INE5421 - Linguagens Formais e Compiladores

Trabalho T1 - Manipulador de LR representadas por AF e ER

Caique Rodrigues Marques
c.r.marques@grad.ufsc.br

Fernando Jorge Mota
contato@fjorgemota.com

1 Introdução

O software possibilita a verificação e a execução de operações relacionadas a autômatos finitos e expressões regulares, através de simples operações é possível verificar a equivalência de dois autômatos finitos, o autômato finito equivalente a uma dada expressão regular, a diferença entre dois autômatos finitos e entre outras operações que serão detalhadas no decorrer deste relatório.

2 Uso do Programa

Ao iniciar o programa, ilustrado na figura 1, uma janela conterá uma aba com a mensagem de boas-vindas com um pequeno resumo de quais passos básicos seguir para a execução do software. Na tela inicial do software também contém três menus na parte superior da janela: `File`, `Operations` e `Tab`, a seguir será detalhado mais sobre os três menus.

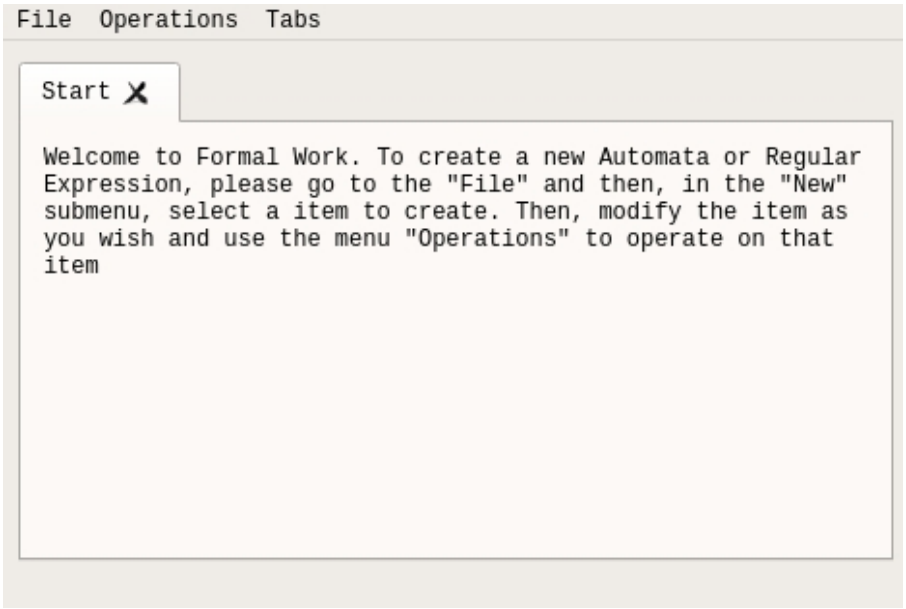

Figura 1: Tela inicial do software

- `File`: este menu contém duas opções: `New` e `Exit`, a primeira opção permite a criação de um novo conjunto regular, ao selecionar a opção, um novo submenu aparece possibilitando a criação de um autômato finito (`Finite Automata`) ou de uma expressão regular (Regular Expression). Clicando em qualquer um, uma nova janela aparecerá fornecendo um campo digitável para a escrita do nome do conjunto regular a ser criado; a segunda opção é para sair do programa.
- `Operations`: Permite realizar as operações entre os conjuntos regulares, tais operações são detalhadas na seção relacionada à implementação (4). Este menu contém um submenu divido por uma linha, onde a parte superior corresponde às operações envolvendo autômatos finitos e a outra envolvendo operações com expressões regulares. Vale citar que dada uma expressão regular, ela será transformada em um autômato finito via algoritmo de Simone (4.2.2) e, assim, conseguirá realizar a todas as operações relacionadas a autômatos finitos.

- `Tab`: Menu relacionado a aba em que se está trabalhando, contendo duas opções: `Change Name` e `Duplicate`. A primeira opção permite a mudança de nome da aba, obviamente, tal mudança só será aceita se for um nome único, que não tenha sido utilizado em outra aba; a segunda opção permite a duplicação de conteúdo, ou seja, o conteúdo da aba é copiado para uma nova, com um outro nome que pode ser escolhido pelo usuário.

3 Operações

3.1 Operações com autômatos finitos

O foco do software está na manipulação de expressões regulares e autômatos finitos, portanto, como já descrito na seção anterior (2) o menu `Operations` contém as operações para manipulação dos conjuntos regulares. As operações possíveis para autômatos finitos são:

1. `Union` (União)
2. `Intersection` (Interseção)
3. `Difference` (Diferença)
4. *`Complement` (Complemento)
5. `Containment` (Contenção)
6. `Equivalence` (Equivalência)
7. *`Check if accept` (Checar se aceita)
8. *`Determinize` (Determinização)
9. *`Minimize` (Minimização)

As operações com * são operações que funcionam com apenas um autômato como entrada, ou seja, não requisitam outro autômato ou expressão regular e simplesmente geram a entrada solicitada. Note que todas as operações exigem autômatos finitos ou expressões regulares válidas, caso contrário, o software irá destacar em amarelo o problema com o autômato e, ao passar o cursor na região em destaque, na parte inferior da janela irá detalhar qual o problema relatado. A figura 2 ilustra dois avisos diferentes notados pelo software.

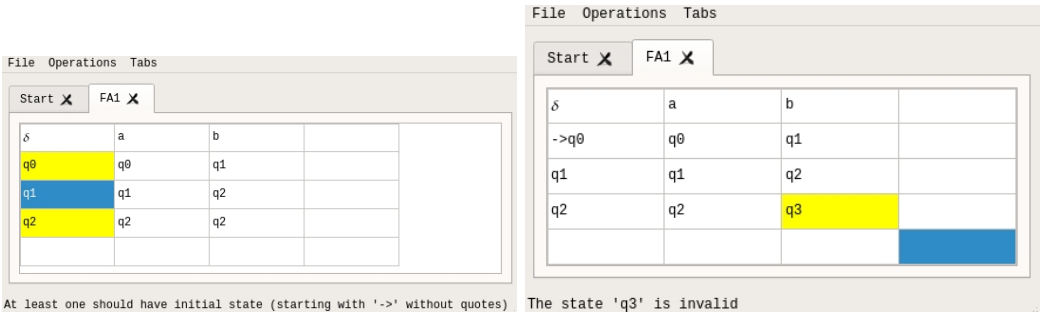

Figura 2: Dois exemplos de avisos apontados pelo software. À esquerda está o aviso da falta de um estado inicial na criação de um autômato, à direita descreve sobre a inexistência de um estado.

Com exceção da operação de número 7 da lista, todas as operações realizadas explicitam os passos usados para a geração da saída especificada, na figura 3 ilustra o resultado de uma interseção entre a expressão regular $(a|b)^*b^+(c)^?$ e o autômato finito de teste gerado ao iniciar o software. Cada passo intermediário apresentado pelo software vem acompanhado de um botão `Move to tab`, que permite que o autômato em questão seja copiado para um nova aba e, assim, seja trabalhado em particular.

3.2 Operações com expressões regulares

A única operação possível que envolve diretamente expressões regulares está na transformação delas em um autômato finito. Assim, tendo um autômato equivalente à expressão regular, é possível realizar as operações listadas na seção anterior.

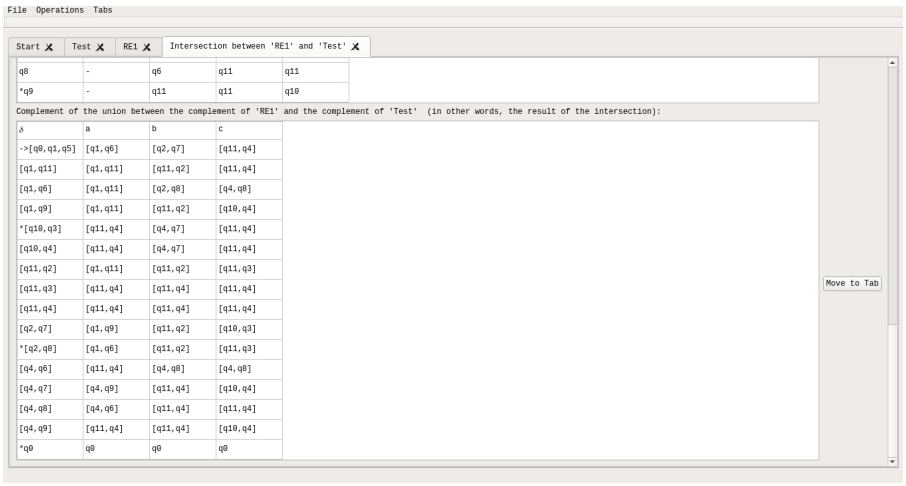

Figura 3: Resultado da interseção de uma expressão regular com um autômato finito

4 Implementação

4.1 Autômatos Finitos

Todas as operações relacionadas aos autômatos finitos estão explicitadas na classe `FiniteAutomata`. A seguir, um detalhamento das operações relacionadas aos autômatos finitos.

4.1.1 Propriedades básicas de autômatos finitos

O conjunto de autômatos finitos regulares são fechados a quatro operações: união, complemento, concatenação e interseção. Tais operações básicas possibilitam a realização de outras operações como a diferença, equivalência e contenção. A seguir é apresentado como o software consegue realizar tais operações partindo das propriedades básicas.

- **Diferença**: A diferença de dois autômatos finitos regulares T1 e T2 é equivalente a $T1 \cap \overline{T2}$ que é equivalente a $\overline{\overline{T1} \cup T2}$ (prova por teorema de De Morgan);
- **Interseção**: A interseção de dois autômatos finitos regulares T1 e T2 é equivalente a $\overline{\overline{T1} \cup \overline{T2}}$ (prova por teorema de De Morgan);
- **Contenção**: Dados dois autômatos finitos quaisquer T1 e T2, $T1 \subseteq T2 \leftrightarrow T1 \cap \overline{T2} = \varphi$;
- **Equivalência**: Dados dois autômatos finitos T1 e T2 quaisquer, $T1 = T2 \leftrightarrow T1 \subseteq T2 T2 \subseteq T1$.3

4.1.2 Determinização de um autômato finito

Um autômato é dito não determinístico se ele possui pelo menos um estado cujas transições levam a dois estados diferentes por um mesmo símbolo do alfabeto. A determinização de um autômato finito não determinístico acontece definindo cada a transição de cada estado e, quando um símbolo leva a dois estados diferentes, estes são unidos, formando um só - tais etapas são repetidas ate que todas as transições do autômato finito sejam determinísticas.

4.1.3 Minimização de um autômato finito

Um autômato finito é dito mínimo se ele não possui estados mortos, não possui estados inalcançáveis e não possui estados equivalentes, assim, para a minimização de um autômato qualquer as três condições devem ser seguidas.

A remoção de estados mortos é feita a partir da verificação de cada estado, se um estado não possui transições de saída e não é estado final, então, ele é considerado morto e é removido, retornando um novo autômato finito sem estados mortos. Para remoção de estados inalcançáveis é feita uma varredura no autômato, partindo do estado inicial seguindo todas as transições possíveis, o novo autômato é retornado contendo apenas os estados que foram alcançados por essa varredura.

Para o último passo para a minimização de um autômato finito é a remoção dos estados equivalentes. Como primeiro passo, os estados são divididos em duas classes (grupos) de equivalência: de estados finais e de estados não finais, logo em seguida, cada estado tem suas transições verificadas, se um ou mais estados do mesmo

tipo "'apontam"para o mesmo grupo, então eles são agrupados numa mesma classe de equivalência. Após, novamente cada estado é avaliado e se a condição novamente for verdadeira em relação ao então número de classes de equivalência, novas são formadas, assim segue sucessivamente até que todos os estados equivalentes estejam devidamente agrupados. Esses novos grupos formam um único estado e, assim, é retornado o autômato finito sem estados equivalentes.

4.2 Expressão Regular

Todas as operações relacionadas às expressões regulares estão explicitadas na classe `RegularExpression`. A seguir, as operações relacionadas às expressões regulares.

4.2.1 Árvore associada

Dada uma expressão regular, sempre é possível associá-lo a uma árvore binária, esta é formada a partir das operações de menor para maior prioridade. A árvore associada à expressão regular acaba sendo útil para a obtenção do autômato finito associado à mesma expressão regular, usando-se o algoritmo baseado no método de Simone que será detalhado na subseção posterior.

Para a obtenção da árvore associada à expressão regular, primeiro a expressão em si é normalizada, ou seja, todas as operações se tornam explícitas, por exemplo, a expressão regular $(ab)^*(ba)^+$ possui forma normalizada como $(a.b)^*.(b.a)^+$. A seguir, a expressão regular é examinada e é coletada a operação de menor prioridade, da esquerda para a direita e fora de parênteses, como nó raiz da árvore, disto, também é definido os nós filhos desse nó raiz - no exemplo citado, "."será a raiz e as subexpressões "$(a.b)^*$"e "$(b.a)^+$"serão os nós filhos à esquerda e à direita, respectivamente, de ".".

O algoritmo de obtenção da árvore associada é recursivo, assim, as subexpressões serão analisadas da mesma forma descrita antes. O algoritmo para quando não há mais subexpressões para analisar.

4.2.2 Método de Simone

O método de Simone possibilita a transformação de uma expressão regular qualquer a um autômato finito associado, tal autômato resultante não necessariamente é mínimo. A execução deste algoritmo começa com a obtenção de uma árvore de Simone associada à expressão regular, os resultados gerados pelo algoritmo serão provindos por informações obtidas nesta árvore; em seguida, a partir da árvore, é coletada as composições que são formadas partindo de cada nó folha, estas composições serão úteis posteriormente; o primeiro percorrimento da árvore é feito, assim, é gerado o primeiro estado (normalmente nomeado como "`q0`") e a primeira composição de nó folhas deste estado inicial, disto é possível inferir por quais símbolos `q0` terá transições.

Para os estados posteriores, não é necessário nenhum percorrimento à arvore de Simone, pois, com as composições alcançadas por nós folhas coletadas no começo do algoritmo, é necessário apenas detalhar a composição do estado partindo das composições alcançáveis já conhecidas. Assim, durante esses passos, é criado o estado em questão e suas transições já são definidas a partir de suas composições. Vale citar que o algoritmo já trata de casos onde, se a composição de dois estados forem exatamente iguais, um é automaticamente descartado evitando a inclusão de estados equivalentes (quaisquer referências a equivalentes, sempre serão associadas ao primeiro estado que teve tal composição). Após todo o processamento da composição dos estados, resultará no autômato finito equivalente, que será o retorno deste algoritmo.

Uma observação em questão de implementação, o algoritmo também trata casos onde o método pode resultar em *loops* infinitos, como por exemplo as expressões regulares $((a^*)^*)$ e $(1|0)^?((10)^*(01)^*)^*(1|0)^?$. Essa verificação é feita usando uma fila de ações já realizadas em um nó (rotinas de subida ou descida), portanto, uma ação não é repetida duas vezes por um mesmo nó.

5 Testes realizados

Toda a validação das estruturas que compõem o programa foram feitas usando testes unitários, para isto foi utilizado o framework gtest[1]. No total, foram realizados 49 testes unitários para autômatos finitos e 22 testes unitários para expressões regulares, muitos desenvolvidos de maneira iterativa conforme os métodos foram sendo criados e problemas foram encontrados. Além disso, um serviço denominado Travis CI[2] testou cada commit realizado no projeto, compilando e realizando os testes em plataformas como o Linux e o macOS, de forma a garantir que problemas fossem encontrados rapidamente no código.

[1] Google Test - Framework para testes de códigos em C++
[2] Travis CI - Serviço de integração contínua