UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Relatório - AS

Construção de Analisador Sintático

Artur Ribeiro Alfa [17103919]

Augusto Vieira Coelho Rodrigues [19100517]

Leonardo Vieira Nunes [19102923]

Thainan Vieira Junckes [19100545]

FLORIANÓPOLIS, 19 DE JUNHO DE 2022

1. CC-2022-1 está na forma BNF. Coloque-a na forma convencional de gramática. Chame tal gramática de ConvCC-2022-1.

ConvCC-2022-1:

PROGRAM → STATEMENT | FUNCLIST | &

FUNCLIST → FUNCDEF FUNCLIST | FUNCDEF

FUNCDEF → def ident(PARAMLIST) {STATELIST}

DATATYPE → int | float | string

PARAMLIST → DATATYPE ident, PARAMLIST | DATATYPE ident | &

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

VARDECL → DATATYPE ident OPT_VECTOR

OPT_VECTOR → [int_constant] OPT_VECTOR | &

ATRIB_RIGHT → EXPRESSION | ALOCEXPRESSION | FUNCCALL

ATRIBSTAT → LVALUE = ATRIB_RIGHT

FUNCCALL → ident(PARAMLISTCALL)

PARAMLISTCALL → ident, PARAMLISTCALL | ident | &

PRINTSTAT → print EXPRESSION

READSTAT → read LVVALUE

RETURNSTAST → return

IFSTAT → if (EXPRESSION) STATEMENT OPT_ELSE

OPT_ELSE → (STATEMENT) | &

FORSTAT → for (ATRIBSTAT; EXPRESSION; ATRIBSTAT) STATEMENT

STATELIST → STATEMENT OPT_STATELIST

OPT_STATELIST → STATELIST | &

ALLOCEXPRESSION → new DATATYPE [NUMEXPRESSION] OPT_ALLOC_NUMEXP

OPT_ALLOC_NUMEXP → [NUMEXPRESSION] OPT_ALLOC_NUMEXP | &

REL_OP → < | > | <= | >= | == | !=

NUMEXPRESSION → TERM REC_PLUS_MINUS_TERM

REC_PLUS_MINUS_TERM → PLUS_OR_MINUS TERM REC_PLUS_MINUS_TERM | &

PLUS_OR_MINUS → + | -

TERM → UNARYEXPR REC_UNARYEXPR

REC_UNARYEXPR → UNARYEXPR REC_UNARYEXPR

UNARYEXPR_OP → * | / | %

UNARYEXPR → PLUS_OR_MINUS FACTOR | FACTOR

FACTOR → int constant | float constat | string constat | null | LVALUE | (NUMEXPRESSION)

LVALUE → ident OPT_ALLOC_NUMEXP

2. A sua ConvCC-2022-1 possui recursão à esquerda? Justifique detalhadamente sua resposta.
Se ela tiver recursão à esquerda, então remova tal recursão.

Não possui recursão à esquerda, uma vez que a gramática apresentada acima não possui produções a esquerda capazes de gerar o gerador inicial.
Exemplo: X =*> Xy

3. A sua ConvCC-2022-1 está fatorada à esquerda? Justifique detalhadamente sua resposta.
Se ela não estiver fatorada à esquerda, então fatore.

Não está fatorada, pois temos produções do tipo X => zY | zZ

Exemplo em ConvCC-2022-1:
FUNCLIST → FUNCDEF FUNCLIST | FUNCDEF
Após a fatoração à esquerda, obtemos
FUNCLIST → FUNCDEF FUNCLISTAUX
FUNCLIST AUX → FUNCLIST | &

A seguir a ConvCC-2022-1 fatorada:

PROGRAM → STATEMENT | FUNCLIST | &

FUNCLIST → FUNCDEF FUNCLISTAUX

FUNCLIST AUX → FUNCLIST | &

FUNCDEF → def ident(PARAMLIST) {STATELIST}

PARAMLIST → DATATYPE ident PARAMLISTAUX | &

PARAMLISTAUX → , PARAMLIST | &

DATATYPE → int | float | string

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

VARDECL → DATATYPE ident OPT_VECTOR

OPT_VECTOR → [int_constant] OPT_VECTOR | &

ATRIBSTAT → LVALUE = ATRIB_RIGHT

ATRIB_RIGHT → FUNCCALL_OR_EXPRESSION | ALOCEXPRESSION

FUNCCALL_OR_EXPRESSION →
+ FACTOR REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR |
- FACTOR REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR |
int_constant REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR |
float_constant REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR |
string_constant REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR |
null REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR |
(NUMEXPRESSION) REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR |
ident FOLLOW_IDENT

FOLLOW_IDENT → OPT_ALLOC_NUMEXP REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR | (PARAMLISTCALL)

FUNCCALL → ident(PARAMLISTCALL)

PARAMLISTCALL → ident PARAMLISTCALLAUX | &

PARAMLISTCALLAUX → , PARAMLISTCALL | &

PRINTSTAT → print EXPRESSION

READSTAT → read LVALUE

RETURNSTAT → return

IFSTAT → if (EXPRESSION) {STATELIST} OPT_ELSE

OPT_ELSE → else {STATELIST} | &

FORSTAT → for (ATRIBSTAT; EXPRESSION; ATRIBSTAT) STATEMENT

STATELIST → STATEMENT OPT_STATELIST

OPT_STATELIST → STATELIST | &

ALLOCEXPRESSION → new DATATYPE [NUMEXPRESSION] OPT_ALLOC_NUMEXP

OPT_ALLOC_NUMEXP → [NUMEXPRESSION] OPT_ALLOC_NUMEXP | &

EXPRESSION → NUMEXPRESSION OPT_REL_OP_NUM_EXPR

OPT_REL_OP_NUM_EXPR → REL_OP NUMEXPRESSION | &

REL_OP → < | > | <= | >= | == | !=

NUMEX RESSION → TERM REC_PLUS_MINUS_TERM

REC_P LUS_MINUS_T ERM → PLUS_OR_MINUS TERM REC_PLUS_MINUS_TERM | &

PLUS_OR_MINUS → + | -

TERM → UNARYEXPR REC_UNARYEXPR

REC_UNARYEXPR → UNARYEXPR_OP TERM | &

UNARYEXPR_OP → * | / | %

UNARYEXPR → PLUS_OR_MINUS FACTOR | FACTOR

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

LVALUE → ident OPT_ALLOC_NUMEXP

4. Faça ConvCC-2022-1 ser uma gramática em LL(1). É permitido adicionar novos terminais na gramática, se achar necessário. Depois disso, mostre que ConvCC-2022-1 está em LL(1)
(você pode usar o Teorema ou a tabela de reconhecimento sintático vistos em videoaula).

Pelo teorema, uma gramática está em LL(1) se, e somente se, sempre que tivermos duas produções distintas A-> α | β, os seguintes condições são satisfeitas:

1. First(α) ∩ First(β): ∅
2. Se β =*> &, então First (α) ∩ Follow(A) = ∅

3. Se α =*> &, então First(β) ∩ Follow(A) = ∅

O programa desenvolvido faz a verificação de cada condição, mas abaixo apresentamos os firsts e follows para checagem manual para verificar se ConvCC-2022-1 está em LL(1). Explicação: Se não há nada após o símbolo de igual (=), como por exemplo ‘Follow([) = ‘, é porque não possui follows. First(A) = {‘;’, ‘)’} significa que os firsts de A são os símbolos ‘;’ e ‘)’, sem aspas.

First(OPT_ELSE) = {'&', 'else'}
Follow(OPT_ELSE) = {'read', 'string', 'break', ';', 'return', '}', 'ident', 'float', '{', 'print', 'if', 'for', 'int', '$'}
First(REL_OP) = {'>=', '>', '<', '!=', '==', '<='}
Follow(REL_OP) = {'-', '(', 'null', 'string_constant', 'ident', '+', 'int_constant', 'float_constant'}
First(IFSTAT) = {'if'}
Follow(IFSTAT) = {'read', 'string', 'break', ';', 'return', '}', 'ident', 'float', '{', 'print', 'if', 'for', 'int', '$'}
First(EXPRESSION) = {'-', '(', 'null', 'string_constant', 'ident', '+', 'int_constant', 'float_constant'}
Follow(EXPRESSION) = {';', ')'}
First(OPT_VECTOR) = {'&', '['}
Follow(OPT_VECTOR) = {';'}
First(UNARYEXPR_OP) = {'%', '/', '*'}
Follow(UNARYEXPR_OP) = {'-', '(', 'null', 'string_constant', 'ident', '+', 'int_constant', 'float_constant'}
First(PROGRAM) = {'read', 'string', 'break', ';', 'return', 'ident', 'float', '{', 'print', 'if', '&', 'for', 'int', 'def'}
Follow(PROGRAM) = {'$'}
First(DATATYPE) = {'int', 'float', 'string'}
Follow(DATATYPE) = {'ident', '['}
First(PARAMLISTCALLAUX) = {',', '&'}
Follow(PARAMLISTCALLAUX) = {')'}
First(FACTOR) = {'null', 'string_constant', 'ident', '(', 'int_constant', 'float_constant'}
Follow(FACTOR) = {'%', '/', '>', '-', '!=', ';', '<=', ')', '==', '*', '>=', '<', '+', ']'}
First(FUNCLIST) = {'def'}
Follow(FUNCLIST) = {'$'}
First(ATRIB_RIGHT) = {'new', '(', 'null', 'ident', 'string_constant', 'int_constant', 'float_constant'}
Follow(ATRIB_RIGHT) = {';', ')'}
First(RETURNSTAT) = {'return'}
Follow(RETURNSTAT) = {';'}
First(REC_PLUS_MINUS_TERM) = {'&', '-', '+'}
Follow(REC_PLUS_MINUS_TERM) = {'>', '<=', '!=', ';', '==', '>=', '<', ']', ')'}
First(OPT_REL_OP_NUM_EXPR) = {'&', '>', '!=', '==', '>=', '<', '<='}
Follow(OPT_REL_OP_NUM_EXPR) = {';', ')'}
First(FUNCCALL_OR_EXPRESSION) = {'null', 'string_constant', 'ident', '(', 'int_constant', 'float_constant'}
Follow(FUNCCALL_OR_EXPRESSION) = {';', ')'}
First(PARAMLISTCALL) = {'&', 'ident'}

Follow(PARAMLISTCALL) = {')'}
First(ALLOCEXPRESSION) = {'new'}
Follow(ALLOCEXPRESSION) = {';', ')'}
First(PLUS_OR_MINUS) = {'-', '+'}
Follow(PLUS_OR_MINUS) = {'-', '(', 'null', 'string_constant', 'ident', '+', 'int_constant', 'float_constant'}
First(LVALUE) = {'ident'}
Follow(LVALUE) = {'=', '%', '/', '>', '-', '!=', ';', ')', '<=', '==', '*', '>=', '<', '+', ']'}
First(ATRIBSTAT) = {'ident'}
Follow(ATRIBSTAT) = {';', ')'}
First(PRINTSTAT) = {'print'}
Follow(PRINTSTAT) = {';'}
First(UNARYEXPR) = {'-', '(', 'null', 'string_constant', 'ident', '+', 'int_constant', 'float_constant'}
Follow(UNARYEXPR) = {'%', '/', '>', '-', '!=', ';', ')', '<=', '==', '*', '>=', '<', '+', ']'}
First(STATELIST) = {'read', 'string', 'break', ';', 'return', 'ident', 'float', '{', 'print', 'if', 'for', 'int'}
Follow(STATELIST) = {'}'}
First(REC_UNARYEXPR) = {'&', '%', '/', '*'}
Follow(REC_UNARYEXPR) = {'>', '-', '!=', ';', '<=', ')', '==', '>=', '<', '+', ']'}
First(TERM) = {'-', '(', 'null', 'string_constant', 'ident', '+', 'int_constant', 'float_constant'}
Follow(TERM) = {'>', '-', '!=', ';', '<=', ')', '==', '>=', '<', '+', ']'}
First(STATEMENT) = {'if', 'for', 'read', 'string', 'break', ';', 'return', 'ident', 'int', 'float', '{', 'print'}
Follow(STATEMENT) = {'read', 'string', 'break', ';', 'return', '}', 'ident', 'float', '{', 'print', 'if', 'for', 'int', '$'}
First(PARAMLIST) = {'&', 'int', 'float', 'string'}
Follow(PARAMLIST) = {')'}
First(FUNCDEF) = {'def'}
Follow(FUNCDEF) = {'def', '$'}
First('-') = set()
Follow('-') = {'(', 'null', 'string_constant', 'ident', 'int_constant', 'float_constant'}
First(OPT_STATELIST) = {'read', 'string', 'break', ';', 'return', 'ident', 'float', '{', 'print', 'if', '&', 'for', 'int'}
Follow(OPT_STATELIST) = {'}'}
First(NUMEXPRESSION) = {'-', '(', 'null', 'string_constant', 'ident', '+', 'int_constant', 'float_constant'}
Follow(NUMEXPRESSION) = {'>', '<=', '!=', ';', '==', '>=', '<', ']', ')'}
First(OPT_ALLOC_NUMEXP) = {'&', '['}
Follow(OPT_ALLOC_NUMEXP) = {'=', '%', '/', '>', '-', '!=', ';', ')', '<=', '==', '*', '>=', '<', '+', ']'}
First(VARDECL) = {'int', 'float', 'string'}
Follow(VARDECL) = {';'}
First(READSTAT) = {'read'}
Follow(READSTAT) = {';'}
First(FORSTAT) = {'for'}
Follow(FORSTAT) = {'read', 'string', 'break', ';', 'return', '}', 'ident', 'float', '{', 'print', 'if', 'for', 'int', '$'}
First(FOLLOW_IDENT) = {'%', '/', '>', '-', '!=', '<=', '&', '[', '(', '==', '*', '>=', '<', '+'}
Follow(FOLLOW_IDENT) = {';', ')'}
First(PARAMLISTAUX) = {',', '&'}
Follow(PARAMLISTAUX) = {')'}

First(FUNCLISTAUX) = {'&', 'def'}
Follow(FUNCLISTAUX) = {'$'}
First('+') = set()
Follow('+') = {'(', 'null', 'string_constant', 'ident', 'int_constant', 'float_constant'}
First(%) = {'%'}
Follow(%) =
First(/) = {'/'}
Follow(/) =
First(-) = {'-'}
Follow(-) =
First(string) = {'string'}
Follow(string) =
First(return) = {'return'}
Follow(return) =
First(!=) = {'!='}
Follow(!=) =
First(null) = {'null'}
Follow(null) =
First(float) = {'float'}
Follow(float) =
First({) = {'{'}
Follow({) =
First()) = {')'}
Follow()) =
First(<=) = {'<='}
Follow(<=) =
First(if) = {'if'}
Follow(if) =
First(for) = {'for'}
Follow(for) =
First(string_constant) = {'string_constant'}
Follow(string_constant) =
First(int) = {'int'}
Follow(int) =
First(,) = {','}
Follow(,) =
First(def) = {'def'}
Follow(def) =
First(=) = {'='}
Follow(=) =
First(new) = {'new'}
Follow(new) =
First(>) = {'>'}
Follow(>) =
First(read) = {'read'}
Follow(read) =
First(break) = {'break'}
Follow(break) =

First(;) = {';'}
Follow(;) =
First(}) = {'}'}
Follow(}) =
First(ident) = {'ident'}
Follow(ident) =
First(int_constant) = {'int_constant'}
Follow(int_constant) =
First(print) = {'print'}
Follow(print) =
First(else) = {'else'}
Follow(else) =
First([) = {'['}
Follow([) =
First(() = {'('}
Follow(() =
First(==) = {'=='}
Follow(==) =
First(*) = {'*'}
Follow(*) =
First(>=) = {'>='}
Follow(>=) =
First(<) = {'<'}
Follow(<) =
First(+) = {'+'}
Follow(+) =
First(]) = {']'}
Follow(]) =
First(float_constant) = {'float_constant'}
Follow(float_constant) =
First(&) = {'&'}
Follow(&) = {'%', '/', '-', 'string', '!=', 'return', 'float', '{', ')', '<=', 'if', 'for', 'int', '$', '=', '>', 'read', 'break', ';', '}', 'ident', 'print', '==', '*', '>=', '<', '+', ']'}

5. se não usou ferramenta, uma descrição da implementação do analisador sintático (Usou uma tabela de reconhecimento sintático para gramáticas em LL(1)? Se não, então o que foi usado?)

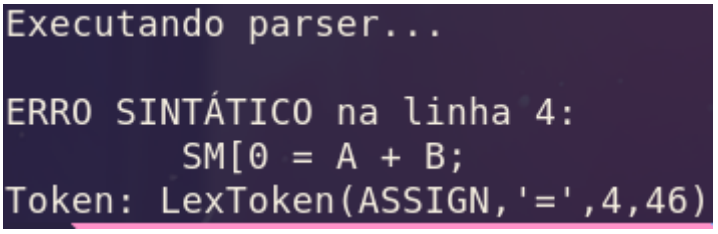
O arquivo src/utils/cfg_processor.py implementa a geração de first e follow, testa se a gramática é LL(1) por meio dos teoremas vistos em aula e por fim gera uma tabela de análise que é passada para o analisador sintático (src/parser.py). Além dos tokens, o analisador sintático recebe a gramática ConvCC-2022-1 como entrada.

Após obter a tabela de análise, ocorre a análise sintática de fato, feita na função parse().

Saída do programa:

1. Se tiver erro léxico, é apresentado na tela os erros léxicos e o programa é abortado;

2. Se não houver erro léxico, é apresentado a tabela de tokens e a tabela de símbolos, e vamos realizar a análise sintática;
3. Se houver erro sintático, é apresentado a linha e o respectivo trecho de código que pode ter causado o erro. Na ausência de um token, o programa pode apontar para o próximo token, e na ausência de um ponto e vírgula, o programa pode apresentar a próxima linha;



   Exemplo: Ausência de um ‘]’ após o ‘0’ ocasionando um erro sintático.
4. Se não houver erro sintático, é apresentada uma mensagem de sucesso.