```
type 'a exp =                    (* 'a é o tipo das letras *)
| Vazia                          (* linguagem vazia *)
| Palavra of ('a vect)           (* linguagem de uma palavra *)
| Soma of ('a exp) * ('a exp)    (* união de linguagens *)
| Concat of ('a exp) * ('a exp)  (* concatenação de linguagens *)
| Estrela of ('a exp)            (* exponenciação duma linguagem *)

(* "pertence" diz se a palavra u pertence a
   linguagem regular e *)

let rec pertence (e, u) =
    match e with
      | Vazia -> false
      | Palavra(m) -> u = m
      | Soma (e1, e2) -> pertence (e1, u) or pertence (e2, u)
      | Concat (e1, e2) -> conta (e1, e2, u, 0)
      | Estrela(e1) -> u = [| |] or conta (e1, e1, u, 1)
and conta (e1, e2, u, i) =
    if i > vect_length(u) then false
    else let v = sub_vect u 0 i
         and w = sub_vect u i ((vect_length u) - i) in
         (pertence (e1, v)) && (pertence (e2, w)) or conta (e1, e2, u, i+1)
```

Autómatos finitos.

- Sabemos gerar palavras de uma determinada linguagem L. → gramáticas
- Sabemos descrever linguagens
  - → gramáticas
  - → expressões regulares (para as linguagens regulares)

Como reconhecer palavras duma determinada linguagem? → autómatos.

Vamos nos concentrar nas linguagens regulares

→ Autómatos finitos

Autómato = máquina com estados (em número finito) finito) aceitando dados em entrada e que muda de estado em função desses dados.

3 tipos de estados:

- estados iniciais
- estados finais
- estados intermédios

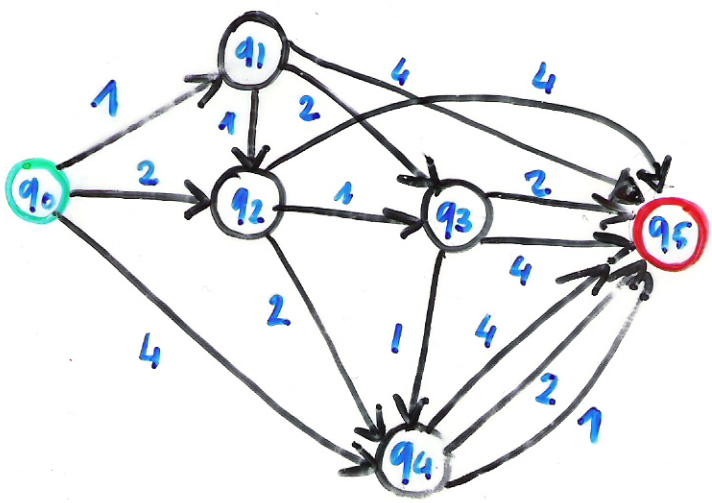
Exemplos:

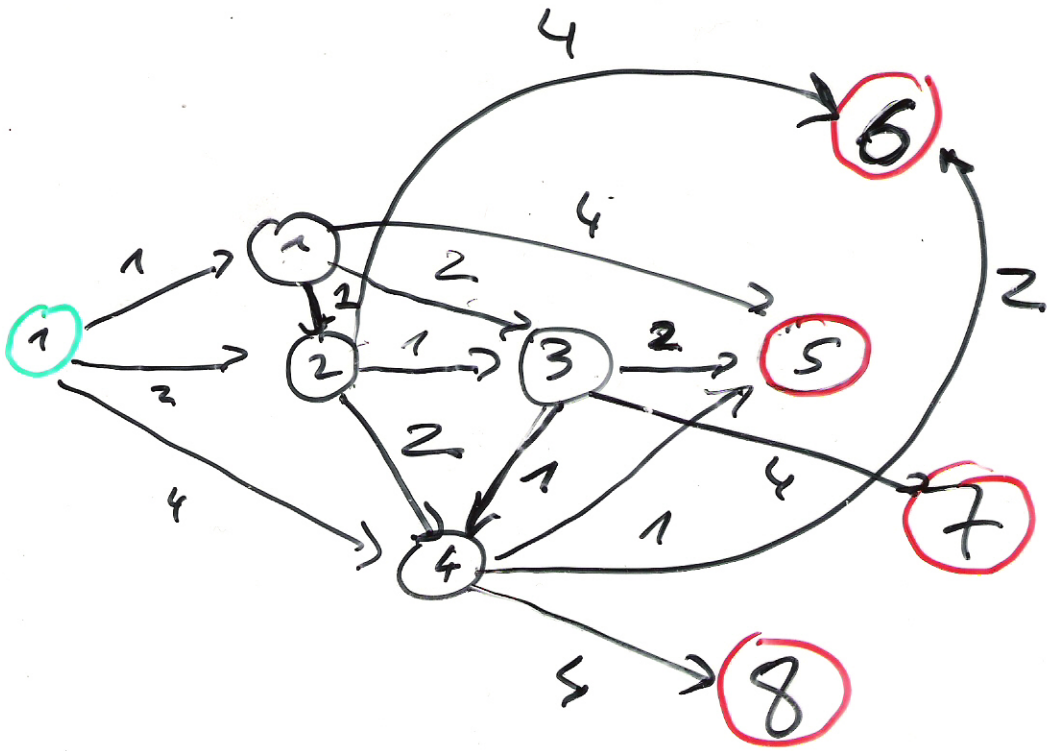
Máquina de Café

moedas: 1, 2, 4

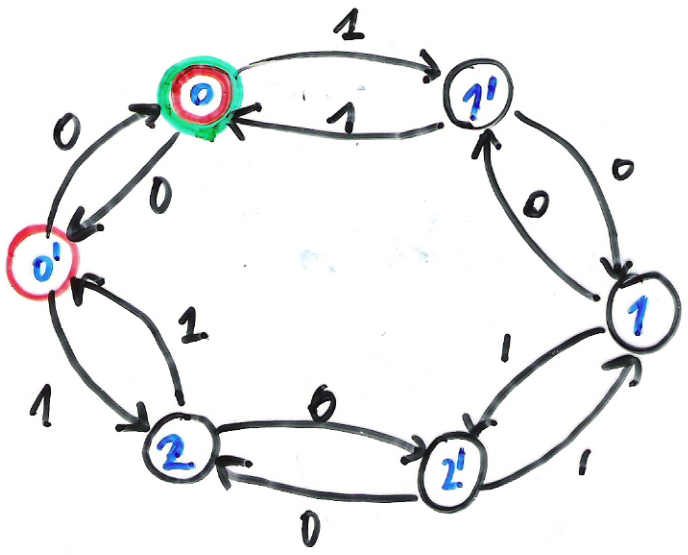
Café: 5

Soma máxima: 8




4
6
4
1
2
1
2
2
1
2
3
5
2
2
1
4
4
1
4
7
4
8

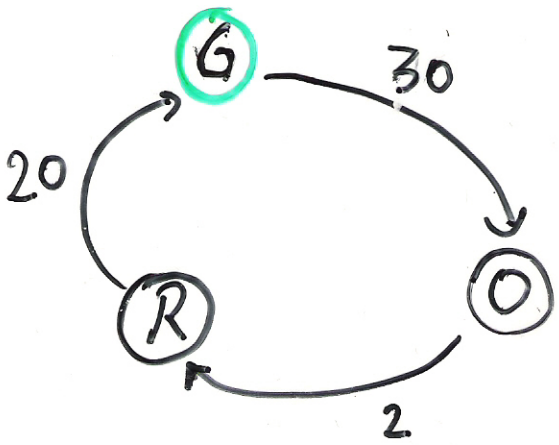
Exemplo 2:



Autómato que lê um número binário (da direita para a esquerda) e que aceita o número se este é divisível por 3.

19 → 10011 acaba em 1'

87 → 1010111 acaba em 0'



G = Verde

R = Vermelho

O = Laranja

i: segundos

Definição: Autómatos finitos (não determinísticos)

Seja A um alfabeto. Seja $A_\varepsilon = A \cup \{\varepsilon\}$

Um autómato finito sobre o alfabeto A é um 4-tuplo $\mathcal{A} = (Q, T, I, F)$

Abreviatura: NDFA

- Q: Conjunto finito (conjunto dos estados)
- $T \subseteq Q \times A_\varepsilon \times Q$: Conjunto das transições
  - T é uma relação
- $I \subseteq Q$: conjunto dos estados iniciais
- $F \subseteq Q$: estados finais

uma transição (q, ε, q') é uma ε-transição.

exemplo: $\mathcal{A} = \{Q, T, I, F\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ $\quad I = \{q_0\}$

$F = \{q_5\}$

$T = \{(q_0, 1, q_1), (q_0, 2, q_2), (q_0, 4, q_4), (q_1, 1, q_2)$
$(q_1, 2, q_3), (q_1, 4, q_5), (q_2, 1, q_3), (q_2, 2, q_4)$
$(q_2, 4, q_5), (q_3, 1, q_4), (q_3, 2, q_5), (q_3, 4, q_5)$
$(q_4, 1, q_5), (q_4, 2, q_5), (q_4, 4, q_5)\}$

Linguagem aceite por um autómato.

Definição:

- Seja $\mathcal{A} = \{Q, T, I, F\}$ um autómato.
  Um **caminho** em $\mathcal{A}$ é uma sequência c de arestas consecutivas,

$$c = (q_0, a_1, q_1)(q_1, a_2, q_2) \cdots (q_{n-1}, a_n, q_n)$$

ou

$$c: \; q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \; \cdots \xrightarrow{a_n} q_n$$

Diz-se então que o caminho c tem o **comprimento** n. Seja u a palavra $a_1 a_2 \ldots a_n$.
Diz-se que u é a **etiqueta** de c e que

q_0 é o **estado inicial** de c.
q_n é o **estado final**

Diz-se também que c é um caminho de q_0 para q_n (notação $q_0 \xrightarrow{u} q_n$)

Convenção: existe sempre um caminho vazio de q para q de etiqueta ε.

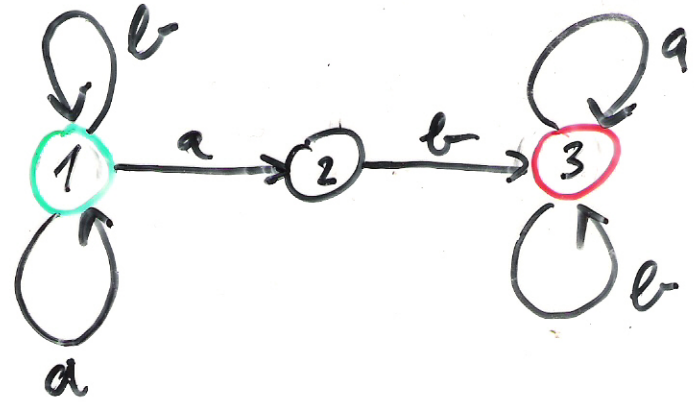
Definição:

Seja $\mathcal{A} = \{Q, T, I, F\}$ um autómato.

- Um caminho c é chamado de bem sucedido se o seu estado inicial pertence a I e se o seu estado final pertence a F.
- Uma palavra w é reconhecida / aceite por $\mathcal{A}$ se existe um caminho bem sucedido de etiqueta w.
- A linguagem reconhecida por $\mathcal{A}$ é o conjunto das etiquetas dos caminhos bem sucedidos. notação: $\mathcal{L}(\mathcal{A})$

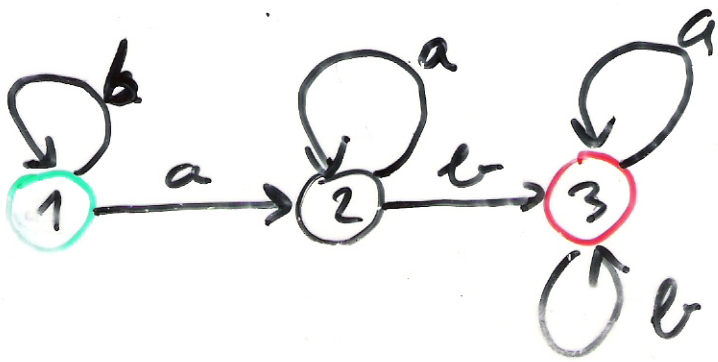
diz-se também que $\mathcal{A}$ aceita / reconhece $\mathcal{L}(\mathcal{A})$

A_1



$\mathcal{L}(A_1) = (a+b)^* a\, b(a+b)^*$

A_2



$\mathcal{L}(A_2) = ?$

Definição: Linguagens | aceitáveis
reconhecíveis

Seja A um alfabeto.

- Uma linguagem L sobre o alfabeto A é aceitável se existe um autómato finito sobre A que reconhece L.
- o conjunto de todas linguagens aceitáveis é notado $Rec(A^*)$.

↑ conjunto das linguagens (subconjuntos de A^*)

Reconhecíveis

Teorema de Kleene:

Seja A um alfabeto:

$$Reg(A^*) = Rec(A^*)$$

Demonstração: (... só uma ilustração de uma parte)

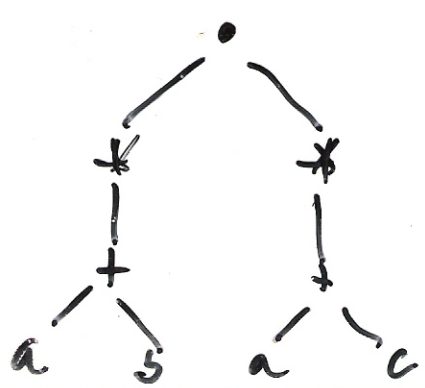
$Reg(A^*) \subseteq Rec(A^*)$

ou seja qualquer expressão racional pode ser descrita por um autómato:

Vamos descrever um processo automático (algoritmo) de construção de autómatos a partir de expressões regulares R.

(1) podemos descrever uma expressão regular por uma árvore:

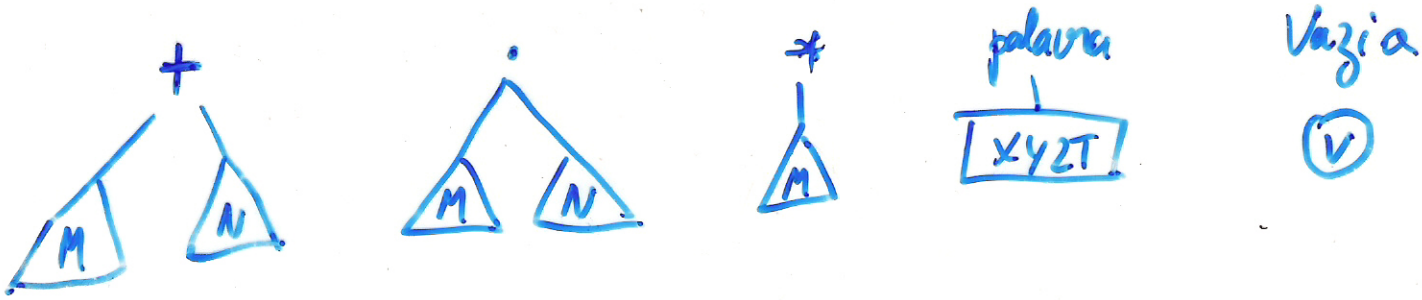
$(a+b)^* . (a+c)^* =$

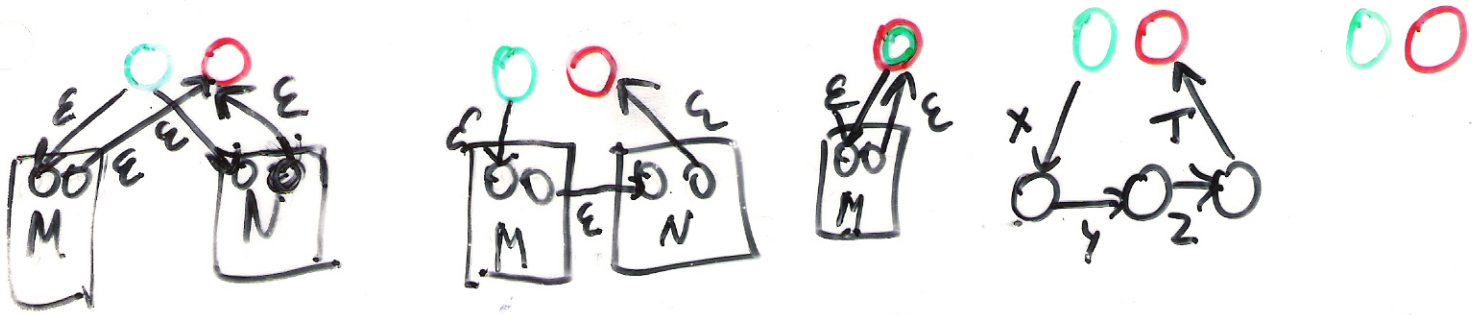


- A cada nó desta árvore associamos dois estados: | um de entrada / um de saída

para a estrela, um só estado

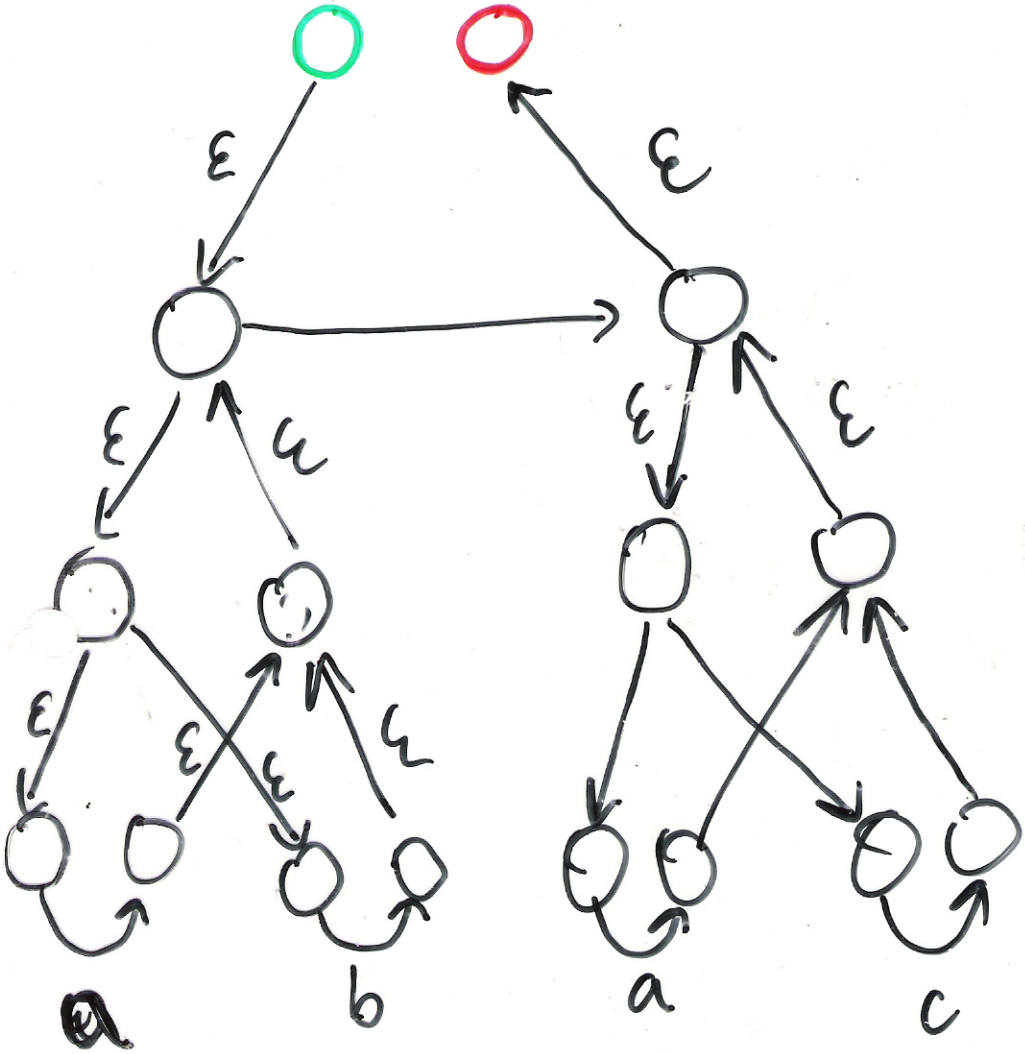
- colocamos transições entre esses estados e os estados dos nodos filhos conforme a figura seguinte:

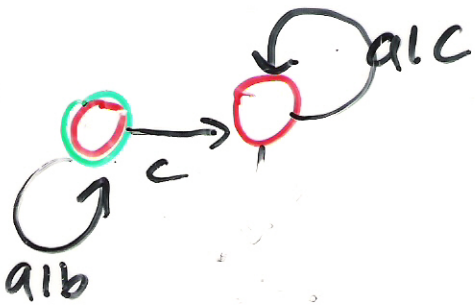




(M.+N) (MN) M* XYZT ∅

palavra vazia ε





← sem ε-transições
. determinístico
. mínimal.

$$(a+b)^* \cdot (a+c)^*$$

Autómatos determinísticos: (DFA)

Seja $\mathcal{A} = \{Q, T, I, F\}$ um autómato (com ε-transições).

Se: ① $\#(I) = 1$ (um só estado inicial)

② Seja $q \in Q$ e $a \in A$.

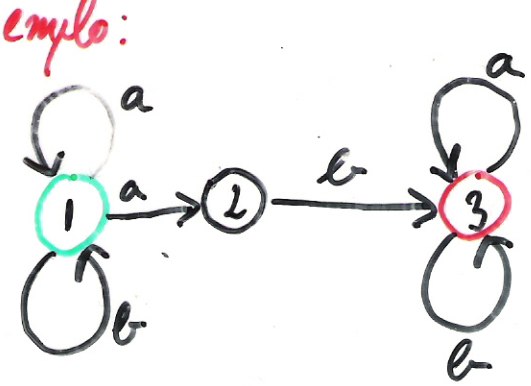
$$(\forall\, q', q'' \in Q,\ (q, a, q') \in T \ \text{e}\ (q, a, q'') \in T \implies q' = q'')$$

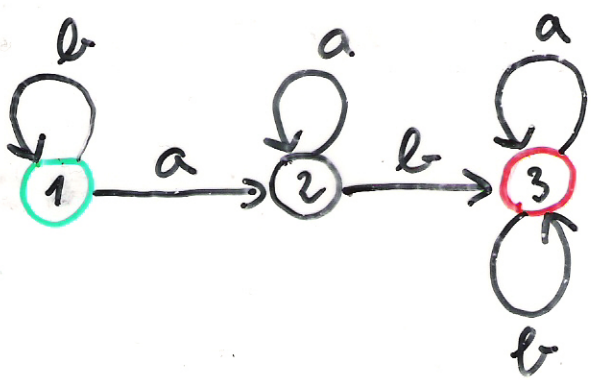
③ $\mathcal{A}$ não tem ε-transições

então diz-se que

$\mathcal{A}$ é **determinístico**

exemplo:





$\mathcal{A}_1$ não é determinístico $\mathcal{A}_2$ é determinístico.

$(1, a, 2) \in T_{\mathcal{A}_1} \wedge (1, a, 1) \in T_{\mathcal{A}_1}$

Vantagens dos autómatos determinísticos

→ pretendemos verificar se uma palavra w é reconhecida por um autómato A

- se A não for determinístico:
  - verificar todos os caminhos que partem dum estado inicial e que têm w como etiqueta. Verificar se um deles chega a um estado final
- se A for determinístico:
  verificar se o caminho etiquetado por w e partindo do único estado inicial é bem sucedido

Proposição:

. Seja $\mathcal{A}$ um autómato determinístico $\mathcal{A} = \{Q, T, I, F\}$

. Seja w uma palavra de A^*

① $\forall q \in Q$, existe no máximo um caminho partindo de q com a etiqueta w.

② Se $w \in \mathcal{L}(\mathcal{A})$ então w é a etiqueta dum caminho bem sucedido.

(e único)

i.e. não há dois caminhos $\neq$ bem sucedidos com w como etiqueta.

De facto, o que é importante ver, num autómato determinístico, é que a relação T é uma relação funcional: (i.e pode ser vista como uma relação)

Definição

Seja $\mathcal{A} = \{Q, T, I, F\}$ um autómato determinístico sobre um alfabeto A.

A **função de Transição** $\delta : Q \times A_\varepsilon \longrightarrow Q$ é a função que associa a $q \in Q$ e $a \in A_\varepsilon$ o estado q' tq $(q, a, q') \in T$.

δ é uma função parcial (este q' pode não existir).

Definição • Seja $\mathcal{A} = \{Q, T, I, F\}$ um autómato determinístico sobre um alfabeto A.

• Seja δ a sua função de transição.

A **função de transição estendida** de $\mathcal{A}$ é a função $\bar{\delta} : Q \times A^*_\varepsilon \longrightarrow Q$ que associa a todo o par (q, w) o estado q' se existir um caminho de q para q' com etiqueta

Proposição: $\bar{\delta}$ definida de forma única por recursão estrutural sobre a palavra u.

$$\begin{cases} \bar{\delta}(q, \varepsilon) = q \\ \bar{\delta}(q, ua) = \delta(\bar{\delta}(q, u), a) \text{ se } \bar{\delta}(q,u) \text{ existe e } \delta(\bar{\delta}(q,u), a) \text{ !!} \\ \text{indefinido senão} \end{cases}$$

com $q \in Q$
$a \in A$
$u \in A^*$

O que vem a seguir?

- representar e executar autómatos
  - não determinísticos
  - determinísticos

$NDFA_\varepsilon$ ↓ ← • remover ε-transições dos autómatos ND

NDFA ↓ ← • determinisar NDFA

DFA ↓ ← • minimizar DFA

DFA_{min}

Eliminação das ε-transições:

passo 1) para todo (p, a, q) de T onde $a \in A$ $(\neq \varepsilon)$ acrescentar as transições seguintes:

(p, a, q') onde q' é um estado

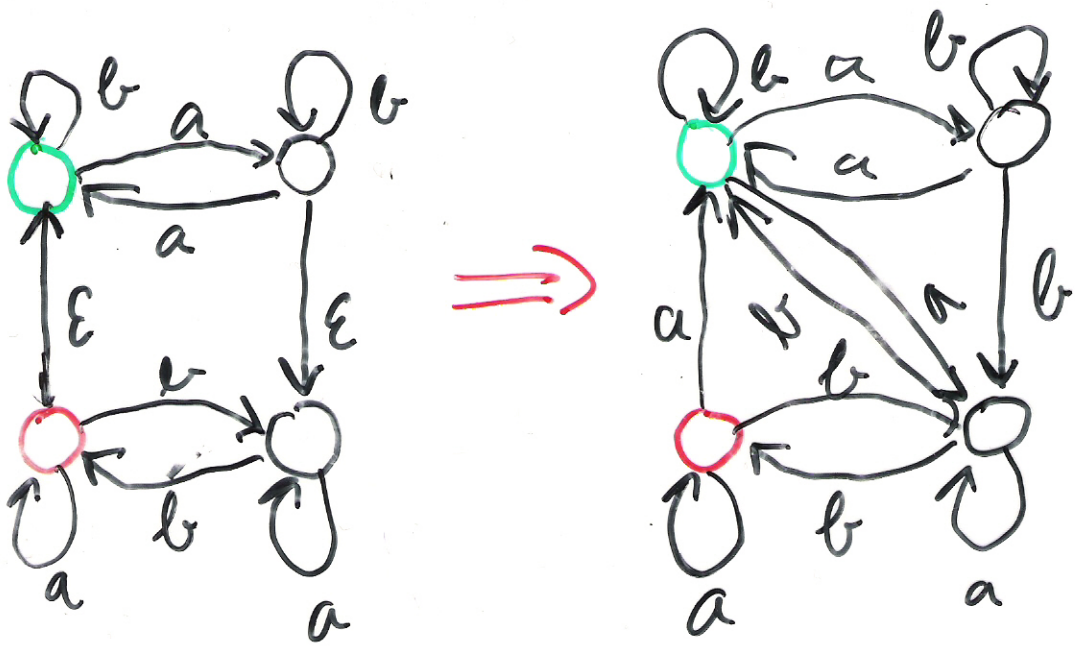
t.q. $\exists$ um caminho de q a q' etiquetado exclusivamente por ε-transições.

propriedade de correção

passo 2) • acrescentar a I todo o estado acessível exclusi. por ε-transições a partir de estados iniciais

passo 3) • acrescentar a F todo o estado que acede aos estados finais com transições ε.

Prop: As Linguagens reconhecidas coincidem!



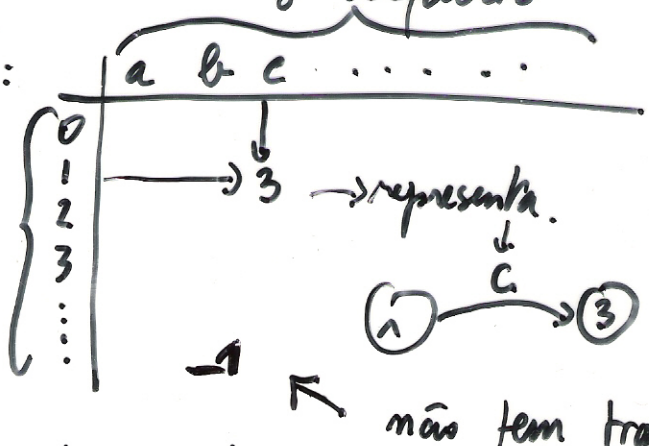
Representação e execução de autómatos (DFA e NDFA)

DFA: $A = \{Q, \delta, I, F\}$

↑ preferível a T.

A dificuldade: representar a função de transição.

- por uma matriz:



- por uma função "directa".

```
(* e: estado,  l: letra *)
let transição (e,l) =
    match (e,l) with
      | (0,'a') -> ....
      | ... | (1,'c') -> 3 ... | (42,'c')
      | _ -> fail with "impossível"
    end
```

```
(* 'a : tipo dos estados,  'b: tipo das letras *)
type ('a,'b) autómato =
  { inicial : 'a;
    final : 'a -> bool   (* alternativa: 'a list *)
    transição : ('a * 'b) -> 'a
  };;
```

open Array;;
length(m)

```
let análise (a, m) =
    try  let e = ref (a.inicial) in
         for i=0 to vect_length(m) - 1 do
             e := a.transição (!e, m.(i))
         done;
         a.final(!e)   (* mem !e a.final *)
    with Failure "_" -> false
```

Outra solução:

```
let rec estado_0 u =
    match u with
    | 'a' :: v -> estado_x v
    | ...
    | []       -> false
and estado_1 u =
    match u with
    | 'a' :: v -> estado_y v
    | ...
    | 'c' :: v -> estado_3 v
    | ...
    | []       -> true
```

← o estado 0 não é final.

← o estado 1 é final

...

Autómatos não deterministicos:

- Caso genérico: baseado em grafos.

```
type 'a letra = Epsilon | Letra of 'a
and ('a,'b,'c) automato =
    {
      g : ('a, 'b, ('c letra list)) grafo ;
      inicial : 'a list ;
      final : 'a list
    }
```

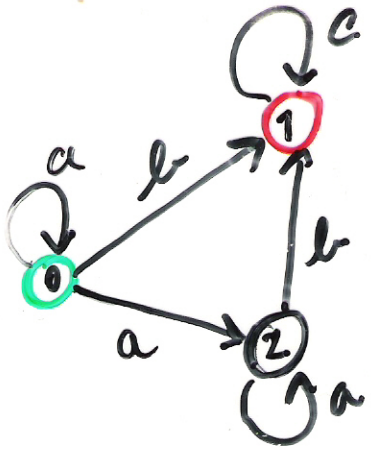
tipo dos nodos — tipo das etiquetas (dos nodos) — tipo das etiquetas dos arcos

- Caso "mais simples".

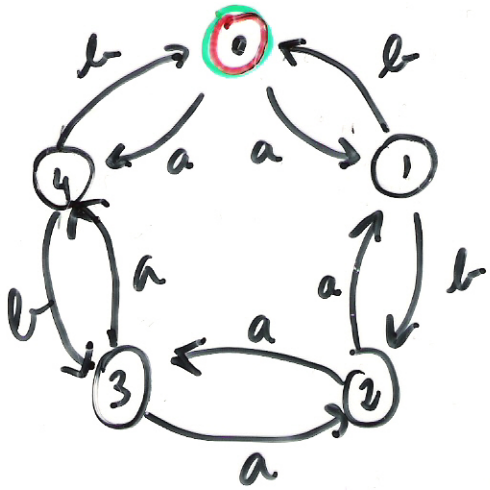
```
type ('a,'b) autómato = {
        inicial : 'a list ;
        final : 'a -> bool ;
        transição : ('a * 'b) -> 'a list } ;;
```

```
let rec sucessores (a, lista, letra) =
    match lista with
      | [] -> []
      | x::resto -> union (a.transição (x, letra))
                          (sucessores (a, resto, letra))
;;

let analise (a, u) =
    let l = ref (a.inicial) in
      for i = 0 to vect.length (u) - 1 do
          l := sucessores (a, !l, u.(i))
      done ;
      exist a.final (fun x -> mem x !l)
;;
```



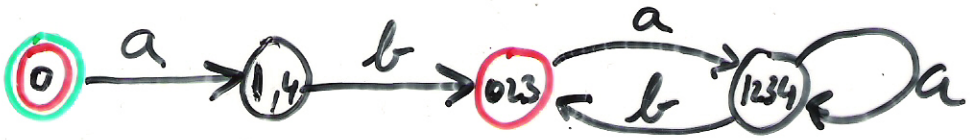

```
let a = {
    inicial = [0];
    final = (fun x -> x = 1);
    transições = (fun (0,'a') -> [0;2]
                  | (0,'b') -> [1]
                  | (1,'c') -> [1]
                  | (2,'a') -> [2]
                  | (2,'b') -> [1]
                  | _       -> [])
};;
```



Determinisação:

NDFA → DFA

Partida	Letra	Chegada
{0}	a	{1,4}
{0}	b	∅
{1,4}	a	∅
{1,4}	b	{0,2,3}
{0,2,3}	a	{1,2,3,4}
{0,2,3}	b	∅
{1,2,3,4}	a	{1,2,3,4}
{1,2,3,4}	b	{0,2,3}



Algoritmo: $\mathcal{A} = \{Q, T, i, F\}$

Seja $\mathcal{A}$ um NDFA sobre o alfabeto A.

O "truque" é considerar um autómato $\mathcal{A}'$ no qual os estados são subconjuntos de estados de $\mathcal{A}$.

(Se $\mathcal{A}$ tem n estados então $\mathcal{A}'$ tem no máximo 2^n estados)

- o estado inicial de $\mathcal{A}'$ é o subconjunto de estados acessíveis por ε-transições a partir dos estados iniciais de $\mathcal{A}$.

ex: Se 0, 1, 3 são acessíveis em $\mathcal{A}$ por ε-transições então o estado inicial de $\mathcal{A}'$ é: $\{0, 1, 3\}$

- Para um conjunto E de estados de $\mathcal{A}$ pertencendo a $\mathcal{A}'$, e uma letra a.

pertence a $\mathcal{A}'$ se existir um conjunto não vazio E' de estados de $\mathcal{A}$ acessíveis dos estados de E por uma transição de etiqueta a.

- repetir o processo até não produzir nenhum estado novo.

- os estados finais de A' são os estados contendo estados finais de A.

Ex: $\{xyz\} \in A'$

se x é final é final em A
então xyz é final em A'.

minimização de um DFA.

o objectivo: . Sejam $A_1, \ldots A_m$ DFA que reconhecem a mesma linguagem $\mathcal{L}$

→ Mostrar (e construir) um autómato "minimal" ou "canónico" A_{min} que tem um número mínimo de estados e que reconhece a mesma linguagem $\mathcal{L}$.

→ este autómato tem boas propriedades.

É único (módulo o nome dos estados)

assim verificar se dois autómatos são equivalentes (se reconhecem a mesma linguagem)

é decidível.

Algoritmo: calcular os autómatos minimais e comparé-los. Se forem iguais então são equivalentes

Simulação:

Um autómato A_1 simula um autómato A_2 se $\forall$ c caminho bem sucedido em A_2 de etiqueta w então existe um caminho c' em A_1 de etiqueta w igualmente bem sucedido.

Definição: (simulação)

$A_1 = \{Q_1, \delta_1, q_0^1, F_1\}$

$A_2 = \{Q_2, \delta_2, q_0^2, F_2\}$

A_1 e A_2 são DFA sobre o alfabeto A

A_2 simula A_1 se existir uma aplicação φ de $Q_1 \xrightarrow{\text{para}} Q_2$ tq

① $\forall q \in Q_1, \forall a \in A$, se $\delta_1(q,a)$ existe então $\delta(\varphi(q),a)$ também existe e $\varphi(\delta_1(q,a)) = \delta_2(\varphi(q),a)$

② $\varphi(q_0^1) = q_0^2$

(3) $\varphi^{-1}(F_2) = F_1$

φ é designada de simulação de A_1 por A_2

Lema: Se A_2 simula A_1 então

$$\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$$

Definição (isomorfismo de autómatos)

Sejam A_1 e A_2 dois DFAs.

Se A_1 simula A_2 por φ (bijectiva) e A_2 simula A_1 por φ^{-1} então diz-se que A_1 e A_2 são isomórficos.

Equivalência de Nerode

Seja $\mathcal{A} = \{Q, \delta, I, F\}$ um DFA.

- definimos para $q \in Q$ a linguagem $\mathcal{L}(q)$ como:

$$\mathcal{L}(q) = \{u \in A^* \mid \overline{\delta}(q,u) \in F\}$$

↑ conjunto das etiquetas (palavras) de caminhos bem sucedidos partindo de q.

- $\boxed{q \sim q' \text{ se } \mathcal{L}(q) = \mathcal{L}(q')}$

↑ equivalência de Nerode.

trata-se de facto duma relação de equivalência (é reflexiva, transitiva e simétrica)

Sendo $\sim$ uma relação de equivalência podemos então falar de classes de equivalência.

$$[q] = \{ q' \in Q \mid q' \sim q \}.$$

↑ classe de equivalência de q.

repara que

$$q \sim q' \Rightarrow \delta(q,a) \sim \delta(q',a)$$

O Autómato de Nerode $A_\sim$ associado a um autómato A é assim o autómato quociente de A pela relação $\sim$:

$$A_\sim = \{ [Q], \delta_\sim, [q_0], [F] \}$$

onde:

$$[Q] = \{ [q] \mid q \in Q \}$$

$$[F] = \{ [q] \mid q \in F \}$$

Para todo o $q \in Q$ e $a \in A$ $\quad \delta_{\sim}([q], a) = [\delta(q, a)]$

Se $\mathcal{A}$ for completo $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_{\sim})$

o autómato $\mathcal{A}_{\sim}$ é minimal

(sem demonstração ...)

Existirá um algoritmo prático de cálculo da relação $\sim$?

Sim!

Algoritmo para calcular $\sim$:

→ método "iterativo" baseado sobre um cálculo de ponto fixo!

- para qualquer $n \in \mathbb{N}$, seja $A^{\leq n}$ o conjunto das palavras de A^* de comprimento $\leq n$.

- definimos a relação $\sim_n$ por

$$q \sim_n q' \triangleq L(q) \cap A^{\leq n} = L(q') \cap A^{\leq n}$$

objectivo: calcular $\sim$ "por refinamento":

$\sim_0, \sim_1, \sim_2 \cdots \sim_i \sim_{i+1} \cdots \cdots$

até um j tal que $\sim_j = \sim_{j+1}$

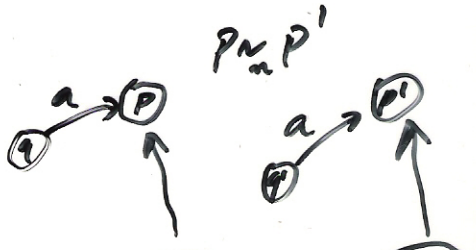
neste caso $\sim = \sim_j$

repare que:

- $\forall n \in \mathbb{N},\ \forall q, q' \in Q \quad q \nu_{n+1} q' \Rightarrow q \sim_n q'$

- $\forall q q' \in Q,\ q \nu q' \iff \forall n \in \mathbb{N},\ (q \nu_n q')$

Temos então:



Lema: Seja $n \in \mathbb{N}$.

$$q \nu_{n+1} q' \iff q \nu_0 q' \wedge (\forall a \in A,\ \delta(q,a) \nu_n \delta(q',a))$$

$$\iff q \nu_n q' \wedge (\forall a \in A,\ \delta(q,a) \nu_n \delta(q',a))$$

$$(\nu_n \subseteq \nu_{n+1})$$

Corolário:

$$\forall n \in \mathbb{N} \quad \nu_n = \nu_{n+1} \Rightarrow \forall k \in \mathbb{N} \quad \nu_{n+k} = \nu_n$$

Temos assim todos os elementos que justificam que ν pode ser obtido como o menor ponto fixo.

$\perp$ ← relação vazia

$$\nu_0 \to \nu_1 \to \nu_2 \to \cdots \boxed{\nu_k \to \nu_{k+1}}$$

até $\nu_k = \nu_{k+1}$

neste caso $\boxed{\nu_k = \nu}$

Por outras palavras

Seja $C_Q = (Q \times Q, \subseteq)$ o conjunto das relações binárias sobre Q ordenado pela relação de inclusão.

→ C_Q é bem fundado.

→ C_Q é um Domínio (cpo com $\perp$)

Seja $T : Q \times Q \longrightarrow Q \times Q$

$$T\,R \triangleq q\,R\,q' \wedge (\forall a \in A,\ \delta(q,a)\,R\,\delta(q',a))$$

repare que

a) $T \perp = T \emptyset = \nu_0$

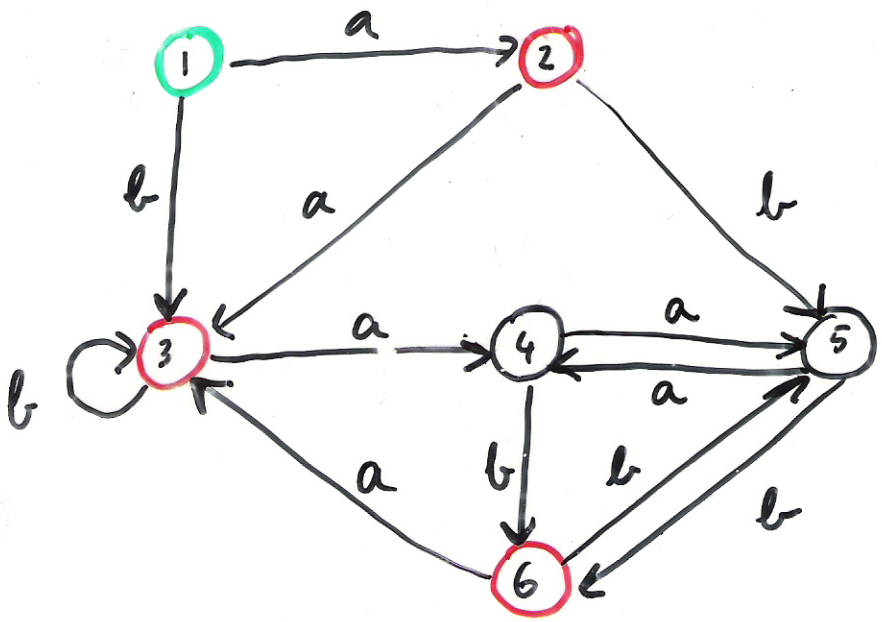
b) $T\,\nu_n = \nu_{n+1}$

c) $\forall R \subseteq Q \times Q$
$R \subseteq T\,R$

ν : mais pequeno ponto fixo de T

→ justifica o algoritmo: calcular os sucessivos $\nu_0\ \nu_1 \ldots \nu_k\ \nu_{k+1}$ até $\nu_k = \nu_{k+1}$.
$\nu_{k+1} = T(\nu_k) = \nu_k$!!

exemplo: A : determinista completo:



ν_0 : 2 classes de equivalencia (F e Q/F)

- $N_0 = \{2, 3, 6\} \quad \{1, 4, 5\}$

1	$L(1) \cap A^{\leq 0}$	$= \emptyset$
2	"	$= \{\varepsilon\}$
3	"	$= \{\varepsilon\}$
4	"	$= \emptyset$
5	"	$= \emptyset$
6	"	$= \{\varepsilon\}$

- ν_1 ?

$L(1) \cap A^{\leq 1} = \{a, b\}$
$L(2) \cap A^{\leq 1} = \{\varepsilon, a\}$
$L(3) \cap A^{\leq 1} = \{\varepsilon, b\}$
$L(4) \cap A^{\leq 1} = \{b\}$
$L(5) \cap A^{\leq 1} = \{b\}$
$L(6) \cap A^{\leq 1} = \{\varepsilon, a\}$

Logo ν_1 : 4 classes de equivalência :

$N_1 = \{2, 6\} \quad \{3\} \quad \{1\} \quad \{4, 5\}$

ν_2 ?

$L(1) \cap A^{\leq 2} = \{a, b, aa, bb\}$

$L(2) \cap A^{\leq 2} = \{\varepsilon, a, ab, bb\}$

$L(3) \cap A^{\leq 2} = \{\varepsilon, b, ab, bb\}$

$L(4) \cap A^{\leq 2} = \{b, ab, ba\}$

$L(5) \cap A^{\leq 2} = \{b, ab, ba\}$

$L(6) \cap A^{\leq 2} = \{\varepsilon, a, ab, bb\}$

4 classes de equivalências

ν_2: $\{2, 6\}$ $\{3\}$ $\{1\}$ $\{4, 5\}$

→ igual a ν_1 !

algumas considerações: 1) não extender os cálculos aos elementos das classes com 1 elemento (ex: 1, 3)

2) o algoritmo, de facto, tenta partir as classe de equivalência até ao máximo.

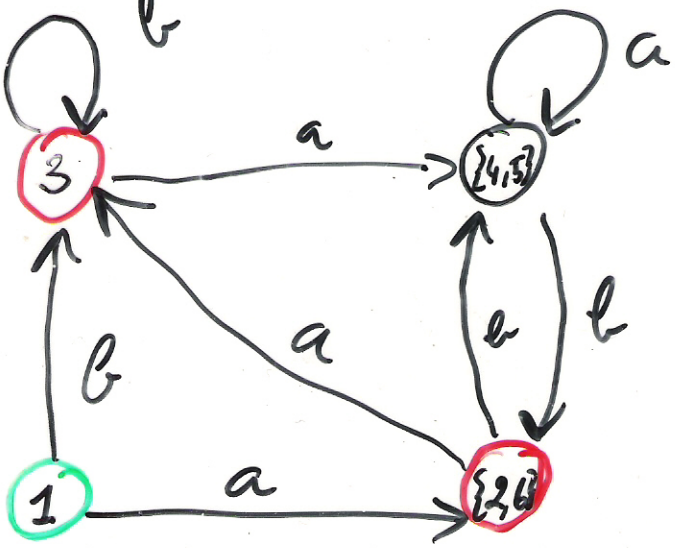
$\nu = \{\{2,6\}\ \{3\}\ \{1\}\ \{4,5\}$ $I = 1$ $F = \{\{1,3,6\}$ (4)

	ε	a	b
1	1	2	3
2	2	3	5
3	3	4	3
4	4	5	6
5	5	4	6
6	6	3	5

ν_0 ν_1

	aa	ab	bb	ba
1				
2	4	3	4	6
3				
4	4	6	3	5
5	6	6	3	5
6	4	3	4	6

ν_2



Calcular o autómato de Nerode (o autómato mínimo) de