AFε e Transformação de AFε para AFD

Autômato Finito com Movimento Vazio (AFϵ)

- São autômatos que utilizam a transição em vazio na sua representação
  - transições sem leitura de símbolos na fita
- Auxilia a construção de um autômato
- Caso particular de não-determinismo
- qualquer AFϵ pode ser simulado por um AFN

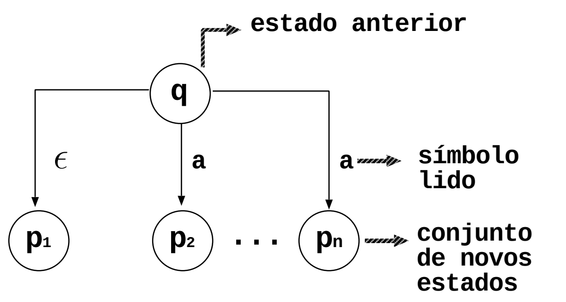
Movimentos vazios constituem uma generalização dos modelos de máquinas não-determinística. Pode ser interpretado como um não-determinismo interno ao autômato o qual é encapsulado, ou seja, nada é observado a não ser a alteração de um estado para o outro. Uma das vantagens dos AFϵs é o fato de facilitar algumas construções e demonstrações relacionadas com os autômatos.

AFϵ - Formalismo

- $M = (\Sigma, Q, \delta, q_0, F)$
  - Σ – alfabeto de símbolos de entrada
  - Q – conj. de estados possíveis do autômato o qual é finito
  - δ – função de transição ($\delta : Q$ x $(\Sigma \cup \{\epsilon\})$)
  - q_0 – estado inicial ($q_0 \in Q$)
  - F – conjunto dos estados finais tal que F está contido em Q

Portanto, excetuando-se pela função programa, as componentes Σ, Q, F e q_0 são como na definição de AFN. Um movimento vazio (ou transição vazia) é representado pela aplicação da função programa, em um dado estado q ao símbolo especial ϵ.

Autômato Finito Não-Determinístico



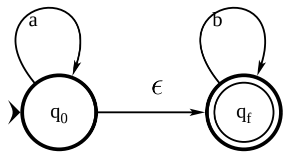
Um movimento em vazio não se implementa, computacionalmente.

Exemplo

$L = \{$w | qualquer símbolo a antecede qualquer símbolo b $\}$

$M = (\{$ a, b$\}, \{q_0, q_f\}, \delta, q_0, \{q_f\})$

δ	a	b	ϵ
q_0	$\{q_0\}$	-	$\{q_f\}$
q_f	-	$\{q_f\}$	-



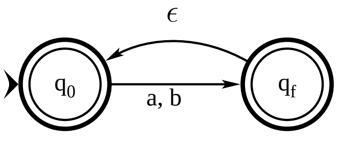
O processamento de um AFϵ é análogo ao de um AFN. O processamento de uma transição para uma entrada vazia também é não-determinística. Assim, um AFϵ ao processar uma entrada vazia assume simultaneamente os estados destino e origem. Ou seja, a origem de um movimento vazio sempre é um caminho alternativo.

Exercício 1

$L = (\mathtt{a} \cup \mathtt{b})^*$

$M = (\Sigma, Q, \delta, q_0, F)$
$\Sigma = \{\ \mathtt{a},\ \mathtt{b}\}$
$Q = \{q_0, q_f\}$
$F = \{q_0, q_f\}$

δ	a	b	ϵ
q_0	q_f	q_f	-
q_f	-	-	q_0

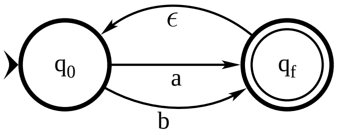


Exercício 2

$L = (\mathtt{a} \cup \mathtt{b})^+$

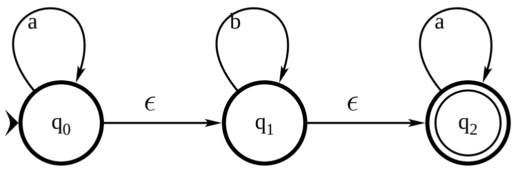
$M = (\Sigma, Q, \delta, q_0, F)$
$\Sigma = \{\ \mathtt{a},\ \mathtt{b}\}$
$Q = \{q_0, q_f\}$
$F = \{q_f\}$

δ	a	b	ϵ
q_0	q_f	q_f	-
q_f	-	-	q_0



Definições

Utilizaremos o autômato abaixo para exemplificar as o uso das definições.



- `edge(s,c)` — conjunto de todos os estados do AFN alcançáveis do estado s pelo rótulo c

Exemplo:

- edge(q_0, a) $= \{q_0\}$
- edge(q_0, ϵ) $= \{q_1\}$
- edge(q_1, b) $= \{q_1\}$
- edge(q_1, ϵ) $= \{q_2\}$
- edge(q_2, a) $= \{q_2\}$

Definições

- `closure(S)` — conjunto de estados que podem ser alcançados a partir do estado S sem consumir nenhuma entrada. Isto é, menor conjunto T, onde:

$$T = S \cup (\bigcup_{s \in T} \text{edge}(s, \epsilon))$$

```
T ← S
repeat T' ← T
       T ← T' ∪(⋃_{s∈T'} edge(s,ϵ))
until T = T'
```

- closure(q_0) $= \{q_0, q_1, q_2\}$
- closure(q_1) $= \{q_1, q_2\}$
- closure(q_2) $= \{q_2\}$

Definições

- $d = \{s_i, s_k, s_l\}$
- `DFAedge(d,c)` — estados alcançados a partir de d e consumindo a entrada c.

$$\text{DFAedge(d,c)} = \text{closure}(\bigcup_{s \in d} \text{edge(s,c)})$$

```
d ← closure[s1]
for i ← 1 to k
   d ← DFAedge(d, c_i)
```

DFAedge verifica os estados alcançados a partir de um conjunto de estados consumindo uma determinada entrada, incluindo os possíveis movimentos com o ϵ. Como o nosso exemplo não utiliza ϵ, o resultado para DFAedge é similar ao resultado obtido por edge.

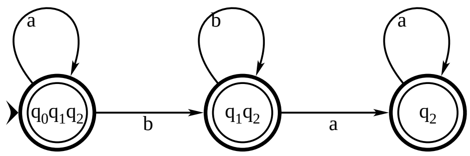
- DFAedge($\{q_0\}$, a) $= \{q_0, q_1, q_2\}$
- DFAedge($\{q_1\}$, b) $= \{q_1, q_2\}$
- DFAedge($\{q_2\}$, a) $= \{q_2\}$
- DFAedge($\{q_0, q_1\}$, a) $= \{q_0, q_1, q_2\}$
- DFAedge($\{q_0, q_1\}$, b) $= \{q_1, q_2\}$
- DFAedge($\{q_0, q2\}$, a) $= \{q_0, q_1, q_2\}$
- DFAedge($\{q_1, q2\}$, a) $= \{q_2\}$
- DFAedge($\{q_1, q2\}$, b) $= \{q_1, q_2\}$
- DFAedge($\{q_0, q_1, q2\}$, a) $= \{q_0, q_1, q_2\}$
- DFAedge($\{q_0, q_1, q2\}$, b) $= \{q_1, q_2\}$

Algoritmo AFN ⇒ AFD

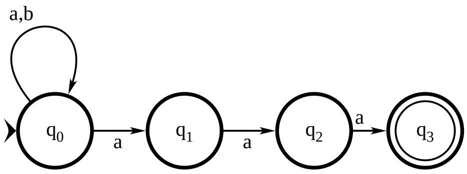
```
states[0] ← {}
states[1] ← closure(s₁)
p ← 1
j ← 1
while j ≤ p
  foreach c ∈ ∑
    e = DFAedge(states[j],c)
    if e = states[i] for some i ≤ p
      then trans[j,c] ← i
      else p ← p + 1
          states[p] ← e
          trans[j,c] ← p
  j ← j + 1
```

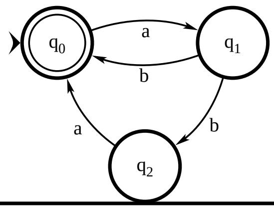
Este algoritmo transforma qualquer AFN em um AFD equivalente. Você deve transformar todo estado que possui um estado final do AFN como estado final do AFD.

Ao transformar o AFN do exemplo em AFD, teremos o seguinte resultado:



Transforme os AFNs em AFDs

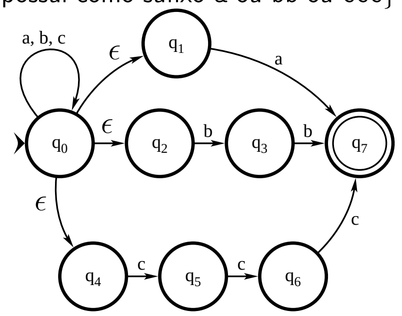




O resultado deste exercício encontra-se na aula 4. Observe que ao aplicar este algoritmo em um AFN, teremos como resultado o mesmo AFD obtido pelo primeiro algoritmo (visto na aula4).

Sugestão de exercícios: você pode transformar os AFNs da aula 4 em AFDs.

Escreva um AFϵ para a linguagem abaixo. Transforme o AFϵ em um AFD. $L = \{w \mid w$ possui como sufixo a ou bb ou ccc$\}$



AFD:

$p_0 = \{q_0q_1q_2q_4\}$
$p_1 = \{q_0q_1q_2q_4q_7\}$
$p_2 = \{q_0q_1q_2q_4q_3\}$
$p_3 = \{q_0q_1q_2q_4q_5\}$
$p_4 = \{q_0q_1q_2q_3q_4q_7\}$
$p_5 = \{q_0q_1q_2q_4q_5q_6\}$
$p_6 = \{q_0q_1q_2q_4q_5q_6q_7\}$