

Übersetzerbau: Übung 08

von

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

Aufgabe 1

Um überhaupt prädiktiv parsen zu können, müssen wir zunächst die Linksrekursion in B und T entfernen. Wir erhalten

$$\begin{aligned} B &\rightarrow TB' \\ B' &\rightarrow \underline{\text{or}} TB' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow \underline{\text{and}} FT' \mid \epsilon \\ F &\rightarrow \underline{\text{not}} F \mid (B) \mid \underline{\text{true}} \mid \underline{\text{false}} \end{aligned}$$

Jetzt können wir uns einen Parser überlegen:

```
void B() {
    T(); B2();
}

void B2(){
    if( lookahead == 'or')
        match('or'); T(); B2();
    else //do nothing for epsilon production
}

void T() {
    F(); T2();
}

void T2() {
    if( lookahead == 'and')
        match('and'); F(); T2();
    else //do nothing for epsilon production
}

void F() {
    switch(lookahead) {
        case 'not': match('not'); F(); break;
        case '(': match('('); B(); match(')'); break;
        case 'true': match('true'); break;
        default: match('false');
    }
}
```

Aufgabe 2

Auch hier haben wir wieder eine linksrekursive Grammatik gegeben, die wir zunächst „entrekursivieren“ müssen.

$$\begin{aligned} S &\rightarrow aR \\ R &\rightarrow S + R \mid S * R \mid \epsilon \end{aligned}$$

Jetzt sehen wir allerdings, dass R keine disjunkten FIRST-Mengen hat und wir dadurch nicht mit einem Lookahead von 1 parsen können. Das lässt sich jedoch durch Linksfaktorisierung beheben. Wir erhalten dann

$$\begin{aligned} S &\rightarrow aR \\ R &\rightarrow ST \\ T &\rightarrow +R \mid *R \mid \epsilon \end{aligned}$$

Jetzt können wir uns den Parser überlegen.

```
void S() {
    match('a'); R();
}

void R() {
    S(); T();
}

void T() {
    if (lookahead == '+')
        match('+'); R();
    else if (lookahead == '*')
        match('*'); R();
    else //do nothing for epsilon production
}
```

Aufgabe 3

a) Die Parse-Tabelle kann am schnellsten aufstellen, wenn man sich zuvor schon die FIRST- und FOLLOW-Mengen für alle Nichtterminale überlegt.

$$\begin{aligned} FIRST(S) &= \{if, while, begin, s\} & FOLLOW(S) &= \{\$, else, ;\} \\ FIRST(S') &= \{else, \epsilon\} & FOLLOW(S') &= FOLLOW(S) \\ FIRST(L) &= FIRST(S) & FOLLOW(L) &= \{end\} \\ FIRST(L') &= \{;, \epsilon\} & FOLLOW(L') &= FOLLOW(L) \end{aligned}$$

Nun können wir die Tabelle aufstellen. Aus Platzgründen haben wir nur die rechten Regelseiten hingeschrieben. Wir verwenden die panische Fehlerbehandlung, dazu benutzen wir die Elementen der Follow-Mengen und markieren die entsprechenden Zellen mit „sync“ (falls diese leer sind). Wenn der Parser auf eine solche Zelle trifft, wird das oberste Nichtterminal-Symbol vom Stack entfernt. Trifft er hingegen auf eine leere Zelle, wird das nächste Symbol aus der Eingabe entfernt.

	if	then	while	do	begin	end	else	;	e	s	\$
S	if e then SS'		while e do S		begin L end		sync	sync		s	sync
S'							else S, ϵ	ϵ			ϵ
L	SL'		SL'		SL'	sync				SL'	
L'						ϵ		$;L$			

Wir sehen, dass in einer Zelle zwei Produktionen landen, d. h. dass wir nur nichtdeterministisch parsen können. Wir können aber mit „Methode Brutale“ die Produktion $S' \rightarrow \epsilon$ in der (S'/else) -Zelle streichen.

b) Für die beiden Eingaben vollziehen wir die einzelnen Schritte des Parsers nach, indem wir die Eingabe und den aktuellen Inhalt des Stacks betrachten. Zu Anfang liegt das Startsymbol auf dem Stack. Für jedes Nichtterminal auf dem Stack wird dann in der passenden Zelle der Parser-Tabelle M nachgesehen und die entsprechende Produktion angewendet. Terminalsymbole auf dem Stack werden versucht mit der Eingabe zu matchen und dann entfernt.

1. **if e then s; if e then s end**

Wenn man hier mit dem Startsymbol S auf dem Stack beginnt, wird das Parsen schon bei **; if e then s end** abgebrochen, da der Stack leer wird, bevor die Eingabe vollständig gelesen ist. An dieser Stelle kann man nun wieder von vorne beginnen und zwar mit dem nächsten Nichtterminalsymbol, in dessen First-Menge sich der Anfang der Eingabe (if) noch befindet. Das ist in unserem Fall L . Zusätzlich passen wir dann die Follow-Menge von L an und fügen das Symbol $\$$ hinzu. Dadurch erhält der Tabelleneintrag $M[L, \$]$ und $M[L', \$]$ den Eintrag *sync*.

Eingabe	Stack	Aktion
if e then s; .. \$	$L\$$	$M[L, \text{if}]$
if e then s; .. \$	$SL'\$$	$M[S, \text{if}]$
if e then s; .. \$	if e then $SS'L'\$$	match
s; if .. \$	$SS'L'\$$	$M[S, s]$
s; if .. \$	s $S'L'\$$	match
; if .. \$	$S'L'\$$	$M[S', ;] \rightarrow \epsilon$
; if .. \$	$L'\$$	$M[L', ;]$
; if .. \$; $L\$$	match
if e then s end \$	$L\$$	$M[L, \text{if}]$
if e then s end \$	$SL'\$$	$M[S, \text{if}]$
if e then s end \$	if e then $SS'L'\$$	match
s end \$	$SS'L'\$$	$M[S, s]$
s end \$	s $S'L'\$$	match
end \$	$S'L'\$$	$M[S', \text{end}] \rightarrow \text{leer: delete(end)}$
\$	$S'L'\$$	$M[S', \$] \rightarrow \epsilon$
\$	$L'\$$	$M[L', \$] \rightarrow \text{sync, pop}(L')$
\$	\$	match $\rightarrow \text{OK}$

2. while e do begin s; if e then s; end

Eingabe	Stack	Aktion
while e do begin .. \$	$S\$$	$M[S, \text{while}]$
while e do begin .. \$	while e do $S\$$	match
begin s; .. \$	$S\$$	$M[S, \text{begin}]$
begin s; .. \$	begin L end $S\$$	match
s; if .. \$	L end \$	$M[L, s]$
s; if .. \$	SL' end \$	$M[S, s]$
s; if .. \$	sL' end \$	match
; if .. \$	L' end \$	$M[L', ;]$
; if .. \$	$;L$ end \$	match
if e then s; end \$	L end \$	$M[L, \text{if}]$
if e then s; end \$	SL' end \$	$M[S, \text{if}]$
if e then s; end \$	if e then $SS'L'$ end \$	match
s; end \$	$SS'L'$ end \$	$M[S, s]$
s; end \$	$sS'L'$ end \$	match
; end \$	$S'L'$ end \$	$M[S', ;] \rightarrow \epsilon$
; end \$	L' end \$	$M[L', ;]$
; end \$	$;L$ end \$	match
end \$	L end \$	$M[L, \text{end}] \rightarrow \text{sync, pop}(L)$
end \$	end \$	match $\rightarrow \text{OK}$