Universidade Federal de Santa Catarina - UFSC
Departamento de Informática e Estatística
Curso de Ciências da Computação
INE5426 - Construção de Compiladores

Luiz João Carvalhaes Motta
Maria Eduarda de Melo Hang
Marina Pereira das Neves Guidolin

Relatório EP2 - Analisador Sintático

Florianópolis, 23 de Outubro de 2020

1 INTRODUÇÃO

Esse trabalho possui como objetivo a implementação de um Analisador Sintático para a linguagem ConvCC-2020-1 conforme proposto no enunciado do Exercício Programa 2.

2 CONVERSÃO DE BNF PARA A FORMA CONVENCIONAL

A gramática a seguir é a gramática ConvCC-2020-1, obtida a partir da conversão da gramática CC-2020-1 da forma BNF para a forma convencional. Os tokens referentes à símbolos e pontuação, tipos, keywords, expressões aritméticas e operadores relacionais estão identificados entre ‘ ‘. Logo após a gramática, é explicado o modo como foi feita a transformação.

PROGRAM → STATEMENT | FUNCLIST | &

FUNCLIST → FUNCDEF FUNCLIST | FUNCDEF

FUNDEF → ‘def’ ‘ident’ ‘(‘ PARAMLIST ‘)’ ‘{‘ STATELIST ‘}’

PARAMLIST → ‘ident’ (‘,’ PARAMLISTALL | &) | &

STATEMENT → VARDECL ‘;’ | ATRIBSTAT ‘;’ | PRINTSTAT ‘;‘ | READSTAT ‘;’ | RETURNSTAT ‘;’ | IFSTAT | FORSTAT | ‘{‘ STATELIST ’}’ | ‘break’ ‘;’ | ‘;’

VARDECL → (‘int’ | ‘float’ | ‘string’) ‘ident’ A

A → T1 A | &

T1 → ‘[‘ ‘int_constant’ ‘]’

ATRIBSTAT → LVALUE ‘=’ (EXPRESSION | ALLOCEXPRESSION | FUNCCALL)

FUNCCALL → ‘ident’ ‘(‘ PARAMLISTCALL ‘)’

PARAMLISTCALL →’ident’ (, PARAMLISTCALL | &) | &

PRINTSTAT → ‘print’ EXPRESSION

READSTAT → ‘read’ LVALUE

RETURNSTAT → ‘return’

IFSTAT → ‘if’ (EXPRESSION) STATEMENT (‘else’ STATEMENT | &)

FORSTAT → ‘for’ ‘(‘ ATRIBSTAT; EXPRESSION; ATRIBSTAT ‘)’ STATEMENT

STATELIST → STATEMENT (STATELIST | &)

ALLOCEXPRESSION → ‘new’ (‘int’ | ‘float’ | ‘string’) T2 B

B → T2 B | &

T2 → ‘[‘ NUMEXPRESSION ‘]’

EXPRESSION → NUMEXPRESSION (((‘<’ | ’ >’ | ‘ <=’ | ‘ >=’ | ‘==‘ | ‘!=’) NUMBEXPRESSION) | &)

NUMEXPRESSION → TERM C

C → T3 C | &

T3 → (‘+’ | ‘-’) TERM

TERM → UNARYEXPR D

D → T4 D | &

T4 → (‘*’ | ‘ \’ | ‘%’) UNARYEXPR

UNARYEXPR → (‘+’ | ‘ -’ | &) FACTOR

FACTOR → (‘int_constant’ | ‘float_constant’ | ‘string_constant’ | ‘null’ | LVALUE | ‘(‘ NUMEXPRESSION ‘)’)

LVALUE → ‘ident’ B

Para chegar nessa gramática, o grupo fez substituições referentes às produções que na gramática BNF constavam como: (expressão)*, (expressão1 | expressão2)? e (expressão)+.

Para transformar as produções do tipo (expressão)* na forma convencional, criamos um loop da seguinte forma:

Considere como a produção original:

VARDECL → (‘int’ | ‘float’ | ‘string’) ‘ident’ (‘[‘ ‘int_constant’ ‘]’)*

Criamos um novo não-terminal para substituir uma parte do corpo da produção que contém o operador *, nesse caso o não-terminal A, e outro não-terminal, T1, para representar a expressão dentro do operador:

VARDECL → (‘int’ | ‘float’ | ‘string’) ‘ident’ A
T1 → ‘[‘ ‘int_constant’ ‘]’

Criamos uma nova produção para o não-terminal A para representar um loop:

A → T1 A | &
T1 → ‘[‘ ‘int_constant’ ‘]’

Para o operador + também utilizamos essa mesma técnica, mas no corpo da produção original colocamos o não-terminal T1 para obrigar uma ocorrência dele antes do loop:

VARDECL → (‘int’ | ‘float’ | ‘string’) ‘ident’ T1 A

Para o operador ? apenas adicionamos uma nova produção para o Épsilon (ε) , como por exemplo:

PROGRAM → (STATEMENT | FUNCLIST)? ⇒
PROGRAM → STATEMENT | FUNCLIST | &

3 RECURSÃO À ESQUERDA

Analisando a gramática ConvCC-2020-1 na forma convencional, não foi observado nenhuma produção do tipo $A \rightarrow^{1} Aa$, sendo assim, esta gramática não possui recursão **direta** à esquerda.

Para analisar se a gramática possui recursão **indireta** à esquerda, o grupo fez a análise de cada não-terminal que estivesse no início de cada produção da expressão de maneira individual. Para isso, cada produção não-terminal que se encontra à esquerda foi substituída. Tendo como exemplo a produção a seguir, a verificação de existência ou não de recursão à esquerda deu-se da seguinte forma:

LVALUE → ‘ident’ B

Primeiramente, é possível ter certeza de que essa produção nunca apresentará uma recursão à esquerda visto que começa com um terminal. Com isso, é possível descartar algumas outras produções que não irão apresentar recursão a esquerda: UNARYEXPR , T4, T3, T2, ALLOCEXPRESSION, FORSTAT, IFSTAT, RETURNSTAT, READSTAT, PRINTSTAT, PARAMLISTCALL, FUNCCALL, T1, VARDECL, PARAMLIST e FUNDEF.

Agora, observando todas as outras produções que começam com os não-terminais que estão presentes na lista acima, visto que eles não possuem recursão à esquerda, também não irão apresentar a recursão à esquerda. São eles: FACTOR, D, TERM, C, B, ATRIBSTAT, A, RETURNSTAT, FUNCLIST.

Mais uma vez, realizando o mesmo procedimento, agora nos utilizando de ambas as listas, temos mais dois não-terminais: NUMEXPRESSION e STATEMENT.

Finalizando, temos os últimos não-terminais: EXPRESSION, STATELIST e PROGRAM.

Com isso, conseguimos mostrar a gramática ConvCC-2020-1 não possui recursões à esquerda diretas e indiretas.

4 FATORAÇÃO À ESQUERDA

Analisando a gramática ConvCC-2020-1 na forma convencional, é possível observar que a gramática não está fatorada à esquerda. Isso se torna visível no produtor FUNCLIST, que pode gerar duas produções que começam com o FUNCDEF, gerando ambiguidade.

FUNCLIST → **FUNCDEF** FUNCLIST | **FUNCDEF**

Podemos fatorá-la facilmente criando uma nova produção FUNCLIST2:

FUNCLIST → FUNCDEF FUNCLIST2

FUNCLIST2→ FUNCLIST | &

Após a fatoração completa da gramática, foi observado que a seguinte produção ficou inalcançável:

FUNCCALL → ‘ident’ ‘(‘ PARAMLISTCALL ‘)’

Uma vez que o não-terminal ATRIBSTAT2, que continha uma produção com o FUNCCALL, precisou ser fatorado novamente devido à uma fatoração indireta com o terminal ‘ident’, o FUNCCALL acabou sendo removido do corpo das produções de ATRIBSTAT2 e não aparecendo nas produções de ATRIBSTAT3. As produções antigas do ATRIBSTAT2 e as resultantes da fatoração do ATRIBSTAT2 se encontram logo abaixo.

Produções antigas:

ATRIBSTAT2 → EXPRESSION | ALLOCEXPRESSION |

FUNCCALL

Novas produções devido à fatoração:

ATRIBSTAT2 → ident ATRIBSTAT3 | ALLOCEXPRESION | …

ATRIBSTAT3 → B D C EXPRESSION2 | ‘(‘ PARAMLISTCALL ‘)’

5 CONVERSÃO PARA LL(1)

A transformação da ConvCC-2020-1 em uma gramática LL(1) apenas necessitou alterar a seguinte produção:

IFSTAT → ‘if’ ‘(‘ EXPRESSION ‘)’ STATEMENT IFSTAT2

Para a seguinte:

IFSTAT → ‘if’ ‘(‘ EXPRESSION ‘)’ ‘{‘ STATELIST ‘}’ IFSTAT2

Essa mudança foi necessária porque a tabela de reconhecimento sintático estava apresentando um conflito na entrada [IFSTAT2, else], uma vez que a interseção de FOLLOW(IFSTAT2) com FIRST(‘else’ STATEMENT) estava dando {‘else’}, sendo que o conjunto FIRST da segunda produção de IFSTAT2 continha ε.

Para resolver esse problema, foi trocado o STATEMENT para ‘{‘ STATELIST ‘}’ no corpo da produção para retirar o ‘else’ do FOLLOW(IFSTAT2) e colocar o terminal ‘{‘ no lugar, tornando a interseção vazia. A tabela de FIRST e FOLLOW se

encontra no arquivo *First_Follow.pdf* e a tabela de reconhecimento sintático no arquivo *Tabela_LL(1).ods*. Caso não seja possível abrir os documentos, os links para acesso estão no arquivo links-de-compartilhamento.txt, localizado no diretório do arquivo enviado.

6 GRAMÁTICA FINAL

PROGRAM → STATEMENT | FUNCLIST | &

FUNCLIST → FUNCDEF FUNCLIST2

FUNCLIST2 → FUNCLIST | &

FUNCDEF → ‘def’ ‘ident’ ‘(‘ PARAMLIST ‘)’ ‘{‘ STATELIST ‘}’

PARAMLIST → ‘int’ ‘ident’ PARAMLIST2 | ‘float’ ‘ident’ PARAMLIST2 | ‘string’ ‘ident’ PARAMLIST2 | &

PARAMLIST2 → ‘,’ PARAMLIST | &

STATEMENT → VARDECL ‘;’ | ATRIBSTAT ‘;’ | PRINTSTAT ‘;’ | READSTAT ‘;’ | RETURNSTAT ‘;’ | IFSTAT | FORSTAT | ‘{‘ STATELIST ’}’ | break ‘;’ | ‘;’

VARDECL → ‘int’ ‘ident’ A | ‘float’ ‘ident’ A | ‘string’ ‘ident’ A

A → T1 A | &

T1 → ‘[‘ ‘int_constant’ ‘]’

ATRIBSTAT → LVALUE ‘=’ ATRIBSTAT2

ATRIBSTAT2→ ‘ident’ ATRIBSTAT3 | ALLOCEXPRESSION | ‘+’ FACTOR | ‘-’ FACTOR | ‘int_constant’ | ‘float_constant’ | ‘string_constant’ | ‘null’ | ‘(‘ NUMEXPRESSION ‘)’

ATRIBSTAT3 → B D C EXPRESSION2 | ‘(‘ PARAMLISTCALL ‘)’

PARAMLISTCALL → ‘ident’ PARAMLISTCALL2 | &

PARAMLISTCALL2 → ‘,’ PARAMLISTCALL | &

PRINTSTAT → ‘print’ EXPRESSION

READSTAT → ‘read’ LVALUE

RETURNSTAT → ‘return’

IFSTAT → ‘if’ ‘(‘ EXPRESSION ‘)’ ‘{‘ STATELIST ‘}’ IFSTAT2

IFSTAT2→ ‘else’ STATEMENT | &

FORSTAT → ‘for’ ‘(‘ ATRIBSTAT ‘;’ EXPRESSION ‘;’ ATRIBSTAT ‘)’ STATEMENT

STATELIST → STATEMENT STATELIST2

STATELIST2→ STATELIST | &

ALLOCEXPRESSION → ‘new’ ALLOCEXPRESSION2

ALLOCEXPRESSION2 → ‘int’ T2 B | ‘float’ T2 B | ‘string’ T2 B

B → T2 B | &

T2 → ‘[‘ NUMEXPRESSION ‘]’

EXPRESSION → NUMEXPRESSION EXPRESSION2

EXPRESSION2 → ‘<’ NUMEXPRESSION | ‘>’ NUMEXPRESSION | ‘<=’ NUMEXPRESSION |

‘>=’ NUMEXPRESSION | ‘==’ NUMEXPRESSION | ‘!=’ NUMEXPRESSION | &

NUMEXPRESSION → TERM C

C → T3 C | &

T3 → ‘+’ TERM | ‘-’ TERM

TERM → UNARYEXPR D

D → T4 D | &

T4 → ‘*’ UNARYEXPR | ‘\’ UNARYEXPR | ‘%’ UNARYEXPR

UNARYEXPR → ‘+’ FACTOR | ‘-’ FACTOR | FACTOR

FACTOR → ‘int_constant’ | ‘float_constant’ | ‘string_constant’ | ‘null’ | LVALUE | ‘(‘ NUMEXPRESSION

‘)’

LVALUE → ‘ident’ B

7 FERRAMENTA UTILIZADA

A ferramenta escolhida pelo grupo foi o ANTLR versão 4, a mesma utilizada para o analisador léxico. O objetivo dessa escolha deu-se pelo fato do desenvolvimento incremental a partir do analisador léxico, e também, de a ferramenta possuir integração com a linguagem escolhida para o desenvolvimento do trabalho, além da possível geração da árvore de análise sintática.

8 ENTRADA/SAÍDA

8.1 ENTRADA

Para utilizar a ferramenta, é preciso que um arquivo de extensão *.g4 seja criado. Nesse arquivo, é necessário que a gramática seja declarada na sintaxe da ferramenta como mostra a imagem.

```
program
  :
    ( statement
    | funclist
    )?
  ;
```

A partir do diretório principal do trabalho, seguindo o caminho /src/main/antlr4, pode-se encontrar o arquivo ConvCC20201.g4, que foi utilizado para a definição da gramática da linguagem CC-2020-1.

Para que o grupo pudesse adaptar a gramática às exigências da ferramenta, no arquivo ConvCC20201.g4, as produções que apresentavam o & precisaram ser escritas com ?, visto que não foi possível encontrar suporte ao & na versão utilizada.

8.2 SAÍDA

Ao executar o programa, a partir do arquivo ConvCC20201.g4, a ferramenta gera como saída duas classes ConvCC20201Lexer.java e ConvCC20201Parser.java**.** A classe ConvCC20201Lexer é a responsável por implementar os métodos referentes ao analisador léxico. Sua responsabilidade é, a partir do código de entrada, fazer a

identificação dos tokens recebidos. Com ela, foi possível a criação de um objeto do tipo CommonTokenStream.

Utilizando essa Stream de tokens, podemos instanciar o parser ConvCC20201Parser. Por fim, devemos dizer qual produção inicial, para então o parser realizar a análise sintática.

```
// Read the file, and start the lexer and parser.
CharStream charStream = CharStreams.fromFileName(filePath);
ConvCC20201Lexer lexer = new ConvCC20201Lexer(charStream);
CommonTokenStream tokens = new CommonTokenStream(lexer);
ConvCC20201Parser parser = new ConvCC20201Parser(tokens);
// Start the parser on the 'program' which is the initial grammar producer.

out.println("===> Starting Syntactic Analysis");
ParseTree tree = parser.program();
```

O ANTRL irá realizar a análise sintática automaticamente, colocando o log de eventuais erros da análise no terminal e abrirá uma janela mostrando a árvore sintática obtida na análise.