k = \underbrace{rr...r}_{k}$, $a = n - m$.
$r\{m, n\} = r^{m+i}$ (mit $i \in [0...a]$) $= r^m r^i$ (mit $i \in [0...a]$) $= r^m r?^a = \underbrace{rr...r}_{m}\underbrace{r?r?...r?}_{a}$

c) `\"\\`

d) Wird ein regulärer Ausdruck zum Matchen von Teilen einer Eingabe verwendet, so können klassische reguläre Ausdrücke nur den zu matchenden String selbst und nicht den Kontext in dem er steht verwenden um zu entscheiden ob der Ausdruck auf den gelesenen String passt. Durch den Lookahead-Operator wird es möglich in begrenztem Umfang den Kontext des Strings mit zu betrachten. Die Entscheidung über einen Match hängt also nicht nur von dem zu matchenden String selbst, sondern auch von seiner Umgebung ab. Dies stellt eine Erweiterung zu klassischen regulären Ausdrücken dar und kann nicht durch andere Operatoren ausgedrückt werden. Dieser Unterschied wirkt sich jedoch nur aus, wenn der reguläre Ausdruck zum Matchen eines Teilstrings verwendet wird.

Aufgabe 2 (3 Punkte)

Folgender Ausdruck erkennt diese Kommentare:

`\/\*([^\*]|\*[^\/])*\*\/`

Der Ausdruck bedeutet Folgendes: Es muss zunächst `/*` kommen, gefolgt von einer beliebigen Anzahl von Zeichen, die entweder nicht `*` sind oder `*` sind und nicht von `/` gefolgt werden. danach muss `*/` kommen.

Aufgabe 3 (7 Punkte)

Die antlr-Grammatik:

```
class WikiParser extends Parser;

text returns [String value = ""]
{String s;}
    : ( s = plaintext {value += s;}
        | s = underline {value += s;}
        | s = boldface {value += s;}
        | s = italics {value += s;}
      )* NEWLINE
    ;

plaintext returns [String value = ""]
    : s:STRING {value += s.getText();}
    ;

underline returns [String value=""]
{String s;}
    : UNDERSCORE {value += "<u>";}
      ( s = plaintext {value += s;}
        | s = boldface {value += s;}
        | s = italics {value += s;}
      )*
      UNDERSCORE {value += "</u>";}
    ;

boldface returns [String value=""]
{String s;}
    : STAR {value += "<b>";}
      ( s = plaintext {value += s;}
        | s = underline {value += s;}
        | s = italics {value += s;}
      )*
      STAR {value += "</b>";}
    ;

italics returns [String value=""]
{String s;}
    : PERCENT {value += "<i>";}
      ( s = plaintext {value += s;}
        | s = underline {value += s;}
        | s = boldface {value += s;}
      )*
      PERCENT {value += "</i>";}
    ;
```

```
class WikiLexer extends Lexer;

UNDERSCORE  : '_' ;
PERCENT     : '%' ;
STAR        : '*' ;

STRING      : ( 'a' .. 'z' | 'A' .. 'Z' | '0' .. '9' | ',' | '.' | ' ')+ ;

NEWLINE     : '\n' ;
```

Das Hauptprogramm:

```
import antlr.*;
import java.io.*;

public class Main {
    public static void main(String[] args) throws Exception {
        WikiLexer lexer = new WikiLexer(new DataInputStream(System.in));
        TokenBuffer buffer = new TokenBuffer(lexer);
        WikiParser parser = new WikiParser(buffer);
        String text =  parser.text();
        System.out.println(text);
    }
}
```

Aufgabe 4 (3 Punkte)

Folgende Symbolfolge könnte aus dem Java-Programm erzeugt werden:

(public,)(int,)(id,p1)(lparent,)(id,p2) (comma,)(id,p3)(rparent,)
(if,)(lparent,)(id,p2)(greater,)(id,p3)(rparent,)(lcurly,)
(return,)(id,p2)(minus,)(id,p3)(sepsymb,)
(rcurly,)(else,)(lcurly,)
(return,)(id,p3)(minus,)(id,p2)(sepsymb,)
(rcurly,)
(rcurly,)

Dabei würden folgende Einträge in der Symboltabelle erzeugt:

ID	Name
p1	symDiff
p2	i
p3	j