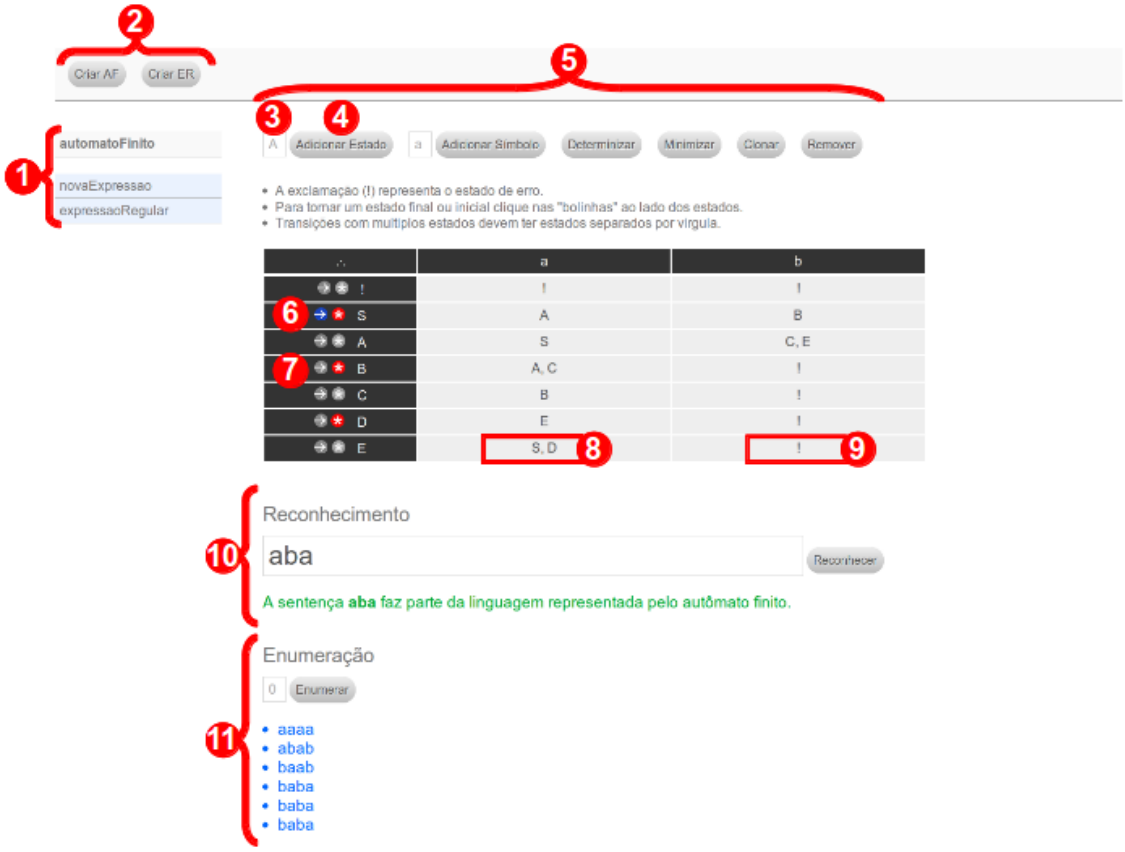
INE 5421: Trabalho 1
Lucas Pereira Da Silva (10100754)
Renan Oliveira Netto (10103126)

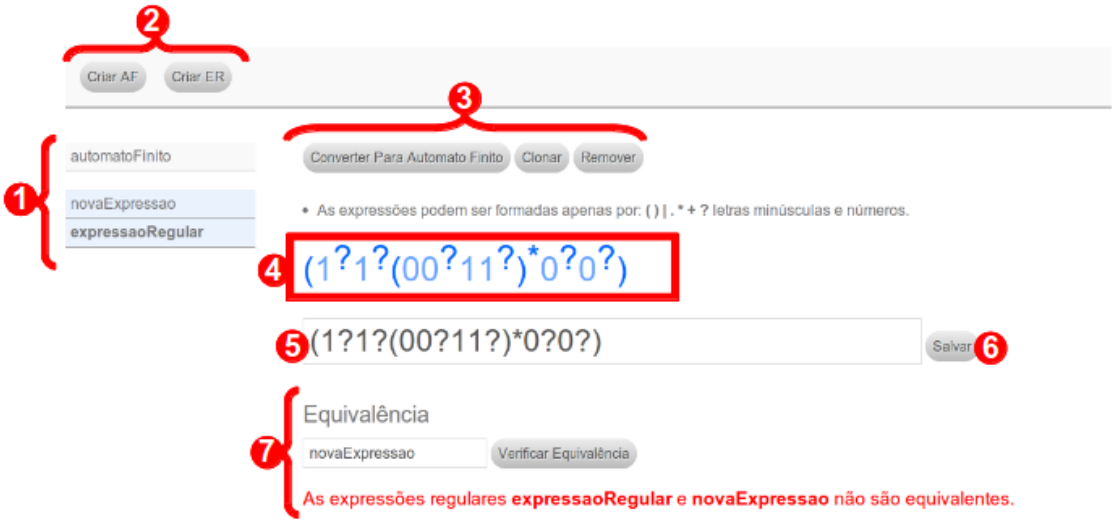
Relatório

O trabalho foi desenvolvido utilizando a linguagem de programação JavaScript. A escolha da linguagem se deu pelo fato do JavaScript ser uma linguagem portável e que pode ser executada por qualquer navegador Web. Assim, o programa desenvolvido poderá ser executado em qualquer plataforma.

Para rodar o programa basta extraír o arquivo compactado enviado junto com esse trabalho e abrir o arquivo `regulares.html` em um navegador Web. Veja abaixo duas imagens do programa mostrando suas funcionalidades:



1. Listas de autômatos finitos e expressões regulares.
2. Botões para criação de autômatos finitos e expressões regulares.
3. Caixa de texto para adição de estado.
4. Botão para adição de estado.
5. Operações disponíveis para autômatos.
6. Estado S marcado como final e inicial.
7. Estado B marcado como final.
8. Transição não determinística
9. Transição para estado de erro.
10. Reconhecimento de sentenças.
11. Enumeração de sentenças de tamanho n.



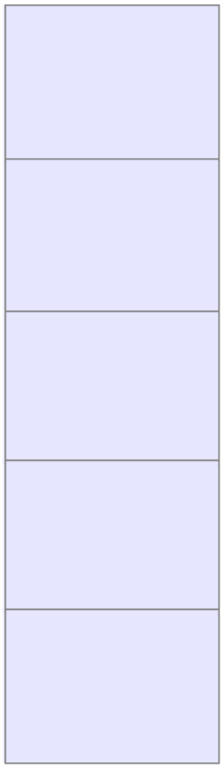
1. Listas de autômatos finitos e expressões regulares.
2. Botões para criação de autômatos finitos e expressões regulares.
3. Operações disponíveis para expressões regulares.
4. Expressão regular.
5. Expressão regular editável.
6. Botão salvar expressão regular.
7. Transição para estado de erro.
8. Verificação de equivalência entre expressões regulares.

Iniciamos a implementação do trabalho pelas operaçoes envolvendo os autômatos finitos. Primeiramente foi feita a criação, edição e leitura de autômatos finitos. Posteriormente, partimos para o desenvolvimento das operações básicas: determinização, remoção de estados inalcançáveis, remoção de estados mortos e junção de estados equivalentes. A implementação dessas operações foi relativamente trivial e os algorítimos foram os mesmos vistos em aula. Dentre as operações mencionadas, a implementação que apresentou a maior dificuldade foi a junção de estados equivalentes, principalmente na tarefa de determinar a classe de equivalência de um determinado estado.

Após a implementação das operações envolvendo autômatos finitos, iniciamos o desenvolvimento das expressões regulares, começando pela criação, edição e leitura. Após isso, iniciamos a parte que consideramos a mais difícil do trabalho: a conversão de expressões regulares para autômatos finitos. Utilizamos o método De Simone e para facilitar a implementação dividimos a o método em duas partes: a criação da árvore costurada e a criação da tabela de composição.

Para a criação da árvore, utilizamos a forma infixada da expressão regular. Boa parte da literatura cita a utilização da forma pós-fixada para a análise de expressões regulares, porém decidimos fazer nossa implementação utilizando a própria forma infixada. Para isso, temos dois elementos básicos, uma pilha e a uma árvore. Na pilha colocamos símbolos e operações que ao serem consumidos são transformados em nodos da árvore. Como utilizamos a forma infixada, então o responsável por consumir valores da pilha são os símbolos e operações unárias. Segue abaixo um exemplo mostrando um caso de reconhecimento de uma expressão regular de acordo com nossa implementação:

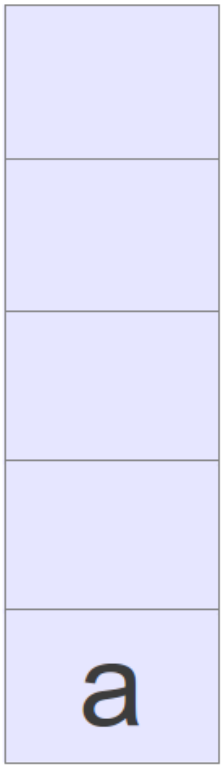
a|bc|de*f



Pilha **Árvore**

Pilha inicia vazia e árvore sem nenhum nodo.

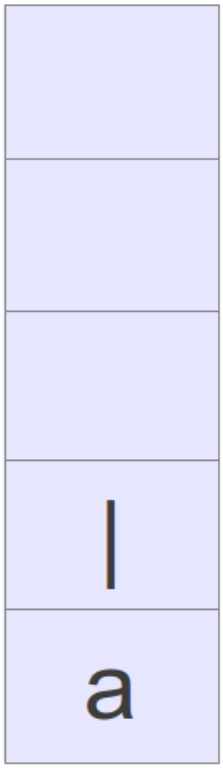
a|bc|de*f



Pilha **Árvore**

Como a pilha está vazia o primeiro símbolo é apenas colocano nela.

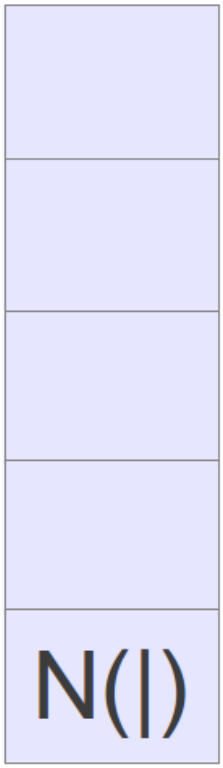
a|bc|de*f



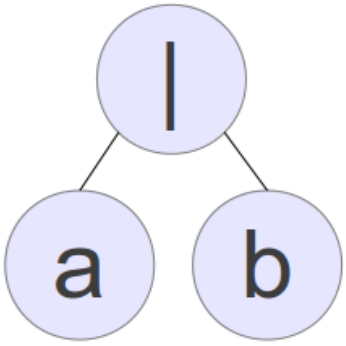
Pilha **Árvore**

| é apenas colocado na pilha. A árvore ainda continua sem nenhum nodo, pois, nenhuma operação foi efetivamente concluída.

a | **b**c | de*f



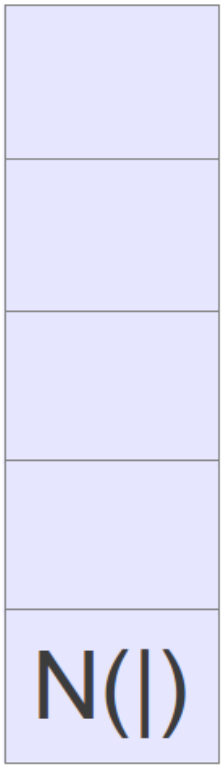
Pilha



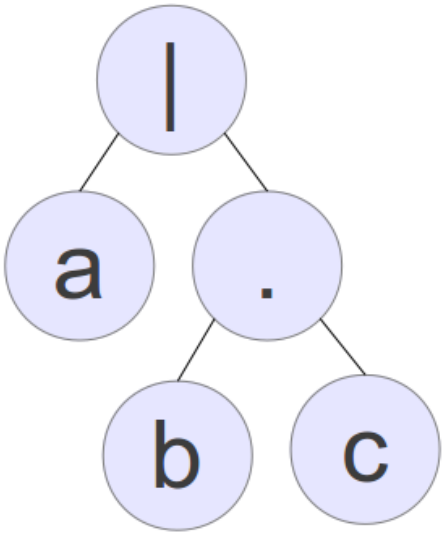
Árvore

Ao analisar **b**, é verificado que a pilha contém uma operação pendente. O símbolo **a** e a operação | são consumidos e deixam na pilha o nodo criado que representa a operação.

a|bc|de*f



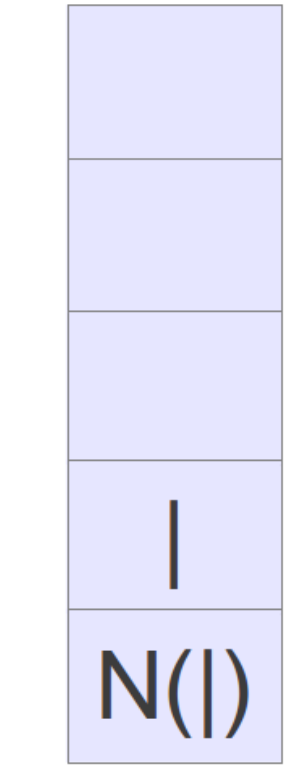
Pilha



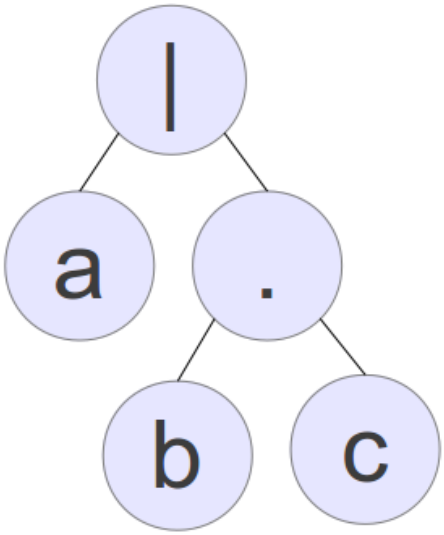
Árvore

Ao analisar **c**, é verificado que a pilha contém um nodo de uma operação. Como dois símbolos justapostos representam uma operação de concatenação, então o nodo **N(|)** é consumido. Outro fator é que como a operação **.** possui uma ordem de precedência maior que a operação **|**, então, o nodo **N(.)** será filho de **N(|)** e **N(|)** será mantido no topo da pilha, pois, é o nodo raiz.

a|bc|e*f

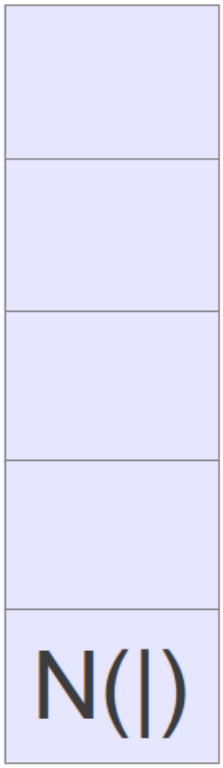


Pilha

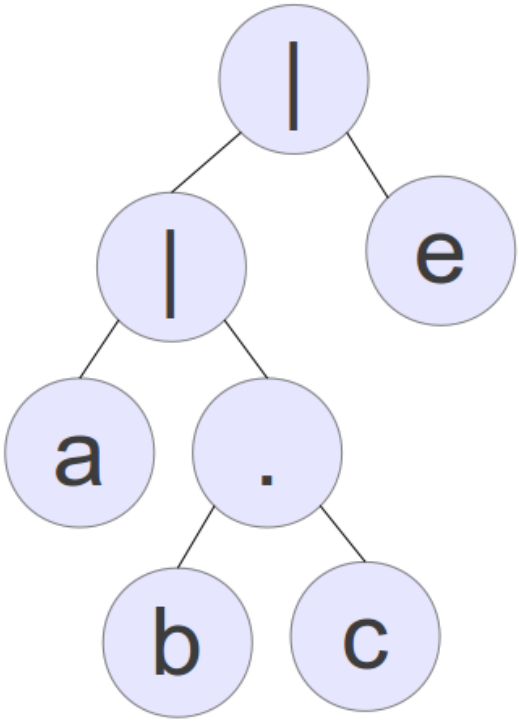


Árvore

a|bc|e*f



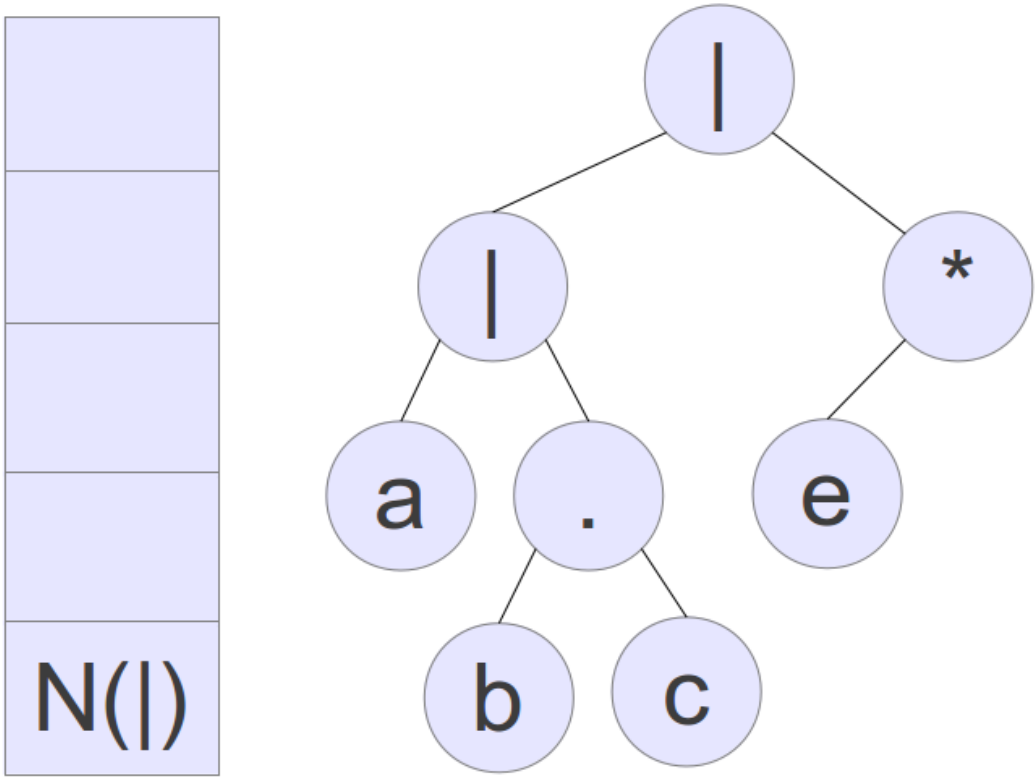
Pilha



Árvore

O símbolo **e** é lido e verifica que há uma operação pendente. Essa operação será a união do nodo já existente na pilha **N(|)** com o símbolo **e** lido. Note que como a operação pendente possui mesma ordem de precedência a operação **N(|)** que está na pilha, então a árvore é construída para cima. Sendo assim o **N(|)** que fica no topo da pilha é o da nova operação criada.

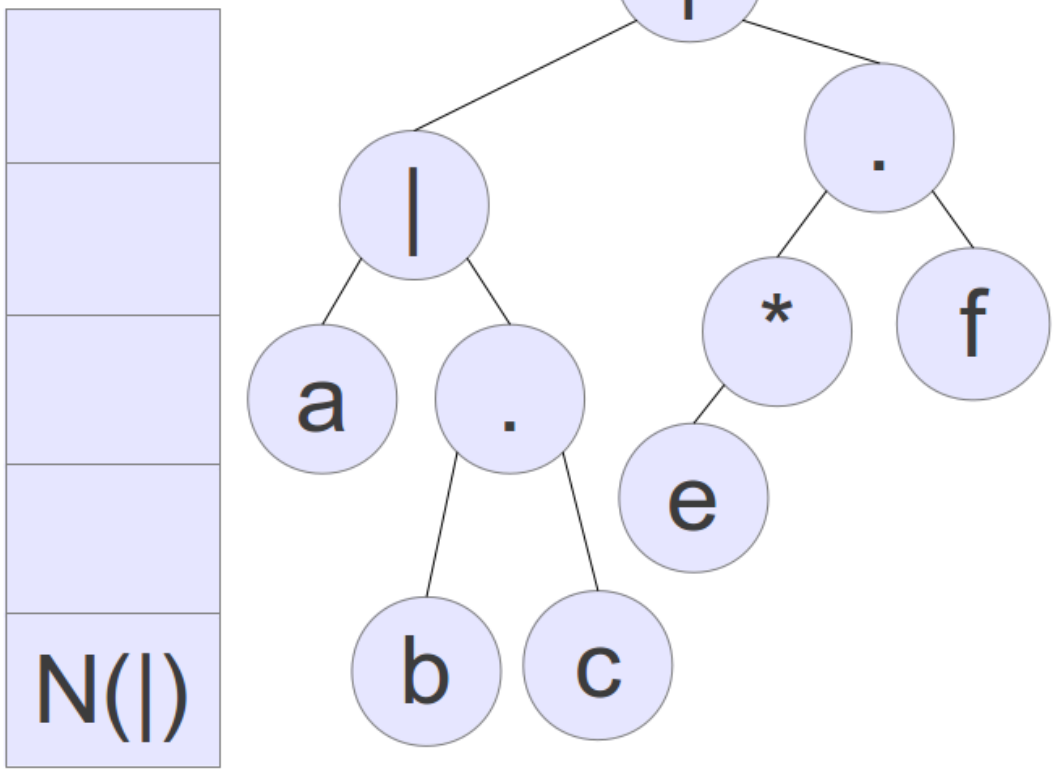
a|bc|e*f



Pilha **Árvore**

Operadores binários, assim como os símbolos consomem elementos da pilha. Vale lembrar a questão da ordem de precedência. Como a ordem de precedência da operação * é maior que **N(|)** então o novo nodo **N(*)** será filho de **N(|)** e este último permanecerá no topo da pilha.

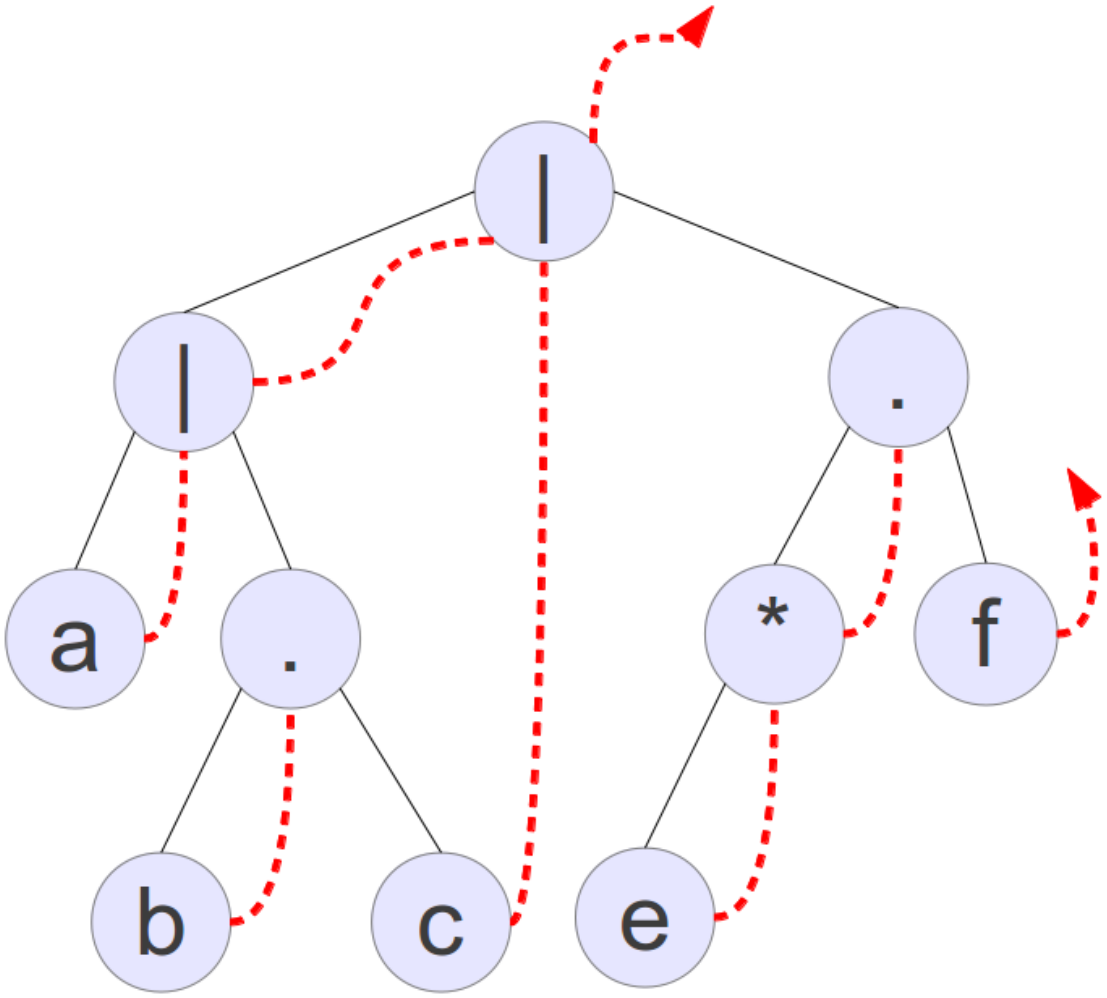
a | bc | e*f



Pilha

Árvore

a|bc|e*f



Por fim, após ter sido construída a árvore é costurada.

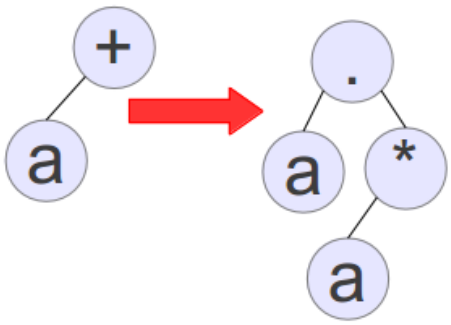
Observações

- Sempre que a ordem de precedência de uma nova operação for menor ou igual que a ordem de precedência da operação que está no topo da pilha, então a árvore é construída para cima, caso contrário é construída para baixo.

- O funcionamento do reconhecimento dos parênteses ocorre da seguinte forma: Ao ler um **(**, então esse **(** é colocado no topo da pilha juntamente com a árvore que está sendo utilizada até então. Uma nova árvore vaiza é criada e a expressão regular é reconhecida normalmente até ser lido lido um **)**. Ao ler o **)**, o nodo raiz da nova árvore criada é anexado como filho da direita da operação pendente que estiver no topo da pilha. Além disso, após ler o **)**, a árvore atual passa a ser a árvore que foi empilhada anteriormente, juntamente com o **(**.

- Os nodos das operações unárias são sempre pai de nodos de símbolos ou de nodos de agrupamento. Nodos de agrupamento são aqueles formados devido aos parênteses e possuem um atributo especial indicando que se trata de um agrupamento. Assim, as operações unárias procuram o primeiro nodo de símbolo ou nodo de agrupamento que encontrar.

- O operador + não possui rotinas de subir e descer. Assim, ele é decomposto em outras duas operações, por exemplo: a+ será transformado em aa*, veja como fica na árvore:



Após a geração da árvore foi relativamente fácil construir a tabela de composição e assim gerar o autômato. Essa facilidade se deu devido ao fato de existir as rotinas subir e descer para gerar as composições. Essas rotinas de subir e descer são bastante simples e de fácil implementação.

O reconhecimento e enumeração de sentenças foram as atividades seguintes. O reconhecimento se deu da mesma forma que apresentado em aula e foi utilizado uma estrutura de repetição analisando símbolo a símbolo da sentença a ser reconhecida. Já para as enumerações fizemos um algorítimo recursivo que envia como parâmetro um valor que inicialmente é o tamanho das sentenças a serem enumeradas e esse valor vai sendo decrementado a cada chamada recursiva no algorítimo. A recursão termina quando esse valor chega a zero. É importante lembrar que além desse valor também é fornecido como parâmetro uma lista onde são colocadas as sentenças enumeradas.

A última operação implementada foi a verificação de equivalência entre expressões regulares. Para realizar essa operação utilizamos a função de conversão de expressão regular para autômato finito e através de operações de complemento, união, determinização e minimização de autômatos finitos, foi possível realizar a verificação de equivalência. Assim, apenas foi necessário implementar a união e o complemento de autômatos, já que, as demais operações já estavam implementadas.