Tradutores trabalho 1

NUR COREZZI[1]

[1] *Departamento de Ciência da Computação, Campus Darcy Ribeiro, Asa Norte, Brasília - DF, 70910-900*

Compiled March 19, 2020

Escolha do Tema

1. INTRODUÇÃO

Neste trabalho será apresentada a especificação de uma linguagem a ser desenvolvida que se destina á área de processamento de grafos. A mesma tem como objetivo fornecer funcionalidades básicas de uma linguagem declarativa como comandos iterativos *for* e *while*, condicionais *if* , tratamento de expressões booleanas e aritméticas e também algumas funcionalidades orientadas as grafos. Seu diferencial está na introdução de um tipo básico *graph* que possui uma interface básica para inserção de arestas direcionais sem peso e também um comando *for* facilitador que permite iterar fazendo percorrimentos *DFS* (Depth First Search) e *BFS* (Breadth First Search). Este tipo básico poupa a necessidade do programador manter e cuidar do uso de sua própria estrutura e ter que carrega-lá a todo momento que decidir iniciar um novo programa, permitindo que algum comportamento seja aplicado de forma prática a topologia de um grafo especificado.

2. GRAMÁTICA

A. Definição

⟨*program*⟩ ::= ⟨*program*⟩ ⟨*function*⟩ | ''

⟨*function*⟩ ::= ⟨*type*⟩ ⟨*id*⟩ '(' ⟨*opt_params*⟩ ')' '{' ⟨*statements*⟩ '}'

⟨*type*⟩ ::= 'boolean' | 'int' | 'float' | 'void' | 'graph' '[' ⟨*number_int*⟩ ']'

⟨*opt_params*⟩ ::= '' | ⟨*params*⟩ ⟨*params*⟩ ::= ⟨*params*⟩ ',' ⟨*declaration*⟩ | ⟨*declaration*⟩

⟨*statements*⟩ ::= '' | ⟨*statements*⟩ ⟨*statement*⟩ ⟨*statement*⟩ ::= ⟨*declaration_and_atrib*⟩ ';'
| ⟨*expr_decl*⟩ ';'
| ⟨*for*⟩
| ⟨*while*⟩
| ⟨*if*⟩
| 'return' ⟨*expr_decl*⟩ ';'
| ⟨*read*⟩
| ⟨*write*⟩
| ⟨*id*⟩ '.' ⟨*function_call*⟩ ';'
| '{' ⟨*statements*⟩ '}'

⟨*for*⟩ ::= 'for' '(' ⟨*declaration_and_atrib*⟩ ';' ⟨*expr_decl*⟩ ';' ⟨*expr_decl*⟩ ')' ⟨*statement*⟩
| 'for' '(' ⟨*declaration*⟩ ':' ⟨*graph_op*⟩ ')' ⟨*statement*⟩

⟨*while*⟩ ::= 'while' '(' ⟨*expr_decl*⟩ ')' ⟨*statement*⟩

⟨*if*⟩ ::= 'if' '(' ⟨*expr_decl*⟩ ')' ⟨*statement*⟩

⟨*read*⟩ ::= '»' ⟨*value*⟩ ';'

⟨*write*⟩ ::= '«' ⟨*value*⟩ ';'

⟨*function_call*⟩ ::= ⟨*id*⟩ '(' ⟨*opt_params_call*⟩ ')'

⟨*opt_params_call*⟩ ::= '' | ⟨*params_call*⟩

⟨*params_call*⟩ ::= ⟨*params_call*⟩ ',' ⟨*value*⟩ | ⟨*value*⟩

⟨*graph_op*⟩ ::= 'dfs' '(' ⟨*id*⟩ ',' ⟨*id*⟩ ')'
| 'bfs' '(' ⟨*id*⟩ ',' ⟨*id*⟩ ')'

⟨*declaration_and_atrib*⟩ ::= ⟨*declaration*⟩
| ⟨*declaration*⟩ '=' ⟨*expr_decl*⟩

⟨*declaration*⟩ ::= ⟨*type*⟩ ⟨*id*⟩

⟨*expr_decl*⟩ ::= ⟨*id*⟩ '=' ⟨*expr_relational*⟩ | ⟨*expr_relational*⟩

⟨*expr_relational*⟩ ::= ⟨*expr_relational*⟩ ⟨*compare_op*⟩ ⟨*expr_and*⟩ | ⟨*expr_and*⟩

⟨*expr_and*⟩ ::= ⟨*expr_and*⟩ '&&' ⟨*expr_or*⟩ | ⟨*expr_or*⟩

⟨*expr_or*⟩ ::= ⟨*expr_or*⟩ '||' ⟨*expr_add*⟩ | ⟨*expr_add*⟩

⟨*expr_add*⟩ ::= ⟨*expr_add*⟩ ⟨*add_op*⟩ ⟨*expr_mult*⟩ | ⟨*expr_mult*⟩

⟨*expr_mult*⟩ ::= ⟨*expr_mult*⟩ ⟨*mul_op*⟩ ⟨*expr_not*⟩ | ⟨*expr_not*⟩

⟨*expr_not*⟩ ::= '!' ⟨*factor*⟩ | ⟨*factor*⟩

⟨*factor*⟩ ::= '(' ⟨*expr_decl*⟩ ')'
| ⟨*function_call*⟩
| ⟨*value*⟩

⟨*value*⟩ ::= ⟨*id*⟩ | ⟨*number*⟩

⟨*compare_op*⟩ ::= '<' | '<=' | '>=' | '=='

⟨*add_op*⟩ ::= '*' | '/'

⟨*mul_op*⟩ ::= '+' | '-'

⟨*number*⟩ ::= ⟨*number_int*⟩ | ⟨*number_list*⟩ '.' ⟨*digit*⟩ ⟨*number_list*⟩

⟨*number_int*⟩ ::= ⟨*digit*⟩ ⟨*number_list*⟩

⟨*number_list*⟩ ::= ⟨*number_list*⟩ ⟨*digit*⟩ | ''

⟨*digit*⟩ ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

⟨*id*⟩ ::= ⟨*letter*⟩ ⟨*letter_list*⟩ ⟨*number_list*⟩

⟨*letter_list*⟩ ::= ⟨*letter_list*⟩ letter | ''

⟨*letter*⟩ ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' |
'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
| 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n'
| 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | '_'

B. Aspectos informais da gramática

A gramática em questão define uma lista de funções como um programa válido, tendo ponto de execução inicial em alguma função específica, cada função possui um conjunto de argumentos que são declarações que envolvem um tipo e um identificador e possui também um tipo de retorno. Serão também delimitadas por um conjunto de abre e fecha chaves *{}* que possui uma lista de declarações.

Uma declaração constitui um comando da linguagem que podem ser qualquer um dos especificados em <*statements*>. Existe também um comando diferenciado de *for* para o percorrimento dos grafos que contempla também chamadas de funções *BFS* e *DFS* na regra <*graph_op*>.

Os comandos de *for*, *while* e *if* contemplam expressões que avaliaram os fluxos de execução e também uma regra <*statement*> que permite encadear uma lista de comandos a sua execução.

Expressões podem ser consideradas como atribuições, operações lógicas e aritméticas que seguem uma precedência nessa mesma ordem. Existe também a possibilidade de definir prioridades distintas a partir do uso de parêntesis definidos na regra <*factor*>.

Os tipos básicos da linguagem contemplam *boolean*, *int*, *float*, *void graph*. O tipo graph é um tipo especial que também é unicamente identificado pelo seu tamanho especificado entre colchetes *[<number_int>]*.

Por último os numerais da linguagem são definidos pelos terminais numéricos como listas que definem inteiros e reais e os identificadores são definidos pelos carácteres alfabéticos minúculos e maiúsculos além do underline.

3. DESCRIÇÃO SEMÂNTICA

1. O ponto inicial de execução será sempre a função que possui o identificador de main e sua falta implica em não execução.

2. Parâmetros serão passados sempre por cópia e atribuições também.

3. Cópia de variáveis do tipo *graph* apenas serão possíveis para aqueles que possuem um mesmo tamanho especificado entre parêntesis.

4. Cast implícito apenas será possível *int* e *float* portanto expressões lógicas não se misturam com aritméticas.

5. expressões de atribuição possuem um valor de retorno que será o do próprio identificador sendo atribuido.

A. Dificuldades da implementação

A implementação da nova feature talvez não seja tão complexa por parte do analisador sintático, porém adiciona complexidade na verificação semântica pois será necessário verificar atribuições entre grafos se atentando a definição para o que seria algo válido neste caso. Outro problema pode se apresentar no momento de gerar um código intermediário uma vez que um grafo pode ser visto como uma lista de listas ou uma matriz bidimensional que pode não ser diretamente implementavel na linguagem de destino.

REFERENCES

1. jutta, “Ansi c grammar, lex specification,” http://www.quut.com/c/ANSI-C-grammar-l-2011.html. Accessed: 2020-03-17.
2. jutta, “Ansi c yacc grammar,” http://www.quut.com/c/ANSI-C-grammar-y-2011.html. Accessed: 2020-03-17.
3. A. V. Aho, *Compilers: Principles, Techniques, and Tools* (Addison Wesley, 1986).