Tp2 - Análise Sintática

Matheus Alves de Resende

May 14, 2019

1 Introdução

Este trabalho descreve um analisador sintático implementado em java para a linguagem de programação Tiger[1]. No intuito de realizar tal tarefa, fora utilizada a ferramenta Cup[2] para a geração do analisador sintático.

Para a realização do trabalho, foram tomados como referência o material [1] do curso de Compiladores I, ministrado pela professora Mariza Bigonha, da Universidade Federal de Minas Gerais e o livro texto [2] do curso.

2 Ferramentas de Apoio

Para a criação do compilador foram usadas apenas as ferramentas Jlex e Cup.

3 Modo de uso

Para gerar o analisador sintático, é necessário rodar o comando *java java_cup.Main -parser Grm -expect 6 <Grm.cup >Grm.out 2>Grm.err* . Ao final da execução do comando, dois códigos fontes são gerados, Grm.java, que possui a classe que é o analisador sintático propriamente dito e a classe sym.java, contendo as constantes relativas a cada terminal da linguagem. Os arquivos fonte gerados podem ser compilados normalmente com o comando *java*.

4 Suposições

Para a realização do trabalho, assumi-se a existência de uma classe lexer, contendo o método nextToken(), que assumiria o papel do analisador léxico provendo tokens de entrada para o analisador sintático. A classe assume o formato do exemplo de arquivo .cup disponibilizado pela especificação deste trabalho.

[1] https://homepages.dcc.ufmg.br/mariza/Cursos/CompiladoresI/Comp2019-1/Pagina/Dia-a-Dia/comp2019-1.html

[2] https://www.cs.princeton.edu/ appel/modern/java/CUP/

Também foram criadas associações de tokens para terminais, da seguinte forma:

Token	**Lexema**
EF	EOF
EMPTY	ϵ
LPAR	(
RPAR	)
MINUS	-
PLUS	+
TIMES	*
DIVIDE	\
ASSIGN	:=
BIGGER	>
SMALLER	<
EQUAL	=
BIGGER_EQUAL	=>
SMALLER_EQUAL	=<
DIFF	<>
AND	&
OR	|
LBRA	[
RBRA	]
LKEY	{
RKEY	}
COLON	:
VIRG	,
DOT	.
NIL	NIL
OF	of
IF	if
THEN	then
ELSE	else
WHILE	while
DO	do
FOR	for
TO	to
SEMI	;
BREAK	break
LET	let
IN	in
END	end
TYPE	type
ARRAY_OF	array of
VAR	var
FUNCTION	function

5 Gramática modificada

A gramática que originalmente especificava a linguagem era altamente ambigua, consequentemente não era LALR e seu uso direto implicava numa grande quantidade de conflitos reduce/reduce. Para resolver este problema a mesma precisou de passar por uma refatoração. A gramática refatorada pode ser averiguada nas diretivas presentes no arquivo Grm.cup, na última sessão deste documento.

Tendo esta questão da ambiguidade em vista, foram criados não terminais adicionais para sempre favorecer uma associação mais a esquerda. Conjuntamente, foi estabelecido um símbolo de EOF para facilitar o reconhecimento.

Na gramática original também não havia precedencia nos operadores aritméticos. No nosso analisador tal precedência é garantida pelas diretivas *precedence left*, que define a prioridade entre os operadores.

5.1 Conflito Shift-Reduce

Ao fim do processo de desambiguação a gramática apresentou um conflito shift/reduce na expressão *ID (*) "[" exp "]" of exp*, onde ID poderia ser reduzido para l_value, ou poderia ser expandido para conseguir o match *type_id "[" exp "]" of exp* e ser posteriormente reduzido para type_id.

Este conflito foi resolvido através da remoção da produção type_id ::= ID, e criação de um não terminal intermediário que teria a produção type_id2 ::= ID "[" exp "]", e modificação da produção *type_id "[" exp "]" of exp* para, *type_id2 of exp*. Desta forma resolvendo o problema de shift/reduce com o uso do lookahead **of**.

References

[1] Roberto Bigonha, Mariza e Bigonha. Compiladores, análise sintática. `https://homepages.dcc.ufmg.br/~mariza/Cursos/CompiladoresI/Geral/Slides/SlidesPagComp/2019-1/comp-05-sintatica-p2.pdf`. Accessed: 2019-05-14.

[2] J Ulman, AV Aho, and R Sethi. Compiler design: Principles, tools, and techniques. *Reading, Pearson Education Inc*, 1986.

6 Código

```
package Parse;
import java_cup.runtime.*;

init with {: lexer.init();:};
scan with {: return lexer.nextToken(); :};

terminal Integer NUM;
terminal EF, EMPTY,SEMI, LPAR, RPAR, MINUS, PLUS, TIMES, DIVIDE, ASSIGN, BIGGER, SMALLER,
terminal NIL, OF, IF, THEN, ELSE, WHILE, DO, FOR, TO, BREAK, LET, IN, END, TYPE, ARRAY_OF,
terminal String ID, STRING;
non terminal type_id2, exp1, whileexp1, ifexp3, forexp, orexp, orexp1, andexp1, ebiggerexp

precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE;
precedence left WHILE, DO, FOR, IN;

precedence left LPAR, RPAR;
precedence left LKEY, RKEY;
precedence left LBRA, RBRA;

start with exp1;

exp1 ::= exp ;

exp ::= l_value EF
| NIL EF
| plusexp EF
| LPAR expseq RPAR EF
| NUM EF
| STRING EF
| ID LPAR args RPAR EF
| minusexp EF
| timesexp EF
| equalexp EF
| divexp EF
| diffexp EF
| biggerexp EF
| smallerexp EF
| ebiggerexp EF
| esmallerexp EF
| andexp EF
| orexp EF
| ID LKEY ID EQUAL exp idexps RKEY EF
| type_id2  OF exp EF
| l_value ASSIGN exp EF
| ifexp1 EF
| ifexp2 EF
| whileexp EF
```

```
| forexp EF
| BREAK EF
| LET decs IN expseq END EF
;


ifexp1 ::= IF ifexp3 THEN exp;

ifexp2 ::= IF ifexp3 THEN exp ELSE exp;

ifexp3 ::= l_value
| NIL
| plusexp
| LPAR expseq RPAR
| STRING
| minusexp
| equalexp
| divexp
| diffexp
| biggerexp
| smallerexp
| ebiggerexp
| esmallerexp
| andexp
| orexp
| type_id2 OF exp
| ID LKEY ID EQUAL exp idexps RKEY
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;

whileexp ::= WHILE whileexp1 DO exp;


whileexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2  OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| forexp
| BREAK
| LET decs IN expseq END
;
```

```
plusexp ::= plusexp1 PLUS exp;

plusexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;



minusexp ::= minusexp1 MINUS exp;

minusexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
;



timesexp ::= timesexp1 TIMES exp;

timesexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
```

```
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| BREAK
| LET decs IN expseq END
;


equalexp ::= equalexp1 EQUAL exp;

equalexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;


divexp ::= divexp1 DIVIDE exp;

divexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;


diffexp ::= diffexp1  DIFF exp;

diffexp1 ::= l_value
```

```
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;



orexp ::= orexp1 OR exp;

orexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2  OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;


andexp ::= andexp1 AND exp;

andexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
```

```
| BREAK
| LET decs IN expseq END
;


ebiggerexp ::= ebiggerexp1 BIGGER_EQUAL exp;


ebiggerexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;


esmallerexp ::= esmallerexp1 SMALLER_EQUAL exp;

esmallerexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;


smallerexp ::= smallerexp1 SMALLER exp;

smallerexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
```

```
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;

biggerexp ::=  biggerexp1 BIGGER exp;

biggerexp1 ::= l_value
| NIL
| LPAR expseq RPAR
| NUM
| STRING
| ID LKEY ID EQUAL exp idexps RKEY
| type_id2 OF exp
| l_value ASSIGN exp
| ifexp1
| ifexp2
| whileexp
| forexp
| BREAK
| LET decs IN expseq END
;



forexp ::= FOR ID ASSIGN exp TO exp DO;


decs ::=
| dec decs
;

dec ::= tydec
| vardec
| fundec
;

tydec ::= TYPE ID EQUAL ty;

ty ::= ID
| LKEY ID COLON type_id tyfields1 RKEY
| ARRAY_OF ID
;
```

```
tyfields ::= EMPTY
| ID COLON type_id tyfields1
;

tyfields1 ::= EMPTY
| VIRG ID COLON type_id tyfields1
;

vardec ::= VAR ID ASSIGN exp
| VAR ID COLON type_id ASSIGN exp
;

fundec ::= FUNCTION ID LPAR tyfields RPAR EQUAL exp
| FUNCTION ID LPAR tyfields RPAR COLON type_id EQUAL exp
;

l_value ::= ID
| l_value DOT ID
| l_value LBRA exp RBRA
;


type_id ::= ID;

type_id2 ::= ID LBRA exp RBRA;

expseq ::= EMPTY
|  exp expseq1
;

expseq1 ::= EMPTY
| SEMI  exp expseq1
;

args ::= EMPTY
|  exp args1
;

args1 ::= EMPTY
| VIRG args1 exp
;

idexps ::= EMPTY
| VIRG ID EQUAL idexps
;
```