Analisador Sintático

Brendon Vicente Rocha Silva
Graduando em Ciências da Computação
Universidade Federal de Santa Catarina - UFSC
Florianópolis, SC, Brasil
<brendon.vicente@grad.ufsc.br>

Para este trabalho foi necessário o desenvolvimento de um analisador sintático para a linguagem *LCC-2022-2*, gerada pela gramática *CC-2022-2*.

Para executar tal tarefa, foi preciso a implementação de um analisador léxico, para que fosse gerada a sequência de *tokens* utilizada para alimentar o analisador sintático.

O projeto foi desenvolvido em linguagem de programação *Python 3.10*, com auxílio da ferramenta *PLY* e de um módulo para manipulação de gramáticas, parte de um trabalho prévio do autor, disponível neste link.

MODIFICAÇÕES EM *CC-2022-2*

Para a segunda parte do presente trabalho foram feitas algumas modificações na gramática proposta, a fim de proporcionar melhor usabilidade e mais comodidade ao se desenvolver códigos na linguagem estudada.

As seguintes modificações foram feitas:

```
STATEMENT        → (VARDECL; | ATRIBSTAT; | PRINTSTAT; | READSTAT;
                    | RETURNSTAT; | IFSTAT | FORSTAT | {STATELIST}
                    | break; | FUNCCALL; | ;)
PARAMLISTCALL    → (ident, PARAMLISTCALL | ident | FACTOR)?
RETURNSTAT       → return (EXPRESSION)?
ATRIBSTAT        → LVALUE = (EXPRESSION | ALLOCEXPRESSION | ~~FUNCCALL~~)
FACTOR           → (int_constant | float_constant | string_constant
                    | null | LVALUE | (NUMEXPRESSION) | FUNCCALL)
```

CC-2022-2 NA FORMA CONVENCIONAL

Para transformar *CC-2022-2* para a forma convencional foi necessária a adição de alguns símbolos não-terminais (***TYPE***, ***INT_INDEX***, ***ALLOC_SIZE***, ***RELOP***, ***SUM***, ***MULTI*** e ***NUM_INDEX***), além de variações “linha” de símbolos já existentes ou recém-adicionados (***ATRIBSTAT’***, ***IFSTAT’***, ***STATELIST’***, ***ALLOC_SIZE’***, ***EXPRESSION’***, ***NUMEXPRESSION’***, ***TERM’***).

Por fim, a *ConvCC-2022-2* tomou a seguinte forma:

```
PROGRAM           → STATEMENT | FUNCLIST | &
FUNCLIST          → FUNCDEF FUNCLIST | FUNCDEF
FUNCDEF           → DEF ID ( PARAMLIST ) { STATELIST }
PARAMLIST         → & | TYPE ID , PARAMLIST | TYPE ID
TYPE              → INT | FLOAT | STRING
STATEMENT         → VARDECL ; | ATRIBSTAT ; | PRINTSTAT ; | READSTAT ;
                    | RETURNSTAT ; | IFSTAT | FORSTAT | { STATELIST }
                    | BREAK ; | FUNCCALL ; | ;
VARDECL           → TYPE ID INT_INDEX
INT_INDEX         → [ INT_CONSTANT ] INT_INDEX | &
ATRIBSTAT         → LVALUE = ATRIBSTAT'
ATRIBSTAT'        → EXPRESSION | ALLOCEXPRESSION
FUNCCALL          → ID ( PARAMLISTCALL )
PARAMLISTCALL     → ID , PARAMLISTCALL | ID | FACTOR | &
PRINTSTAT         → PRINT EXPRESSION
READSTAT          → READ LVALUE
RETURNSTAT        → RETURN EXPRESSION | RETURN
IFSTAT            → IF ( EXPRESSION ) STATEMENT IFSTAT'
IFSTAT'           → & | ELSE STATEMENT
FORSTAT           → FOR ( ATRIBSTAT ; EXPRESSION ; ATRIBSTAT ) STATEMENT
STATELIST         → STATEMENT STATELIST'
STATELIST'        → & | STATELIST
ALLOCEXPRESSION   → NEW TYPE ALLOC_SIZE
ALLOC_SIZE        → [ NUMEXPRESSION ] ALLOC_SIZE'
ALLOC_SIZE'       → ALLOC_SIZE | &
EXPRESSION        → NUMEXPRESSION EXPRESSION'
EXPRESSION'       → & | RELOP NUMEXPRESSION
RELOP             → < | > | LESS_EQUAL_THAN | GREATER_EQUAL_THAN | EQUAL
                    | DIFFERENT
NUMEXPRESSION     → TERM NUMEXPRESSION'
NUMEXPRESSION'    → SUM TERM NUMEXPRESSION' | &
SUM               → + | -
TERM              → UNARYEXPR TERM'
TERM'             → & | MULTI UNARYEXPR TERM'
MULTI             → * | / | %
UNARYEXPR         → SUM FACTOR | FACTOR
FACTOR            → INT_CONSTANT | FLOAT_CONSTANT | STRING_CONSTANT | NULL
                    | LVALUE | ( NUMEXPRESSION ) | FUNCCALL
LVALUE            → ID NUM_INDEX
NUM_INDEX         → [ NUMEXPRESSION ] NUM_INDEX | &
```

RECURSÃO À ESQUERDA DE *ConvCC-2022-2*

Qualquer produção da gramática *ConvCC-2022-2* tem em sua cabeça somente um símbolo não terminal. Além disso, uma produção só terá recursão à esquerda se um não terminal derivar mais à esquerda, em qualquer nível de derivação, a si próprio.

Tendo isso em mente e analisando-se a gramática produzida na seção anterior, é possível constatar que esse não é o caso de nenhuma das produções apresentadas. *ConvCC-2022-2*, portanto, não é recursiva à esquerda.

FATORAÇÃO À ESQUERDA DE *ConvCC-2022-2*

ConvCC-2022-2 não está fatorada à esquerda, como evidenciado nas produções:

```
FUNCLIST          → FUNCDEF FUNCLIST | FUNCDEF

PARAMLIST         → TYPE ID , PARAMLIST | TYPE ID

STATEMENT         → ATRIBSTAT ; | FUNCCALL ;
  ATRIBSTAT       -*→ ID …
  FUNCCALL        -*→ ID …

PARAMLISTCALL     → ID , PARAMLISTCALL | ID | FACTOR
  FACTOR          -*→ ID …

RETURNSTAT        → RETURN EXPRESSION | RETURN

FACTOR            → LVALUE | FUNCCALL
  LVALUE          → ID …
  FUNCCALL        → ID …
```

Para eliminar as ambiguidades foram feitas as seguintes mudanças:

```
FUNCLIST          → FUNCDEF FUNCLIST1
FUNCLIST1         → FUNCLIST | &

PARAMLIST         → & | TYPE ID PARAMLIST1
PARAMLIST1        → , PARAMLIST | &

STATEMENT         → VARDECL ; | PRINTSTAT ; | READSTAT ; | RETURNSTAT ;
                    | IFSTAT | FORSTAT | { STATELIST } | BREAK ; | ;
                    | ID STATEMENT1 ;
STATEMENT1        → NUM_INDEX = ATRIBSTAT' | ( PARAMLISTCALL )

PARAMLISTCALL     → ID PARAMLISTCALL1 | &
PARAMLISTCALL1    → & | , PARAMLISTCALL | NUM_INDEX | ( PARAMLISTCALL)

RETURNSTAT        → RETURN RETURNSTAT1
RETURNSTAT1       → EXPRESSION | &

FACTOR            → INT_CONSTANT | FLOAT_CONSTANT | STRING_CONSTANT | NULL
                    | ( NUMEXPRESSION ) | ID FACTOR1
FACTOR1           → NUM_INDEX | ( PARAMLISTCALL )
```

LL(1)

A gramática não é *LL(1)*, pois para a entrada da tabela de parsing na célula de ***IFSTAT'*** *x* ***ELSE*** existem duas produções. Para solucionar o problema, foi adicionado um novo terminal à gramática: ***endif***, representado pelo *token* ***ENDIF***.

Com tal mudança, duas produções foram modificadas:

```
STATEMENT         → VARDECL ; | PRINTSTAT ; | READSTAT ; | RETURNSTAT ;
                    | IFSTAT ; | FORSTAT | { STATELIST } | BREAK ; | ;
                    | ID STATEMENT1 ;
IFSTAT'           → ENDIF | ELSE STATEMENT ENDIF
```

ConvCC-2022-2 LL(1)

Por fim, a gramática tomou a seguinte forma:

```
PROGRAM          → STATEMENT | FUNCLIST | &
FUNCLIST         → FUNCDEF FUNCLIST1
FUNCLIST1        → FUNCLIST | &
FUNCDEF          → DEF ID ( PARAMLIST ) { STATELIST }
PARAMLIST        → & | TYPE ID PARAMLIST1
PARAMLIST1       → , PARAMLIST | &
TYPE             → INT | FLOAT | STRING
STATEMENT        → VARDECL ; | PRINTSTAT ; | READSTAT ; | RETURNSTAT ;
                   | IFSTAT ; | FORSTAT | { STATELIST } | BREAK ; | ;
                   | ID STATEMENT1 ;
STATEMENT1       → NUM_INDEX = ATRIBSTAT' | ( PARAMLISTCALL )
VARDECL          → TYPE ID INT_INDEX
INT_INDEX        → [ INT_CONSTANT ] INT_INDEX | &
ATRIBSTAT        → LVALUE = ATRIBSTAT'
ATRIBSTAT'       → EXPRESSION | ALLOCEXPRESSION
FUNCCALL         → ID ( PARAMLISTCALL )
PARAMLISTCALL    → ID PARAMLISTCALL1 | &
PARAMLISTCALL1   → & | , PARAMLISTCALL | NUM_INDEX | ( PARAMLISTCALL)
PRINTSTAT        → PRINT EXPRESSION
READSTAT         → READ LVALUE
RETURNSTAT       → RETURN RETURNSTAT1
RETURNSTAT1      → EXPRESSION | &
IFSTAT           → IF ( EXPRESSION ) STATEMENT IFSTAT'
IFSTAT'          → ENDIF | ELSE STATEMENT ENDIF
FORSTAT          → FOR ( ATRIBSTAT ; EXPRESSION ; ATRIBSTAT ) STATEMENT
STATELIST        → STATEMENT STATELIST'
STATELIST'       → & | STATELIST
ALLOCEXPRESSION  → NEW TYPE ALLOC_SIZE
ALLOC_SIZE       → [ NUMEXPRESSION ] ALLOC_SIZE'
ALLOC_SIZE'      → ALLOC_SIZE | &
EXPRESSION       → NUMEXPRESSION EXPRESSION'
EXPRESSION'      → & | RELOP NUMEXPRESSION
RELOP            → < | > | LESS_EQUAL_THAN | GREATER_EQUAL_THAN | EQUAL
                   | DIFFERENT
NUMEXPRESSION    → TERM NUMEXPRESSION'
NUMEXPRESSION'   → SUM TERM NUMEXPRESSION' | &
SUM              → + | -
TERM             → UNARYEXPR TERM'
TERM'            → & | MULTI UNARYEXPR TERM'
MULTI            → * | / | %
UNARYEXPR        → SUM FACTOR | FACTOR
FACTOR           → INT_CONSTANT | FLOAT_CONSTANT | STRING_CONSTANT | NULL
                   | ( NUMEXPRESSION ) | ID FACTOR1
FACTOR1          → NUM_INDEX | ( PARAMLISTCALL )
LVALUE           → ID NUM_INDEX
NUM_INDEX        → [ NUMEXPRESSION ] NUM_INDEX | &
```

IMPLEMENTAÇÃO

Para o desenvolvimento do analisador sintático, foi utilizado um módulo (módulo Grammar), parte de um esforço prévio do autor. Na ferramenta em questão, se implementou um analisador preditivo *LL(1)*, assim como um analisador *SLR(1)*.

Foi desenvolvida uma nova função, para o trabalho em questão, baseada na solução já implementada, capaz de construir o analisador *LL(1)* utilizando-se do analisador léxico, construído com a ferramenta *PLY*.

Como entrada esperada, a nova função deve receber um arquivo de código, escrito na linguagem *LCC-2022-2*.

Caso não haja erros sintáticos, a função retornará o valor booleano ***True***.

Caso erros sintáticos sejam encontrados, uma exceção indicando qual o local do erro, assim como qual entrada na tabela de reconhecimento sintático está vazia será exibida.