Universidade do Porto

Faculdade de Engenharia

FEUP

CFLOW

Compiladores

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Grupo: 3MIEIC04-G25

- Nome: João Paulo Moreira Barbosa
  Número: up201406241
  Nota: 20
  Contribuição: 35%

- Nome: José Luís Pacheco Martins
  Número: up201404189
  Nota: 20
  Contribuição: 35%

- Nome: Miguel Lira Barbeitos Luís
  Número: up201405324
  Nota: 17
  Contribuição: 15%

- Nome: Miriam Cristiana Meireles Campos Gonçalves
  Número: up201403441
  Nota: 17
  Contribuição: 15%

Sumário:

O ***CFlow*** consiste numa aplicação para *desktop* com a intenção de permitir a análise do fluxo de um programa através de uma expressão regular.

O utilizador poderá importar todo um projeto java para a ferramenta ou mesmo escrever código na ferramenta de modo a que este seja avaliado.

O código deverá conter pragmas ***@BasicBlock*** seguidos de um identificador do respetivo bloco, este identificador deverá ser constituído apenas por letras e algarismos, começando por uma letra maiúscula. Estes identificadores podem então ser utilizados na construção de uma expressão regular para avaliar o código.

Execução:

Para executar o programa é necessário instalar o *NodeJS* na máquina. De seguida, na linha de comandos, deverá ser executado o comando 'npm start' no diretório 'CFlow-Electron' (*root* do projeto *electron*).

Tratamento de Erros Sintáticos:

Possíveis erros sintáticos têm origem em 2 situações:

- No *parsing* da expressão regular
- No *geração de código*

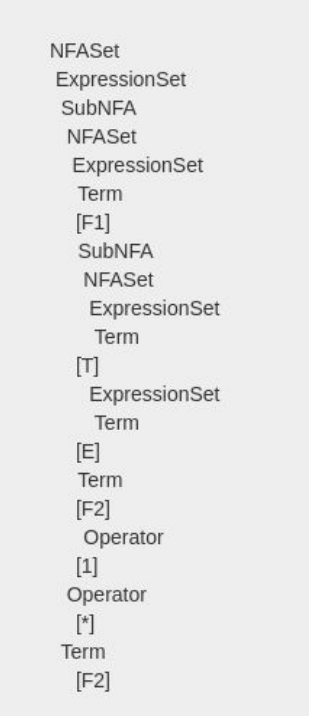
Sendo utilizadas, respetivamente, as ferramentas *JavaCC* e *Kadabra*, os erros sintáticos são detetados por estas de forma automática, sendo só necessário, da nossa parte, saber qual o resultado da sua execução.

Análise Semântica:

Este projeto não implementa qualquer tipo de validação semântica, uma vez que esta não foi considerada relevante em nenhum módulo do projeto em questão.

Representação Intermédia:

Na inserção de uma expressão regular, utilizando a ferramenta *JavaCC*, é gerada uma *HIR* que representa a estrutura base para a geração do *NFA*. Um exemplo de *HIR* do programa é apresentado na figura seguinte:



As representações intermédias do ***CFlow*** possuem três conjuntos importantes: *NFASet*, *ExpressionSet* e *Term*.

- **NFASet:** Este conjunto é sempre a estrutura inicial durante o *parsing*, i.e. todas as *HIRs* geradas no programa começam em *NFASet*. Os filhos na árvore desta estrutura são sempre *ExpressionSet.* Pode também ter associado um *Operator*.

- **ExpressionSet:** Conjunto que engloba sub-elementos. Filhos diretos da árvore podem ser *Term* ou *SubNFA(*utilizado para identificar parênteses*)*, sendo este último o equivalente a um *NFASet*.

- **Term:** Conjunto base da árvore, sendo a única estrutura nas folhas da árvore. Representa o operador da *regex*, podendo ter como filho um *Operator*, que associa uma operação ao conjunto.

Geração de Código:

Para o ***CFlow*** controlar com sucesso o fluxo, é necessário inserção de código no programa do utilizador. Sempre que é encontrado um *@BasicBlock* (na forma de comentário), é inserido código capaz de transitar no autómato gerado pela ferramenta, a partir do conteúdo desse mesmo comentário.

Além disso, são também geradas duas linhas, no início e no fim do programa (na função *main*), para gerar o autómato e produzir as estatísticas, respetivamente. A linha inserida no fim do *main* prevê se existe uma linha de *return*, inserindo o código de acordo com a situação.

A inserção de código é auxiliada pela ferramenta *Kadabra*, fornecida pelo professor nas aulas práticas, com a utilização de ficheiros *lara* gerados dinamicamente.

Visão Geral:

A abordagem seguida pela grupo assenta essencialmente em 4 pontos que serão descritos de seguida:

- **Parse da expressão regular:**
  - A gramática desenvolvida aceitas expressões regulares com os principais operadores como *****, **+**, **?**, **.** (ponto), **{num,}**, **{num,num}**, **{num}**. Todos os operadores devem estar associados a identificadores, ou até mesmo parênteses - que foram também implementados para permitir ao utilizador definir níveis de isolamento. De modo a facilitar o desenvolvimento do projeto, foi retirado o máximo partido da classe SimpleNode de forma a estruturar internamente a expressão regular da melhor maneira possível.

- **Tradução da Regex para um DFA:**
  - Uma vez realizado o *parse* e o armazenamento de toda a expressão regular, foi implementada uma tradução do NFA de forma recursiva, em que os termos base da nossa representação (**Term**) geram pequenos NFAs tendo em conta os respetivos operadores e identificadores. Num nível intermédio (**ExpressionSet**) o NFA é gerado pela interligação dos NFAs dos nós filho com ligações *epsilon* por *append*, i.e. os filhos são ligados de forma sequencial. Por fim, no último nível (**NFASet**) todos os NFAs filhos são interligados e caso existam operadores, são também clonados de forma a gerar o NFA

pretendido. A interligação neste nível funciona em disjunção (todos os filhos se ligam a um nó comum, por caminhos independentes). Uma vez gerado o NFA, a tradução para DFA é realizada tendo em conta os métodos lecionados na unidade curricular de Teoria da Computação integrada no 2º ano do MIEIC@FEUP.

- **Integração do código da ferramenta em código do utilizador:**
  - Para analisarmos se o código do utilizador verifica a expressão regular introduzida, foi então necessário integrar código em 3 situações:
    - **Início da execução do projeto do utilizador:** Código responsável pela inicialização do DFA.
    - **A seguir a todos os pragmas BasicBlock:** Código responsável pelas transições no DFA.
    - **Antes do fim da execução do projeto do utilizador:** Código responsável pela geração de todas as estatísticas referentes a execução do programa assim como o seu resultado.

    A introdução de código foi efetuada com recurso à ferramenta Kadabra, tal como referido em '**Geração de Código**'.

- **Criação de um interface gráfica:**
  - De modo a facilitar a interação do utilizador com o produto, foram utilizadas duas ferramentas externas **electron** e **vis.js**. A primeira foi utilizada para criar uma interface apelativa e atual, de uma forma mais ágil e menos dispendiosa em termos de tempo - uma vez que a interface não era de todo o foco do projeto. A segunda ferramenta auxiliou na visualização dos autómatos na interface.

Testes:

Ao longo do desenvolvimento do projeto foram realizados vários testes referentes inicialmente à avaliação dos autómatos gerados através das regex de forma a apurar todos os detalhes que poderiam estar a ser mal interpretados. Depois da integração com o kadabra, a validação passou a ser efetuada em pequenos projetos teste desenvolvidos pelo grupo, por forma a validar toda a interação entre o projeto do utilizador e a nossa implementação do ***CFlow***.

Distribuição de Tarefas:

- ➔ João Barbosa:
  Responsável pelo *parser* da expressão regular, e pela configuração do projeto para utilizar *NodeJS, Electron* e *Vis.js*. Também responsável pela estruturação inicial da lógica do programa.

- ➔ José Martins:
  Responsável pela geração do nfa e respetiva tradução para dfa a partir da expressão regular, geração de scripts e do protocolo de comunicação entre a gui(eletron) e o backend(java).

- ➔ Miguel Lira:
  Responsável pelo *handling* dos erros.

- ➔ Miriam Gonçalves:
  Responsável pela interface e *design* do programa.

Pros:

- Analisar o fluxo de código, de uma forma automática e intuitiva.
- Visualização dos *NFA*s e *DFA*s, referentes à expressão regular fornecida.
- Otimização dos *NFA*s gerados, com remoção de transições *epsilon* redundantes.
- Visualizar todas as transações efetuadas no autómato gerado, aquando da execução do código.
- Interface simples e intuitiva.
- Grande variedade de opções, para criar expressões regulares.
- Possibilidade de ver o novo código, gerado pelo ***CFlow***.
- *Feedback* visual, em caso de erro.

Cons:

- O *feedback* visual dado ao utilizador, em caso de erro, pode não ser muito específico.