Engenharia Gramatical TP3

Inês Oliveira Anes Vicente
pg50436

Jorge Miguel Silva Melo
pg50507

Miguel Ângelo Machado Martins
pg50655

4 de junho de 2023

1 Introdução

O objetivo desta terceira fase do projeto era, utilizando no programa já desenvolvido na segunda fase, estudar o comportamento dos programas-fonte com base na construção dos vários *Directed Acyclic Graph* que se usam para estudar o fluxo da execução (controlo) e dos dados (em função das dependências entre as variáveis). Para além disso também fizemos melhoria tanto na *frontend* como nas estruturas de dados.

2 Grafos

Inicialmente, é gerada uma lista de listas através do dicionário gerado pelo interpretador. A lista é uma versão mais "manuseável"do grafo. Cada lista interna representa uma aresta do grafo e tem 2 ou 3 elementos. O primeiro é a origem da aresta, o segundo é o destino da aresta e o terceiro é uma indicação da forma que o destino da aresta deve ter. No caso se ser a forma *default*, não há um terceiro elemento da lista.

A geração desta estrutura é feita no ficheiro *graphsGenerator.py*. Este ficheiro tem uma estrutura simples de perceber:

- **função principal :** percorre o dicionário e encaminha cada parte para a função de escrita respetiva.
- **funções de escrita :** acrescentam arestas ao grafo, ou seja, listas à lista principal.
- **funções de obtenção de string :** percorrem o dicionário até aos seus extremos para formar strings com a informação dispersa no dicionário e direciona-las para as funções de escrita.

A partir dessa lista, são criadas duas estruturas, que serão, depois, utilizadas pelo *frontend*:

- ***dict_sdg :*** Dicionário que contém o grafo SDG em formato *string* e a complexidade *McCabe* do mesmo. O grafo SDG é muito próximo de uma "transcrição"para *string* da estrutura que foi dada como *input*. Ao colocar essa estrutura em *string*, é também calculada a complexidade *McCabe*.
- ***dicts_cfg :*** Lista de dicionários. Cada dicionário na lista contém um grafo CFG em formato *string* para uma parte diferente do programa, assim como a complexidade *McCabe* dessa parte. Cada um dos grafos é obtido de forma semelhante ao grafo SDG, a diferença é que o início do grafo é numa instrução específica, não no início do programa.

3 Mudanças no *frontend*

Na primeira fase do trabalho, o *frontend* do analisador de código dos programas da nossa linguagem já estava bastante completo, tendo uma identação correta e legível e um aspeto visual agradável, com

possibilidade de configuração das cores de diferentes componentes do programa, e *hovers* que avisavam o utilizador se uma variável não tivesse sido declarada ou inicializada.

Além disso, as otimizações a nível das condicionais "SE"já eram efetuadas com sucesso.

Contudo, para esta fase era necessário extender o *HTML* gerado, de forma a que o mesmo pudesse apresentar os grafos gerados pela aplicação, bem como as estatísticas do programa dado como *input*, que na fase anterior eram somente apresentadas no terminal.

Desta forma, o *HTML* gerado foi modificado de forma a ter uma *sidebar* com o analisador de código *per se*, que não necessitou de modificações, e 2 novas páginas: uma para as estatísticas e outra para os grafos.

3.1 Estatísticas

Como supramencionado, na fase anterior o grupo acabou por não colocar as estatísticas como uma das componentes do *HTML* gerado, sendo que na altura da defesa dessa fase, acabamos por nos comprometer a adicionar essa componente nesta fase, algo que não seria complexo de fazer e que enriqueceria o documento gerado pela aplicação desenvolvida.

Assim sendo, cumprimos a promessa e atualmente é possível observar as seguintes estatísticas, cuja informação é proveniente da estrutura de dados *infos*, sobre a qual elaboramos no relatório da fase anterior:

- Variáveis;
- Instruções;
- Aninhamentos;
- Pacotes Importados.

Em baixo temos o aspeto que estas assumem no *HTML* gerado:

Figura 1: Estatística : Variáveis

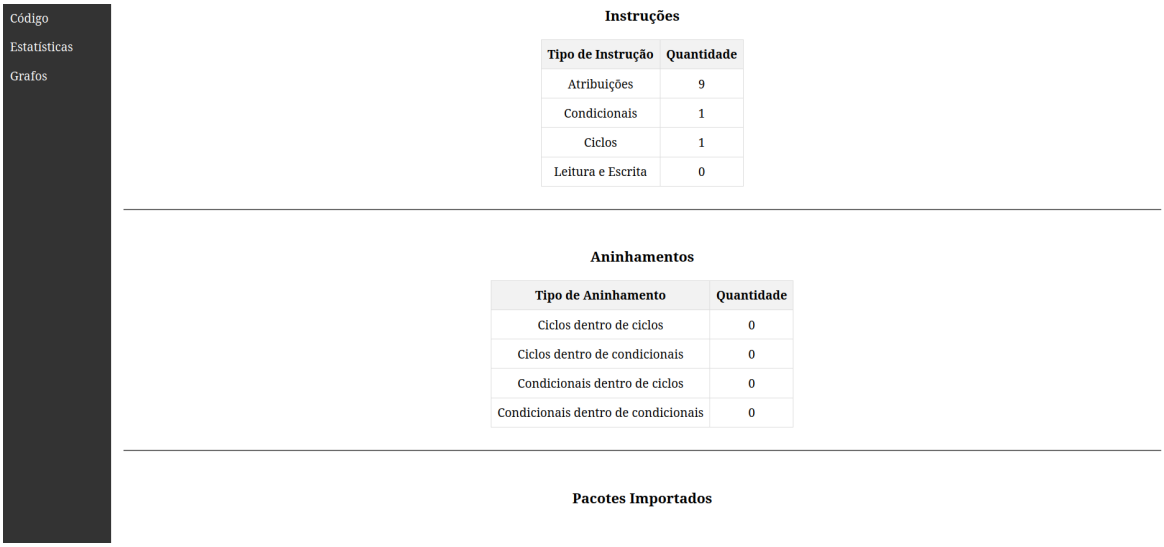

Figura 2: Estatística : Instruções, Aninhamentos e Pacotes Importados

3.2 Grafos

Os grafos gerados são provenientes de estruturas *Digraph* geradas na *backend* do programa e passados para imagem. Os *CFG* são mostrados em formato *slideshow*.

Os exemplos são apressentados abaixo.

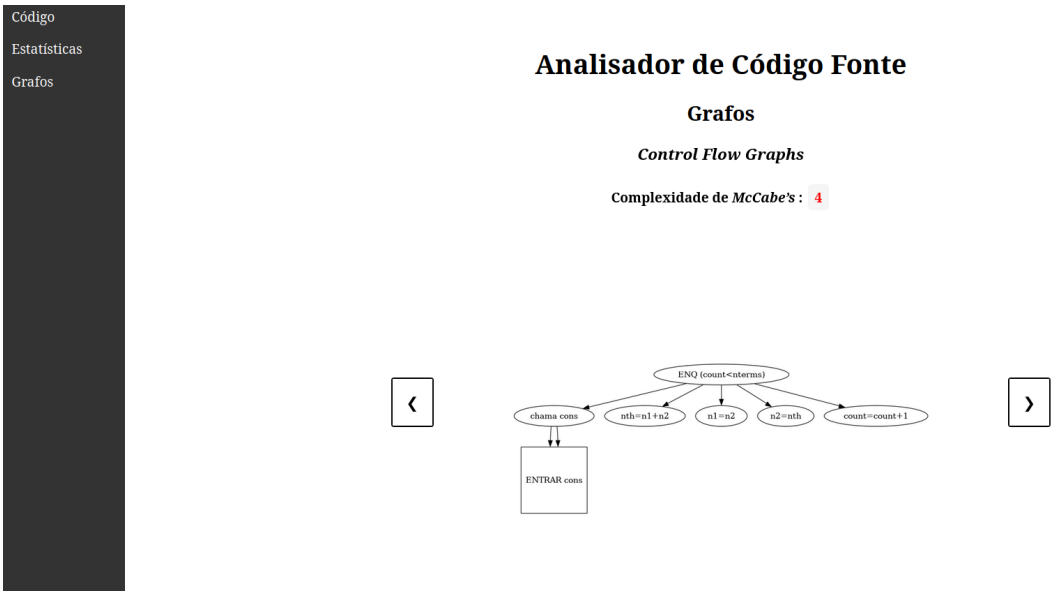

Figura 3: Grafos : *CFG* 1

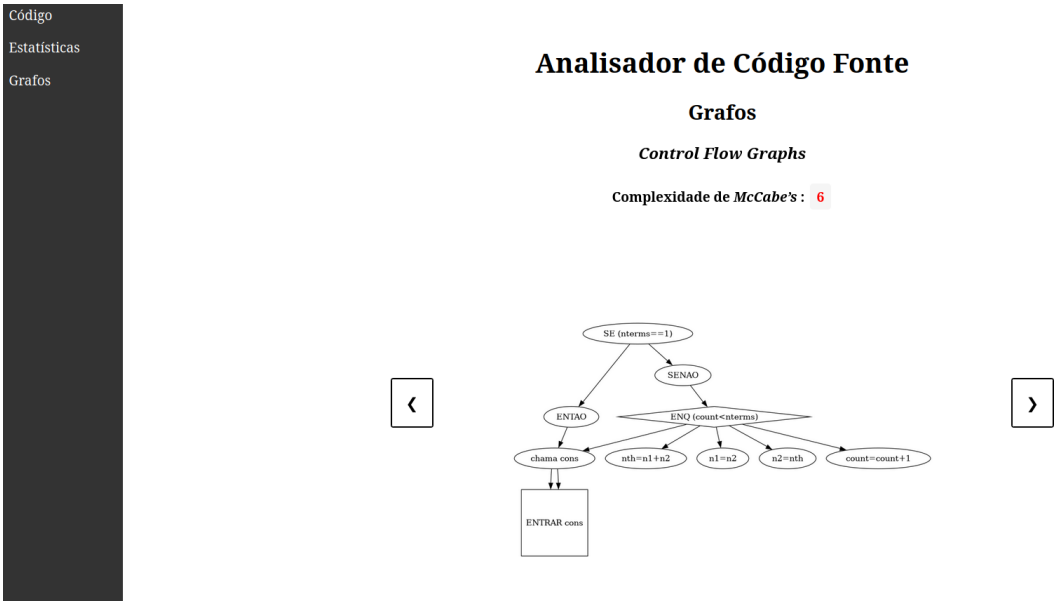

Figura 4: Grafos : *CFG* 2

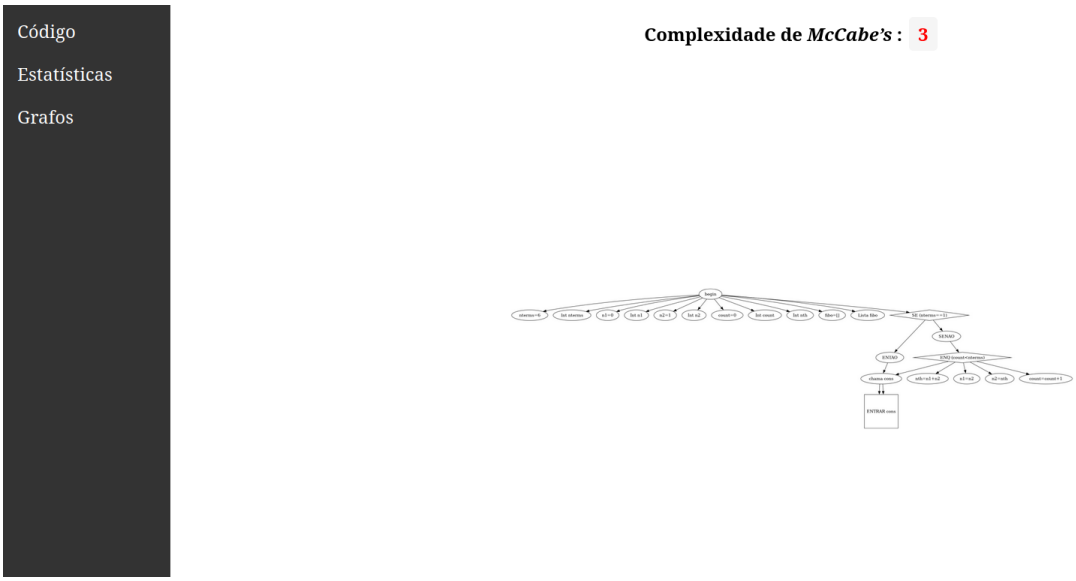

Figura 5: Grafos : *SDG*

4 Mudança das estruturas de dados

Na última fase foi apresentada uma estrutura *infos*, em formato *json*, que tinha, entre outras coisas, informação de todas as variáveis e as suas informações

- **foi_declarada**: *bool* a dizer se a variável foi declarada ou não;
- **foi_inicializada**: *bool* a dizer se a variável foi incicializada ou não;
- **foi_utilizada**: *bool* a dizer se a variável foi utilizada ou não;
- **foi_redeclarada**: *bool* a dizer se a variável foi redeclarada ou não;
- **tipo_de_variavel**: Caso a variável tenha sido declarada, este campo diz o tipo da variável;

- **valores**: Todos os valores que essa variável já teve.

Contudo, essa estrutura apresentava alguns *bugs* na fase anterior, com variáveis a apresentar valores errados e com informações incorretas, o que depois levava a *frontend* a apresentar erradamente algumas dessas informações. Para esta fase essa estrutura foi refeita de forma mais organizada e inteligente, pelo que todos esses *bugs* foram resolvidos e a informação agora é corretamente armazenada.

5 Conclusão

Tendo finalizado este projeto, achamos que conseguimos fazer tudo o que nos foi proposto pela equipa docente. Este trabalho ajudou-nos a consolidar os tópicos dados nas aulas, tais como o módulo *Lark* para a criação e manipulação de gramáticas e grafos de análise estática, bem como ver a sua utilidade num ambiente prático.