

# Übersetzerbau: Übung 04

von

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

## Aufgabe 1

Wie erweitern das Codefragment aus der Vorlesung, welches bereits Whitespaces überspringt, sodass nun auch Zeilen- und Blockkommentare ignoriert werden.

```
for ( ; ; peek = next input character) {
    if (peek is blank or tab) doNothing;
    else if (peek is newline) line = line + 1;
    else if (peek is '/') {
        peek = next input character;

        /* a) Ignore single line comments */
        if (peek is '/') {
            do {
                peek = next input character;
            } while (peek is not newline);
            continue;
        }

        /* b) Ignore multiline comments */
        if (peek is '*') {
            do {
                peek = next input character;
            } while (peek is not '*' && peek+1 is not '/');
            peek = next input character;
            continue;
        }
        break;
    }
}
```

## Aufgabe 2

Jetzt erweitern wir die `scan()` Methode aus der Vorlesung um die Erkennung der Operatoren `<`, `>`, `<=`, `>=`, `==`, `!=`.

```
Token scan() {
    // Ignore whitespaces and comments
    // Read digits
    // Read identifier and key words
    ...
    switch(peek) {
        case '<': case '>':
            if (peek+1 is '=')
                return new Token(peek concat peek+1);
            return new Token(peek);
        case '=': case '!=':
            if (peek+1 is '=')
                return new Token(peek concat peek+1);
            break;
        case default: // assume single character operator
            return new Token(peek);
    }
}
```

### Aufgabe 3

- a)  $((\epsilon | a)b^*)^*$ . Dieser Ausdruck beschreibt die Sprache, die alle Wörter über dem Alphabet  $\{a, b\}$  enthält, d. h. die  $a$ s und  $b$ s können beliebig angeordnet sein.
- b)  $(a | b)^*a(a | b)(a | b)$ . Dieser Ausdruck beschreibt die Sprache über dem Alphabet  $\{a, b\}$ , in der alle Wörter auf  $aaa$ ,  $aab$ ,  $aba$  oder  $abb$  enden. Also alle Wörter, deren drittletzter Buchstabe ein  $a$  ist.
- c)  $a^*ba^*ba^*ba^*$ . Dieser Ausdruck beschreibt die Sprache über dem Alphabet  $\{a, b\}$  deren Wörter genau drei  $b$ s enthalten.

### Aufgabe 4

- a) KfG für boolesche Ausdrücke.

$$\begin{aligned} S &\rightarrow S \wedge S \mid S \vee S \mid \neg S \mid W \\ W &\rightarrow t \mid f \end{aligned}$$

- b) Da die Grammatik aus a) nicht eindeutig ist, beachten wir nun die Prioritäten der Operatoren.  $\neg$  hat eine höhere Priorität als  $\wedge, \vee$ .

$$\begin{aligned} S &\rightarrow S \wedge N \mid S \vee N \mid N \\ N &\rightarrow \neg W \mid W \\ W &\rightarrow t \mid f \mid (S) \end{aligned}$$

- c) Nun überlegen wir uns die syntaxgerichtete Definition für eine Übersetzung der booleschen Ausdrücke in Präfix-Notation.

Produktion	Semantische Regel
$S \rightarrow S_1 \wedge N$	$S.p = \wedge \parallel S_1.p \parallel n.p$
$S \rightarrow S_1 \vee N$	$S.p = \vee \parallel S_1.p \parallel n.p$
$S \rightarrow N$	$S.p = N.p$
$N \rightarrow \neg W$	$N.p = \neg \parallel W.p$
$N \rightarrow W$	$N.p = W.p$
$W \rightarrow t$	$W.p = t$
$W \rightarrow f$	$W.p = f$
$W \rightarrow (S)$	$W.p = S.p$

- d) Das Übersetzungsschema für c) sieht nun wie folgt aus.

$$\begin{aligned} S &\rightarrow \{print(' \wedge ')\} S \wedge N \\ S &\rightarrow \{print(' \vee ')\} S \vee N \\ S &\rightarrow N \\ N &\rightarrow \{print(' \neg ')\} \neg W \\ N &\rightarrow W \\ W &\rightarrow \{print(' t ')\} t \\ W &\rightarrow \{print(' f ')\} f \\ W &\rightarrow (S) \end{aligned}$$

- e) Wie müssen für den prädiktiven Parser wieder die Linksrekursion eliminieren. Trotzdem ist es für den Parser nicht möglich bei der Produktion  $S$  mit einem Lookahead von 1 zu entscheiden welche Produktion angewendet werden muss. Wir können aber auch einen Parser mit Lookahead 2 benutzen, um so die richtige Produktion auswählen zu können.

$$\begin{aligned} S &\rightarrow \{print(' \wedge')\} N R_1 \mid \{print(' \vee')\} N R_2 \\ R_1 &\rightarrow \wedge N \\ R_2 &\rightarrow \vee N \\ N &\rightarrow \{print(' \neg')\} \neg W \mid W \\ W &\rightarrow \{print(' t')\} t \mid \{print(' f')\} f \mid (S) \end{aligned}$$