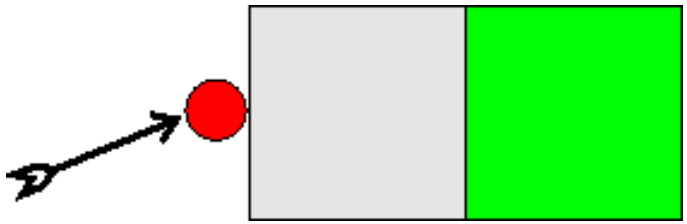
# Operações Regulares e Expressões Regulares

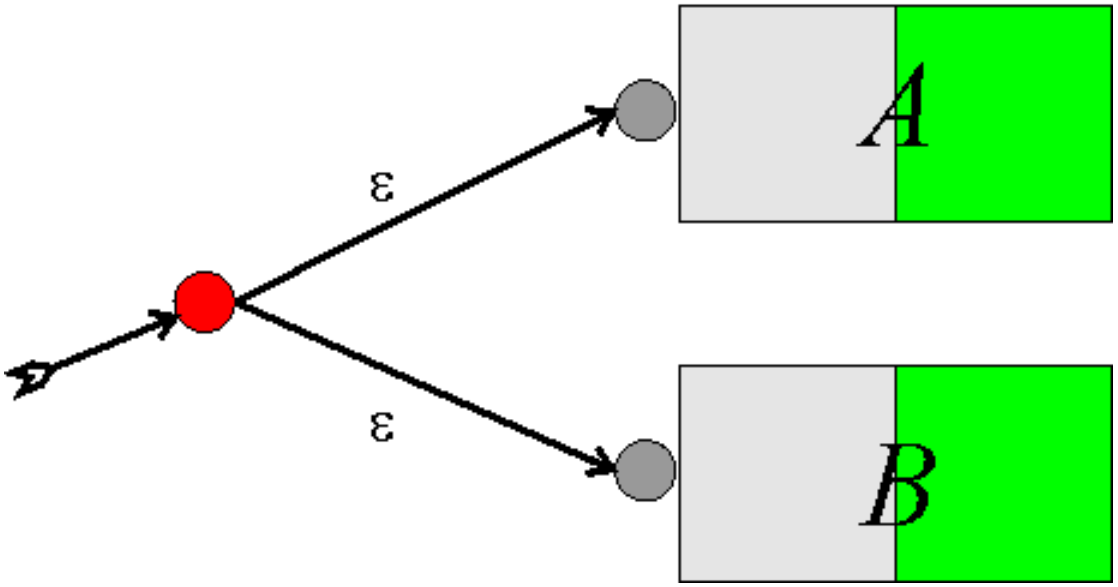
# AFNs. Operações Regulares. Esquematicamente.

O circulo vermelho representa o estado inicial $q_0$, a porção verde representa o conjunto de estados de aceitação *F*, os demais estados são a região cinza



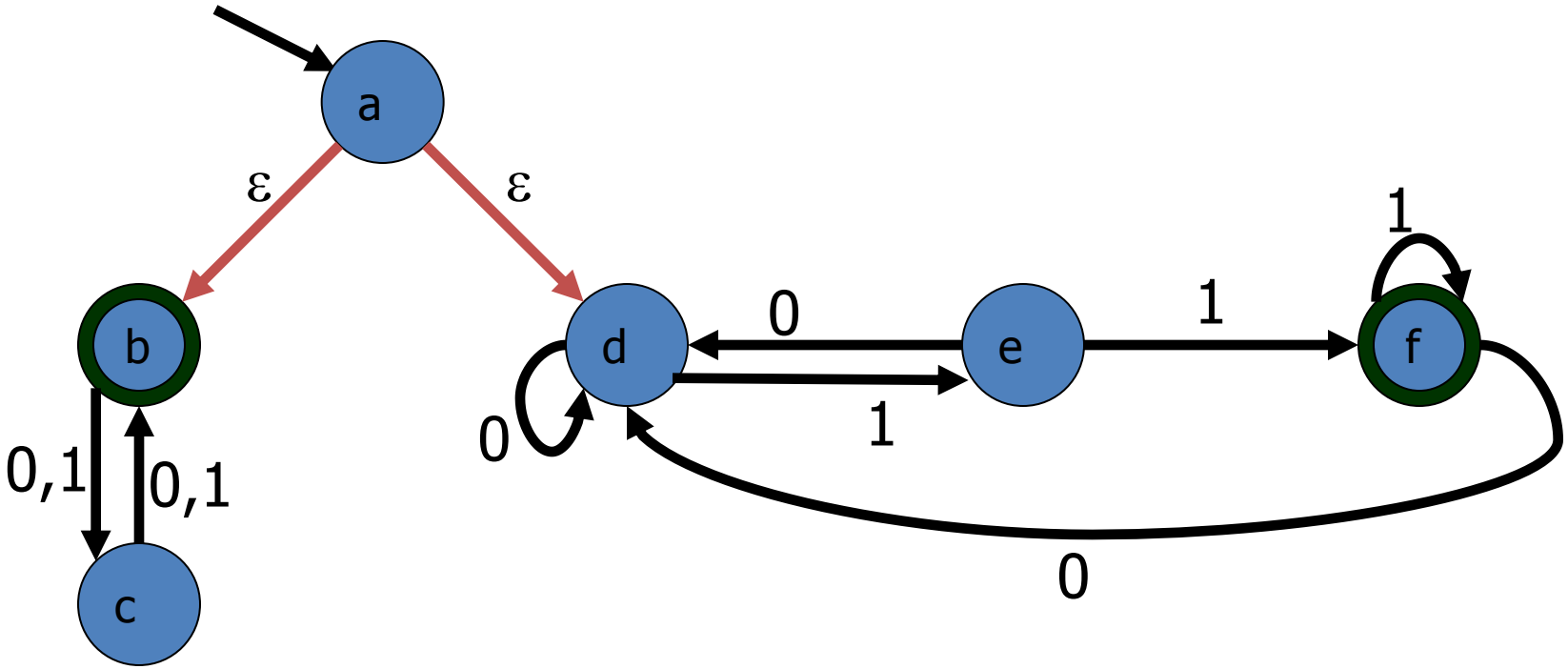
# NFAs. União.

A união $A \cup B$ é formada colocando os automata em paralelo. Criamos um novo estado inicial e o conectamos aos estados iniciais originais por $\varepsilon$-transições:



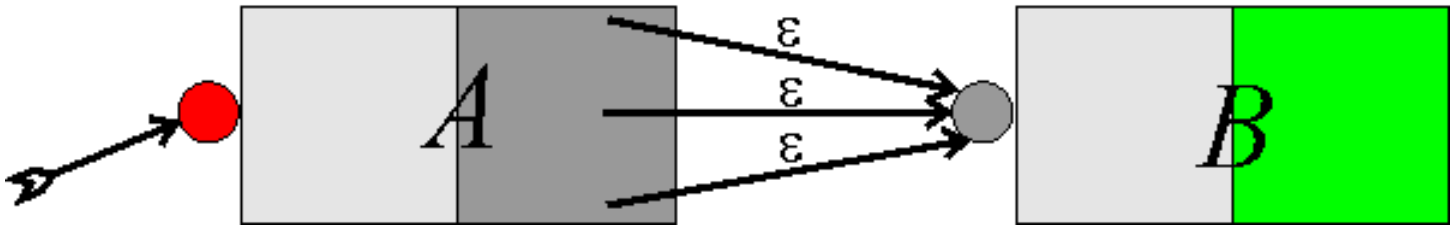
# União: Exemplo

$L$ = {$x$ tem comprimento par} $\cup$ {$x$ termina com 11}



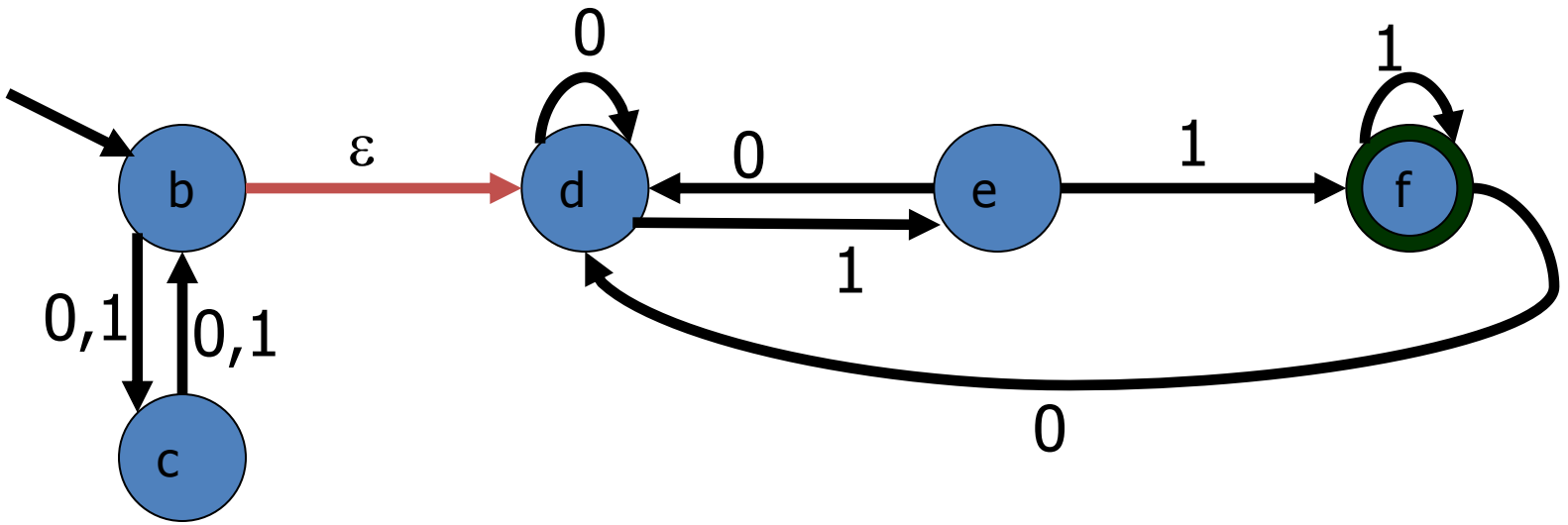
# AFNs. Concatenação.

A concatenação $A \bullet B$ é formada colocando-se os automatos en série. O estado inicial é o de *A* enquanto os estados de aceitação são os de *B*. Os estados de aceitação de *A* deixam de sê-lo e são conectados, por meio $\varepsilon$-transição, ao estado inicial de *B*:



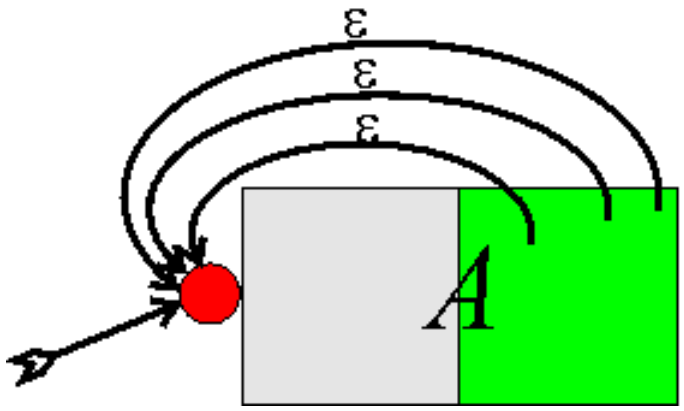
# Concatenação: Exemplo

$L$ = {$x$ tem comprimento par} • {$x$ termina com 11}



# AFNs. Kleene-+.

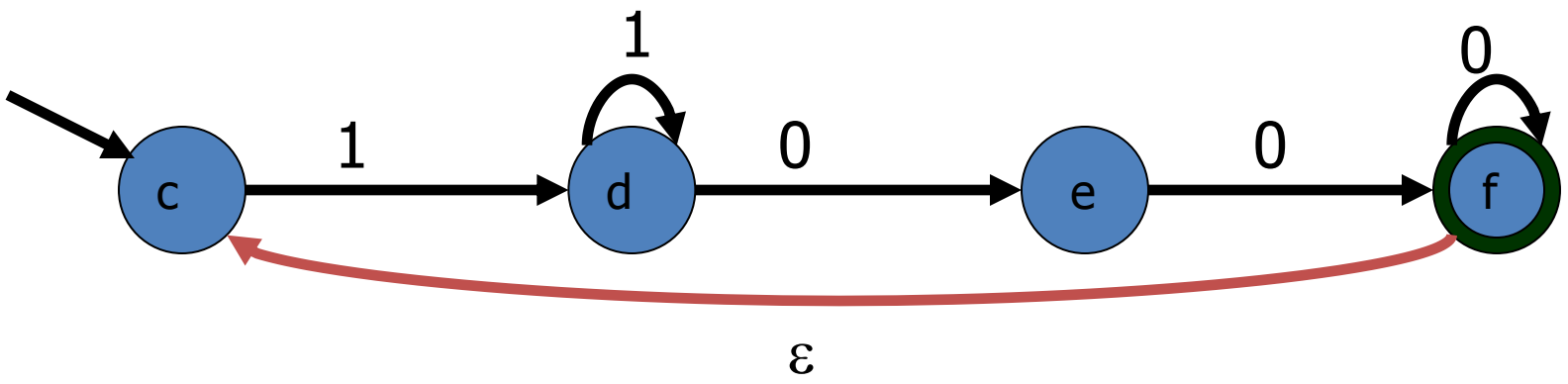
A operação Kleene-+ $A^+$ é formada criando-se um *loop* de retroalimentação. Os estados de aceitação são conectados ao estado inicial por meio de ε-transição:



# Kleene-+ Exemplo

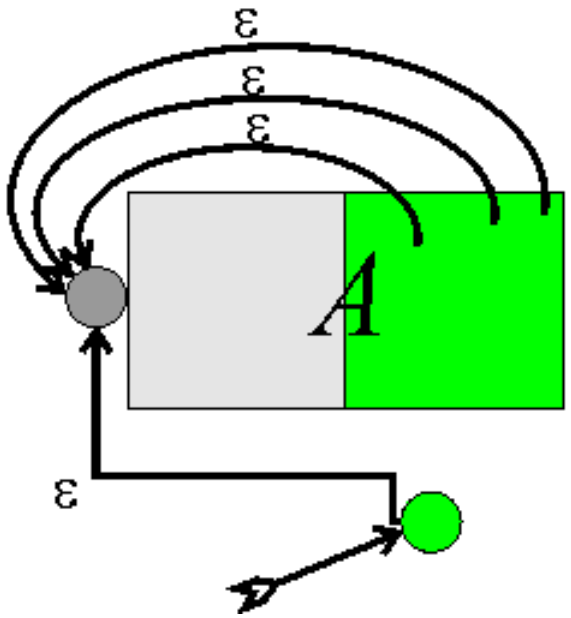
$L = \{$ *x* é uma sequência de 1 ou mais 1's seguida de uma sequência de 2 ou mais 0's $\}^+$

$= \{$ *X* começa com 1, termina com 0, e conciste de repetições de sequências de 1 ou mais 1's seguidos de 2 ou mais 0's $\}$



# AFNs. Kleene-*.

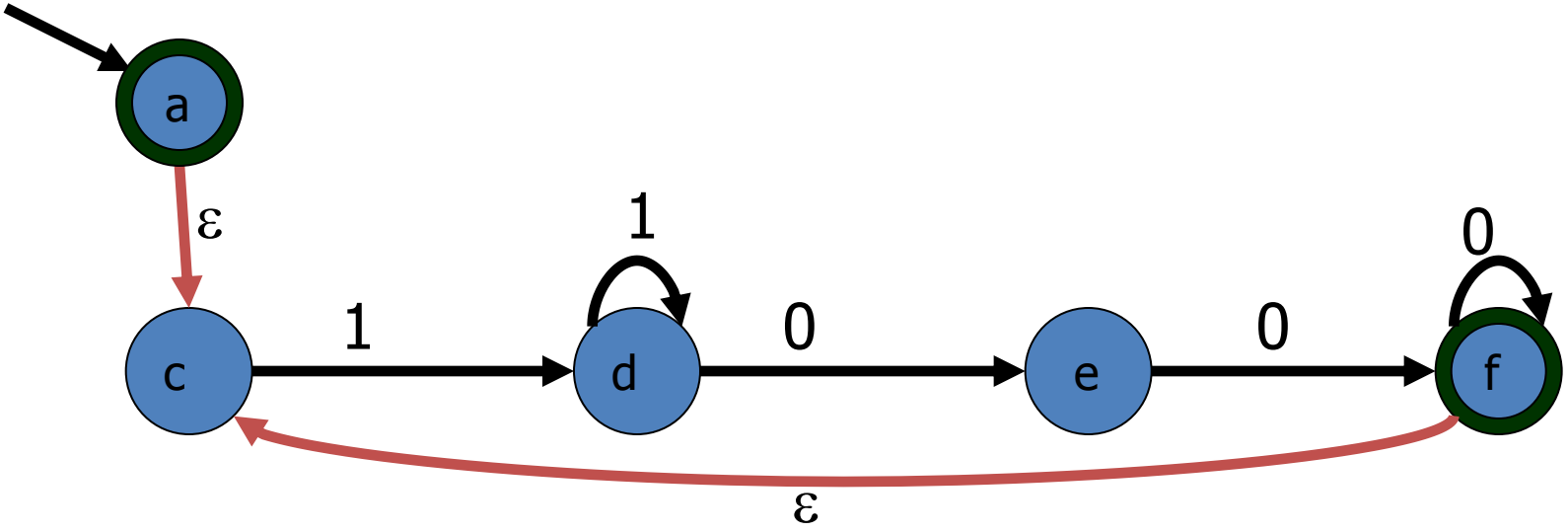
A construção deriva da de Kleene-+ usando-se o fato de que $A^*$ é a união de $A^+$ com a linguagem que contém apenas o string vazio. Construa Kleene-+ e adicione um novo inicial conectado ao antigo estado inicial por meio de ε-transição:



# Kleene-* Exemplo

$L = \{$ *x* é uma sequência de 1 ou mais 1's seguida de uma sequência de 2 ou mais 0's $\}^*$

$= \{$ *X* começa com 1, termina com 0, e conciste de repetições de sequências de 1 ou mais 1's seguidos de 2 ou mais 0's **ou** x é ε $\}$



# Fecho de AFNs sob Operações Regulares

As construções apresentadas mostram que AFNs são fechados sob operações regulares.

THM 1: Se $L_1$ e $L_2$ são aceitas por AFNs, então $L_1 \cup L_2$, $L_1 \bullet L_2$, $L_1^+$ e $L_1^*$ também são.

De fato, os AFNs que aceitam essas linguagens podem ser construídos algoritmicamente, em tempo linear.

Como já mostramos que todo AFN-$\varepsilon$ pode ser convertido em um AFN e todo AFN pode ser convertido em um AFD equivalente, mostramos que AFDs –e portanto linguagens regulares– são fechadas sob operações regulares.

# Expressões Regulares

Já conhecemos as operações regulares. Expressões regulares provêm uma forma de concisa de expressar linguagens construídas por meio de operações regulares. Por exemplo, a linguagem finita {banana,nab}* pode ser expressa, a partir das linguagens atômicas {a},{b} e {n} do seguinte modo:

( ({b}•{a}•{n}•{a}•{n}•{a})∪({n}•{a}•{b}) )*

Ou mais concisamente como a expressão regular

(banana∪nab)*

# Expressões Regulares

DEF: O conjunto das ***expressões regulares*** sobre um alfabeto $\Sigma$ e as linguagens em $\Sigma^*$ que elas geram são definidas recursamente:

- Casos Base: Cada símbolo $a \in \Sigma$, assim como os símbolos $\varepsilon$ e $\varnothing$ são expressões regulares:
  - $a$ representa a linguagem atômica $L(a) = \{ a \}$
  - $\varepsilon$ representa a linguagem $L(\varepsilon) = \{ \varepsilon \}$
  - $\varnothing$ representa a linguagem $L(\varnothing) = \{ \} = \varnothing$
- Casos Inductivos: se $r_1$ e $r_2$ são expressões regulares, também o são $r_1 \cup r_2$, $(r_1)(r_2)$, $(r_1)^*$ e $(r_1)^+$:
  - $L(r_1 \cup r_2) = L(r_1) \cup L(r_2)$ ($r_1 \cup r_2$ denota união)
  - $L((r_1)(r_2)) = L(r_1) \bullet L(r_2)$ ($(r_1)(r_2)$ denota concatenação)
  - $L((r_1)^*) = L(r_1)^*$ ($(r_1)^*$ denota Kleene-*)
  - $L((r_1)^+) = L(r_1)^+$ ($(r_1)^+$ denota Kleene-+)

# Expressões Regulares - Tabela

| Operação | Notação | Linguagem | UNIX |
|---|---|---|---|
| União | $r_1 \cup r_2$ | $L(r_1) \cup L(r_2)$ | $r_1 \mid r_2$ |
| Concatenação | $(r_1)(r_2)$ | $L(r_1) \bullet L(r_2)$ | $(r_1)(r_2)$ |
| Kleene-* | $(r)^*$ | $L(r)^*$ | $(r)*$ |
| Kleene-+ | $(r)^+$ | $L(r)^+$ | $(r)+$ |
| Exponenciação | $(r)^n$ | $L(r)^n$ | $(r)\{n\}$ |

# Expressões Regulares. Simplificação.

Assim como no caso de fórmulas algébricas, expressões regulares também podem ser simplificadas de modo a usar menos parenteses, se a ordem das operações é clara. Conforme a definição de expressões regulares, para expressar a linguagem {banana,nab}* teríamos que escrever algo complicado como:

$$((((b)(a))(n))(((a)(n))(a))\cup(((n)(a))(b)))^*$$

Usando a precedência de operadores *, $\bullet$ , $\cup$ e a associatividade de $\bullet$ , temos a expressão mais simples:

$$(\text{banana}\cup\text{nab})^*$$

# Expressões Regulares. Exemplo.

Q:Obtenha uma expressão regular que representa a linguagem que consiste de todos os bit-strings que contém uma sequência de sete 0's ou duas sequências disjuntas de três 1's.

Nota: Uma sequência de nove 0's seria válida já que contém um substring de sete 0's.

EX:

- Legal: 010000000011010, 01110111001, 111111
- Ilegal: 11011010101, 10011111001010

# Expressões Regulares. Exemplo.

R: $(0 \cup 1)^*(0^7 \cup 1^3(0 \cup 1)^*1^3)(0 \cup 1)^*$

Ou de maneira ainda mais concisa:

$$\Sigma^*(0^7 \cup 1^3\Sigma^*1^3)\Sigma^*$$

A resposta *oficial* , usando apenas as operações regulares padrão seria:

$$(0 \cup 1)^*(0000000 \cup 111(0 \cup 1)^*111)(0 \cup 1)^*$$

Uma expressão regular UNIX seria:

```
(0|1)*(0{7}|1{3}(0|1)*1{3})(0|1)*
```

# Expressões Regulares.

Expressões regulares são strings. Consequentemente, um conjunto de expressões regulares é uma linguagem.

Q: Suponha sejam consideradas apenas as operações de união, concatenação e Kleene-* Qual é o alfabeto da linguagem das expressões regulares sobre o alfabeto $\Sigma$ ?

# Expressões Regulares.

R: $\Sigma \cup \{ (, ), \cup, * \}$

# Exercício

- Retire o máximo de parênteses
  - $(((0) \cup (( (0) \cup (1) ) (0) ) ) \cup ((1)(1)))$
- Descreva em português as linguagens denotadas pelas REXs
  - $0(0 \cup 1)^*1$
  - $0^*(0+1)1^*$
- Forneça REX que denotem:
  - $\{w \in \{a,b\}^* \mid |w| \geq 3 \}$
  - $\{w \in \{a,b\}^* \mid$ w começa com a e tem tamanho par$\}$
- Forneça REX mais simples para:
  - $\emptyset^* \cup \varepsilon^*$
  - $0^* \cup 1^* \cup 0^*1^* \cup (0 \cup 1)^*$
  - $(00^* \cup 10^*)\, 0^*\, (1^* \cup 0\,)^*$

# REX → AFN-ε

Como AFNs-ε são fechados sob as operações regulares obtemos imediatamente:

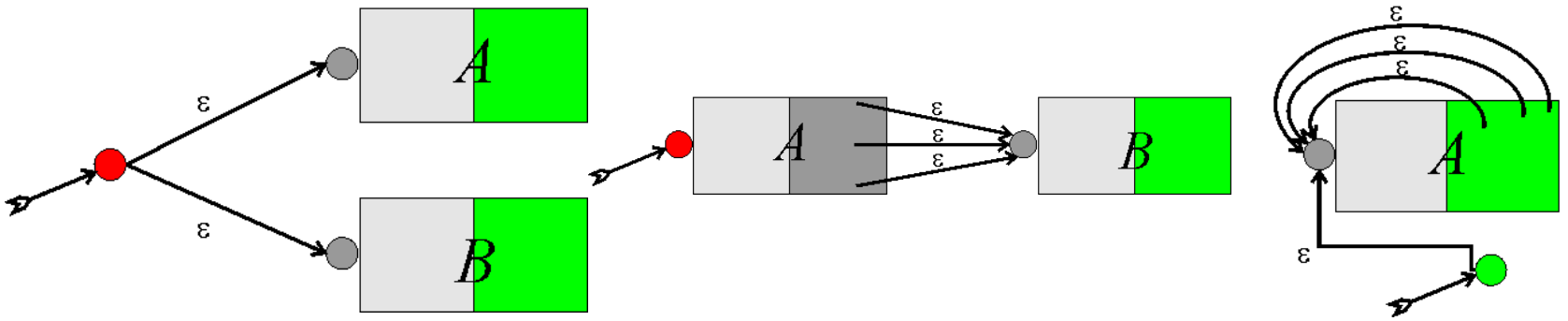
THM 2: Dada uma expressão regular *r* existe um AFN-ε *N* que aceita a linguagem denotada por *r*. I.e. tal que $L(N) = L(r)$. Além disso, esse AFN-ε pode ser construído em tempo linear.

# REX → AFN-ε

*Prova*. A prova é por indução, usando a definição recursiva de expressões regulares. Primeiro precismos mostrar AFNs-ε que aceitam as linguagens denotadas pelas expressões $a \in \Sigma$, ε e ∅. Essas são aceitas pelos AFNs-ε:

# REX → AFN-ε

Agora precisamos mostrar como construir AFNs-ε que aceitam linguagens denotadas por expressões regulares formadas pelas operações regulares. Esses são obtidos pelas construções vistas anteriormente:



# REX → AFN-ε. Exemplo.

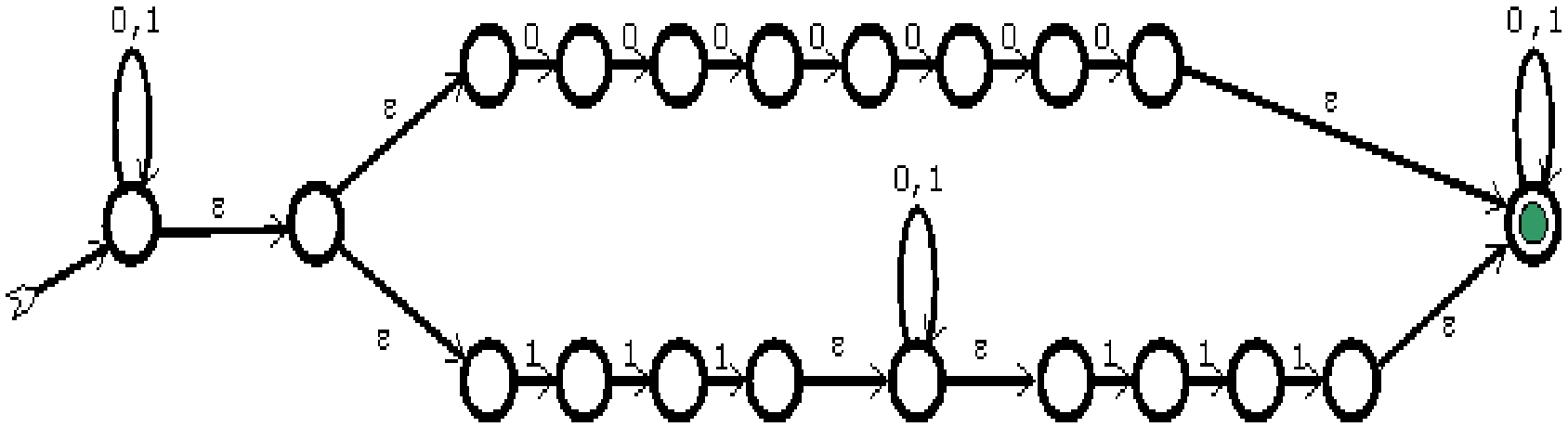
Q: Obtenha um AFN-ε que aceita a linguagem denotada pela expressão regular

$$(0 \cup 1)^*(0000000 \cup 111(0 \cup 1)^*111)(0 \cup 1)^*$$

# REX → AFN-ε. Exemplo.

(0∪1)*(0000000∪111(0∪1)*111)(0∪1)*

R:



# REX → AFN-ε → AFN → AFD

Em resumo: Começando com um AFN-ε, usamos o procedimento de eliminação de ε–transições para obter um AFN equivalente sem ε–transições. Em seguida, usamos a construção de subconjuntos de estados para obter um AFD (*determinístico*) que aceita a mesma linguagem.

Isso implica os seguintes resultados:

# REX → NFA → DFA

THM 3: Se *L* é uma linguagem aceita por um AFN, então é possível construir um AFD que aceita *L*.

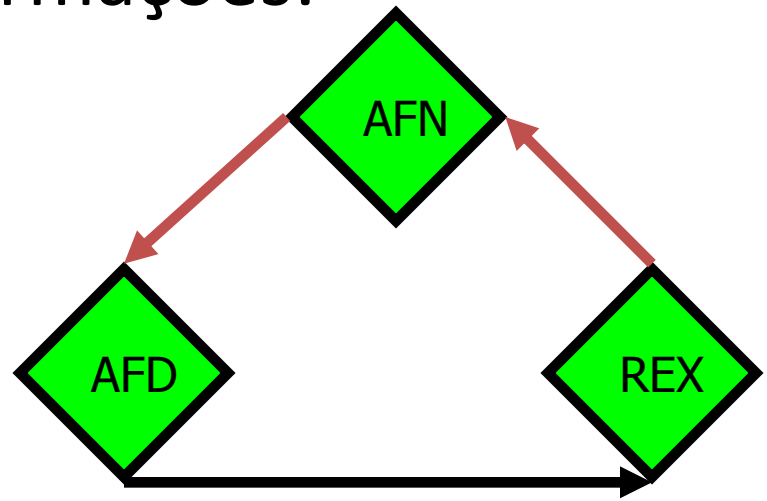
COR: A classe das linguagens regulares é fechada sob as operações regulares.

*Prova* : Como AFNs são fechados sob operações regulares (THM 1) e AFDs são também NFAs, podemos aplicar operações regulares a qualquer AF; então (se necessário) podemos obter um AFD equivalente (THM 3), tendo assim um AFD que aceita a linguagem definida por meio de operações regulares.□

# REX → AFN → AFD → REX …

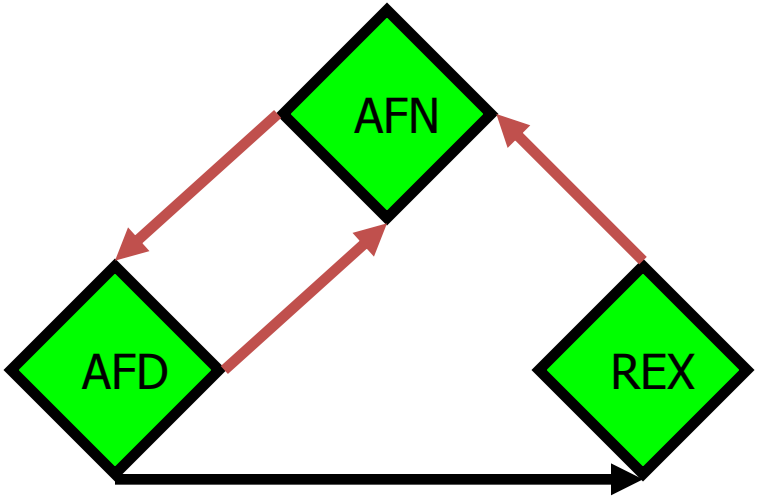
Estamos a um passo de mostrar que AFDs ≈ AFNs ≈ REXs; i.e., as três representações são equivalentes.

Para concluir precisamos completar o círculo de transformações:



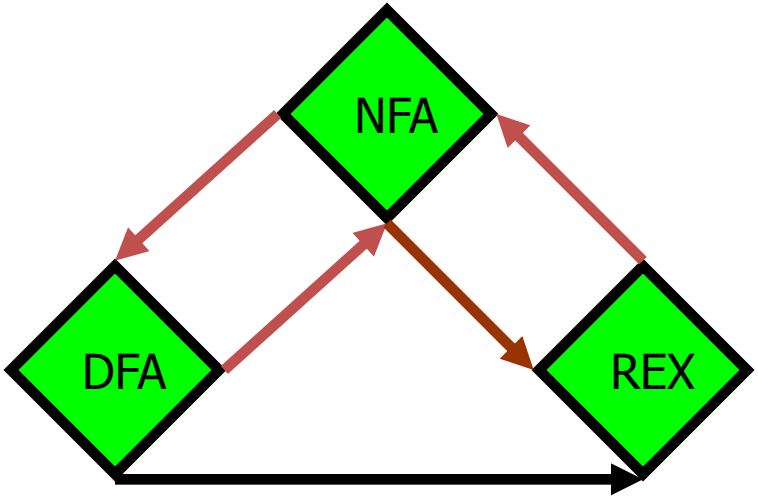
# REX → AFN → AFD → REX …

De fato, podemos adicionar mais informação nessa figura, já que um AFD é automaticamente um AFN:



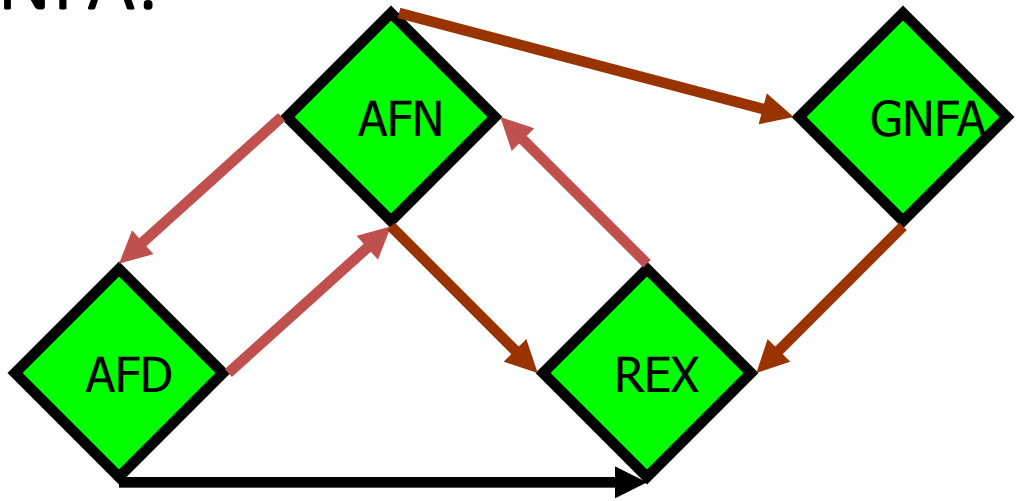
# REX → AFN → AFD → REX …

Vamos mostrar como converter um AFN em uma expressão regular, o que, em particular, mostra como converter um AFD em uma expressão regular e, portanto, completa o círculo de equivalências:



# REX → AFN → AFD → REX …

Para converter um AFN para REX vamos introduzir uma noção mais geral de autômato, chamado “AFN Generalizado” ou “GNFA” ou “Diagrama de Expressão Regular”. Para converter para REX, vamos usar um GNFA:



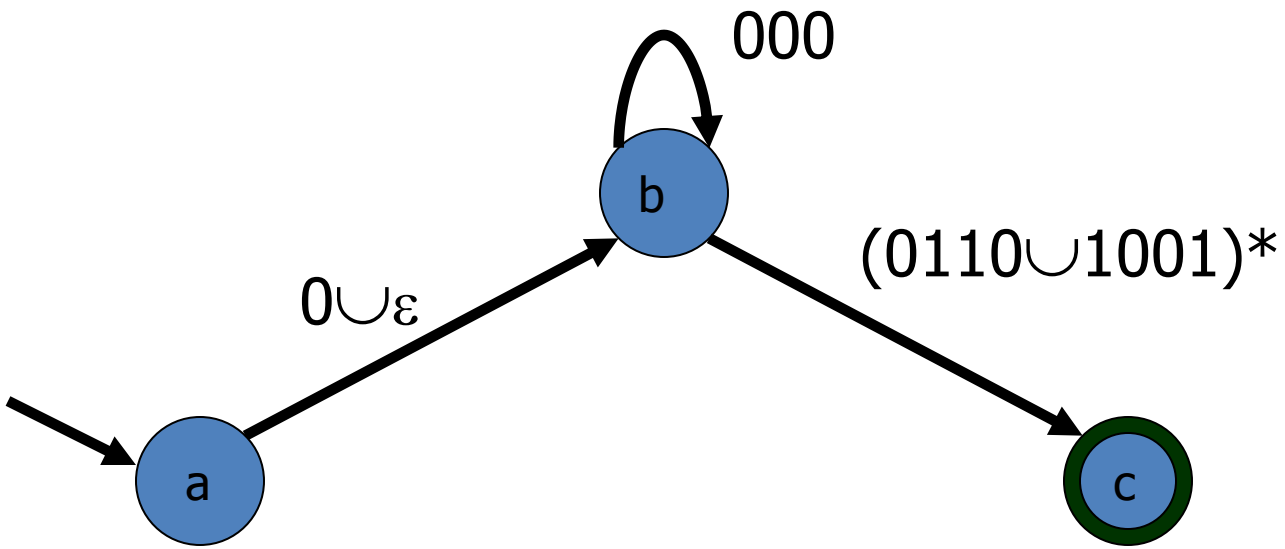
# GNFAs

DEF: A ***autômato finito não determinístico generalizado*** (**GNFA**) é um grafo cujos arcos são rotulados por expressões regulares, tendo um único estado inicial, com grau de entrada 0 e um único estado final, com grau de saída 0.

Um string *u* é ***rótulo*** de um caminho no GNFA, se ele é um elemento da linguagem gerada pela expressão regular obtida concatenando-se todos os rótulos dos arcos nesse caminho.

A ***linguagem aceita*** por um GNFA consiste de todos os strings aceitos por esse GNFA.

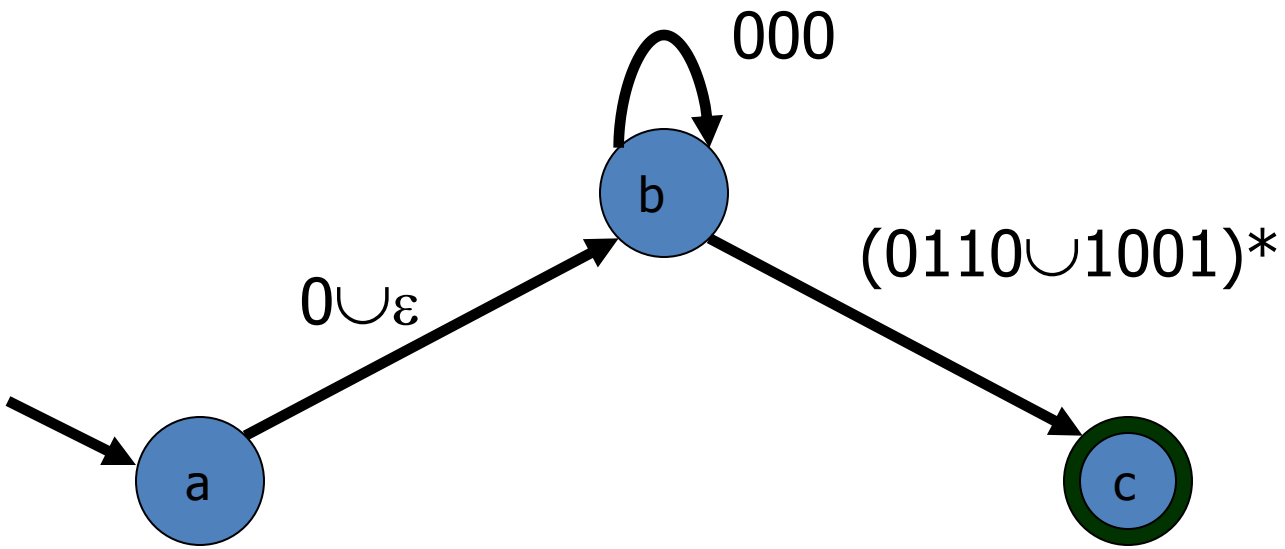
# GNFAs. Exemplo.




É um GNFA porque os arcos são rotulados por REXs, o estado inicial não tem arcos entrando e o *único* estado de aceitação não tem arcos de saída.

Q: O string 000000100101100110 é aceito?

# GNFAs. Exemplo.




R: 000000100101100110 é aceito. O caminho:

$$a \xrightarrow[\varepsilon]{} b \xrightarrow[000]{} b \xrightarrow[000]{} b \xrightarrow[10010110011\,0]{} c$$

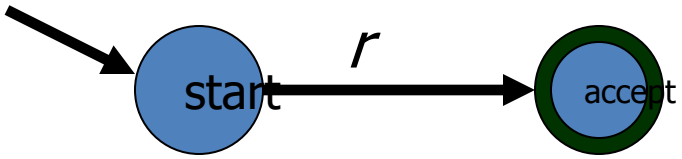
prova isso. (o último rótulo resulta de

100101100110 $\in$ (0110$\cup$1001)* )

# AFN → REX

A conversão AFN para REX é feita do seguinte modo:

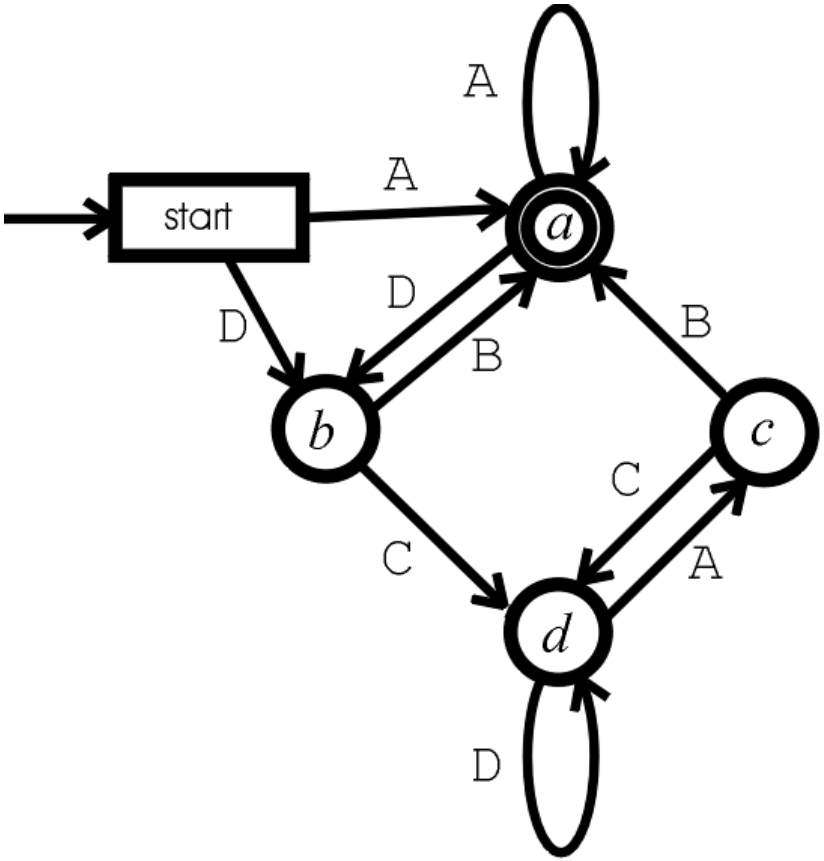
1. Construa um GNFA a partir do AFN.
   A. Unifique arcos com múltiplos rótulos usando "∪"
   B. Crie um estado inicial com grau de entrada 0
   C. Crie o estado de aceitação com grau de saída 0
2. Elimine, passo a passo, estados interiores, modificando os rótulos dos arcos, até obter:




3. A REX correspondente é o rótulo *r*.

# AFN → REX. Exemplo

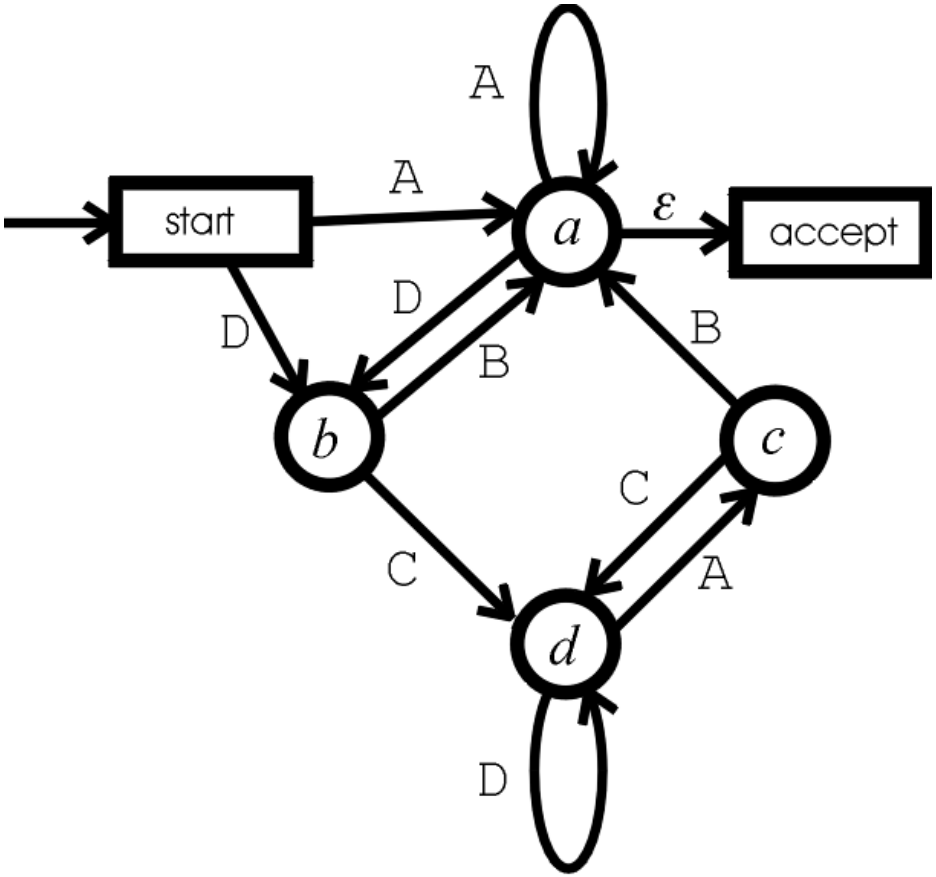
Comece com:




Primeiro converta em um GNFA. Não é ainda um GNFA porque o estado de aceitação, embora único, não tem grau de saída 0. O estado inicial tem grau de entrada 0 e, portanto, está OK.

# AFN → REX. Exemplo

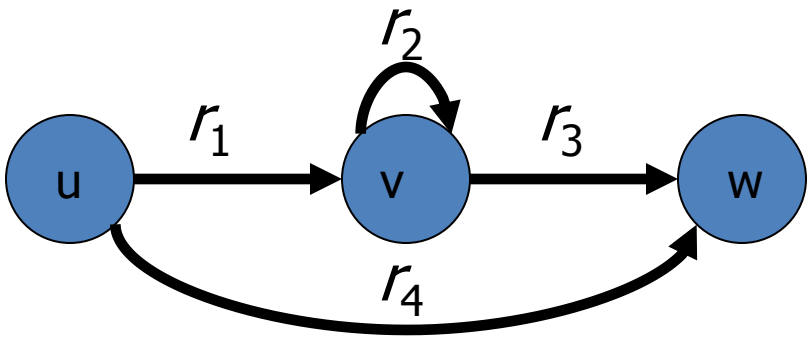
Adicionamos o novo estado de aceitação conectando a ele o a antigo, via ε–transição.



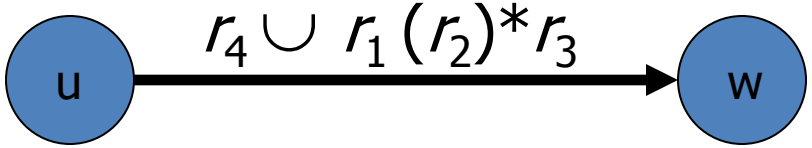

Como os rótulos são simples letras, eles são expressions regulares, is e, portanto, esse é um GNFA. Estamos então prontos para começar a eliminar estados interiores.

# AFN → REX.

A eliminação de estado é feita do seguinte modo. Se quisermos eliminar o estado do meio *v* :




devemos restabelecer todas as possibilidades de ir de *u* para *w*. I.e., adicionamos ao rótulo $r_4$, do arco (*u*,*w*), a expressão regular obtida pela concatenação do rótulo $r_1$, de (*u*,*v* ), com o rótulo $r_2$, do loop (*v*,*v* ), repetido arbitrariamente, seguido do rótulo $r_3$, do arco (*v*,*w* ). O novo rótulo (*u*,*w* ) é portanto:

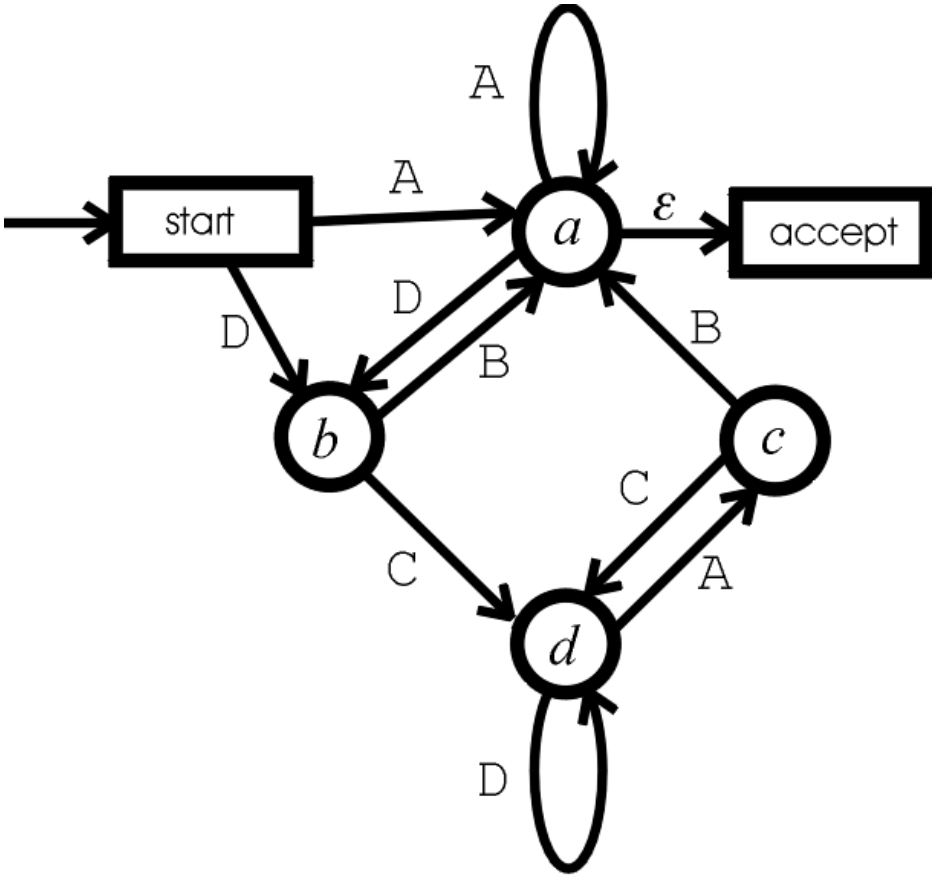


# AFN → REX. Exemplo

Voltando ao nosso exemplo, vamos eliminar *d*.

Q1: Qual deve ser o rótulo de (*b*,*c* )?

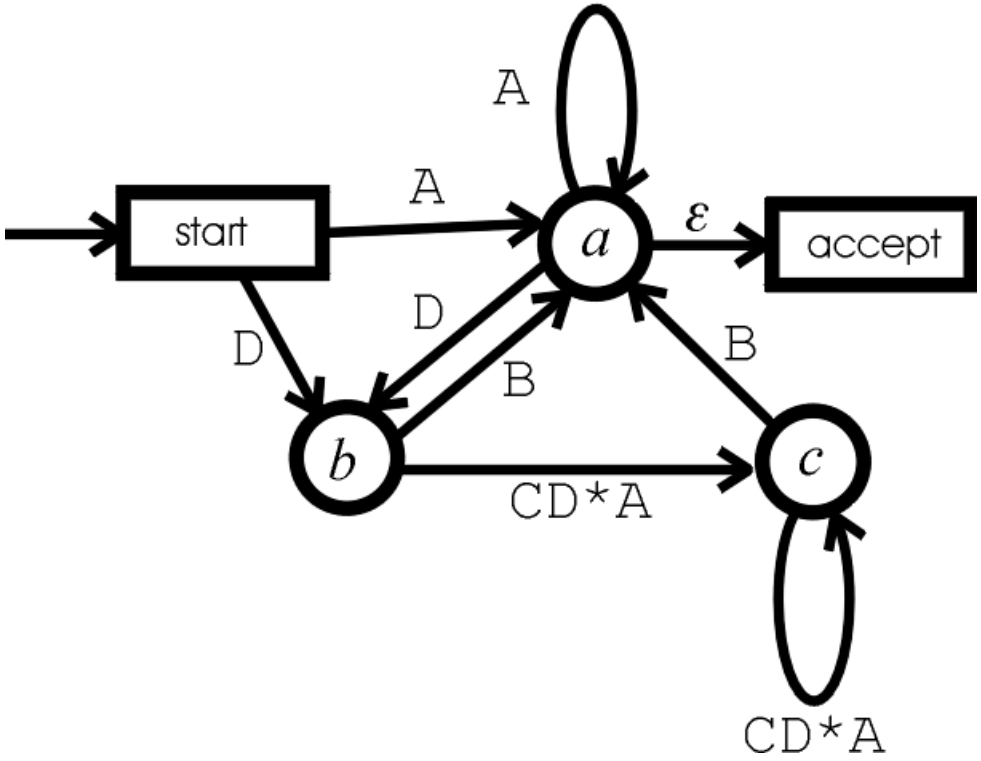
Q2: Qual deve ser o rótulo de (*c*,*c* )?



# AFN → REX. Exemplo

R1: (*b*,*c* ): **CD*A**

R2: (*c*,*c* ): **CD*A**




Em seguida,

eliminar *c.*

Q: Qual deve ser o rótulo de (*b*,*a* )?

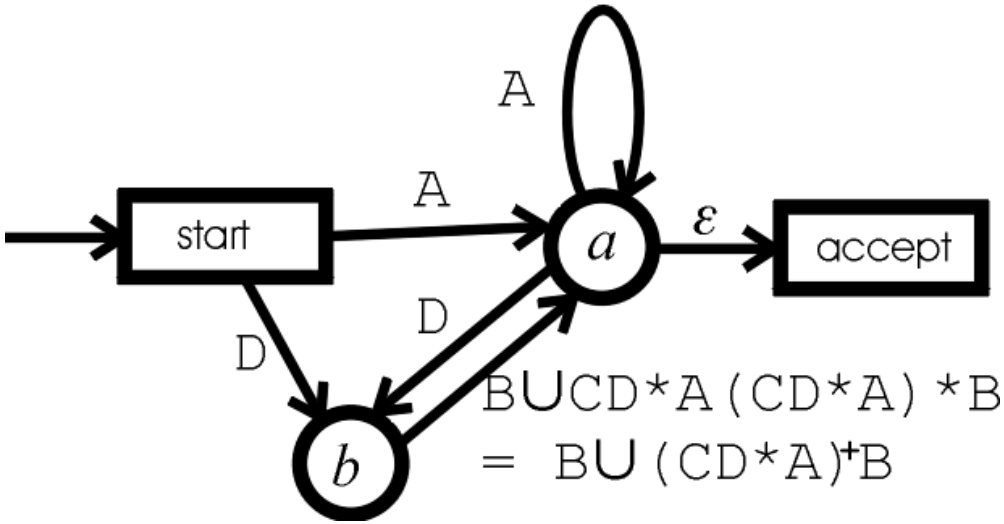
# AFN → REX. Exemplo

R:

**B∪(CD*A) (CD*A)*B**

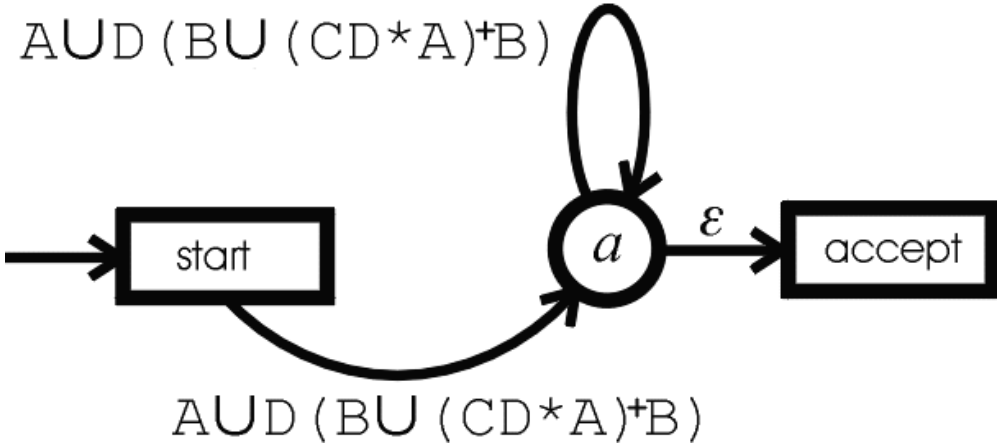
O que simplifica para

**B∪(CD*A)$^+$B**




Agora, vamos eliminar *b*.

# AFN → REX. Exemplo




Finalmente, vamos eliminar *a* :

# AFN → REX. Exemplo

A REX resultante

é o rótulo único

**(A∪D(B∪(CD*A)$^+$B)**

# RESUMO

Isso completa a demonstração de que são *equivalentes* entre si os três métodos de descrever linguagens regulares:

1. AFs Determinísticos
2. AFNs
3. Expressões Regulares