**Projeto Prático de Compiladores I -- 2019**

**microC > Grafos de Fluxo de Controle em DOT, v 1.0**

**Universidade Federal do Amazonas – Instituto de Computação**

**Marco Cristo**

**1. Introdução**

A linguagem *uC* é uma versão da linguagem C usada ao longo do nosso curso de compiladores. A linguagem *DOT* é uma linguagem de descrição de grafos suportada pelo pacote GraphViz. Neste trabalho, você deve completar a construção de um compilador que traduz códigos dados em uma versão simplificada da linguagem microC para grafos de fluxo de controle (CFGs) expressos na linguagem DOT, como ilustra a figura a seguir.

|  |  |
| --- | --- |
| **microC** | **DOT** |
| /\* teste \*/  void main() {  int Y;  if (X() == 1) {  if (Z() == 1)  f(U);  } else  while (W() == 1)  if (V() == 1)  return;  f(Y);  } | digraph {  outputorder=edgesfirst;  1[shape=ellipse, fillcolor=salmon, style=filled, label="start"];  2[shape=ellipse, fillcolor=palegreen2, style=filled, label="int Y"];  3[shape=ellipse, fillcolor=lightyellow, style=filled, label="if (X() == 1)"];  4[shape=ellipse, fillcolor=lightyellow, style=filled, label="if (Z() == 1)"];  5[shape=ellipse, fillcolor=lightyellow, style=filled, label="f(U)"];  7[shape=ellipse, fillcolor=lightyellow, style=filled, label="while (W() == 1)"];  8[shape=ellipse, fillcolor=lightyellow, style=filled, label="if (V() == 1)"];  9[shape=ellipse, fillcolor=lightyellow, style=filled, label="return"];  10[shape=ellipse, fillcolor=salmon, style=filled, label="stop"];  13[shape=ellipse, fillcolor=lightyellow, style=filled, label="f(Y)"];  1 -> 2[style = ""];  2 -> 3[style = ""];  3 -> 4[style = "", label="T"];  3 -> 7[style = "", label="F"];  4 -> 13[style = "", label="F"];  4 -> 5[style = "", label="T"];  5 -> 13[style = ""];  7 -> 13[style = "", label="F"];  7 -> 8[style = "", label="T"];  8 -> 7[style = "", label="F"];  8 -> 9[style = "", label="T"];  9 -> 10[style = ""];  13 -> 10[style = ""];  } |
| **GFC** |
|  |

Há vasta documentação sobre a linguagem DOT na Internet, além de ferramentas online que podem ser úteis no aprendizado da linguagem[[1]](#footnote-1). A gramática da linguagem uC usada neste trabalho é fornecida no Anexo I, neste documento.

**2. Objetivo**

Fornecer uma (I) gramática LL1 para a linguagem microC, usando COCO/R, de forma que fontes em uC sejam traduzidos para *Grafos de Fluxo de Controle* expressosem *DOT*. Todos os códigos fontes irão respeitar uma versão limitada da linguagem microC, cuja gramática é fornecida no Anexo I. Junto com a gramática (e códigos Java auxiliares que tenham sido escritos), devem ser fornecidos os (II) códigos *Compile.java*, *Parser.java* e *Scanner.java*, além de (III) um relatório que descreva (a) como o seu compilador deve ser construído (*building*) e executado e (b) dê exemplos de códigos uC e CFGs correspondentes em DOT.

**3. Avaliação**

A avaliação será feita por meio da análise da gramática fornecida, dos códigos gerados em DOT e das imagens geradas a partir desses códigos. Seu compilador será testado tanto para os fontes de teste fornecidos quanto para os fontes *não* fornecidos, mas com sintaxe coerente com a gramática microC dada no Anexo I. Os fontes de teste em microC (extensão .uc) serão disponibilizados junto ao enunciado do trabalho no site da disciplina.

**4. Observações**

* Trabalho a ser realizado por equipes de, no máximo, duas pessoas.
* Plágio não será tolerado com anulação do trabalho para as equipes envolvidas.
* O aprendizado de DOT e de ferramentas para gerar imagens (ex: PNG) a partir do DOT faz parte das atribuições necessárias para se completar este trabalho.
* Junto com o enunciado também é fornecida a classe Graph.java que pode ser usada para criação de grafos de adjacência. Operações sobre grafos necessárias para construção de GFCs (por exemplo, a remoção de um nó no grafo com preservação de caminhos) já estão implementadas nesta classe. Contudo, se preferir, você pode usar outras bibliotecas de grafos para Java, como a JGraphT. Uma vantagem da JGraphT é a sua capacidade nativa de gerar DOT, além da sua extensa documentação.

**Anexo I – Gramática da linguagem *microC***

Todos os fontes usados neste trabalho seguem a gramática microC dada a seguir.

/\* microc – versão reduzida para trabalho prático \*/

COMPILER microc

CHARACTERS

semAspas = ANY - '"'.

letra = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ\_".

digito = "0123456789".

TOKENS

id = letra {letra | digito}.

strConst = '"' {semAspas} '"'.

num = ['-'] digito { digito } .

COMMENTS FROM "/\*" TO "\*/" NESTED

IGNORE '\r' + '\n' + '\t'

PRODUCTIONS

microc = {DeclConst} {Definicao} .

DeclConst = "const" Tipo id "=" num ";" .

Definicao = ( Tipo | "void" ) DesigI ( DeclVar | DeclFuncao ) .

DeclVar = {"," DesigI } ";" .

DeclFuncao = "(" [ Tipo DesigI {"," Tipo DesigI } ] ")" CBlock .

Tipo = "int" .

DesigI = id .

CBlock = "{" {Definicao | Instrucao} "}" .

Instrucao = Designador ( Atrib | Parametros ) ";"

| While | For | IfElse | Return | Printf | Scanf | CBlock | ";".

Atrib = "=" Expr .

While = "while" "(" Condicao ")" Instrucao .

For = "for" "(" Designador Atrib ";" Condicao";" Designador Atrib ")" Instrucao .

IfElse = "if" "(" Condicao ")" Instrucao ("else" Instrucao | ) .

Return = "return" (Expr | ) ";".

Printf = "printf" "(" ( strConst | Expr ) {"," ( strConst | Expr ) } ")" ";" .

Scanf = "scanf" "(" Designador ";" .

Parametros = "(" [ Expr {"," Expr } ] ")" .

Condicao = Expr OpRel Expr .

OpRel = "==" | "!=" | ">" | ">=" | "<" | "<=" .

Expr = ( Termo | "-" Termo ) {( "+" | "-" ) Termo } .

Termo = Fator {( "\*" | "/" | "%" ) Fator } .

Fator = Designador [ Parametros ] | num | "(" Expr ")" .

Designador = id .

END microc.

Entre as principais diferenças para o microC dado em sala de aula destacamos a ausência de vetores, instruções *repeat* e *switch*. Embora a gramática suporte funções além do *main*(), constantes e variáveis globais, nenhum dos fontes dados devem ter tais construtores.

1. http://sandbox.kidstrythisathome.com/erdos/ [↑](#footnote-ref-1)