1 Expressões Regulares e Linguagens

- "Linguagem de Programação"
  - Pesquisa em Textos
  - Componentes de Compiladores
- Intimamente Relacionadas com AFNDs
- São capazes de definir todas e somente as linguagens regulares
- Servem de linguagem de entrada para muitos sistemas
  - GREP (Unix): busca em texto. Alguns sistemas convertem as expressões regulares em AFD ou AFND e simulam este autômato no arquivo sendo pesquisado.
  - LEX/FLEX: componente de um compilador que divide o programa fonte em unidades lógicas (tokens)
    - Palavras-chaves (while, if, etc...)
    - Identificadores (letra seguida de letras ou dígitos, etc...)
    - Sinais (+, <=, etc...)

1.1 Operadores em Expressões Regulares

01* + 10*: representa todos os strings que são um único 0 seguido por qualquer número de 1's ou um único 1 seguido por qualquer número de 0's

1. **União**: L = {001,10,111} M = {ε, 001}
   L U M = {ε,10,001,111}
   Conjunto de strings que estão em L ou M (ou ambos)
2. **Concatenação**: L = {001,10,111} M = {ε, 001}
   L.M = LM = {001, 10, 111, 001001, 10001, 111001}
3. **Fechamento (Estrela, Fechamento de Kleene):**
   L = {0,1} L* = {ε, 0, 1, 01, 10, 00, 11, 000, 001,...}
   L = {0,11} L* = {ε, 0, 11, 011, 110, 1111, 00, ... }

1.2 Construção de Expressões Regulares

- Constantes ε e Ø são expressões regulares que denotam as linguagens {ε} e Ø respectivamente. Portanto L(ε) = {ε} e L(Ø) = Ø

- Se *a* é símbolo de entrada então *a* é uma expressão regular
- Se E e F são expressões regulares:
  - L(E+F) = L(E) + L(F)
  - L(EF) = L(E)L(F)
  - L(E*) = (L(E))*
  - L((E)) = L(E)

Exemplo: expressão regular para o conjunto de strings que consistem em 0's e 1's alternados

(01)* + (10)* + 0(10)* + 1(01)*

\+ operador de união

Simplificando:

(ε + 1)(01)*(ε+0)

1.3 Precedência de Operadores em Expressões Regulares

1. Operador * tem precedência mais alta, se aplica a menor seqüência de símbolos a sua esquerda
2. Concatenação (ou ponto)
3. Uniões (+)

Utilizamos parênteses para agrupar operandos como queremos

Exemplo: 01*+1 = (0(1*))+1

2 Autômatos Finitos e Expressões Regulares

Autômatos Finitos e Expressões Regulares representam o mesmo conjunto de linguagens (linguagens regulares).

Portanto podemos sempreconverter uma Expressão Regular em Autômato Finito e vice-versa.

2.1 Conversão de AFDs para Expressões Regulares

- Bastante complicada

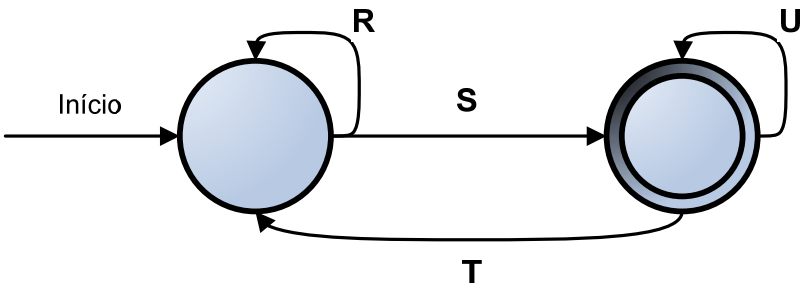
- Começamos descrevendo caminhos que não passam por qualquer estado e construímos indutivamente as expressões que permitem a passagem dos caminhos por conjuntos de estados progressivamente maiores.
- Numera-se os estados de um AFD *A*, de 1 a *n*, para um número *n* de estados.
- Utilizamos $R_{ij}^{(k)}$ como o nome de uma expressão regular cuja linguagem é o conjunto de strings *w* tais que *w* é o rótulo de um caminho que vai do estado *i* ao estado *j* em *A*, e que esse caminho não tem nenhum nó intermediário cujo número seja maior que *k*. (*i* e *j* não são intermediários e portanto podem ser maiores que *k*).
- Começamos a construção dos $R_{ij}^{(k)}$ com $k=0$ e vamos até $k=n$
- Com $k=0$ não podemos ter nenhum estado intermediário. Só dois tipos de caminho satisfazem tal condição:
    - Um arco do nó (estado) *i* para o nó *j*
    - Um caminho de comprimento 0 que consiste em apenas um nó *i*
- Quando $i \neq j$ apenas o caso 1 é possível. Devemos examinar o AFD *A* e encontrar os símbolos de entrada a tais que exista uma transição do estado *i* para o estado *j* com o símbolo *a*.
    - Se não existe tal símbolo *a*, então $R_{ij}^{(0)} = \emptyset$
    - Se existe exatamente um símbolo *a*, então $R_{ij}^{(0)} = a$
    - Se existem símbolos $a_1, a_2, ..., a_k$ que rotulam os arcos do estado i para o estado j, então $R_{ij}^{(0)} = a_1 + a_2 + ... + a_k$.
- Porém quando $i = j$, então os caminhos válidos são o caminho de comprimento 0 e todos os loops de *i* até ele mesmo. O caminho de comprimento 0 é representado por ε.
- Dado um caminho do estado *i* para *j*, que não passe por nenhum estado maior que *k*, há dois casos possíveis:
    - Caminho não passa por *k*, e seu rótulo está em $R_{ij}^{(k-1)}$
    - Caminho passa por *k* ao menos uma vez, e assim podemos dividílo em diversas partes: $R_{ik}^{(k-1)} + R_{kk}^{(k-1)} + R_{kk}^{(k-1)} + ... + R_{kj}^{(k-1)}$ e assim podemos construir a seguinte expressão:
        - $R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}\left(R_{kk}^{(k-1)}\right)^{*} R_{kj}^{(k-1)}$

- Através dela podemos construir todos os $R_{ij}^{(k)}$, começando com k=0 que é construído observando o autômato, e todos os k seguintes até k=n podem ser construídos utilizando a expressão acima.

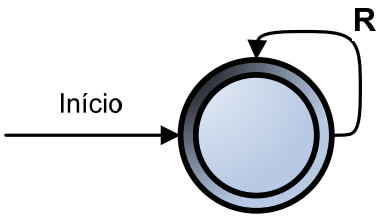
- A expressão regular final será dada pela união de todas as expressões dadas por $R_{ij}^{(n)}$ tal que n é o numero de estados, i é o estado inicial e j é um estado final.

2.2 Conversão de AFDs em ER por eliminação de estados

- O método visto anteriormente sempre funciona e serve não apenas para AFDs, mas também para AFNDs e AFNDεs, entretanto sua construção é dispendiosa.
  - É preciso construir n^3 expressões para um autômato de n estados.
  - Expressões regulares podem chegar a 4^n símbolos se não forem (ou não puderem ser) simplificadas.
- Eliminação de estados
  - Suponha um estado s que é intermediário entre p e q. Eliminamos o estado s e todos os caminhos que passam por ele deixam de existir, assim incluímos um arco que vai direto de q a p para os estados que iam de q a p passando por s. A transição de q a p passa a ser uma expressão regular:
  - A estratégia para construir a expressão regular, portanto é:
    - Para cada estado de aceitação q, aplica-se o processo de redução anterior para produzir um autômato equivalente com rótulos de expressões regulares nos arcos. Eliminam-se todos os estados exceto q e o estado inicial.
    - Se $q \neq q_0$ teremos um autômato de dois estado semelhante ao da figura abaixo. A expressão regular equivalente será dada por: (R+SU*T)*SU*.



- Se o estado final também for estado inicial, então eliminamos todos os estados exceto o estado inicial. Ficaremos com um autômato de apenas um estado como o mostrado abaixo, cuja expressão regular equivalente será R*:



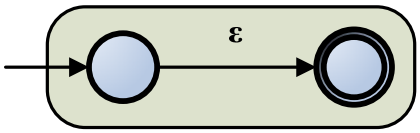
- A expressão regular desejada é a soma (união) de todas as expressões derivadas dos autômatos reduzidos correspondentes a cada estado de aceitação.

2.3 Conversão de expressões regulares em autômatos

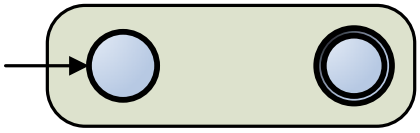
Para qualquer expressão regular *R* temos um AFNDε *E* com:

1. Exatamente um estado de aceitação
2. Nenhum arco chega ao estado inicial
3. Nenhum arco sai do estado de aceitação

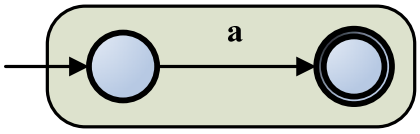
Construção para ε:



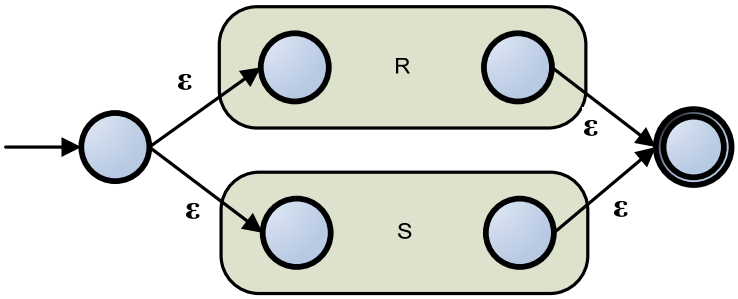
Construção para Ø:



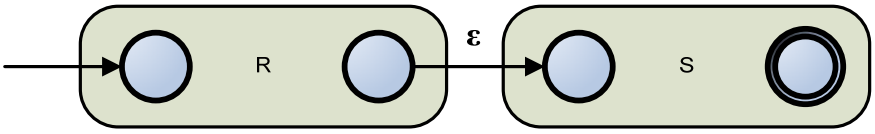
Construção para uma expressão regular **a:**



Construção para uma união R + S:



Construção para uma união RS:



Construção para uma expressão R*: