UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
INE5421 – LINGUAGENS FORMAIS E COMPILADORES

TRABALHO PRÁTICO I

Makhles Reuter Lange

Junho de 2017

Funcionamento do Programa

O programa disponibilizado foi feito na linguagem Java e é composto dos seguintes arquivos (que se encontram no subdiretório *dist*):

- RegexAnalyser.jar
- RegexAnalyser.bat

Para rodar o programa, basta ter uma JRE do Java instalada no computador (pelo menos a versão 7 é aconselhável) e executar o arquivo RegexAnalyser.bat. O arquivo de batch foi criado apenas para a execução do programa no SO Windows. Ele possui o seguinte comando:

java -jar RegexAnalyser.jar

Para a execução do programa em um unix*, basta abrir um terminal e navegar até a pasta onde está o arquivo RegexAnalyser.jar e executar o mesmo comando supracitado.

Operações com Expressões Regulares

Para inserir uma nova expressão regular, basta pressionar o botão `New Regex`, logo abaixo da lista de expressões regulares (inicialmente vazia). Ao ser pressionado, no painel central superior irá aparecer uma área de texto que pode ser utilizada para a entrada das expressões. Se algum símbolo não permitido for inserido, uma mensagem de erro é mostrada em uma caixa de diálogo. Pressionando-se o botão `Add`, e não havendo símbolos não permitidos, a expressão é adicionada à lista de expressões com o nome "Regex X", onde X representa o número de criação da regex.

Selecionada uma expressão regular, pode-se obter o autômato finito determinístico obtido a partir do método de De Simone pressionado-se o botão `Convert to DFA`. Neste momento, o painel *Output* será atualizado com o autômato obtido. Este autômato é adicionado à lista de autômatos (lado direito da interface) com o nome "FA Y from Regex X", onde Y é o seu número de criação.

Para editar o autômato recém-criado, deve-se selecionar esse autômato na lista de autômatos, operação que atualiza o painel de *Input*. Na edição de autômatos já criados (ou seja, não é um novo autômato), pode-se modificar os símbolos do vocabulário e os estados, porém o tamanho da tabela que é mostrada é fixa (inexperiência com o uso de tabelas no Java). Após editado, deve-se pressionar o botão `Add` e um novo autômato será incluído na lista de autômatos.

Havendo pelo menos duas expressões regulares na lista de expressões regulares, é possível verificar a equivalência das linguagens que elas denotam. Para tal, deve-se selecionar uma das expressões da lista, segurar a tecla `<Ctrl>`, segurar uma segunda expressão e clicar no botão `Equivalence`. Será mostrado em uma caixa de diálogo o resultado, *e.g.*, "`Regex 0 ⊆ Regex 1`".

A implementação da verificação de equivalência de duas regexes se encontra no método `checkEquivalenceOfRegularLanguages()` da classe Controller. Finalmente, para excluir uma expressão regular, deve-se pressionar no botão `Remove` logo abaixo da lista de expressões regulares.

Operações com Autômatos

A criação de autômatos é feita pressionando-se o botão `New automaton` logo abaixo da lista de autômatos. Uma tabela será mostrada no painel *Input*. O seu preenchimento deve seguir o padrão mostrado na Figura 1 abaixo.

Embora tenha-se disponibilizado diversas colunas e linhas para serem preenchidas, no exemplo dado são consideradas apenas as células entre A0 – A3 e F0 – F3. Ou seja, a partir do momento em que uma célula estiver em branco após o vocabulário, as demais células na mesma

linha não serão mais consideradas. Em outras palavras, o símbolo ‘d’ não faz parte do vocabulário. O mesmo acontece com os estados definidos na coluna C. Neste caso, são considerados apenas os estados até a célula C3 (o estado E não pertence ao autômato). Células não preenchidas, e.g., D2, serão consideradas como ‘-’ pelo programa.

Além disso, deve-se ter pelo menos um estado inicial e um estado de aceitação, representados pelos símbolos ‘->’ e ‘*’, respectivamente (vide colunas A e B). Finalmente, transições não-determinísticas devem utilizar-se de uma vírgula para separar os estados, como é o caso na célula D1.

A	B	C	D	E	F	G	H
Initial	Accept...	δ	a	b	c		d
->		A	A,B	B	C		
		B		A	C		
	*	C	C	A	B		
		E					

Figura 1 – Preenchimento da tabela que representa um autômato.

Após preenchido, deve-se pressionar o botão `Add` para adicionar o autômato à lista de autômatos. Preenchimentos de símbolos não-permitidos ou de estados que não foram definidos na coluna de estados (coluna C) incidirão em uma mensagem de erro numa caixa de diálogo.

As operações possíveis sobre autômatos estão representadas nos diversos botões que se encontram abaixo da lista de autômatos. Para a determinização, deve-se pressionar o botão `NFA to DFA`. Para a minimização, o botão `Minimise`. Para as operações ditas de conjunto sobre os autômatos, deve-se utilizar os botões `Complement`, `Union`, `Intersection` e `Difference`. Com exceção do complemento de um autômato, as demais necessitam que dois autômatos estejam selecionados. Procede-se como no caso da equivalência entre expressões regulares, com a diferença de que não será mostrado uma caixa de diálogo com o resultado das operações, mas novos autômatos serão adicionados à lista de autômatos com nomes representativos das operações. Algumas das operações irão gerar diversos autômatos, como é o caso da diferença e da intersecção entre autômatos, pois foram implementadas “formalmente”.

Finalmente, é possível remover um autômato da lista através do botão `Remove`.

Implementação

O programa foi feito seguindo o padrão de design MVC (Model View Controller). Neste caso, há apenas uma classe na *view* chamada RegexAnalyser. No *controller*, criou-se a classe Controller. Nela estão implementadas todas as operações sobre autômatos. O *model* inclui as classes Automaton, State, RegExParser e RegExTree:

- **Automaton** – classe que representa um autômato finito (determinístico ou não). As transições são representadas como um `Map<State, List<State>>`. Aqui, `List<State>` representa cada um dos estados aos quais se pode transitar através de um

símbolo de entrada do vocabulário. Essa lista está ordenada de acordo com a lista de símbolos do vocabulário (`List<String>`).

- **State** – classe que representa um estado do autômato. Escolheu-se dar, ao invés de apenas um rótulo, um conjunto de rótulos para o estado, para que fosse possível representar, dessa forma, um autômato não-determinístico.
- **RegExParser** – é a classe responsável por analisar as expressões regulares e transformá-las em uma árvore de De Simone.
- **RegExTree** – é a classe que representa a árvore utilizada no método de De Simone. Nela está implementada a obtenção do DFA.

Além destas classes, há ainda algumas classes que foram utilizadas como Exceções e uma classe de testes chamada TestAutomaton que se encontra no subdiretório *test*.