REO3 – REORaiser

Pedro Antônio de Souza – 201810557

1 GRAMÁTICA ALGUMA

O quadro abaixo apresenta a especificação da Gramática Livre de Contexto (GLC) para representar as regras sintáticas da gramática ALGUMA.

NÃO-TERMINAIS	GRAMÁTICA

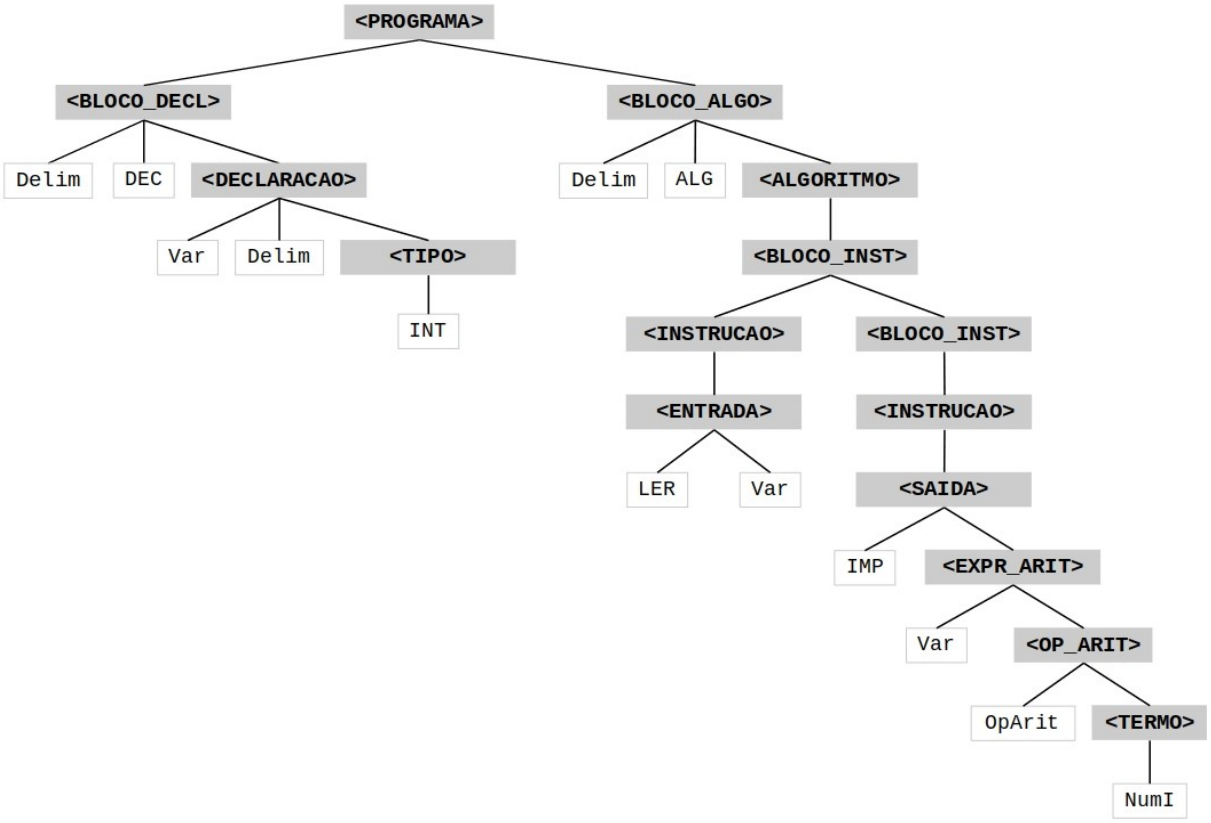
<PROGRAMA>
 	
<BLOCO_DECL>
<DECLARACAO>
<TIPO>
 	
<BLOCO_ALGO>
<ALGORITMO>
<BLOCO_INST>
<INSTRUCAO>
<ESCOPO>
 	
<ENTRADA>
<ATRIBUICAO>
<CONDICIONAL>
<LOOP>
<SAIDA>
 	
<EXPR_ARIT>
<EXPR_ARIT_FP>
<OP_ARIT>
 	
<EXPR_REL>
<OP_REL>
 	
<EXPR_BOOL>
<EXPR_BOOL_FP>
<OP_BOOL>
 	
<CONDICAO>
<TERMO>
	<PROGRAMA> ::= <BLOCO_DECL> <BLOCO_ALGO>;
	//Bloco de DECLARACOES
	<BLOCO_DECL> ::= Delim DEC <DECLARACAO>;
	<DECLARACAO> ::= Var Delim <TIPO> <DECLARACAO> | Var Delim <TIPO>;
	<TIPO> ::= INT | REA;
	//Bloco de ALGORITMO
	<BLOCO_ALGO> ::= Delim ALG <ALGORITMO>;
	<ALGORITMO> ::= <BLOCO_INST>;
	<BLOCO_INST> ::= <INSTRUCAO> | <INSTRUCAO> <BLOCO_INST>;
	<INSTRUCAO> ::= <ENTRADA>
	| <ATRIBUICAO>
	| <CONDICIONAL>
	| <LOOP>
	| <SAIDA>;
	<ESCOPO> ::= INI <BLOCO_INST> FIM;
	//Instruções permitidas
	<ENTRADA> ::= LER Var;
	<ATRIBUICAO> ::= ATR <TERMO> A Var
	| ATR <EXPR_ARIT> A Var
	| ATR Var A Var;
	<CONDICIONAL> ::= SE <CONDICAO> ENT <ESCOPO>
	| SE <CONDICAO> ENT <INSTRUCAO>;
	<LOOP> ::= ENQ <CONDICAO> <ESCOPO>;
	<SAIDA> ::= IMP <TERMO> | IMP <EXPR_ARIT> | IMP Var | IMP Str;
	//Expressoes permitidas (aritmetica, relacional e booleana)
	<EXPR_ARIT> ::= <TERMO> <OP_ARIT>
	| Var <OP_ARIT>
	| AP <EXPR_ARIT_FP>;
	<EXPR_ARIT_FP> ::= <EXPR_ARIT> FP
	| <EXPR_ARIT> FP <OP_ARIT>;
	<OP_ARIT> ::= OpArit <EXPR_ARIT> | OpArit <TERMO> | OpArit Var;
	
	<EXPR_REL> ::= <TERMO> <OP_REL> | Var <OP_REL> | <EXPR_ARIT> <OP_REL>;
	<OP_REL> ::= OpRel <TERMO> | OpRel Var | OpRel <EXPR_ARIT>;
	
	<EXPR_BOOL> ::= Var <OP_BOOL>
	| <EXPR_REL> <OP_BOOL>
	| Str <OP_BOOL>
	| AP <EXPR_BOOL_FP>;
	<EXPR_BOOL_FP> ::= <EXPR_BOOL> FP
	| <EXPR_BOOL> FP <OP_BOOL>;
	<OP_BOOL> ::= OpBool <EXPR_BOOL> | OpBool Var
	| OpBool <EXPR_REL> | OpBool Str;
	
	<CONDICAO> ::= <EXPR_REL> | <EXPR_BOOL>;
	<TERMO> ::= NumI | NumR;

2 ÁRVORE SINTÁTICA

Para exemplificação de construção de uma árvore sintática usando a gramática especificada, será utilizado o código a seguir:

```
1. % Algoritmo que lê um número e imprime seu dobro
2. :DECLARACOES
3. x:INTEIRO
4.
5. :ALGORITMO
6. LER x
7. IMPRIMIR x*2
```

Árvore sintática construída.



3 ABORDAGENS ASCENDENTE E DESCENDENTE

A análise sintática pode ser efetuada a partir de dois métodos: o **ascendente** (ou *bottom-up*) e o **descendente** (*top-down*). No **método ascendente**, é utilizada derivação mais a direita inversa, construindo a árvore partindo das folhas e indo até o símbolo não-terminal inicial. O objetivo é sempre converter partes da entrada em símbolos não-terminais até que se encontre o simbolo não-terminal inicial. Já no **método descendente** parte-se da raiz da árvore até que todas as folhas sejam símbolos terminais (que neste caso, são *tokens*). Executa-se derivações sucessivas até que não haja mais símbolos não-terminais, portanto, a partir da árvore pode-se extrair os *tokens* lendo as folhas da esquerda para a direita.

No exercício anterior foi utilizado o método descendente, no qual, a partir das produções especificadas na gramática, tentou-se produzir os *tokens* do código de entrada. Assim, sabemos que a sequência de *tokens* recebidos foi `Delim DEC Var Delim INT Delim ALG LER Var IMP Var OpArit NumI`. Abaixo, pode-se observar como o seria caso fosse escolhida a abordagem ascendente para essa análise sintática. Os conjuntos de símbolos sublinhados evidencia os que serão substituídos por não-terminais no próximo passo e os símbolos que os substituem estarão destacados com a mesma cor do sublinhado.

```
Delim DEC Var Delim INT Delim ALG LER Var IMP Var OpArit NumI
Delim DEC Var Delim INT Delim ALG LER Var IMP Var OpArit <TERMO>
Delim DEC Var Delim INT Delim ALG LER Var IMP Var <OP_TERMO>
Delim DEC Var Delim INT Delim ALG LER Var IMP <EXPR_ARIT>
Delim DEC Var Delim INT Delim ALG LER Var <SAIDA>
Delim DEC Var Delim INT Delim ALG LER Var <INSTRUCAO>
Delim DEC Var Delim INT Delim ALG LER Var <BLOCO_INST>
Delim DEC Var Delim INT Delim ALG <ENTRADA> <BLOCO_INST>
Delim DEC Var Delim INT Delim ALG <INSTRUCAO> <BLOCO_INST>
Delim DEC Var Delim INT Delim ALG <BLOCO_INST>
Delim DEC Var Delim INT Delim ALG <ALGORITMO>
Delim DEC Var Delim INT <BLOCO_ALGO>
Delim DEC Var Delim <TIPO> <BLOCO_ALGO>
Delim DEC <DECLARACAO> <BLOCO_ALGO>
<BLOCO_DECL> <BLOCO_ALGO>
<PROGRAMA>
```

4 SIMULAÇÃO DE ERROS SINTÁTICOS

Para exemplificar a análise de um erro sintático será utilizado o código a seguir, no qual não há bloco de declarações.

```
1. % Algoritmo com erro sintático.
2. :ALGORITMO
3. IMPRIMIR "erro"
```

Os *tokens* recebidos pelo analisador sintático serão `Delim ALG IMP Str`. Assim, ao tentar derivar a gramática, logo após o delimitador a gramática já encontra o erro, pois se espera um *token* do tipo `DEC`.

<PROGRAMA> ⇒ **<BLOCO_DECL>** **<BLOCO_ALGO>** ⇒ `Delim` *Era esperado: DEC*