Engenharia Gramátical (1º ano do MEI)

Grafos na Análise e Interpretação de Código Fonte

Relatório de Desenvolvimento

João Pereira	Luís Vieira	Pedro Barbosa
PG47325	PG47430	PG47577

30 de maio de 2022

Resumo

O presente trabalho prático, realizado no âmbito da unidade curricular de Engenharia Gramatical inserida no perfil de Engenharia de Linguagens, visa o desenvolvimento de grafos de análise e interpretação de código para uma evolução de uma Linguagem de Programação Imperativa Simples (LPIS), definida no segundo trabalho prático da unidade curricular de Processamento de Linguagens, como continuação do segundo trabalho prático desenvolvido nesta unidade curricular.

Este irá começar com uma introdução e contextualização sobre o tema. De seguida, o mesmo irá ser aprofundado através da sua caracterização, descrição e identificação dos seus principais requisitos, será feita uma breve aborgagem a grafos assim como discutida a conceção e desenho da resolução. Seguidamente, será abordada a construção de ambos os grafos de CGD e SGG. Finalmente, serão apresentados alguns casos de teste realizados e debatidos os resultados obtidos através dos mesmos.

Conteúdo

1 Introdução **2**

2 Análise e Especificação **4**

2.1 Descrição informal do problema 4

2.2 Especificação de Requisitos 4

2.2.1 Módulos para construir grafos 4

2.2.2 Codificação para construir grafos 5

2.2.3 Armazenamento dos grafos em ficheiros 5

2.2.4 Visualização dos grafos 5

3 Concepção/Desenho da Resolução **6**

3.1 Desenho do CFG 6

3.2 Desenho do SGD 8

3.3 Variáveis utilizadas 9

4 Codificação **11**

4.1 Construção do CGD 11

4.2 Construção do SGD 13

4.3 Persistência do código dot em ficheiros 13

4.4 Adição das imagens dos grafos na página html 14

5 Testes e resultados obtidos **15**

5.1 Teste 1 - Várias instruções simples 15

5.2 Teste 2 - Inclusão da instrução condicional 18

5.3 Teste 3 - Ciclo for 21

5.4 Teste 4 - Ciclo repeat 24

5.5 Teste 5 - Mistura de tudo 26

6 Conclusão **29**

A Código do Programa **30**

Capítulo 1

Introdução

O desenvolvimento do tema proposto insere-se na continuação do estudo feito no contexto do segundo trabalho prático. Este consistiu em investigar ferramentas avançadas de análise de código cujo propósito era: desenvolver um Analisador de Código para uma evolução da sua Linguagem de Programação Imperativa Simples (LPIS) de forma a pôr em prática o conhecimento obtido nas aulas teórico-práticas sobre a utilização do *Parser* e dos *Visitors* do módulo de geração de processadores de linguagens *Lark.Interpreter*.

Desta forma, no que concerne ao terceiro trabalho prático, o seu principal objetivo consiste em estudar o comportamento dos programas-fonte com base na construção dos vários DAG (*Directed Acyclic Graph*) que se usam para estudar o fluxo da execução (controlo), CFG, e dos dados (em função da dependência entre as variáveis, DDG, bem como o SDG que permite perceber o fluxo de execução e de dados ao longo das chamadas a funções.

Posto isto, o problema apresentado compreende as seguintes funcionalidades:

a) Criar e representar o CFG para as diferentes estruturas cíclicas;

b) Criar e representar o CFG para a estrutura condicional *if-else*;

c) Criar e representar o CFG para as instruções de declaração, atribuição e *input/output*;

d) Criar e representar o SGD "*lite*", que apenas em consideração o controlo de fluxo (ignorando o fluxo dos dados).

Assim sendo, e tendo em consideração todos os tópicos acima abordados, este relatório visa ajudar a compreender e explicar todos os raciocínios e tomadas de decisão feitas de forma a conseguir realizar todas as tarefas pedidas neste projeto e, assim, atingir o resultado final desejado.

Estrutura do Relatório

Este relatório inicia-se no capítulo 1 onde é feita uma contextualização e enquadramento do tema em estudo e uma explicação do problema a ser tratado e os pressupostos a si inerentes.

Segue-se do capítulo 2 onde é feita uma descrição do problema sobre o qual o projeto assenta e uma posterior análise e planeamento detalhados com o intuito de identificar e especificar os requisitos necessários para o desenvolvimento da ferramenta, sendo eles entradas, resultados e formas de transformação.

Posteriormente, no capítulo 3 será explicada a concepção idealizada para a resolução do problema anteriormente mencionado abordando os diferentes desenhos do CFG, SGD e também as variáveis utilizadas.

Em seguida, no capítulo 4 serão debatidas as principais tomadas de decisão efetuadas bem como possíveis alternativas para as mesmas e alguns problemas enfrentados durante a implementação da solução.

Seguidamente, no capítulo 5 serão apresentados os testes efetuados e os respetivos resultados obtidos

Finalmente o relatório termina com o capítulo 6 onde é feita uma síntese do documento, uma análise crítica generalizada do trabalho e dos resultados obtidos e uma reflexão sobre o trabalho futuro.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O principal desafio que se pretende enfrentar ao desenvolver este trabalho prático é melhorar o analisador estático criado, permitindo o estudo do comportamento dos programas-fonte através da construção de vários DAG. Estes, por sua vez, permitem estudar o fluxo da execução (CFG), dos dados (DDG), em função das dependências de variáveis e o fluxo de execução e de dados ao longo das chamadas das funções (SGD). Estes grafos poderão ser visualizados, mais tarde, nas páginas HTML geradas.

2.2 Especificação de Requisitos

Tendo em conta as diferentes fases possíveis de desenvolvimento deste projeto, este capítulo surge relacionado especialmente com a fase de planeamento e análise do problema em causa. Sendo assim, este incidirá nos requisitos fundamentais para solucionar o problema. Visto que no trabalho prático 2 a equipa já implementou uma ferramenta de análise estática de código, os requisitos levantados serão os mesmos desse projeto, e ainda existirão novos requisitos face ao que é pedido no enunciado do presente trabalho prático.

Desta forma, os novos requisitos identificados para a realização deste projeto são resumidos em 4 categorias, identificadas nas próximas subsecções.

2.2.1 Módulos para construir grafos

Sabe-se que este trabalho prático incide principalmente na construção e apresentação de *Directed Acyclic Graphs* (DAG's), construídos com base na análise estática do código fonte. De acordo com este facto a equipa considera que a utilização de um ou mais módulos do *Python* que nos permitam construir grafos, gerar ficheiros de output, e gerar imagens dos mesmos, é essencial para a realização deste trabalho.

Dois módulos exemplo que nos permitem realizar tais tarefas são o **Graphviz** e o **Pydot**. O primeiro facilita a criação e renderização de descrições de grafos no formato dot. A sua utilização permite criar nós, arestas e retornar a respetiva descrição em dot para um ficheiro. O segundo

funciona em conjunto com o primeiro, sendo que o Pydot funciona como uma interface gráfica do Graphviz e permite gerar imagens a partir dos ficheiros dot.

Depois de os identificarmos como requisito base para o desenvolvimento deste trabalho prático, acabamos por os incluir na nossa solução.

2.2.2 Codificação para construir grafos

Depois de conhecermos uma forma de criar grafos, representá-los em dot, e gerar imagens a partir dessa representação, é necessário codificar todo esse processo. Desta forma são identificadas várias tarefas necessárias na codificação:

- Na visita feita aos nós da árvore gerada pelo Parser do módulo Lark (já implementado no TP2), deve ser incluída a codificação que permita armazenar o *statement* correspondente a cada instrução do código fonte;
- Nessa mesma visita deve ser codificada a criação de nós para cada *statement*, bem como para a criação das arestas existentes entre os nós;
- Durante as visitas aos nós da árvore devem ser gerados dois grafos diferentes, um *Control Flow Graph* (CFG) e um *System Dependency Graph*(SDG) *lite*, tendo em conta apenas o controlo de fluxo.

2.2.3 Armazenamento dos grafos em ficheiros

No trabalho prático 2 eram criados ficheiros html para apresentar os resultados da análise estática. No novo trabalho prático 3, surgem novos requisitos relacionados com a criação de ficheiros, sendo que deverão ser criados ficheiros adicionais com a extensão .dot, de forma a armazenar a representação dos grafos.

2.2.4 Visualização dos grafos

Nos resultados da análise estática, deve existir também alguma forma de apresentar os grafos obtidos, adicionando à página html, criada no TP2, os respetivos grafos e algumas informações adicionais acerca dos mesmos.

Capítulo 3

Concepção/Desenho da Resolução

Com o objetivo de iniciar a concepção da resolução, criamos alguns grafos no papel e verificámos quais nodos deveriam existir nos grafos, e de que forma é que eram construídos. Desta forma seria mais fácil iniciar a codificação. Nesta etapa tomamos várias decisões, relativamente ao CFG e ao SDG.

3.1 Desenho do CFG

Em primeiro lugar, enfrentamos o problema do **CFG** e portanto identificamos quais os nós que deveriam pertencer ao mesmo.

Estrutura geral do grafo CFG

A figura 3.1 apresenta a estrutura geral do nosso grafo de controlo de fluxo, sendo que os nós constituintes incidem no nó de início, fim e um conjunto de nós associados aos *statements* ou instruções que surgem no código. Começamos por considerar os *statements* mais elementares como declaração de variável, atribuição, *print* e *input*. É de notar que cada um destes nós surgem ligados pela ordem em que aparecem no código. O *statement1* corresponde a uma instrução elementar ou uma declaração que aparece antes do *statement2* no código e assim sucessivamente.

Estrutura do CFG quando existe uma condição

Quando existe uma condição no código, decidimos que o nó da condição deveria ter um aspeto diferente para o sinalizar. Adicionalmente criamos dois nós, o *then* e o *else* para indicar o fluxo do programa quando a condição é verdadeira ou quando é falsa, respetivamente. Além desses dois nós incluímos um nó que marca o fim da estrutura condicional. É de notar que o nó *else* só surge se existir realmente um *else* no código, caso contrário o nó da condição pode saltar diretamente para o nó que marca o fim da condição. Na figura 3.2 apresenta-se um exemplo do fluxo quando existe um *else*.

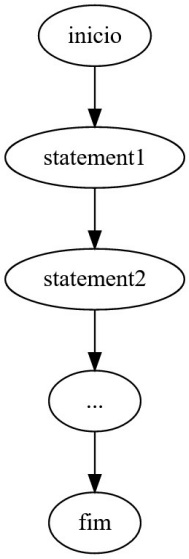


Figura 3.1: Estrutura geral de um CFG

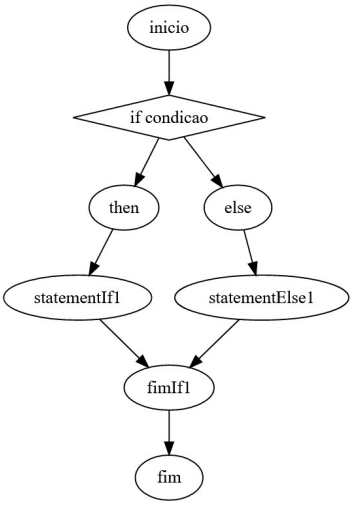


Figura 3.2: Estrutura do CFG com condicional

Estrutura do CFG quando existe um ciclo *while* ou *repeat*

Quando estamos face a uma situação em que surge um *while* ou um *repeat* no código, a estrutura do grafo será igual nos dois casos. Aqui também é criado um novo nó que marca o fim do ciclo, e um nó que representa o próprio ciclo com a respetiva condição. A única diferença entre os dois ciclos é o seu nome e o tipo da condição, já que o *repeat* só aceita como condição um número, correspondente ao número de vezes que o ciclo será executado. Na figura 3.3 está representado este cenário.

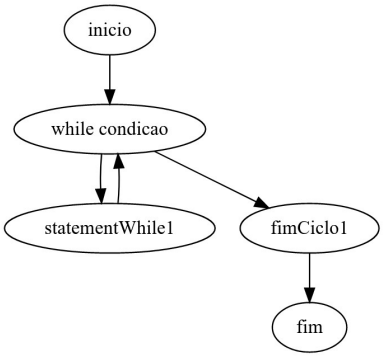


Figura 3.3: Estrutura do CFG com ciclo *while* ou *repeat*

Estrutura do CFG quando existe um ciclo *for*

Por fim, tratamos do cenário do ciclo *for*, que exigiu uma atenção especial visto que este possui uma atribuição antes de iniciar o ciclo, e uma instrução adicional de incrementação em cada iteração do ciclo, portanto a sua estrutura será algo como o que é apresentado na figura 3.4.

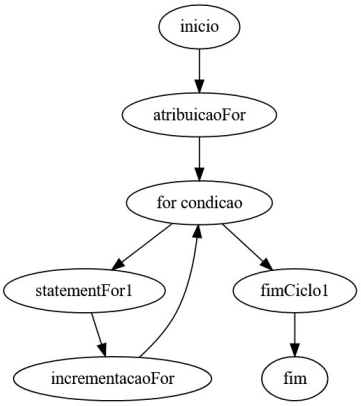


Figura 3.4: Estrutura do CFG com ciclo *for*

3.2 Desenho do SGD

Depois de pensar na resolução do problema para o CFG, passámos para o desenho da solução do **SDG**. Este grafo necessita de um nó especial de entrada, com a designação *entry MAIN*, posteriormente são adicionados ao mesmo nível todos os *statements* principais da esquerda para a direita, de acordo com a sua ordem de aparecimento no código. Se durante a análise estática surgir um *statement* que seja uma estrutura condicional ou cíclica, então serão escritos todos os nós correspondentes às instruções que estão dentro dessa estrutura especial. Só depois é que serão escritos os restantes *statements* principais, à direita da estrutura especial, e ao mesmo nível da mesma. Desta forma, o tipo de visita à árvore utilizado para criar o SDG é a visita em profundidade (**Depth-First Search**).

Nas condições e nos ciclos deixam de existir os nós que marcam o fim da estrutura, e todos os *statements* inseridos nessa estrutura serão escritos também da esquerda para a direita, todos ao mesmo nível. A figura 3.5 apresenta um exemplo de um possível SGD.

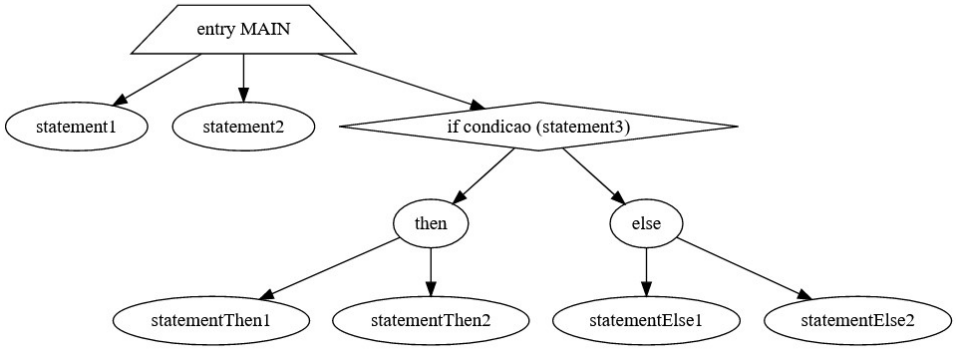


Figura 3.5: Estrutura exemplo do SGD

3.3 Variáveis utilizadas

O próximo passo para a concepção da resolução foi acrescentar ao programa variáveis necessárias para a construção dos grafos. Desta forma adicionamos à função __init__ da nossa classe um conjunto de novas **variáveis de instância**:

- **self.dot**: Instância da classe *Digraph*, pertencente ao módulo **Graphviz**. Representa o grafo CFG que irá ser criado ao longo do código. A partir desta instância conseguimos criar novos nós, bem como mudar as suas formas, e criar arestas;
- **self.dotSdg**: Variável do mesmo tipo da anterior, mas que representa o grafo SDG;
- **self.nodeStatement**: Esta variável é uma string que armazena o texto de cada *statement* do **CFG**, ou seja, o texto que irá ser inserido no seu nó correspondente;
- **self.lastStatement**: Também é uma string, mas esta armazena o id do último nó criado no **CFG**. É através desta variável que conseguimos criar uma aresta entre o nó anterior e o nó atual;
- **self.statementCount**: Esta variável é um inteiro. Este é utilizado para **gerar ids novos** para cada nó do **CFG**. O primeiro nó será o nó 0, e o seu id corresponderá a 0 mas em formato de string. Depois do primeiro nó ser criado este inteiro é incrementado para os ids nunca coincidirem. O mesmo é feito para os nós sucessivos;
- **self.edgeCountCfg**: Inteiro que conta o número de arestas presentes no grafo CFG. É unicamente utilizada para calcular a **complexidade de McCabe**;

- **self.sourceNodeSdg**: Esta variável é uma string que armazena o id do nó origem. É utilizada para ligar este nó ao nó atual no grafo SDG;
- **self.nodeStatementSdg**: Tem a mesma função que a variável **self.nodeStatement**, mas neste caso aplica-se no grafo SDG;
- **self.statementCountSdg**: Tem a mesma função que a variável **self.statementCount**, mas aplicada ao SDG;
- **self.fCfg**: Ficheiro onde é escrito o grafo CFG em formato dot;
- **self.fSdg**: Ficheiro onde é escrito o grafo SDG em formato dot.

Capítulo 4

Codificação

Neste capítulo serão explicados os passos necessários para codificar a construção e apresentação dos grafos tendo em conta a análise estática feita ao código fonte.

4.1 Construção do CGD

Para a codificação da construção deste grafo, seguimos este conjunto de passos:

1. No método **linguagem**, que é o primeiro a ser visitado, criámos imediatamente o nó de início, antes de visitar os nós das declarações. O id do nó é incrementado e este passa a ser o último nó tal como foi explicado na secção anterior.

```
    self.dot.node(str(self.statementCount), label='inicio')
    self.lastStatement = str(self.statementCount)
    self.statementCount += 1

```

2. Visitando declaração a declaração, utilizamos a variável **self.nodeStatement** para armazenar o texto correspondente. No final de cada uma criámos o respetivo nó, e fazemos a ligação entre o nó anterior e este. A variável utilizada para armazenar o texto da declaração é então posta a vazio outra vez para não armazenar vários *statements* em cada nó.

```
    self.dot.node(str(self.statementCount), label=self.nodeStatement)
    self.dot.edge(self.lastStatement,str(self.statementCount))
    self.edgeCountCfg += 1
    self.lastStatement = str(self.statementCount)
    self.statementCount += 1
    self.nodeStatement = ''

```

3. Visitando instrução a instrução, verificamos se é uma instrução simples (atribuição, leitura, escrita). Caso seja o processo é bastante simples pois só temos de ir acumulando o texto do *statement*, e criar o nó no fim, bem como fazer a ligação do nó anterior ao novo nó. Por vezes a escrita na variável **self.nodeStatement** pode ser feita por nós filho, dependendo da instrução.

4. Verificamos se a instrução é um **if**. Se for é imediatamente criado um nó, cujo id será armazenado numa variável **ifNode**.

```
    ifNode = str(self.statementCount)
    self.dot.node(ifNode,label='if ' + condicaoIfAtual, shape='diamond')
    self.lastStatement = ifNode
    self.statementCount += 1
    self.nodeStatement = ''

```

Posteriormente verificamos se o if contém instruções dentro e se estas existirem é criado o nó **then**, e é feita a ligação do nó do if ao nó then. Em seguida é criado um nó para o fim do if. Como estamos numa abordagem de **visitar em profundidade**, as instruções do if são visitadas e no final é feita a ligação entre o último nó criado e o fim do if. Posteriormente o fim do if passa a ser o último nó.

No caso de existir um else é feita a mesma coisa, cria-se um nó else, liga-se o if ao else, depois visitam-se as instruções e no final liga-se o último nó ao fim do if. Ainda existe a possibilidade de não aparecer o else, e portanto, é feita também uma ligação do ifNode para o fim do if, visto que é possível saltar diretamente para o fim se a condição for falsa.

5. Verificamos se a instrução é um **ciclo**. Como na instrução os filhos são percorridos um a um, tivemos de tratar o ciclo *for* de uma forma diferente dos restantes. Isto porque no caso do *for*, surge primeiro a atribuição inicial, depois a condição e depois a incrementação. Desta forma só podemos criar o nó do ciclo quando chegarmos à condição, visto que o *statement* do ciclo deve ter o seu nome e a condição.

```
    if child.type == 'FOR':
        isFor = True

```

Ou seja, neste caso, utilizamos uma *flag* para indicar a presença do *for*. Caso seja só iremos criar o *statement* quando aparecer a condição. Antes desta aparecer, irá aparecer a atribuição inicial, onde será criado um nó para a mesma, e o seu id será armazenado numa variável específica.

```
    if isFor and contadorAtribuicoes == 0:
        #Atribuição do for CFG
        self.dot.node(str(self.statementCount),label=self.nodeStatement)
        initialAtribForNode = str(self.statementCount)
        self.statementCount += 1
        self.nodeStatement = ''

```

Quando finalmente aparece a condição, é criado o nó do *for*, e é feita a ligação entre a atribuição anteriormente armazenada e o próprio ciclo. Além disso, é necessário ligar a última instrução (antes do for) à atribuição inicial.

Quando surge a incrementação do *for*, é criado um nó para a mesma, e armazenado numa variável, tal como acontece na atribuição inicial. No final da instrução faz-se a ligação entre a última instrução do for e a incrementação, e liga-se também a incrementação ao ciclo. Por fim, o ciclo é ligado ao fim de ciclo.

No caso dos **restantes ciclos**, o processo é muito parecido entre eles. No *while* o seu nó é criado quando se visita a condição, enquanto que no *repeat* é criado quando é encontrado um token com o valor NUM, isto porque a condição do *repeat* é um número. Quando os nós são criados, é feita a ligação entre a última instrução e o ciclo. No fim da visita à instrução, o último *statement* é conectado ao nó do ciclo, e este é conectado ao nó de fim de ciclo.

6. No fim da visita às instruções, é criado o nó final, e o nó da última instrução é ligado ao nó final.

4.2 Construção do SGD

Para a construção do SGD foram seguidos estes passos:

1. No método **linguagem** é criado o primeiro nó.

```
1     self.dotSdg.node(str(self.statementCountSdg), label='Entry MAIN', shape
      ='trapezium')
2     self.sourceNodeSdg = str(self.statementCountSdg)
3     self.statementCountSdg += 1
4
```

2. Na visita às declarações o processo é um pouco diferente da construção do CFG. Só são criados nós para as declarações que tenham inicialização, e a ligação feita no final de cada declaração tem como nó origem o nó *entry MAIN*, e portanto não se armazena o último nó a ser criado, contrariamente ao CFG.

3. Nas instruções, a criação dos nós e a criação das arestas, tanto no CFG como no SDG, é feita em simultâneo nos mesmos locais do código.

   Uma exceção do SDG é que os nós de fim de ciclo e de fim de if nunca chegam a ser criados, nem as ligações para eles.

   Outra exceção do SDG é que, sempre que é encontrada uma situação de ciclo ou condicional, o valor da variável **self.sourceNodeSdg** é armazenado numa variável local, antes de ser feita qualquer visita aos filhos dessa instrução. Posteriormente a variável **self.sourceNodeSdg** vai sendo atualizada para fazer as ligações dos nós filhos dessa estrutura. Depois de estar completo o grafo interno à estrutura especial, então a variável **self.sourceNodeSdg** volta a ser o valor anteriormente armazenado, e desta forma é possível continuar a ligar as próximas instruções ao mesmo nível do que as instruções com aninhamentos.

4.3 Persistência do código dot em ficheiros

Depois de ter os grafos todos construídos, é possível escrevê-los para um ficheiro em formato dot, no final da função **linguagem**.

```
1 #Escrever os grafos gerados
2 self.fCfg.write(self.dot.source)
3 self.fSdg.write(self.dotSdg.source)
```

4.4 Adição das imagens dos grafos na página html

No código do trabalho prático anterior, existe uma função responsável por criar a página html com as informações adicionais (**criarSegundaPagina()**). Utilizamos a mesma função para incluir as imagens dos dois grafos construídos, bem como a complexidade de McCabe calculada através do CFG. As imagens dos grafos foram criadas com o auxílio do módulo **Pydot**.

```
#Criar imagens para os mesmos
    (cfg,) = pydot.graph_from_dot_file('cfg.dot')
    cfg.write('cfg.png',format='png')
    (sdg,) = pydot.graph_from_dot_file('sdg.dot')
    sdg.write('sdg.png',format='png')
    #Calcular a complexidade de McCabe
    c = complexidade_McCabe(self.edgeCountCfg,self.statementCount)
    #Preencher a segunda página com informações adicionais
    criarSegundaPagina(self.output, self.f2Html,'cfg.png','sdg.png',c)
```

Capítulo 5

Testes e resultados obtidos

Neste capítulo serão apresentados 5 cenários de teste diferentes, em que a complexidade do código vai aumentando e são cobertos os grafos mais simples, os grafos que incluem condições, os que incluem ciclos e um que inclua uma mistura de tudo.

As figuras que apresentam os grafos obtidos correspondem a capturas de ecrã da página *web* onde são apresentados os resultados.

5.1 Teste 1 - Várias instruções simples

```
1 int y;
2 string s = "string";
3 tuple t = ("a",1);
4
5 y = 2;
6 y = y * (y + 3);
7 print(s);
8 input(s);
9 print(t);
```

Grafos gerados durante a análise estática

Control Flow Graph

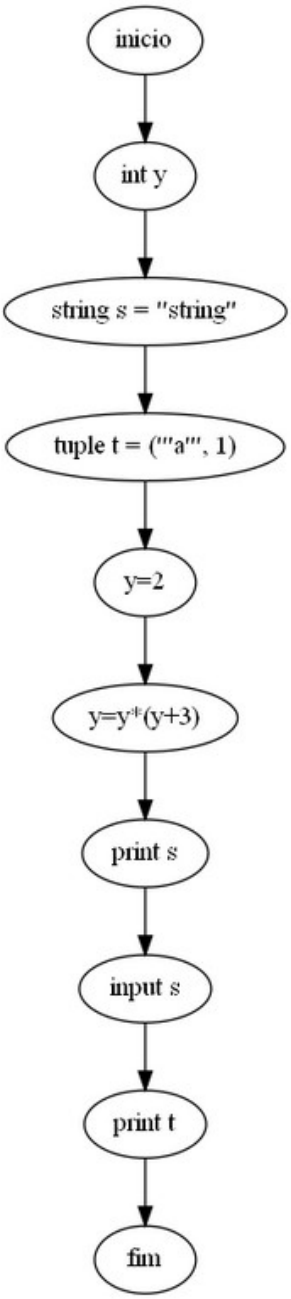


Figura 5.1: CFG obtido no teste 1

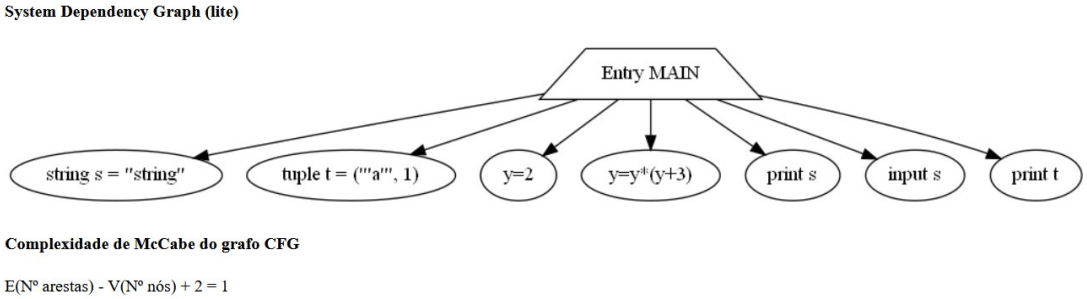


Figura 5.2: SDG e complexidade de McCabe para o teste 1

5.2 Teste 2 - Inclusão da instrução condicional

```
int y = 10;
string s = "";

if(y % 2 == 0){
    print(y);
    if(s != "string"){
        print(s);
    } else {
        input(s);
    }
}
```

Grafos gerados durante a análise estática

Control Flow Graph

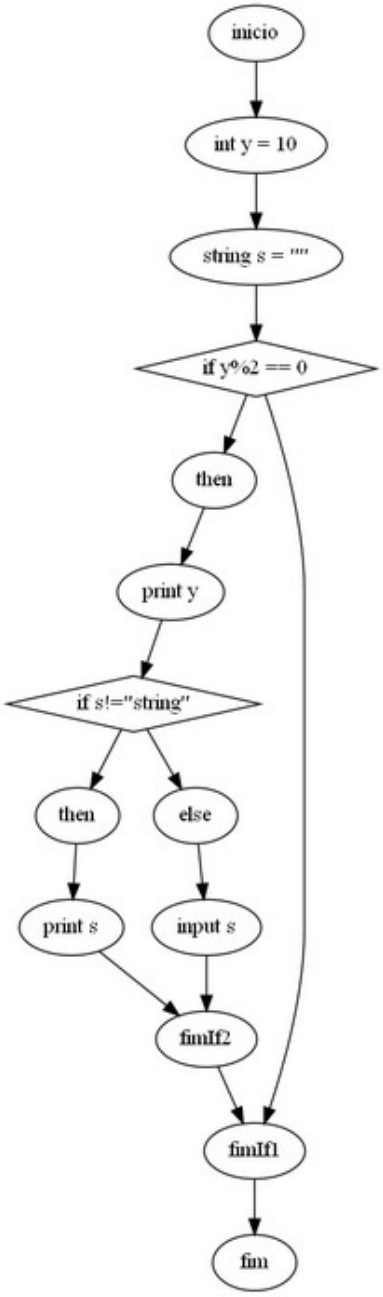
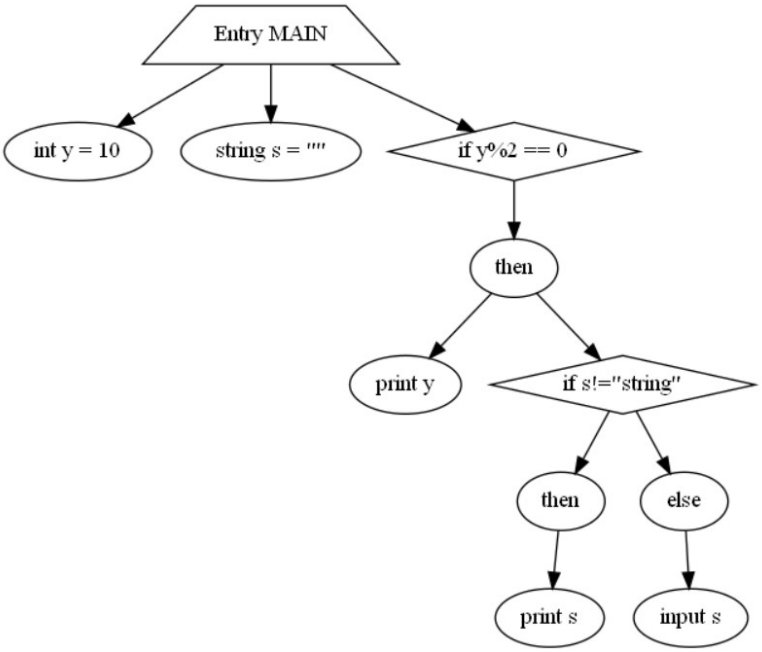


Figura 5.3: CFG obtido no teste 2

System Dependency Graph (lite)



Complexidade de McCabe do grafo CFG

E(Nº arestas) - V(Nº nós) + 2 = 3

Figura 5.4: SDG e complexidade de McCabe para o teste 2

5.3 Teste 3 - Ciclo for

```
int y = 10;
int x;

for(x = 0; x < 20; x = x + 1){
    print(y);
    print(x);
}
```

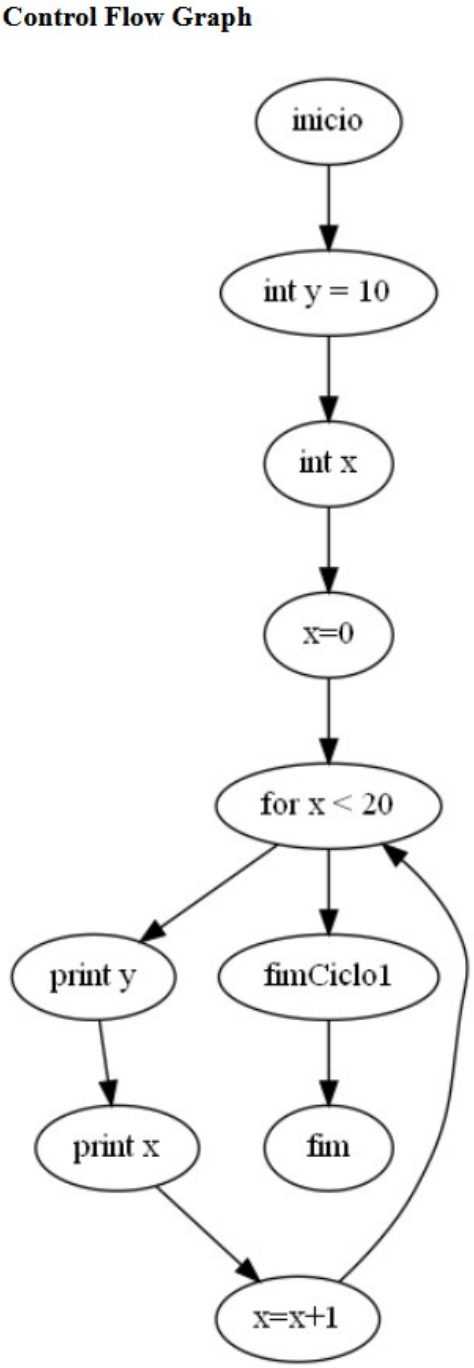
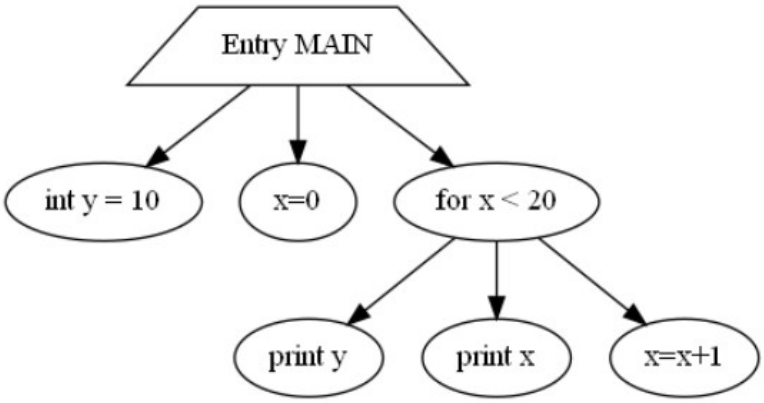


Figura 5.5: CFG obtido no teste 3

System Dependency Graph (lite)



Complexidade de McCabe do grafo CFG

E(Nº arestas) - V(Nº nós) + 2 = 2

Figura 5.6: SDG e complexidade de McCabe para o teste 3

5.4 Teste 4 - Ciclo repeat

```
int y = 10;
int x = 0;

repeat(20){
    print(y);
    print(x);
    x = x + 1;
}
```

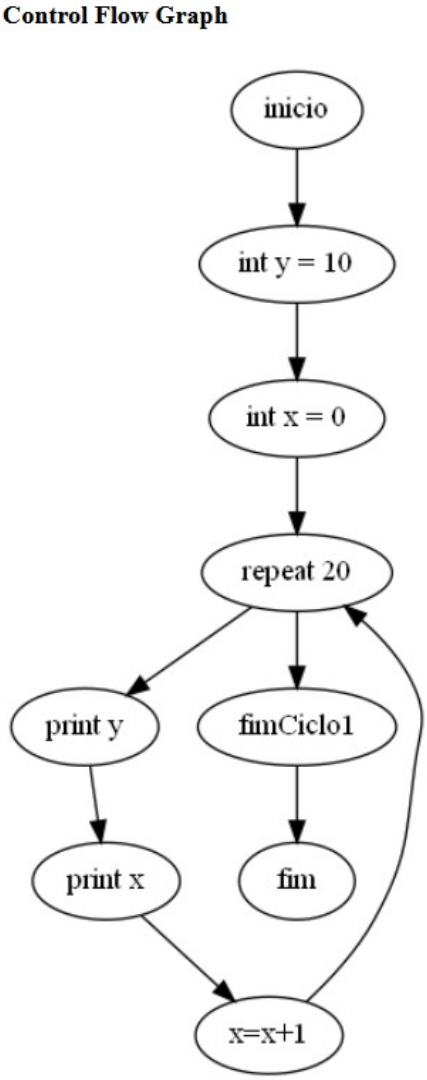
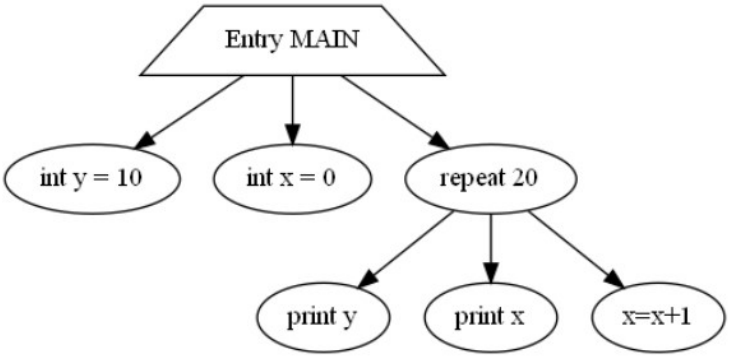


Figura 5.7: CFG obtido no teste 4

System Dependency Graph (lite)



Complexidade de McCabe do grafo CFG

E(Nº arestas) - V(Nº nós) + 2 = 2

Figura 5.8: SDG e complexidade de McCabe para o teste 4

5.5 Teste 5 - Mistura de tudo

```
int y = 10;
int x = 0;

repeat(20){
    if(x % 2 == 0){
        print(x);
        x = x + 1;
    } else {
        while(y > 0){
            print(y);
            y = y - 1;
        }
    }
}
```

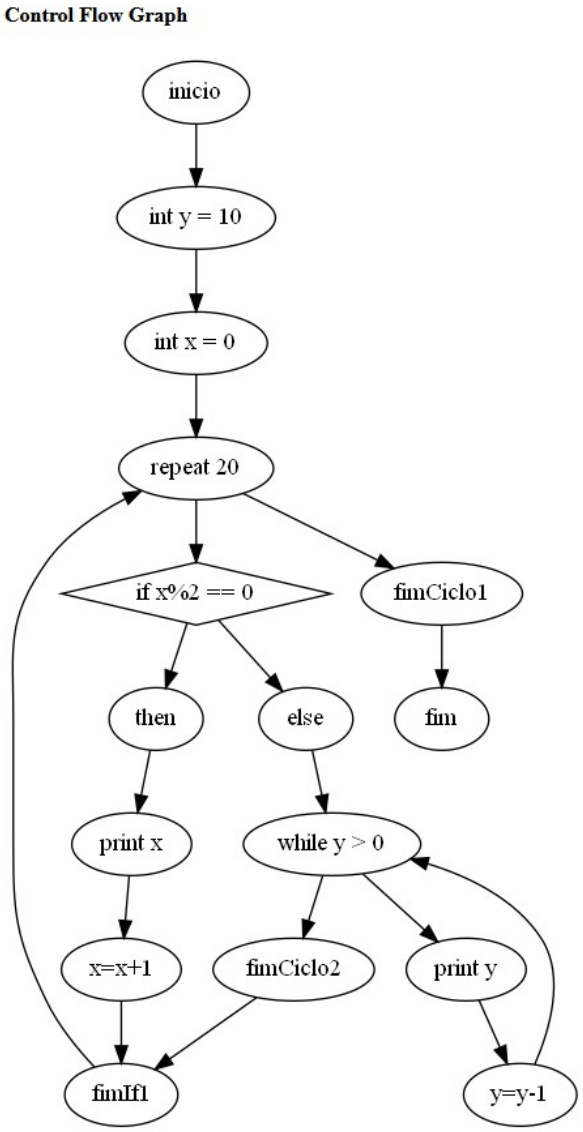
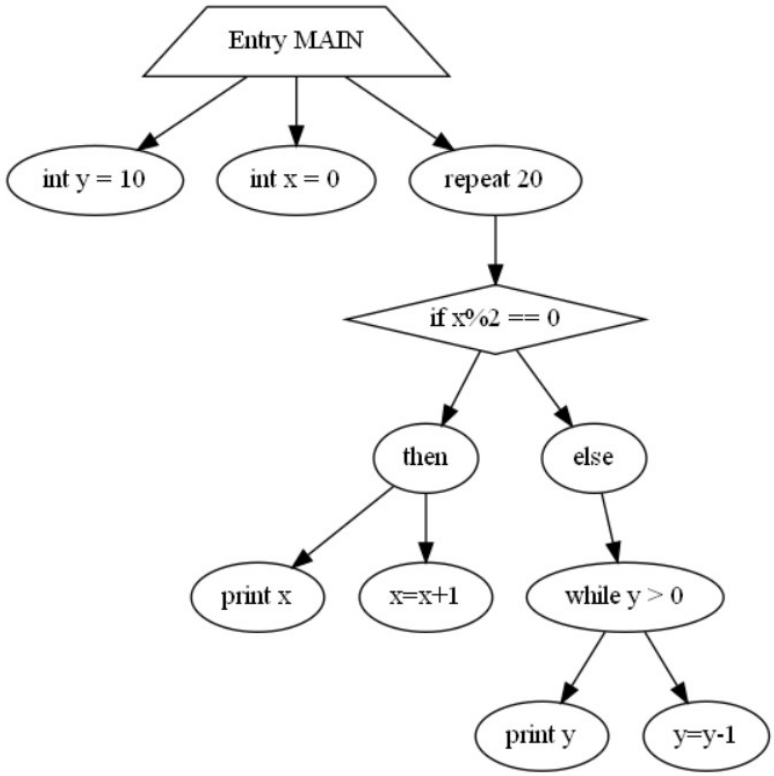


Figura 5.9: CFG obtido no teste 5

System Dependency Graph (lite)



Complexidade de McCabe do grafo CFG

E(Nº arestas) - V(Nº nós) + 2 = 4

Figura 5.10: SDG e complexidade de McCabe para o teste 5

Capítulo 6

Conclusão

O presente documento visa apresentar de forma sucinta e concreta o projeto desenvolvido, desde a formulação do problema até à implementação final e respetivos resultados apresentados, tendo sempre como base a explicação do raciocínio utilizado para a resolução do problema em questão.

Revendo todo o trabalho feito até aqui, o projeto desenvolvido encontra-se completo e a equipa acredita que este está ao nível que pretendia desde o seu início. Todos os resultados citados como objetivo no enunciado foram alcançados e realizados sempre da forma que se pensou ser a mais correta.

No âmbito geral, o grupo sente que desenvolveu o trabalho prático da melhor forma possível e conseguiu alcançar todos os objetivos propostos no mesmo.

Apêndice A

Código do Programa

Lista-se a seguir o código em Python do programa desenvolvido, baseado no código do trabalho prático 2, com adição das ferramentas para construir e apresentar grafos.

construtorGrafos.py

```
from errno import ESOCKTNOSUPPORT
from lark import Lark,Token,Tree
from lark.tree import pydot__tree_to_png
from lark import Transformer
from lark.visitors import Interpreter
from lark import Discard
from funcoesUteis import *
import graphviz
import pydot
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'


grammar = '''
linguagem: declaracoes instrucoes

declaracoes: comentario? declaracao PV comentario? (declaracao PV comentario?)*
declaracao: tipo VAR (ATRIB logic)?

instrucoes: instrucao comentario? (instrucao comentario?)*
instrucao: READ PE conteudoread PD PV
| PRINT PE logic PD PV
| IF PE condicao PD CE instrucoes CD (ELSE CE instrucoes CD)?
| FOR PE atribuicao PV condicao PV atribuicao PD CE instrucoes CD
| WHILE PE condicao PD CE instrucoes CD
| REPEAT PE NUM PD CE instrucoes CD
| atribuicao PV

comentario: C_COMMENT

atribuicao: VAR (PER chave PDR)? ATRIB logic
```

```
32
33 condicao: logic
34
35 conteudoread: VAR (PER chave PDR)?
36
37 logic: PE? logicnot AND logic PD? | PE? logicnot OR logic PD? | PE? logicnot PD?
38 logicnot: PE? NOT logic PD? | PE? relac PD?
39
40 relac: PE? logic EQ exp PD?
41 |  PE? logic DIFF exp PD?
42 |  PE? logic GRT exp PD?
43 |  PE? logic GEQ exp PD?
44 |  PE? logic LWR exp PD?
45 |  PE? logic LEQ exp PD?
46 |  PE? exp PD?
47
48 exp: PE? exp ADD termo PD?
49 | PE? exp SUB termo PD?
50 | PE? termo PD?
51
52 termo: PE? exp MUL termo PD?
53 | PE? exp DIV termo PD?
54 | PE? exp MOD termo PD?
55 | PE? factor PD?
56
57 factor: NUM
58 | BOOLEANO
59 | STRING
60 | NUMDOUBLE
61 | VAR (PER chave PDR)?
62 | PER conteudo? PDR
63 | CE conteudoespecial? CD
64 | PE conteudo? PD
65
66 conteudo: factor (VIR factor)*
67 conteudoespecial: conteudodicionario | conteudo
68 conteudodicionario: entrada (VIR entrada)*
69 entrada: STRING PP factor
70
71 tipo: INT | BOOL | STR | DOUBLE | LIST | SET | TUPLE | DICT
72 chave: NUM | STRING | VAR
73
74 BOOLEANO: "True" | "False"
75 NUM: ("0".."9")+
76 NUMDOUBLE: ("0".."9")+"."("0".."9")+
77 STRING: ESCAPED_STRING
78 INT: "int"
79 STR: "string"
80 BOOL: "bool"
81 DOUBLE: "double"
82 LIST: "list"
83 SET: "set"
84 TUPLE: "tuple"
85 DICT: "dict"
```

```
VIR: ","
PE: "("
PD: ")"
PER: "["
PDR: "]"
CE: "{"
CD: "}"
PV: ";"
PP: ":"
ADD: "+"
SUB: "-"
DIV: "/"
MUL: "*"
MOD: "%"
EQ: "=="
DIFF: "!="
GRT: ">"
GEQ: ">="
LWR: "<"
LEQ: "<="
AND: "and"
OR: "or"
NOT: "not"
READ: "input"
PRINT: "print"
ATRIB: "="
VAR: ("a".."z" | "A".."Z" | "_")+
IF: "if"
ELSE: "else"
FOR: "for"
WHILE: "while"
REPEAT: "repeat"

%import common.WS
%import common.ESCAPED_STRING
%import common.C_COMMENT
%ignore WS
'''

class LinguagemProgramacao(Interpreter):

  def __init__(self):
    #Variáveis para os grafos
    self.dot = graphviz.Digraph('cfg')
    self.dotSdg = graphviz.Digraph('sdg')
    self.nodeStatement = ''
    self.lastStatement = ''
    self.statementCount = 0
    self.edgeCountCfg = 0
    self.sourceNodeSdg = ''
    self.nodeStatementSdg = ''
    self.statementCountSdg = 0
    self.fCfg = criarFicheiro('cfg.dot')
    self.fSdg = criarFicheiro('sdg.dot')
```

```
140
141     #Variáveis para o resto da análise estática
142     self.fHtml = criarFicheiro('codigoAnotado.html')
143     self.f2Html = criarFicheiro('informacoesAdicionais.html')
144     preencherInicio(self.fHtml)
145     self.decls = {}
146     self.naoInicializadas = set()
147     self.utilizadas = set()
148     self.erros = {
149       '1: Não-declaração' : set(),
150       '2: Redeclaração' : set(),
151       '3: Usado mas não inicializado' : set(),
152       '4: Declarado mas nunca mencionado' : set()
153     }
154     self.dicinstrucoes = {
155       'total' : 0,
156       'atribuicoes' : 0,
157       'leitura' : 0,
158       'escrita' : 0,
159       'condicionais' :0,
160       'ciclicas' : 0
161     }
162     self.condicoesIfs = []
163     self.alternativasIfs = []
164     self.niveisIfs = {}
165     self.nivelIf = -1
166     self.nivelProfundidade = 0
167     self.totalSituacoesAn = 0
168     self.nasInstrucoes = False
169     self.instrucaoAtual = ''
170     self.output = {}
171
172   def linguagem(self, tree):
173     #Criação dos primeiros nós dos grafos
174     self.dot.node(str(self.statementCount), label='inicio')
175     self.lastStatement = str(self.statementCount)
176     self.statementCount += 1
177     self.dotSdg.node(str(self.statementCountSdg), label='Entry MAIN', shape='
    trapezium')
178     self.sourceNodeSdg = str(self.statementCountSdg)
179     self.statementCountSdg += 1
180     self.visit(tree.children[0]) #Declarações
181     self.nasInstrucoes = True
182     self.fHtml.write(' '*10 + '<div class="instrucoes">\n')
183     self.visit(tree.children[1]) #Instruções
184     self.dot.node(str(self.statementCount), label='fim')
185     self.dot.edge(self.lastStatement,str(self.statementCount))
186     self.edgeCountCfg += 1
187     self.statementCount += 1
188     self.fHtml.write(' '*10 + '</div>\n')
189     preencherFim(self.fHtml)
190     self.fHtml.close()
191     #Verificar as variáveis declaradas mas nunca mencionadas
192     declaradas = set(self.decls.keys())
```

```
193     self.erros['4: Declarado mas nunca mencionado'] = declaradas - self.
    utilizadas
194     #Criar o output
195     self.output['decls'] = self.decls
196     self.output['naoInicializadas'] = self.naoInicializadas
197     self.output['erros'] = self.erros
198     self.output['niveisIf'] = self.niveisIfs
199     self.output['utilizadas'] = self.utilizadas
200     self.output['instrucoes'] = self.dicinstrucoes
201     self.output['totalSituacoesAn'] = self.totalSituacoesAn
202     self.output['condicoesIf'] = self.condicoesIfs
203     self.output['alternativasIfs'] = self.alternativasIfs
204     #Escrever os grafos gerados
205     self.fCfg.write(self.dot.source)
206     self.fSdg.write(self.dotSdg.source)
207     self.fCfg.close()
208     self.fSdg.close()
209     #Criar imagens para os mesmos
210     (cfg,) = pydot.graph_from_dot_file('cfg.dot')
211     cfg.write('cfg.png',format='png')
212     (sdg,) = pydot.graph_from_dot_file('sdg.dot')
213     sdg.write('sdg.png',format='png')
214     #Calcular a complexidade de McCabe
215     c = complexidade_McCabe(self.edgeCountCfg,self.statementCount)
216     #Preencher a segunda página com informações adicionais
217     criarSegundaPagina(self.output, self.f2Html,'cfg.png','sdg.png',c)
218     return self.output
219 
220   def declaracoes(self, tree):
221     #Visita todas as declarações e cada uma processa a sua parte
222     self.fHtml.write(' '*10 + '<div class="declaracoes">\n')
223     for decl in tree.children:
224       if isinstance(decl, Tree):
225         self.visit(decl)
226     self.fHtml.write(' '*10 + '</div>\n')
227 
228   def declaracao(self, tree):
229     self.fHtml.write('\t')
230     #Inicialização de variáveis
231     var = None
232     tipo = None
233     valor = None
234     #É preciso percorrer cada filho e processar de acordo com as situações
235     for child in tree.children:
236       if isinstance(child, Token) and child.type == 'VAR' and child.value in self
    .decls.keys(): #Visita a variável declarada
237         var = child.value
238         self.erros['2: Redeclaração'].add(var)
239         self.fHtml.write('<div class="error">' + var + '<span class="errortext">
    Variável redeclarada</span></div>')
240         self.nodeStatement += ' ' + var
241         self.nodeStatementSdg += ' ' + var
242       elif  isinstance(child, Token) and child.type == 'VAR':
243         var = child.value
```

```
244         self.fHtml.write(var)
245         self.nodeStatement += ' ' + var
246         self.nodeStatementSdg += ' ' + var
247       elif isinstance(child, Tree) and child.data == 'tipo': #Visita o tipo da
    declaração
248         tipo = self.visit(child)
249         self.fHtml.write(tipo + ' ')
250         self.nodeStatement += tipo
251         self.nodeStatementSdg += tipo
252       elif isinstance(child, Token) and child.type == 'ATRIB': #Se houver atribui
    ção visita o valor que está a ser declarado
253         valor = self.visit(tree.children[3])
254         if valor != None and eDigito(valor):
255           valor = int(valor)
256         elif valor != None and eDouble(valor):
257           valor = float(valor)
258         if valor == None:
259           self.naoInicializadas.add(var)
260           self.decls[var] = tipo
261           self.fHtml.write(';\n')
262         elif isinstance(valor,str) and valor[0] != '"':
263           self.fHtml.write(' = ')
264           self.nodeStatement += ' = '
265           self.nodeStatementSdg += ' = '
266           if ('[' or ']') in valor: #O valor resulta de um acesso a uma estrutura
267             infoEstrutura = valor.split('[')
268             variavel = infoEstrutura[0]
269             if variavel not in self.decls.keys(): #Se a variável não tiver sido
    declarada antes é gerado um erro
270               if variavel != '':
271                 self.erros['1: Não-declaração'].add(variavel)
272               self.fHtml.write('<div class="error">' + variavel + '<span class="
    errortext">Variável redeclarada</span></div>' + '[' + infoEstrutura[1] + ';\n'
    )
273             elif variavel in self.naoInicializadas:
274               self.fHtml.write('<div class="error">' + variavel + '<span class="
    errortext">Variável não inicializada</span></div>' + '[' + infoEstrutura[1] +
    ';\n')
275             else:
276               self.fHtml.write(variavel + '[' + infoEstrutura[1] + ';\n')
277             self.nodeStatement += variavel + '[' + infoEstrutura[1]
278             self.nodeStatementSdg += variavel + '[' + infoEstrutura[1]
279           elif('{' or '}') in valor:
280             self.fHtml.write(valor + ';\n')
281             self.nodeStatement += str(valor)
282             self.nodeStatementSdg += str(valor)
283           elif('(' or ')') in valor:
284             self.fHtml.write(valor + ';\n')
285             self.nodeStatement += str(valor)
286             self.nodeStatementSdg += str(valor)
287           elif valor not in self.decls.keys(): #Verificar se a variável atómica j
    á existe
288             self.erros['1: Não-declaração'].add(valor)
```

```
289             self.fHtml.write('<div class="error">' + valor + '<span class="
    errortext">Variável redeclarada</span></div>;\n')
290             self.nodeStatement += valor
291             self.nodeStatementSdg += valor
292         else:
293           self.fHtml.write(' = ')
294           self.fHtml.write(str(valor) + ';\n')
295           self.nodeStatement += ' = ' + str(valor)
296           self.nodeStatementSdg += ' = ' + str(valor)
297
298     if valor == None: #Caso nunca tenha existido atribuição na declaração
299       self.naoInicializadas.add(var)
300       self.decls[var] = tipo
301       self.fHtml.write(';\n')
302       self.nodeStatementSdg = ''
303     else:
304       #Fazer a ligação entre a entry e a declaração
305       self.dotSdg.node(str(self.statementCountSdg),label=self.nodeStatementSdg)
306       self.dotSdg.edge(self.sourceNodeSdg,str(self.statementCountSdg))
307       self.nodeStatementSdg = ''
308       self.statementCountSdg += 1
309
310     self.decls[var] = tipo
311     self.dot.node(str(self.statementCount), label=self.nodeStatement)
312     self.dot.edge(self.lastStatement,str(self.statementCount))
313     self.edgeCountCfg += 1
314     self.lastStatement = str(self.statementCount)
315     self.statementCount += 1
316     self.nodeStatement = ''
317
318   def instrucoes(self, tree):
319     r = self.visit_children(tree)
320     return r
321
322   def instrucao(self, tree):
323     localLastStatement = self.lastStatement
324     localSourceNode = self.sourceNodeSdg
325     idFimCiclo = ''
326     idFimCondicional = ''
327     estruturaCiclica = False
328     isFor = False
329     #Utilizado para saber quais as atribuições do for
330     contadorAtribuicoes = 0
331     incrementNode = ''
332     incrementNodeSdgStatement = ''
333     initialAtribForNode = ''
334     initialAtribForNodeSdg = ''
335     cicleNode = ''
336     cicleNodeSdg = ''
337     ifNode = ''
338     ifNodeSdg = ''
339     endifNode = ''
340     nivelIf = self.nivelIf
341     nivelProfundidade = self.nivelProfundidade
```

```
342     numTabs = (nivelProfundidade * '\t') + '\t'
343     self.fHtml.write(numTabs)
344     condicoesParaAninhar = []
345     corpo = ''
346     sairDoCiclo = False
347     resultado = ''
348     for child in tree.children:
349         if not sairDoCiclo:
350             if isinstance(child,Token) and child.type == 'IF':
351                 existeElse = False
352                 self.instrucaoAtual = "condicional"
353                 self.dicinstrucoes['condicionais'] += 1
354                 idFimCondicional = str(self.dicinstrucoes['condicionais'])
355                 self.dicinstrucoes['total'] += 1
356 
357                 if self.nivelIf == -1: #Primeiro if do código
358                     nivelIf = self.nivelIf = 0
359                 elif nivelProfundidade > 0:
360                     self.totalSituacoesAn += 1
361 
362                 self.niveisIfs.setdefault(nivelIf, list())
363                 self.niveisIfs[nivelIf].append(self.dicinstrucoes['condicionais'
    ])
364 
365                 self.fHtml.write('<div class="info">' + child.value + '<span
    class="infotext">Nível de aninhamento: ' + str(nivelIf) + '</span></div>')
366                 self.fHtml.write('(')
367                 condicaoIfAtual = self.visit(tree.children[2])[0]
368                 self.fHtml.write('){\n')
369 
370                 #Criação do nó para o grafo CFG
371                 ifNode = str(self.statementCount)
372                 self.dot.node(ifNode,label='if ' + condicaoIfAtual, shape='
    diamond')
373                 self.lastStatement = ifNode
374                 self.statementCount += 1
375                 self.nodeStatement = ''
376 
377                 #Criação do nó para o grafo SDG
378                 ifNodeSdg = str(self.statementCountSdg)
379                 self.dotSdg.node(ifNodeSdg,label='if ' + condicaoIfAtual, shape='
    diamond')
380                 self.sourceNodeSdg = ifNodeSdg
381                 self.statementCountSdg += 1
382                 self.nodeStatementSdg = ''
383 
384                 if len(tree.children[5].children) > 0: #Se existirem instruções
    dentro do if
385                     #Criar o nó then CFG
386                     self.dot.node(str(self.statementCount),label='then')
387                     self.dot.edge(ifNode,str(self.statementCount))
388                     self.edgeCountCfg += 1
389                     self.lastStatement = str(self.statementCount)
390                     self.statementCount += 1
```

```
                    #Criar o nó then SDG
                    self.dotSdg.node(str(self.statementCountSdg),label='then')
                    self.dotSdg.edge(ifNodeSdg,str(self.statementCountSdg))
                    self.sourceNodeSdg = str(self.statementCountSdg)
                    self.statementCountSdg += 1

                    #Criar o nó para fim do if
                    endifNode = str(self.statementCount)
                    self.dot.node(endifNode,label='fimIf' + idFimCondicional)
                    self.statementCount += 1

                    existeElse = 'else' in tree.children[5].children[0].children
                    proxInstrucao = str(tree.children[5].children[0].children[0])
                    if proxInstrucao != 'if' or existeElse or len(tree.children
    [5].children) > 1:
                        self.condicoesIfs.append(condicaoIfAtual)
                        condicoesParaAninhar = self.condicoesIfs
                        self.condicoesIfs = []
                        self.nivelProfundidade += 1
                        self.nivelIf += 1
                        res = self.visit(tree.children[5])
                        self.nivelIf = nivelIf
                        self.nivelProfundidade = nivelProfundidade
                        numTabs = (self.nivelProfundidade * '\t') + '\t'
                        self.fHtml.write(numTabs + '}')
                        corpo = res[0]
                        for r in res[1:]:
                            corpo += '\n' + r

                        #Ligar ao fim do if
                        self.dot.edge(self.lastStatement,endifNode)
                        self.edgeCountCfg += 1
                        self.lastStatement = endifNode

                        if 'else' in tree.children:
                            existeElse = True
                            #Criar o nó else CFG
                            self.dot.node(str(self.statementCount),label='else')
                            self.dot.edge(ifNode,str(self.statementCount))
                            self.edgeCountCfg += 1
                            self.lastStatement = str(self.statementCount)
                            self.statementCount += 1

                            #Criar o nó else SDG
                            self.dotSdg.node(str(self.statementCountSdg),label='
    else')
                            self.dotSdg.edge(ifNodeSdg,str(self.statementCountSdg
    ))
                            self.sourceNodeSdg = str(self.statementCountSdg)
                            self.statementCountSdg += 1

                            self.fHtml.write('else{\n')
                            self.nivelProfundidade += 1
                            self.nivelIf += 1
```

```
                            resElse = self.visit(tree.children[9])
                            self.nivelIf = nivelIf
                            self.nivelProfundidade = nivelProfundidade
                            numTabs = (self.nivelProfundidade * '\t') + '\t'
                            self.fHtml.write(numTabs + '}')
                            resultado += child.value + '(' + condicaoIfAtual + ')
    {\n' + str(res) + '}else{\n' + str(resElse) + '}'

                            #Ligar ao fim do if
                            self.dot.edge(self.lastStatement,endifNode)
                            self.edgeCountCfg += 1
                            self.lastStatement = endifNode
                        else:
                            resultado += child.value + '(' + condicaoIfAtual + ')
    {\n' + str(res) + '}'
                    else:
                        self.condicoesIfs.append(condicaoIfAtual)
                        self.nivelProfundidade += 1
                        self.nivelIf += 1
                        res = self.visit(tree.children[5])
                        self.nivelIf = nivelIf
                        self.nivelProfundidade = nivelProfundidade
                        numTabs = (self.nivelProfundidade * '\t') + '\t'
                        self.fHtml.write(numTabs + '}')
                        corpo = res[0]
                        for r in res[1:]:
                            corpo += '\n' + r

                        #Ligar ao fim do if
                        self.dot.edge(self.lastStatement,endifNode)
                        self.edgeCountCfg += 1
                        self.lastStatement = endifNode

                        if 'else' in tree.children:
                            self.fHtml.write('else{\n')
                            self.nivelProfundidade += 1
                            self.nivelIf += 1
                            resElse = self.visit(tree.children[9])
                            self.nivelIf = nivelIf
                            self.nivelProfundidade = nivelProfundidade
                            numTabs = (self.nivelProfundidade * '\t') + '\t'
                            self.fHtml.write(numTabs + '}')
                            resultado += child.value + '(' + condicaoIfAtual + ')
    {\n' + str(res) + '}else{\n' + str(resElse) + '}'

                            #Ligar ao fim do if
                            self.dot.edge(self.lastStatement,endifNode)
                            self.edgeCountCfg += 1
                            self.lastStatement = endifNode
                        else:
                            resultado += child.value + '(' + condicaoIfAtual + ')
    {\n' + str(res) + '}'
                sairDoCiclo = True
```

```
491                 self.sourceNodeSdg = localSourceNode #Dar reset ao nó origem para
    voltar ao nível anterior
492             elif isinstance(child,Token) and child.type == 'CE':
493                 #O \n é necessário porque a seguir a este token chegam instruções
    aninhadas
494                 self.fHtml.write(child.value)
495                 self.fHtml.write('\n')
496                 resultado += '{\n' + numTabs
497             elif isinstance(child,Token) and child.type == 'CD':
498                 self.fHtml.write(numTabs + child.value)
499                 resultado += numTabs + '}'
500             elif isinstance(child, Token) and (child.type == 'FOR' or child.type
    == 'WHILE' or child.type == 'REPEAT'):
501                 estruturaCiclica = True
502                 if child.type == 'FOR':
503                   isFor = True
504                 if self.nivelProfundidade > 0:
505                   self.totalSituacoesAn += 1
506                 self.fHtml.write(child.value)
507                 self.dicinstrucoes['ciclicas'] += 1
508                 idFimCiclo = str(self.dicinstrucoes['ciclicas'])
509                 self.dicinstrucoes['total'] += 1
510                 self.instrucaoAtual = "ciclo"
511                 resultado += child.value
512                 if not isFor:
513                   self.nodeStatement += child.value + ' '
514                   self.nodeStatementSdg += child.value + ' '
515             elif isinstance(child, Token) and child.type == 'READ':
516                 self.fHtml.write(child.value)
517                 self.instrucaoAtual = "leitura"
518                 self.dicinstrucoes['leitura'] += 1
519                 self.dicinstrucoes['total'] += 1
520                 resultado += child.value
521                 self.nodeStatement += child.value + ' '
522                 self.nodeStatementSdg += child.value + ' '
523             elif isinstance(child, Token) and child.type == 'PRINT':
524                 self.fHtml.write(child.value)
525                 self.instrucaoAtual = "escrita"
526                 self.dicinstrucoes['escrita'] += 1
527                 self.dicinstrucoes['total'] += 1
528                 resultado += child.value
529                 self.nodeStatement += child.value + ' '
530                 self.nodeStatementSdg += child.value + ' '
531             elif isinstance(child,Token):
532                 self.fHtml.write(child.value)
533                 resultado += child.value
534                 if child.value not in [';','(',')','{','}']:
535                   self.nodeStatement += child.value
536                   self.nodeStatementSdg += child.value
537                   if child.type == 'NUM' and estruturaCiclica:
538                     cicleNode = str(self.statementCount)
539                     self.dot.node(cicleNode,label=self.nodeStatement)
540                     self.dot.edge(self.lastStatement,cicleNode)
541                     self.edgeCountCfg += 1
```

```
                    self.lastStatement = str(self.statementCount)
                    self.statementCount += 1
                    self.nodeStatement = ''

                    cicleNodeSdg = str(self.statementCountSdg)
                    self.dotSdg.node(cicleNodeSdg,label=self.nodeStatementSdg)
                    self.dotSdg.edge(self.sourceNodeSdg,cicleNodeSdg)
                    self.sourceNodeSdg = str(self.statementCountSdg)
                    self.statementCountSdg += 1
                    self.nodeStatementSdg = ''
            elif isinstance(child, Tree) and child.data == 'atribuicao':
                self.instrucaoAtual = "atribuicao"
                self.dicinstrucoes['atribuicoes'] += 1
                self.dicinstrucoes['total'] += 1
                res = str(self.visit(child))
                resultado += res
                if isFor and contadorAtribuicoes == 0:
                  #Atribuição do for CFG
                  self.dot.node(str(self.statementCount),label=self.nodeStatement
)
                  initialAtribForNode = str(self.statementCount)
                  self.statementCount += 1
                  self.nodeStatement = ''
                  #Atribuição do for SDG
                  self.dotSdg.node(str(self.statementCountSdg),label=self.
nodeStatementSdg)
                  initialAtribForNodeSdg = str(self.statementCountSdg)
                  self.statementCountSdg += 1
                  self.nodeStatementSdg = ''
                elif isFor and contadorAtribuicoes == 1:
                  #Incrementação do for CFG
                  self.dot.node(str(self.statementCount),label=self.nodeStatement
)
                  incrementNode = str(self.statementCount)
                  self.statementCount += 1
                  self.nodeStatement = ''
                  #Incrementação do for SDG
                  incrementNodeSdgStatement = self.nodeStatementSdg
                  self.nodeStatementSdg = ''
                contadorAtribuicoes += 1
            elif isinstance(child, Tree):
                if child.data == 'conteudoread':
                    res = str(self.visit(child))
                    resultado += res
                elif child.data == 'logic':
                    res = str(self.visit(child))
                    resultado += res
                    self.nodeStatement += res
                    self.nodeStatementSdg += res
                elif child.data == 'condicao':
                    res = str(self.visit(child))
                    resultado += res
                    if estruturaCiclica:
                      if isFor:
```

```
593                         #Ligar a última instrução antes do for à primeira atrib
    do for
594                         self.dot.edge(self.lastStatement,initialAtribForNode)
595                         self.edgeCountCfg += 1
596                         self.dotSdg.edge(self.sourceNodeSdg,
    initialAtribForNodeSdg)
597                         #Ligar a primeira atrib ou nó origem ao for
598                         cicleNode = str(self.statementCount)
599                         cicleNodeSdg = str(self.statementCountSdg)
600                         self.nodeStatement = 'for ' + self.nodeStatement
601                         self.nodeStatementSdg = 'for ' + self.nodeStatementSdg
602                         self.dot.node(cicleNode,label=self.nodeStatement)
603                         self.dotSdg.node(cicleNodeSdg,label=self.nodeStatementSdg
    )
604                         self.dot.edge(initialAtribForNode,cicleNode)
605                         self.edgeCountCfg += 1
606                         self.dotSdg.edge(self.sourceNodeSdg,cicleNodeSdg) #Ligar
    o nó origem ao for
607                         self.lastStatement = str(self.statementCount)
608                         self.sourceNodeSdg = str(self.statementCountSdg)
609                         self.statementCount += 1
610                         self.statementCountSdg += 1
611                         self.nodeStatement = ''
612                         self.nodeStatementSdg = ''
613                       else:
614                         cicleNode = str(self.statementCount)
615                         cicleNodeSdg = str(self.statementCountSdg)
616                         self.dot.node(cicleNode,label=self.nodeStatement)
617                         self.dotSdg.node(cicleNodeSdg,label=self.nodeStatementSdg
    )
618                         self.dot.edge(self.lastStatement,cicleNode)
619                         self.edgeCountCfg += 1
620                         self.dotSdg.edge(self.sourceNodeSdg,cicleNodeSdg)
621                         self.lastStatement = str(self.statementCount)
622                         self.sourceNodeSdg = str(self.statementCountSdg)
623                         self.statementCount += 1
624                         self.statementCountSdg += 1
625                         self.nodeStatement = ''
626                         self.nodeStatementSdg = ''
627                 elif child.data == 'instrucoes':
628                     self.nivelProfundidade += 1
629                     self.nivelIf = 0
630                     res = str(self.visit(child))
631                     resultado += res
632                     self.nivelIf = nivelIf
633                     self.nivelProfundidade = nivelProfundidade
634                     self.sourceNodeSdg = localSourceNode
635                     #self.nodeStatement += res
636
637     #Se a lista de condições tiver mais do que um elemento então podemos aninhar
    os ifs
638     if len(condicoesParaAninhar) > 1:
639       alternativaIf = 'if(' + condicoesParaAninhar[0]
640       for cond in condicoesParaAninhar[1:]:
```

```
641         alternativaIf += ' && ' + cond
642       alternativaIf += '){\n' + numTabs + '\t\t'
643       alternativaIf += corpo
644       alternativaIf += '\n' + numTabs + '}'
645       #print('ALTERNATIVA PARA O IF: ', alternativaIf)
646       self.alternativasIfs.append(alternativaIf)
647     self.fHtml.write('\n')
648 
649     #Verificar se há um novo statement para adicionar ao grafo
650     #Se existir um ciclo...
651     if cicleNode != '': #CFG
652         if isFor:
653           #Ligar a última instrução dentro do ciclo à atribuição de incrementação
654           self.dot.edge(self.lastStatement, incrementNode)
655           self.edgeCountCfg += 1
656           #Ligar a incrementação ao for
657           self.dot.edge(incrementNode,cicleNode)
658           self.edgeCountCfg += 1
659         else:
660           #Ligar a última instrução dentro do ciclo ao próprio ciclo
661           self.dot.edge(self.lastStatement,cicleNode)
662           self.edgeCountCfg += 1
663         #Ligar o ciclo ao fim de ciclo
664         self.dot.node(str(self.statementCount),label='fimCiclo' + idFimCiclo)
665         self.dot.edge(cicleNode,str(self.statementCount))
666         self.edgeCountCfg += 1
667         self.lastStatement = str(self.statementCount)
668         self.nodeStatement = ''
669         self.statementCount += 1
670 
671     #Se não existir um if e houver um statement para escrever...
672     if self.nodeStatement != '' and ifNode == '': #CFG
673       self.dot.node(str(self.statementCount),label=self.nodeStatement)
674       self.dot.edge(self.lastStatement,str(self.statementCount))
675       self.edgeCountCfg += 1
676       self.lastStatement = str(self.statementCount)
677       self.nodeStatement = ''
678       self.statementCount += 1
679     #Se existir um if...
680     elif ifNode != '': #CFG
681       self.dot.edge(localLastStatement,ifNode)
682       self.edgeCountCfg += 1
683       self.lastStatement = endifNode
684       self.nodeStatement = ''
685       if not existeElse:
686         self.dot.edge(ifNode,endifNode)
687         self.edgeCountCfg += 1
688 
689     #Se existir um ciclo...
690     if cicleNodeSdg != '': #SDG
691         if isFor:
692           #Ligar o for à incrementação
693           self.dotSdg.node(str(self.statementCountSdg),label=
   incrementNodeSdgStatement)
```

```
694           self.dotSdg.edge(cicleNodeSdg,str(self.statementCountSdg))
695           self.statementCountSdg += 1
696
697     #Se não existir um if e houver um statement para escrever...
698     if self.nodeStatementSdg != '' and ifNodeSdg == '': #SDG
699       self.dotSdg.node(str(self.statementCountSdg),label=self.nodeStatementSdg)
700       self.dotSdg.edge(self.sourceNodeSdg,str(self.statementCountSdg))
701       self.nodeStatementSdg = ''
702       self.statementCountSdg += 1
703     #Se existir um if...
704     elif ifNodeSdg != '': #SDG
705       self.dotSdg.edge(localSourceNode,ifNodeSdg)
706       self.nodeStatementSdg = ''
707     return resultado
708
709   def condicao(self,tree):
710     r = self.visit_children(tree)
711     self.nodeStatement += str(r[0])
712     self.nodeStatementSdg += str(r[0])
713     return r
714     #print('CONDICAO: ', r)
715
716   def atribuicao(self,tree):
717     res = ''
718     for child in tree.children:
719       if (isinstance(child,Token) and child.type == 'VAR') and child.value in
      self.naoInicializadas:
720         self.fHtml.write(child.value)
721         self.naoInicializadas.remove(child.value)
722         res += child.value
723         self.nodeStatement += child.value
724         self.nodeStatementSdg += child.value
725       elif (isinstance(child,Token) and child.type == 'VAR') and child.value not
      in self.decls.keys():
726         self.fHtml.write('<div class="error">' + child.value + '<span class="
      errortext">Variável não declarada</span></div>')
727         res += '<div class="error">' + child.value + '<span class="errortext">
      Variável não declarada</span></div>'
728         self.nodeStatement += child.value
729         self.nodeStatementSdg += child.value
730       elif (isinstance(child,Token) and child.type == 'VAR'):
731         self.fHtml.write(child.value)
732         res += child.value
733         self.nodeStatement += child.value
734         self.nodeStatementSdg += child.value
735       elif isinstance(child,Token):
736         self.fHtml.write(child.value)
737         res += str(child.value)
738         self.nodeStatement += child.value
739         self.nodeStatementSdg += child.value
740       elif isinstance(child,Tree) and child.data == 'chave':
741         chave = self.visit(child)
742         self.fHtml.write(str(chave))
743         res += str(chave)
```

```
744         self.nodeStatement += str(chave)
745         self.nodeStatementSdg += str(chave)
746       elif isinstance(child,Tree):
747         r = str(self.visit(child))
748         res += r
749         self.nodeStatement += r
750         self.nodeStatementSdg += r
751     return res
752 
753   def conteudoread(self,tree):
754     res = ''
755     for child in tree.children:
756       if isinstance(child,Token) and child.type == 'VAR' and child.value in self.
    naoInicializadas:
757         self.fHtml.write(child.value)
758         self.naoInicializadas.remove(child.value)
759         self.utilizadas.add(child.value)
760         res += child.value
761         self.nodeStatement += child.value
762         self.nodeStatementSdg += child.value
763       elif isinstance(child,Token) and child.type == 'VAR' and child.value not in
     self.decls.keys():
764         self.fHtml.write('<div class="error">' + child.value + '<span class="
    errortext">Variável não declarada</span></div>')
765         res += '<div class="error">' + child.value + '<span class="errortext">
    Variável não declarada</span></div>'
766         self.nodeStatement += child.value
767         self.nodeStatementSdg += child.value
768       elif isinstance(child,Token):
769         self.fHtml.write(child.value)
770         res += str(child.value)
771         self.nodeStatement += str(child.value)
772         self.nodeStatementSdg += str(child.value)
773       elif isinstance(child,Tree) and child.data == 'chave':
774         chave = str(self.visit(child))
775         self.fHtml.write(chave)
776         res += chave
777         self.nodeStatement += chave
778         self.nodeStatementSdg += chave
779     return res
780 
781   def tipo(self, tree):
782     return str(tree.children[0])
783 
784   def logic(self,tree):
785     res = ''
786     for child in tree.children:
787       if isinstance(child,Token) and self.nasInstrucoes:
788         self.fHtml.write(child.value)
789         res += str(child.value)
790       elif isinstance(child,Tree) and child.data == 'logicnot':
791         visit = self.visit(child)
792         if visit != None:
793           res += str(visit)
```

```
794       elif isinstance(child,Tree):
795         visit = self.visit(child)
796         if visit != None:
797           res += str(visit)
798     return res
799 
800   def logicnot(self,tree):
801     res = ''
802     for child in tree.children:
803       if isinstance(child,Token) and self.nasInstrucoes:
804         self.fHtml.write(child.value)
805         res += str(child.value)
806       elif isinstance(child,Tree) and child.data == 'relac':
807         visit = self.visit(child)
808         if visit != None:
809           res += str(visit)
810       elif isinstance(child,Tree):
811         visit = self.visit(child)
812         if visit != None:
813           res += str(visit)
814     return res
815 
816   def relac(self,tree):
817     res = ''
818     for child in tree.children:
819       if isinstance(child,Token) and (child.value == '<' or child.value == '>' or
    child.value == '<=' or child.value == '>=' or child.value == '==') and self.
   nasInstrucoes:
820         self.fHtml.write(child.value)
821         res += str(' ' + child.value + ' ')
822       elif isinstance(child,Token) and self.nasInstrucoes:
823         self.fHtml.write(child.value)
824         res += str(child.value)
825       elif isinstance(child,Tree) and child.data == 'exp':
826         visit = self.visit(child)
827         if visit != None:
828           res += str(visit)
829       elif isinstance(child,Tree):
830         visit = self.visit(child)
831         if visit != None:
832           res += str(visit)
833     return res
834 
835   def exp(self,tree):
836     res = ''
837     for child in tree.children:
838       if isinstance(child,Token) and self.nasInstrucoes:
839         self.fHtml.write(child.value)
840         res += str(child.value)
841       elif isinstance(child,Tree) and child.data == 'termo':
842         visit = self.visit(child)
843         if visit != None:
844           res += str(visit)
845       elif isinstance(child,Tree):
```

```
846         visit = self.visit(child)
847         if visit != None:
848           res += str(visit)
849     return res
850 
851   def termo(self,tree):
852     res = ''
853     for child in tree.children:
854       if isinstance(child,Token) and self.nasInstrucoes:
855         self.fHtml.write(child.value)
856         res += str(child.value)
857       elif isinstance(child,Tree) and child.data == 'factor':
858         visit = self.visit(child)
859         if visit != None:
860           res += str(visit)
861       elif isinstance(child,Tree):
862         visit = self.visit(child)
863         if visit != None:
864           res += str(visit)
865     return res
866 
867   def factor(self, tree):
868     for child in tree.children:
869       if isinstance(child, Token) and child.type == 'VAR' and len(tree.children)
    > 1:
870         self.utilizadas.add(child.value)
871         chave = self.visit(tree.children[2])
872         if self.nasInstrucoes and child.value not in self.decls.keys():
873           self.fHtml.write('<div class="error">' + child.value + '<span class="
    errortext">Variável não declarada</span>\</div>' + '[' + str(chave) + ']')
874           self.erros['1: Não-declaração'].add(child.value)
875         elif self.nasInstrucoes and child.value in self.naoInicializadas:
876           self.fHtml.write('<div class="error">' + child.value + '<span class="
    errortext">Variável não inicializada</span></div>' + '[' + str(chave) + ']')
877           self.erros['3: Usado mas não inicializado'].add(child.value)
878         elif self.nasInstrucoes:
879           self.fHtml.write(child.value)
880           self.fHtml.write('[' + str(chave) + ']')
881         return str(child.value) + '[' + str(chave) + ']'
882       elif isinstance(child, Token) and child.type == 'VAR':
883         if self.nasInstrucoes and child.value not in self.decls.keys():
884           self.fHtml.write('<div class="error">' + child.value + '<span class="
    errortext">Variável não declarada</span></div>')
885           self.erros['1: Não-declaração'].add(child.value)
886         elif self.nasInstrucoes and child.value in self.naoInicializadas:
887           self.fHtml.write('<div class="error">' + child.value + '<span class="
    errortext">Variável não inicializada</span></div>')
888           self.erros['3: Usado mas não inicializado'].add(child.value)
889         elif self.nasInstrucoes:
890           self.fHtml.write(child.value)
891         #Adicionar a variável à lista das utilizadas
892         self.utilizadas.add(child.value)
893         return str(child.value)
894       elif isinstance(child, Tree) and child.data == 'chave':
```

```
895         #Obter o índice da estrutura se existir
896         chave = self.visit(child)
897         if self.nasInstrucoes:
898           self.fHtml.write(chave)
899       elif isinstance(child, Token) and child.type == 'NUM':
900         if self.nasInstrucoes:
901           self.fHtml.write(child.value)
902         return int(child.value)
903       elif isinstance(child, Token) and child.type == 'BOOLEANO':
904         if self.nasInstrucoes:
905           self.fHtml.write(child.value)
906         if child.value == "False":
907           return False
908         elif child.value == "True":
909           return True
910       elif isinstance(child, Token) and child.type == 'STRING':
911         if self.nasInstrucoes:
912           self.fHtml.write(child.value)
913         return str(child.value)
914       elif isinstance(child, Token) and child.type == 'NUMDOUBLE':
915         if self.nasInstrucoes:
916           self.fHtml.write(child.value)
917         return float(child.value)
918       elif isinstance(child, Token) and child.type == 'PER' and isinstance(tree.
    children[1], Token) and (tree.children[1]).type == 'PDR':
919         if self.nasInstrucoes:
920           self.fHtml.write('[]')
921         return list()
922       elif isinstance(child, Token) and child.type == 'PER' and isinstance(tree.
    children[1], Tree):
923         lista = None
924         if self.nasInstrucoes:
925           self.fHtml.write('[')
926           lista = self.visit(tree.children[1])
927           self.fHtml.write(']')
928         else:
929           lista = self.visit(tree.children[1])
930         return lista
931       elif isinstance(child, Token) and child.type == 'CE' and isinstance(tree.
    children[1], Token) and (tree.children[1]).type == 'CD':
932         if self.nasInstrucoes:
933           self.fHtml.write("{}")
934         return {}
935       elif isinstance(child, Token) and child.type == 'CE' and isinstance(tree.
    children[1], Tree):
936         estrutura = None
937         if self.nasInstrucoes:
938           self.fHtml.write('{')
939           estrutura = self.visit(tree.children[1])
940           self.fHtml.write('}')
941         else:
942           estrutura = self.visit(tree.children[1])
943         if isinstance(estrutura,dict):
944           return estrutura
```

```
945         else:
946           return set(estrutura)
947       elif isinstance(child, Token) and child.type == 'PE' and isinstance(tree.
    children[1], Token) and (tree.children[1]).type == 'PD':
948         if self.nasInstrucoes:
949           self.fHtml.write('()')
950         return tuple()
951       elif isinstance(child, Token) and child.type == 'PE' and isinstance(tree.
    children[1], Tree):
952         tuplo = None
953         if self.nasInstrucoes:
954           self.fHtml.write('(')
955           tuplo = self.visit(tree.children[1])
956           self.fHtml.write(')')
957         else:
958           tuplo = self.visit(tree.children[1])
959         return tuple(tuplo)
960       elif isinstance(child,Token) and self.nasInstrucoes:
961         self.fHtml.write(child.value)
962 
963   def conteudoespecial(self,tree):
964     r = self.visit(tree.children[0])
965     return r
966 
967   def conteudodicionario(self,tree):
968     estrutura = dict()
969     i = 0
970     for child in tree.children:
971       if i == 0 and isinstance(child, Tree):
972         entrada = self.visit(child)
973         chave = entrada[0][1:-1]
974         estrutura[chave] = entrada[2]
975       elif isinstance(child, Tree):
976         if self.nasInstrucoes:
977           self.fHtml.write(',')
978           entrada = self.visit(child)
979           chave = entrada[0][1:-1]
980           estrutura[chave] = entrada[2]
981         else:
982           entrada = self.visit(child)
983           chave = entrada[0][1:-1]
984           estrutura[chave] = entrada[2]
985       i += 1
986     return estrutura
987 
988   def entrada(self,tree):
989     res = []
990     for child in tree.children:
991       if self.nasInstrucoes and isinstance(child,Token):
992         self.fHtml.write(child.value)
993         res.append(child.value)
994       elif isinstance(child,Token):
995         res.append(child.value)
996       elif isinstance(child,Tree):
```

```
997         r = self.visit(child)
998         res.append(r)
999     return res
1000
1001   def conteudo(self, tree):
1002     # print('Entrei no conteúdo de uma estrutura...')
1003     res = list()
1004     i = 0
1005     for child in tree.children:
1006       if i == 0 and isinstance(child, Tree):
1007         r = self.visit(child)
1008         res.append(r)
1009       elif isinstance(child, Tree):
1010         r = None
1011         if self.nasInstrucoes:
1012           self.fHtml.write(',')
1013           r = self.visit(child)
1014         else:
1015           r = self.visit(child)
1016         res.append(r)
1017       i += 1
1018     return res
1019
1020   def chave(self,tree):
1021     if tree.children[0].type == 'NUM':
1022       return int(tree.children[0].value)
1023     elif self.nasInstrucoes and tree.children[0].type == 'STRING':
1024       r = (tree.children[0].value)
1025       return r
1026     elif tree.children[0].type == 'STRING':
1027       r = (tree.children[0].value)[1:-1]
1028       return r
1029     elif tree.children[0].type == 'VAR':
1030       return str(tree.children[0].value)
1031
1032 l = Lark(grammar, start='linguagem')
1033
1034 f = open('codigoFonte.txt', 'r')
1035 input = f.read()
1036
1037 tree = l.parse(input)
1038
1039 data = LinguagemProgramacao().visit(tree)
1040 print('-'*100)
1041 print('Output: ', data)
1042 print('-'*100)
```

funcoesUteis.py

```
1 def criarFicheiro(nome):
2   try:
3     f = open(nome, 'r+', encoding='utf-8')
```

```
    f.truncate(0)
    return f
  except:
    f = open(nome, 'a', encoding='utf-8')
    return f

def preencherInicio(ficheiro):
  conteudo = '''
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Análise Estática</title>
  </head>
  <style>
    .info {
      position: relative;
      display: inline-block;
      border-bottom: 1px dotted black;
      color: rgb(142, 142, 248);
    }
    .info .infotext {
      visibility: hidden;
      width: 200px;
      background-color: #555;
      color: #fff;
      text-align: center;
      border-radius: 6px;
      padding: 5px 0;
      position: absolute;
      z-index: 1;
      bottom: 125%;
      left: 50%;
      margin-left: -40px;
      opacity: 0;
      transition: opacity 0.3s;
    }
    .info .infotext::after {
      content: "";
      position: absolute;
      top: 100%;
      left: 20%;
      margin-left: -5px;
      border-width: 5px;
      border-style: solid;
      border-color: #555 transparent transparent transparent;
    }
    .info:hover .infotext {
      visibility: visible;
      opacity: 1;
    }
    .error {
      position: relative;
      display: inline-block;
```

```
58        border-bottom: 1px dotted black;
59        color: red;
60      }
61      .code {
62        position: relative;
63        display: inline-block;
64      }
65      .error .errortext {
66        visibility: hidden;
67        width: 200px;
68        background-color: #555;
69        color: #fff;
70        text-align: center;
71        border-radius: 6px;
72        padding: 5px 0;
73        position: absolute;
74        z-index: 1;
75        bottom: 125%;
76        left: 50%;
77        margin-left: -40px;
78        opacity: 0;
79        transition: opacity 0.3s;
80      }
81      .error .errortext::after {
82        content: "";
83        position: absolute;
84        top: 100%;
85        left: 20%;
86        margin-left: -5px;
87        border-width: 5px;
88        border-style: solid;
89        border-color: #555 transparent transparent transparent;
90      }
91      .error:hover .errortext {
92        visibility: visible;
93        opacity: 1;
94      }
95    </style>
96    <body>
97      <h2>Análise de código</h2>
98      <pre>
99          <code>'''
100   ficheiro.write(conteudo + '\n')
101
102 def preencherFim(ficheiro):
103   conteudo = '''
104       </code>
105     </pre>
106   </body>
107 </html>'''
108   ficheiro.write(conteudo)
109
110 def criarSegundaPagina(dicionario,ficheiro,cfg,sdg,complexidade):
111   inicio = '''
```

```
<!DOCTYPE html>
<html lang="pt">
<head>
  <link rel="stylesheet" href="w3.css">
  <meta charset="UTF-8">
  <title>Informações</title>
</head>
<body class="w3-container">
'''

#Variáveis e os tipos
ficheiro.write(inicio)
ficheiro.write('\t<h2>Variáveis declaradas e os seus tipos</h2>\n')
ficheiro.write('\t<table class="w3-table-all">\n\t\t<tr>\n')
ficheiro.write('\t\t\t<th>Variável</th>\n')
ficheiro.write('\t\t\t<th>Tipo</th>\n')
ficheiro.write('\t\t</tr>\n')
for k,v in dicionario['decls'].items():
  ficheiro.write('\t\t<tr>\n')
  ficheiro.write('\t\t\t<td>' + k + '</td>\n')
  ficheiro.write('\t\t\t<td>' + v + '</td>\n')
  ficheiro.write('\t\t</tr>\n')
ficheiro.write('\t</table>\n')

ficheiro.write('\t<h2>Outras informações sobre as variáveis</h2>\n')
ficheiro.write('\t<ul class="w3-ul">\n')

naoInicializadas = dicionario['naoInicializadas']
if len(naoInicializadas) > 0:
  ficheiro.write('\t\t<li><b>Variáveis sem inicialização: </b>' + str(
 naoInicializadas) + '</li>\n')
else:
  ficheiro.write('\t\t<li><b>Variáveis sem inicialização: </b>Nenhuma</li>\n')

naoDeclaradas = dicionario['erros']['1: Não-declaração']
if len(naoDeclaradas) > 0:
  ficheiro.write('\t\t<li><b>Variáveis não declaradas: </b>' + str(
 naoDeclaradas) + '</li>\n')
else:
  ficheiro.write('\t\t<li><b>Variáveis não declaradas: </b>Nenhuma</li>\n')

redeclaradas = dicionario['erros']['2: Redeclaração']
if len(redeclaradas) > 0:
  ficheiro.write('\t\t<li><b>Variáveis redeclaradas: </b>' + str(redeclaradas)
 + '</li>\n')
else:
  ficheiro.write('\t\t<li><b>Variáveis redeclaradas: </b>Nenhuma</li>\n')

usadasNaoInicializadas = dicionario['erros']['3: Usado mas não inicializado']
if len(usadasNaoInicializadas) > 0:
  ficheiro.write('\t\t<li><b>Variáveis usadas e não inicializadas: </b>' + str(
 usadasNaoInicializadas) + '</li>\n')
else:
```

```
    ficheiro.write('\t\t<li><b>Variáveis usadas e não inicializadas: </b>Nenhuma
   </li>\n')

  nuncaMencionadas = dicionario['erros']['4: Declarado mas nunca mencionado']
  if len(nuncaMencionadas) > 0:
    ficheiro.write('\t\t<li><b>Variáveis declaradas mas nunca mencionadas: </b>'
   + str(nuncaMencionadas) + '</li>\n')
  else:
    ficheiro.write('\t\t<li><b>Variáveis declaradas mas nunca mencionadas: </b>
   Nenhuma</li>\n')
  ficheiro.write('\t</ul>\n')

  ficheiro.write('\t<h2>Informações sobre as instruções</h2>\n')
  ficheiro.write('\t<ul class="w3-ul">\n')
  ficheiro.write('\t\t<li><b>Total de instruções: </b>' + str(dicionario['
   instrucoes']['total']) + '</li>\n')
  ficheiro.write('\t\t<li><b>Total de atribuições: </b>' + str(dicionario['
   instrucoes']['atribuicoes']) + '</li>\n')
  ficheiro.write('\t\t<li><b>Total de leituras: </b>' + str(dicionario['
   instrucoes']['leitura']) + '</li>\n')
  ficheiro.write('\t\t<li><b>Total de escritas: </b>' + str(dicionario['
   instrucoes']['escrita']) + '</li>\n')
  ficheiro.write('\t\t<li><b>Total de instruções condicionais: </b>' + str(
   dicionario['instrucoes']['condicionais']) + '</li>\n')
  ficheiro.write('\t\t<li><b>Total de instruções cíclicas: </b>' + str(dicionario
   ['instrucoes']['ciclicas']) + '</li>\n')
  ficheiro.write('\t\t<li><b>Total de situações de aninhamento: </b>' + str(
   dicionario['totalSituacoesAn']) + '</li>\n')
  ficheiro.write('\t</ul>\n')

  ficheiro.write('\t<h2>Informações sobre os ifs e os seus aninhamentos</h2>\n')
  ficheiro.write('\t<h4>Níveis de aninhamento dos ifs</h4>\n')
  ficheiro.write('\t<ul class="w3-ul">\n')
  niveis = dicionario['niveisIf']
  for k, v in niveis.items():
    ficheiro.write('\t\t<li><b>Nível ' + str(k) + ': </b>Ifs pela ordem em que
   aparecem no código - ' + str(v) + '</li>\n')
  ficheiro.write('\t</ul>\n')

  alternativas = dicionario['alternativasIfs']
  if len(alternativas) > 0:
    ficheiro.write('\t<h4>Alternativa para os ifs aninhados</h4>\n')
    for a in alternativas:
      ficheiro.write('\t<p>' + str(a) + '</p>\n')

  ficheiro.write('\t<h2>Grafos gerados durante a análise estática</h2>\n')
  ficheiro.write('\t<h4>Control Flow Graph</h4>\n')
  ficheiro.write('\t<img src="' + cfg + '" alt="Control Flow Graph">\n')
  ficheiro.write('\t<h4>System Dependency Graph (lite)</h4>\n')
  ficheiro.write('\t<img src="' + sdg + '" alt="System Dependency Graph">\n')

  ficheiro.write('\t<h4>Complexidade de McCabe do grafo CFG</h4>\n')
  ficheiro.write('\t<p> E( N   arestas) - V( N   nós) + 2 = ' + str(complexidade) +
    '</p>\n')
```

```
203
204 def eDigito(palavra):
205   res = True
206   i = 0
207   size = len(palavra)
208   while res and i < size:
209     if palavra[i] < "0" or palavra[i] > "9":
210       res = False
211     i += 1
212   return res
213
214 def eDouble(palavra):
215   res = True
216   ponto = False
217   i = 0
218   size = len(palavra)
219   while res and i < size:
220     if palavra[i] == ".":
221       ponto = True
222     elif palavra[i] < "0" or palavra[i] > "9":
223       res = False
224     i += 1
225   return res and ponto
226
227 def complexidade_McCabe(arestas,nos):
228   return arestas - nos + 2
```