Teoria da Computação

Autómatos finitos

Simão Melo de Sousa

Computer Science Department
University of Beira Interior, Portugal

Plano

Plano

Objectivos

- Autómatos finito: primeira abordagem a modelização da noção de procedimento efectivo. Mas igualmente úteis noutros contextos (processamento de strings, análise léxica, etc...)
- Derivar da noção de autómatos finitos a noção de programa executado num mecanismo (como os computadores).
- Nestes mecanismos é importante realçar as componentes seguintes:
  - o dispositivo de aquisição de dados (I/O, eventualmente infinito)
  - dispositivos de armazenamento de dados (Memória - RAM por exemplo)
  - dispositivo de armazenamento do programa (Memória - ROM por exemplo)
  - dispositivo de execução (o processador)
  - registo do estado da execução (Program counter, registos do processador, etc,)
- Qualquer dispositivo, mesmo sendo teórico, deve contemplar estes aspectos.

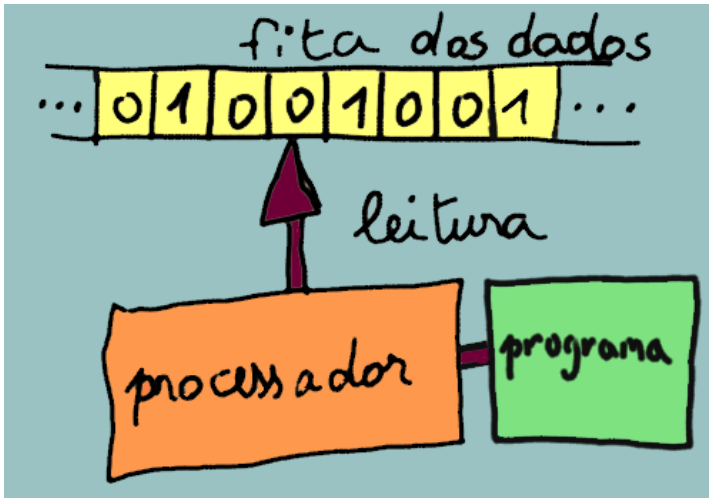
Objectivos

- um estado = configuração dos registos do processador, do *program counter*, da memória e de todos os outros recursos
- Um passo de execução = modificação do estado
- Um exemplo. Uma máquina com um processador com 8 Mb de RAM, 16 registos de 32 bits:
  - 1 byte = 8 bits = 2^3 bits
  - 1 Mb = 1024×1024 bytes = $2^{10} \times 2^{10}$ bytes = 2^{20} bytes = 2^{23} bits
  - 8 Mb = $2^3 \times 2^{23}$ bits = 2^{26} bits
  - 16 registos de 32 bits = $2^4 \times 2^5$ bits = 2^9 bits
  - um estado = $2^{26} + 2^9$ bits = 67109376
  - número de estados possíveis = $2^{67109376}$!!!!!!! (número estimado de átomos no universo = 10^{80})

Objectivos

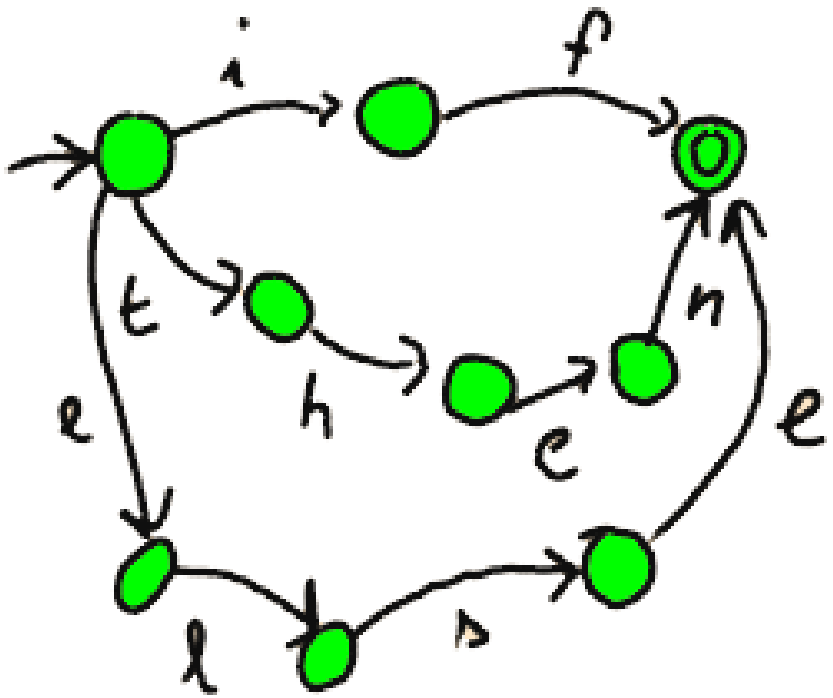
- um passo de execução (bis) = uma **relação R de transição** entre os estados. Ou seja $a\ R\ b$ significa "o processo de execução modifica o estado a no estado b". Quando a execução é determinista (há um só estado "destino"possível em cada passo de execução) a relação é uma **função de transição**.
- Como se constrói R? A partir do programa e da máquina que executa.

Primeira abordagem a noção de *Processo Efectivo*



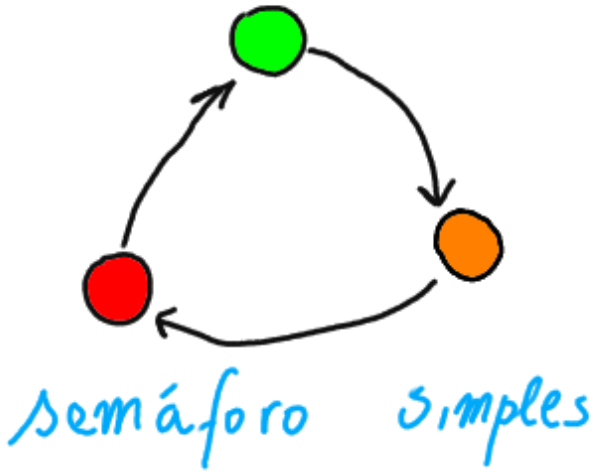
automáto (um grafo particular)	programa/algoritmo
estado/nodo do autómato	uma configuração da memória
conjunto dos vértices	relação de transição
percurso do autómato	execução
relação de transição	execuções possíveis
estado inicial	a configuração inicial da memória
fita	dados de entrada

Exemplos de autómato

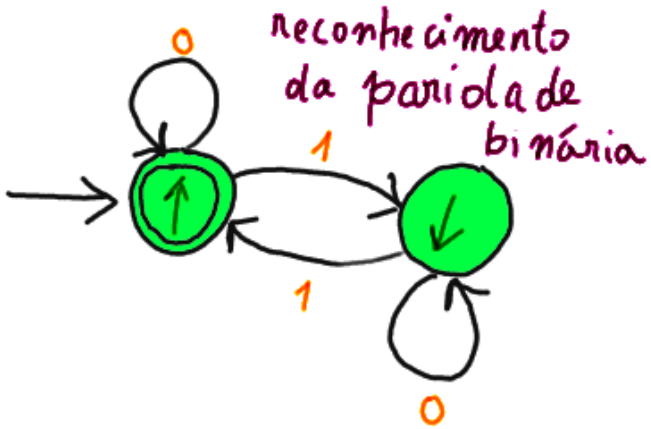


Reconhecimento léxico
(if then else)

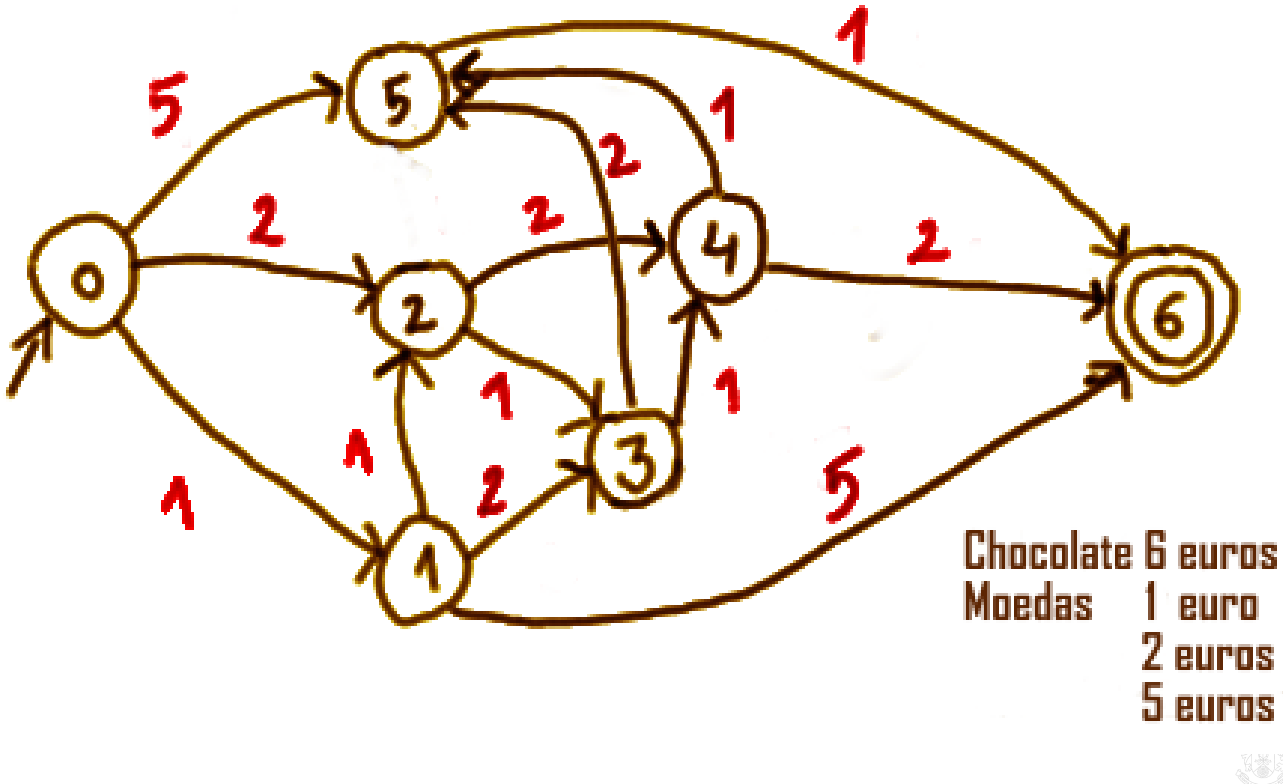
Exemplos de autómato



Exemplos de autómato

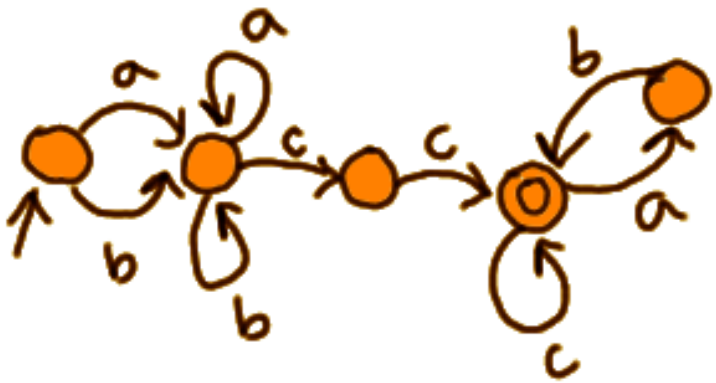


Exemplos de autómato



Exemplos de autómato

$$(a+b)^+ cc\ (ab+c)^*$$



Exemplos de autómato

O Fantástico Prémio

Representação dos dados

- Problema : reconhecer uma linguagem (i.e. as palavras constituintes)
- Dados : uma palavra, colocada numa fita.
- um ciclo de execução: processamento duma letra da palavra por reconhecer.
- fim da execução: fim da palavra por analisar.
- Destaca-se do conjunto de estados os estados **iniciais** e **finais**

Definição Formal

Um autómato finito **não-determinista** (abreviatura **NDFA**) sobre um alfabeto A é definido por um 5-tuplo $M = \{Q, \Sigma, \delta, S, F\}$ onde:

- Q é o conjunto dos estados
- $\Sigma = A \cup \{\epsilon\}$ é o conjunto dos labels das transições
- $\delta \subseteq Q \times \Sigma \times Q$ é a relação de transição
- $S \subseteq Q$ é o conjunto dos estados iniciais
- $F \subseteq Q$ é o conjunto dos estados finais

Diz-se duma transição (q, ϵ, q') que é uma ϵ-transição.

Não determinismo e relação de transição

Um autómato não determinista é um autómato com eventualmente

- mais do que um estado inicial
- transições ϵ (ou seja, transições que não precisam de consumir input para serem exploradas)
- transições com a mesma letra partindo do mesmo estado

Isto implica que a sua execução possa ter de considerar simultaneamento vários estado simultaneamente. Por isso é necessário contemplar uma relação de transição e não somente uma função: para uma dada letra e para um dado estado de partida se relacionam com eventualemnte vários estados de chegadas.

Um Exemplo

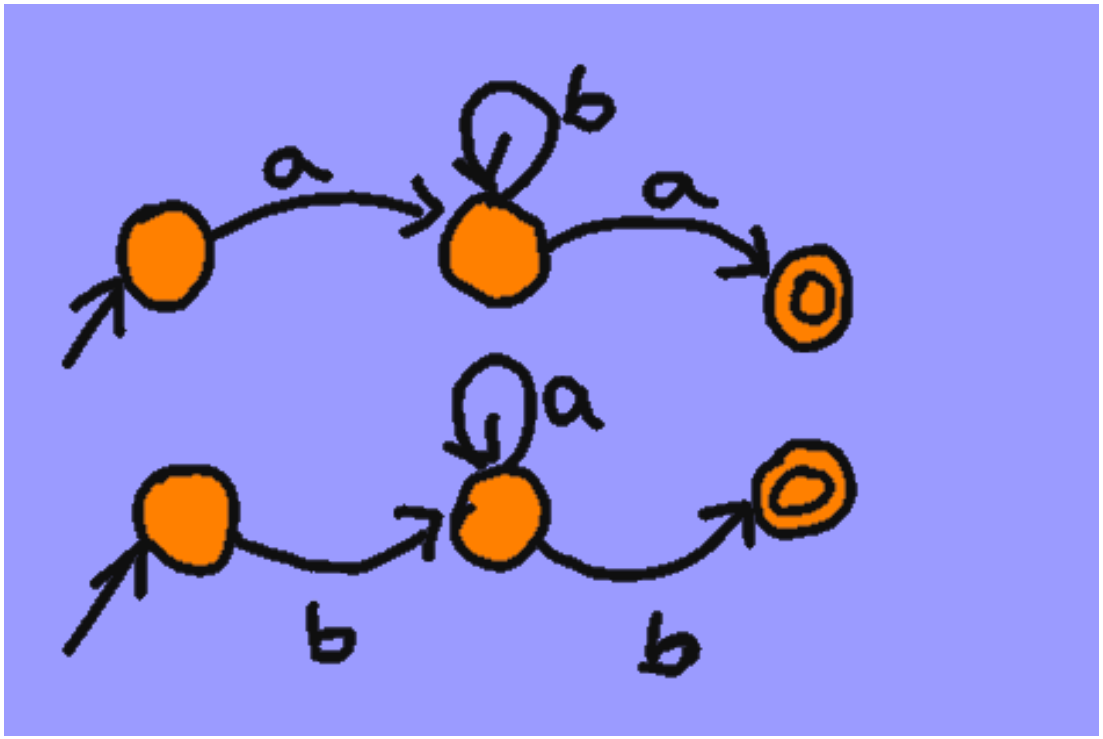


Figura : Autómato Não Determinista

Definição Formal

Um autómato finito **determinista** (abreviatura **DFA**) sobre um alfabeto A é definido por um 5-tuplo $M = \{Q, \Sigma, \delta, S, F\}$ onde:

- Q é o conjunto dos estados
- $\Sigma = A$ é o conjunto dos labels das transições (sem transições ϵ)
- $\delta : Q \times \Sigma \rightarrow Q$ é a função de transição
- $s \in Q$ é o estado inicial
- $F \subseteq Q$ é o conjunto dos estados finais

Um Exemplo

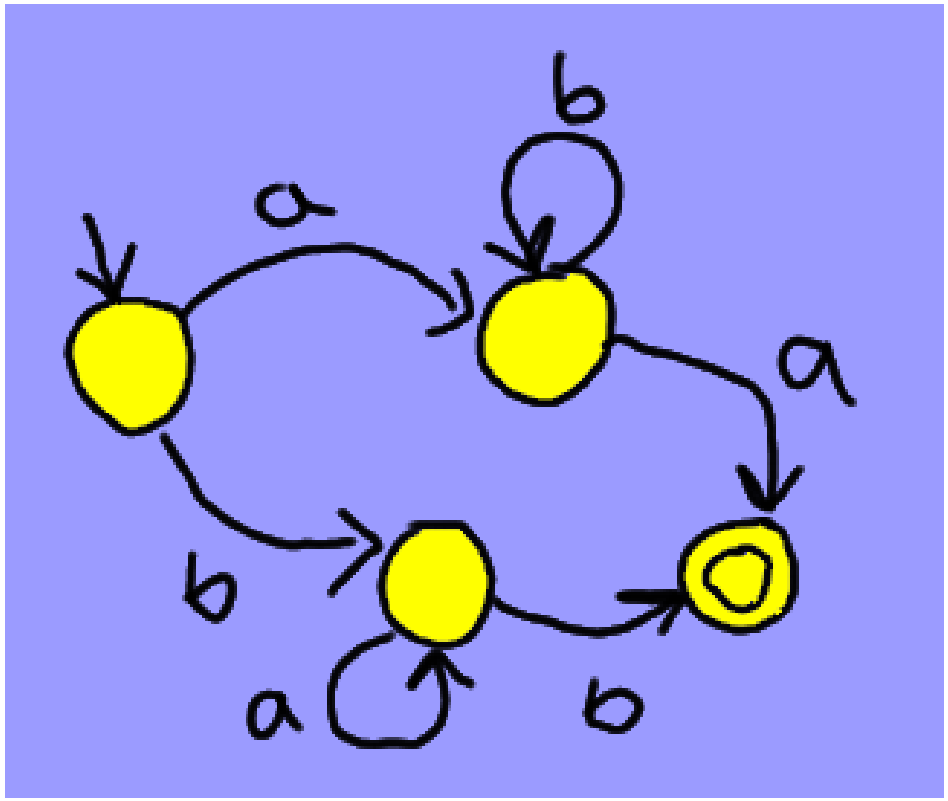


Figura : Autómato Determinista

Mais Um Exemplo

Especificação

- $Q = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- $s = q_0$
- $F = \{q_1\}$
- δ :

q	σ	$\delta(q, \sigma))$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_1

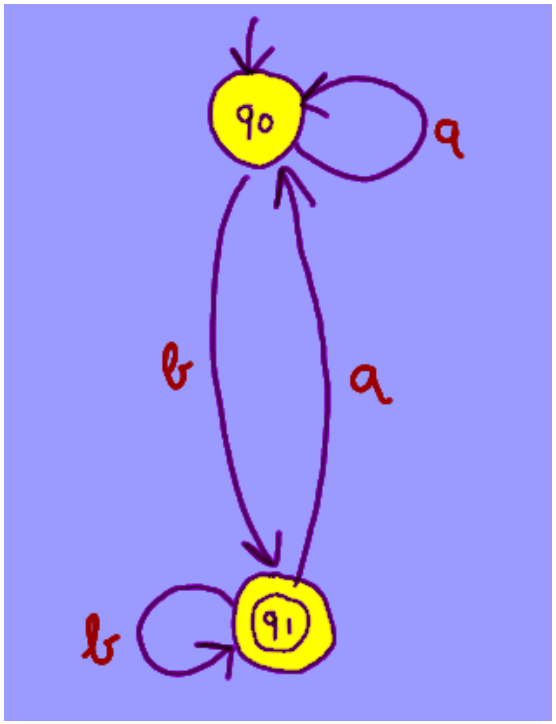
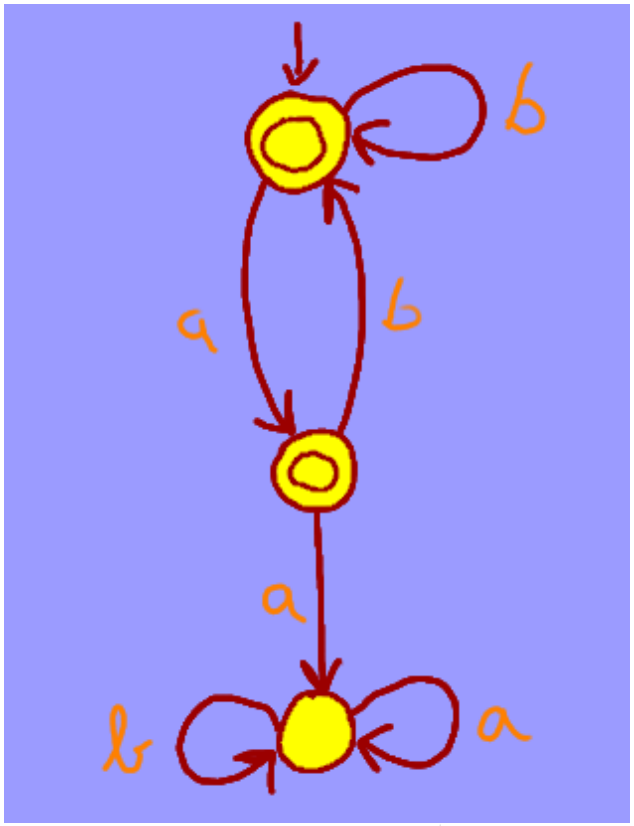


Figura : Autómato reconhecendo palavras de Σ^* que terminam por b

Outro Exemplo

Especificação

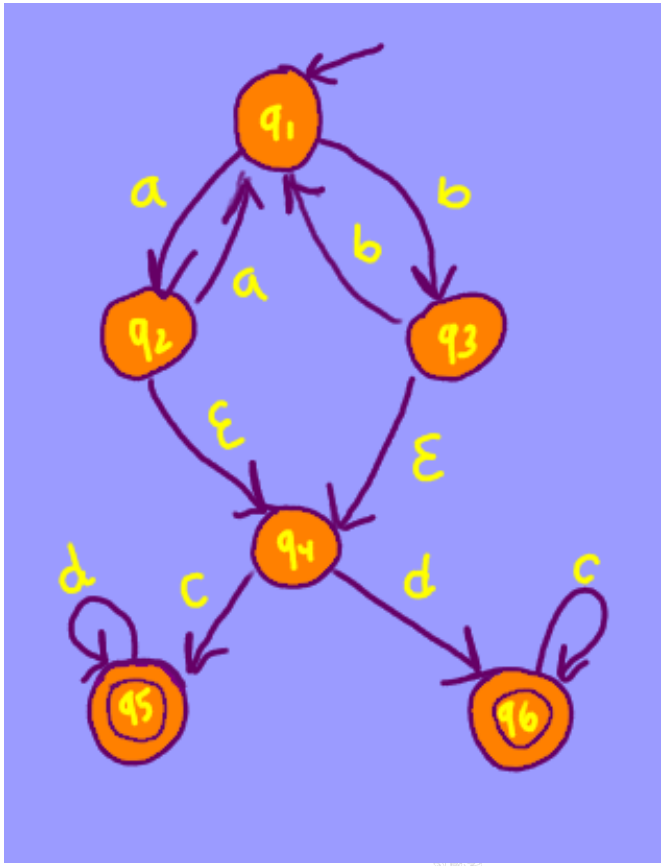
- De que tipo é este autómato?
- Qual é a definição formal deste autómato?
- Que tipo de palavras este autómato reconhece?



Mais Outro Exemplo

Especificação

- De que tipo é este autómato?
- Qual é a definição formal deste autómato?
- Que tipo de palavras este autómato reconhece?



Noção de execução de um autómato

- Consideremos um autómato $M = \{Q, \Sigma, \delta, s, F\}$
- Configuração: $(q, w) \in Q \times \Sigma^*$. Representa o estado completo do autómato (em que estado se encontra actualmente e o input que resta por analisar)
- Configuração derivável num passo: $(q, a.w) \vdash_M (q', w)$ se $(q, a, q') \in \delta$. Representa um passo de execução a partir de q com a entrada a. Esta entrada é de dois tipos: $a \in \Sigma$ ou $a = \epsilon$. No primeiro caso, diz-se que a derivação num passo consumiu um elemento da fita de entrada (a leitura do elemento actualmente activo na fita dos dados). No caso seguinte (transição (q, ϵ, q')), ϵ é *prefixo* de qualquer palavra ($\forall p \in \Sigma^*,\ p = \epsilon.p$), logo a configuração obtida não altera (não consome) a fita de entrada ($(q, \epsilon.w) \vdash_M (q', w)$).

Noção de execução de um autómato

- Configuração derivável num número qualquer de passo (eventualmente 0): $(q, w) \vdash_M^* (q', w')$. Fecho reflexivo-transitivo da relação $\vdash_M$, isto é, $(q, w) \vdash_M^* (q', w')$ se $(q, w) \vdash_M (q_1, w_1) \vdash_M (q_2, w_2) \vdash_M \cdots \vdash_M (q', w')$. Designaremos $\vdash_M^*$ por **derivação**.
- Execução dum autómato:$(s, w) \vdash_M (q_1, w_1) \vdash_M (q_2, w_3) \vdash_M \cdots \vdash_M (q_{n-1}, w_{n-1}) \vdash_M (q_n, \epsilon)$
- No caso dum autómato sem transições ϵ temos a simplificação seguinte (cada passo consome de facto um elemento da fita de entrada): $(s, a_1 a_2 a_3 \ldots a_n) \vdash_M (q_1, a_2 a_3 \ldots a_n) \vdash_M (q_2, a_3 \ldots a_n) \vdash_M \cdots \vdash_M (q_{n-1}, a_n) \vdash_M (q_n, \epsilon)$
- Neste contexto, diz-se que $a_1 a_2 a_3 \ldots a_n$ é aceite se q_n é um estado final ($q_n \in F$).

Linguagem aceite por um autómato

- O conjunto de todas as palavras aceites por um autómato M, designado de $L(M)$ é o conjunto de todas as palavras tais que:

$$\{w \in \Sigma^* \mid (s, w) \vdash_M^* (q, \epsilon) \ com \ q \in F\}$$

- Diz-se de $L(M)$ que é a linguagem **reconhecida** ou **aceite** ou ainda **gerada** por M.
- De notar que esta noção de derivação codifica a noção de caminho num grafo (ver mais além). Passar (derivar) duma configuração para outra = escolher uma transição (e não necessariamente consumir um elemento da fita de dados).

Linguagem aceite por um autómato

- No caso dos autómatos não determinístico é possível existirem várias derivações que permitam reconhecer uma determinada palavra p. Assim para determinar se uma palavra é aceite, basta que exista uma. Por isso é necessário explorar todas as possibilidades até encontrar uma tal derivação.
- No caso determinístico, se existir uma derivação que determine que uma palavra é aceite, então essa é única. Mais, derivação num passo = (consumo dum elemento na fita de entrada + escolha duma transição)

Noção alternativa de execução de um autómato

- Consideremos um autómato $M = \{Q, \Sigma, R_\delta, s, F\}$
- Configuração extendida: $(r, w) \in Q^* \times \Sigma^*$. Representa o estado completo do autómato (em que **conjunto de estados** este se encontra actualmente e o input que resta por analisar)
- Diz-se do estado q' que é **alcançável** por q se existir um caminho de q para q' só com transições ϵ
- Configuração extendida derivável num passo: $(r, a.w) \vdash'_M (r', w)$, ($r$ e r' conjunto de estados) r' é constituído de todos os estados q' que podem ser atingidos a partir dos estados q por exactamente uma transição a (com $a \neq \epsilon$) assim como de todos os estados alcançáveis a partir de q'.

Noção alternativa de execução de um autómato

- Configuração extendida derivável num número qualquer de passo (eventualmente 0): Fecho reflexivo-transitivo da relação $\vdash'_M$. Designaremos $\vdash'^*_M$ de **derivação extendida**.
- Execução dum autómato: $(S, a_1 a_2 a_3 \ldots a_n) \vdash'_M (Q_1, a_2 a_3 \ldots a_n) \vdash'_M (Q_2, a_3 \ldots a_n) \vdash'_M \cdots \vdash'_M (Q_{n-1}, a_n) \vdash'_M (Q_n, \epsilon)$

Linguagem aceite por um autómato (segunda versão)

- Extende-se a noção de palavra **reconhecida** ou **aceite** da seguinte forma: diz-se que $a_1 a_2 a_3 \ldots a_n$ é aceite se o conjunto Q_n contém um estado final $\exists q \in Q_n.\ q \in F$.
- O conjunto de todas as palavras aceites por um autómato M, forma a linguagem reconhecida $L(M)$.
- De forma resumida, esta noção de derivação captura o facto que um passo de execução consome **exactamente** um caracter na fita de entrada. Assim neste contexto uma derivação que comprove que uma determinada palavra p é aceite tem por comprimento o número de caracteres de p (ou seja o comprimento de p).
- Outro aspecto importante desta noção de derivação: não há diferenças entre derivação extendida num autómato determinístico e autómato não determinístico. Uma palavra é aceite via uma derivação extendida única.

NDFA, DFA e execução - Em resumo...

- No caso dos autómatos deterministas, existe no máximo uma execução para uma palavra em entrada
- No caso dos autómatos não determinísticos, é possível que existam execuções possíveis para a mesma palavra em entrada, mesmo na ausência de transições ϵ. É o que tenta capturar a noção de configuração extendida. Neste caso a execução é única.

Exemplo

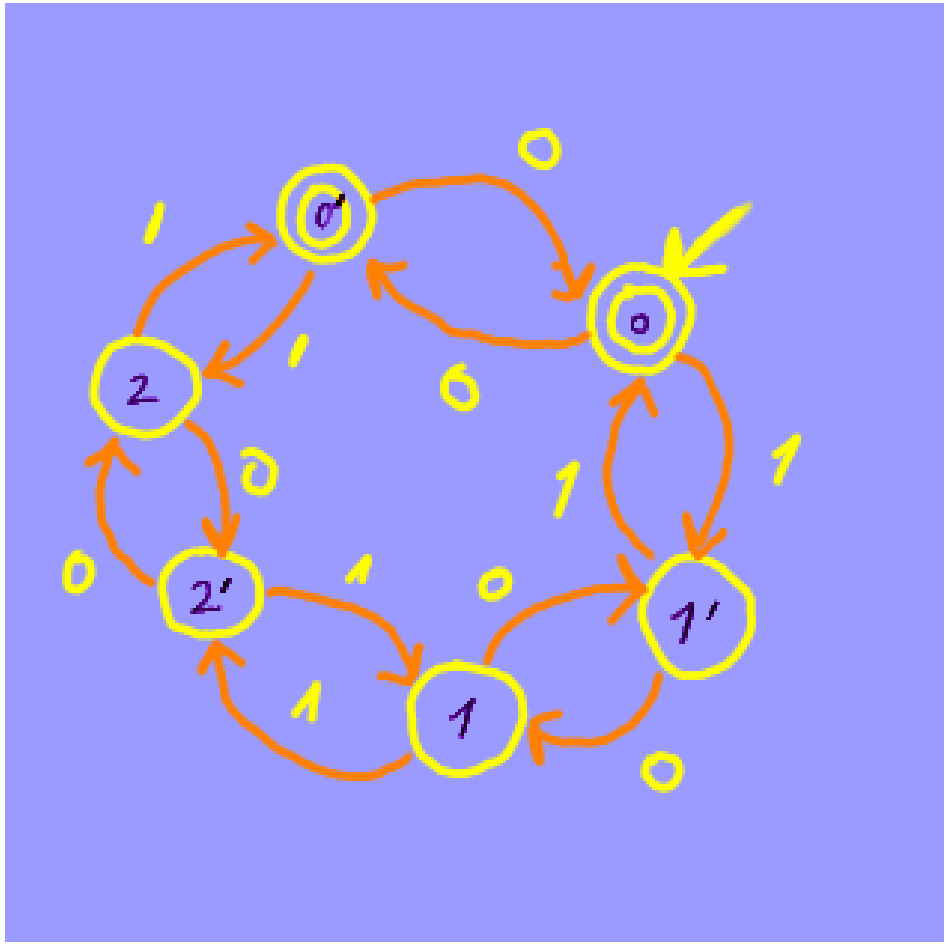


Figura : Qual é a linguagem reconhecida por este autómato?

Exemplo

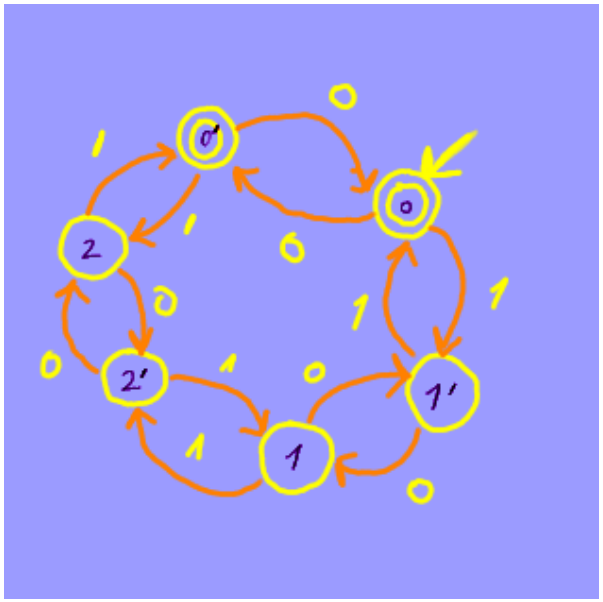


Figura : Qual é a linguagem reconhecida por este Autómato?

Autómato que lê um número binário (fornecido da direita para a esquerda) e que aceita o número se este for divisível por 3.
$19 \rightarrow 10011 \rightarrow$ *acaba em* $1'$, e $87 \rightarrow 1010111 \rightarrow$ *acaba em* $0'$

Caminho

Revisitemos a noção de configuração e de linguagem gerada e ver a execução como um percurso de grafo.
Seja $M = \{Q, \Sigma, R_\delta, S, F\}$ um NDFA (ou de forma indiferente, um DFA). Um caminho em M é uma sequência c de arestas consecutivas (no grafo subjacente ao autómato) de forma seguinte:

$$c = (q_0, a_1, q_1)(q_1, a_2, q_2)(q_2, a_3, q_3)(q_3, a_4, q_4) \cdots (q_{n-1}, a_n, q_n)$$

ou

$$c = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots \xrightarrow{a_n} q_n$$

Diz-se que c tem como **comprimento** n, que $u = a_1 a_2 \ldots a_n$ é a **etiqueta** do caminho c, que q_0 é o **estado inicial** de c e q_n o seu **estado final**.
Assim sendo diz-se que c é um caminho de q_0 para q_n (notação $q_0 \xrightarrow{u} q_n$)
Por convenção, existe sempre um caminho vazio de q para q com etiqueta ϵ

Caminho bem sucedido

- Seja $M = \{Q, \Sigma, R_\delta, S, F\}$. Um caminho $c = q_0 \ldots q_n$ é designado de **bem sucedido** se o seu estado inicial q_0 pertence a S e o seu estado final q_n a F.
- Assim sendo, uma palavra p é reconhecida se é etiqueta dum caminha bem sucedido. A linguagem reconhecida $L(M)$ é o conjunto das etiquetas de todos os caminhos bem sucedidos.
- Diz-se duma linguagem L definida sobre um alfabeto A que é **aceitável** ou **reconhecida** se existir um autómato M sobre A que a reconhece, i.e. $L(M) = L$.
- O conjunto de todas as linguagem reconhecidas por um autómato é designado de $Rec(A^*)$

Exemplo

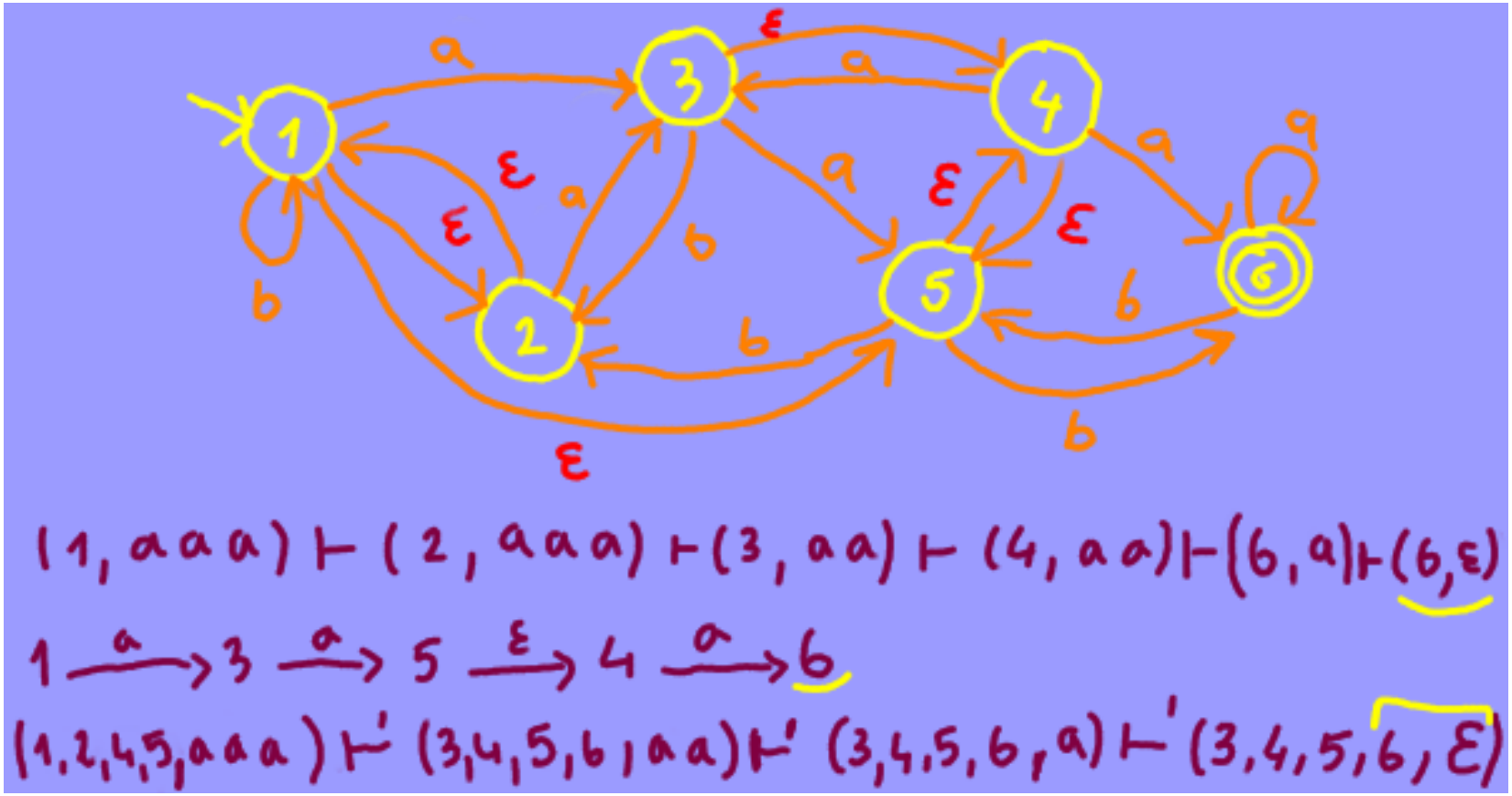


Figura : Um exemplo ilustrativo

Plano

- estado poço
- remoção das transições ϵ ($NDFA_\epsilon \rightarrow NDFA$)
- remoção dos estados não produtivos e dos estados inacessíveis
- completação

Ver acetatos manuscritos distribuídos nas aulas

- noção de autómatos equivalentes
- Determinisação ($NDFA \rightarrow DFA$)
- Minimisação ($DFA \rightarrow DFA_{min}$)
- Unicidade do autómato minimal
- Os autómatos não determínisticos tem o mesmo poder expressivo que os autómatos determinísticos ($L(NDFA) = L(DFA)$)

Ver acetatos manuscritos distribuídos nas aulas

Plano

- Propriedades de fecho das linguagens reconhecidas por autómatos finitos

Ver acetatos manuscritos distribuídos nas aulas

Teorema de Kleene

- Linguagens Regulares = Linguagens reconhecidas por autómatos

Ver acetatos manuscritos distribuídos nas aulas

Objectivos

O teorema de Kleene afirma que o conjunto das linguagens reconhecidas pelos autómatos finitos coincide com o conjunto das linguagens regulares. Sabemos que todas as linguagens podem ser representadas por expressões regulares. Assim o teorema de Kleene pode ser demonstrado:

1. mostrando que qualquer linguagem regular pode ser reconhecida por um autómato. Isto passa pela exibição dum algoritmo que permite transformar uma expressão regular num autómato. A correcção do algoritmo passa pela demonstração de que o domínio do algoritmo é o conjunto das expressões regulares (todas as expressões regulares podem ser alvo do algoritmo) e que a linguagem reconhecida pelo autómato resultante é exactamente a da expressão regular em entrada.
2. mostrando que qualquer autómato pode ser transformado em qualquer expressão regular. O algoritmo resultante deverá ser comprovado correcto no sentido de que deve poder transformar qualquer autómato e que a linguagem regular resultante é igual a linguagem reconhecida pelo autómato original.

Das linguagens regulares aos autómatos

De facto podemos reforçar a implicação (das linguagens regulares aos autómatos) com a seguinte propriedade (cuja demonstração é feita com base no algoritmo (e na sua correcção) que permite transformar uma expressão regular num autómato): A classe das linguagens aceites por autómatos finitos é fechada por

- união
- concatenação
- fecho de Kleene (estrela)
- complementação
- intersecção

Ou seja (para o primeiro caso): Sejam M_1 e M_2 dois autómatos então existe um autómato que reconhece a linguagem $L(M_1) \cup L(M_2)$

Objectivos

Retomemos a nossa digressão sobre o teorema de Kleene.

- Designemos de *aut2regexp* o primeiro algoritmo, e de *regexp2aut* o segundo algoritmo.
- A demonstração relacionada com *aut2regexp* já foi abordada nas aulas anteriores. Vamos a seguir exibir a segunda parte da demonstração.
- mas antes: uma forma simples de comprovar que os dois algoritmos são correctos é de considerar um autómato M e uma expressão regular e. Os dois algoritmos são correctos se $L(e) = L(aut2regexp(regexp2aut(e)))$ e $L(M) = L(regexp2aut(aut2regexp(M)))$ qualquer que sejam M e e.

Algoritmo de Mac Naughton-Yamada

Seja M um autómato, $Q = \{q_1, q_2, q_3, \ldots, q_n\}$ o conjunto dos seus estados e Δ a sua relação de transição. Designamos por $R(i,j,k)$ o conjunto das palavras que permitam passar do estado q_i ao estado q_j passando exclusivamente pelos estados $\{q_1, \ldots, q_{k-1}\}$

$$R(i,j,1) = \begin{cases} \{w|(q_i, w, q_j) \in \Delta\} & se\ i \neq j \\ \{\epsilon\} \cup \{w|(q_i, w, q_j) \in \Delta\} & se\ i = j \end{cases}$$

$$R(i,j,k+1) = R(i,j,k) \cup R(i,k,k)R(k,k,k)^* R(k,j,k)$$

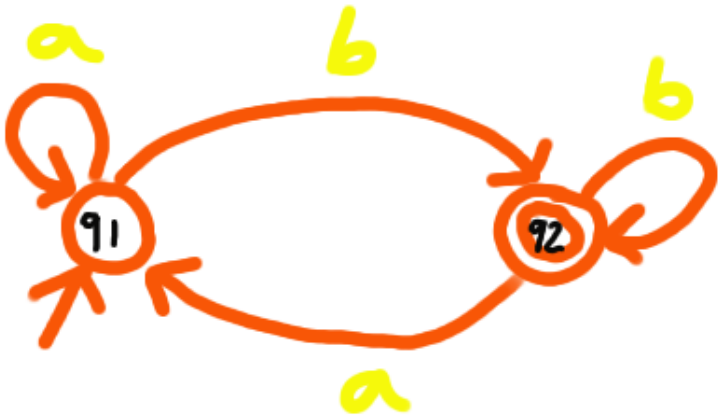
Temos então $L(M) = \bigcup_{q_j \in F} R(1,j,n+1)$ (para ter a expressão regular correspondente basta subsituir $\cup$ nas definições anteriores por $+$)

Dos autómatos às expressões regulares

- Para formular as linguagens $R(i, j, k)$ só foram utilizados conjuntos finitos, união, concatenação e o fecho de kleene (o fecho reflexivo, transitivo). A linguagem resultante é assim por definição uma linguagem regular.
- Este algoritmo pertence à família dos algoritmos de *programação dinâmica*.

Exemplo

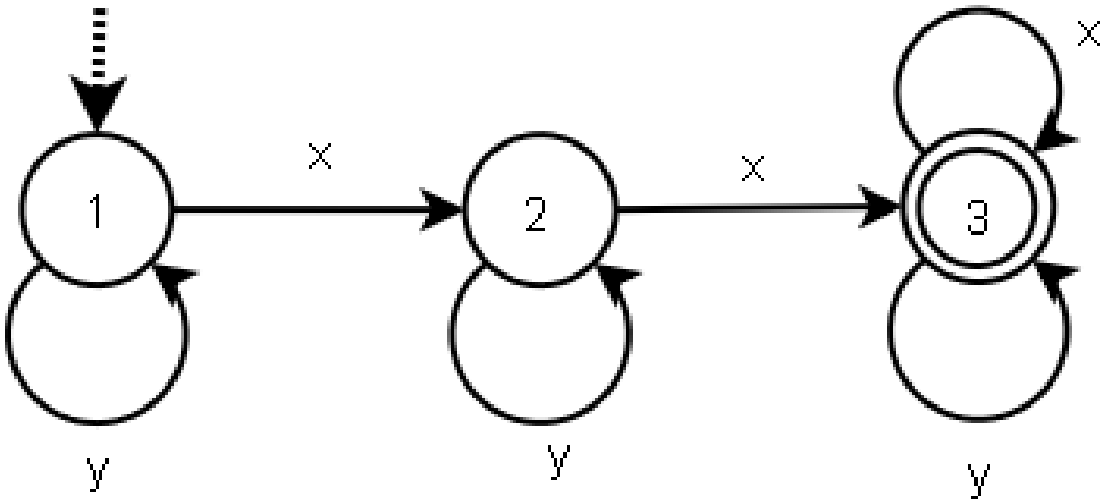
.



	$k=1$	$k=2$
$R(1,1,k)$	$\epsilon + a$	$(\epsilon + a) + (\epsilon + a)(\epsilon + a)^*(\epsilon + a)$
$R(1,2,k)$	b	$b + (\epsilon + a)(\epsilon + a)^* b$
$R(2,1,k)$	a	$a + a(\epsilon + a)^*(\epsilon + a)$
$R(2,2,k)$	$\epsilon + b$	$(\epsilon + b) + a(\epsilon + a)^* b$

Assim $L(M) = R(1,2,3) = (b + (\epsilon + a)(\epsilon + a)^* b) + (b + (\epsilon + a)(\epsilon + a)^* b)((\epsilon + b) + a(\epsilon + a)^* b)^*((\epsilon + b) + a(\epsilon + a)^* b)$

Outro Exemplo



Temos $L(M) = R(1, \{1, 2, 3\}, 3)$. Calculemos estes valores por partes.

$R(1, \emptyset, 1) = \{y, \epsilon\}$ $R(2, \emptyset, 1) = \epsilon$ $R(3, \emptyset, 1) = \epsilon$

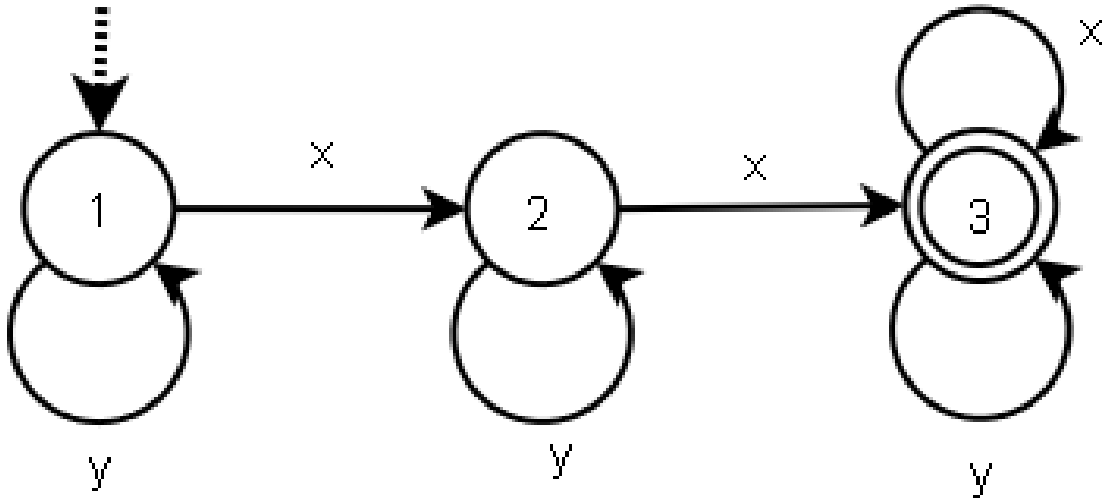
$R(1, \emptyset, 2) = \{x\}$ $R(2, \emptyset, 2) = \{\epsilon, y\}$ $R(3, \emptyset, 2) = \epsilon$

$R(1, \emptyset, 3) = \emptyset$ $R(2, \emptyset, 3) = \{x\}$ $R(3, \emptyset, 3) = \{\epsilon, x, y\}$

$R(1, \{1, 2, 3\}, 3) = R(1, \{2, 3\}, 3) \cup$

$R(1, \{2, 3\}, 1).R(1, \{2, 3\}, 1)^*.R(2, \{3\}, 1) = R(1, \{2, 3\}, 1)^*.R(2, \{3\}, 1)$

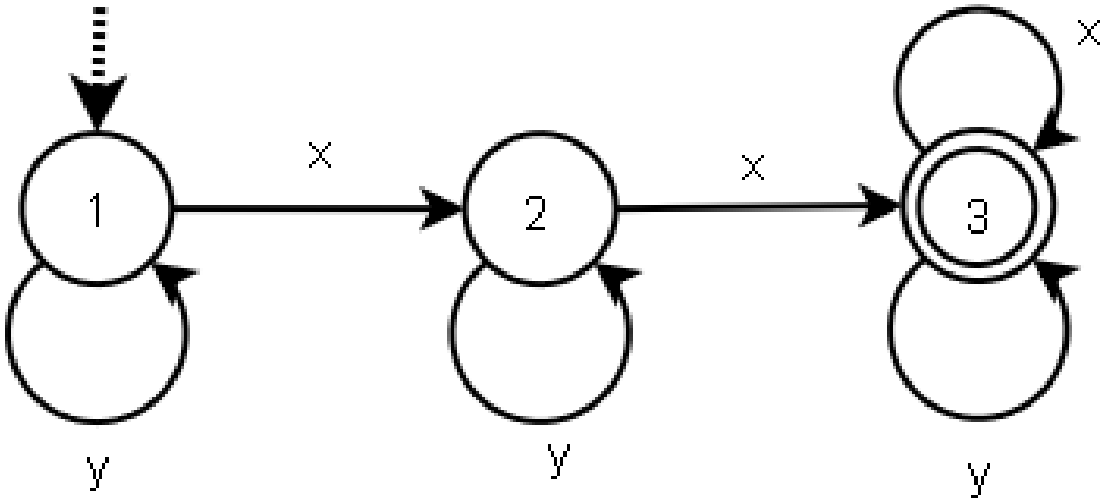
Outro Exemplo



$$
\begin{array}{lll}
R(1,\{2,3\},1) & = R(1,\{3\},1) \cup R(1,\{3\},2).R(2,\{3\},2)^*.R(2,\{3\},1) & \\
R(1,\{3\},1) & = R(1,\emptyset,1) \cup R(1,\emptyset,3).R(3,\emptyset,3)^*.R(3,\emptyset,1) & = \{\epsilon, y\} \\
R(1,\{3\},2) & = R(1,\emptyset,2) \cup R(1,\emptyset,3).R(3,\emptyset,3)^*.R(3,\emptyset,2) & = \{x\} \\
R(2,\{3\},2) & = R(2,\emptyset,2) \cup R(2,\emptyset,3).R(3,\emptyset,3)^*.R(3,\emptyset,2) & = \{\epsilon, y\} \\
R(2,\{3\},1) & = R(2,\emptyset,1) \cup R(2,\emptyset,3).R(3,\emptyset,3)^*.R(3,\emptyset,1) & = \emptyset
\end{array}
$$

Assim $R(1,\{2,3\},1) = \{\epsilon, y\}$

Outro Exemplo



$$
\begin{array}{lll}
R(1,\{2,3\},3) & = R(1,\{3\},3) \cup R(1,\{3\},2).R(2,\{3\},2)^*.R(2,\{3\},3) & \\
R(1,\{3\},3) & = R(1,\emptyset,3) \cup R(1,\emptyset,3).R(3,\emptyset,3)^*.R(3,\emptyset,3) & = \emptyset \\
R(2,\{3\},3) & = R(2,\emptyset,2) \cup R(2,\emptyset,3).R(3,\emptyset,3)^*.R(3,\emptyset,3) & = x(x+y)^*
\end{array}
$$

Assim $R(1,\{2,3\},3) = xy^*x(x+y)^*$.

Por consequente, $L(M) = y^*xy^*x(x+y)^*$

Variante do algoritmo: Técnica da eliminação de estados

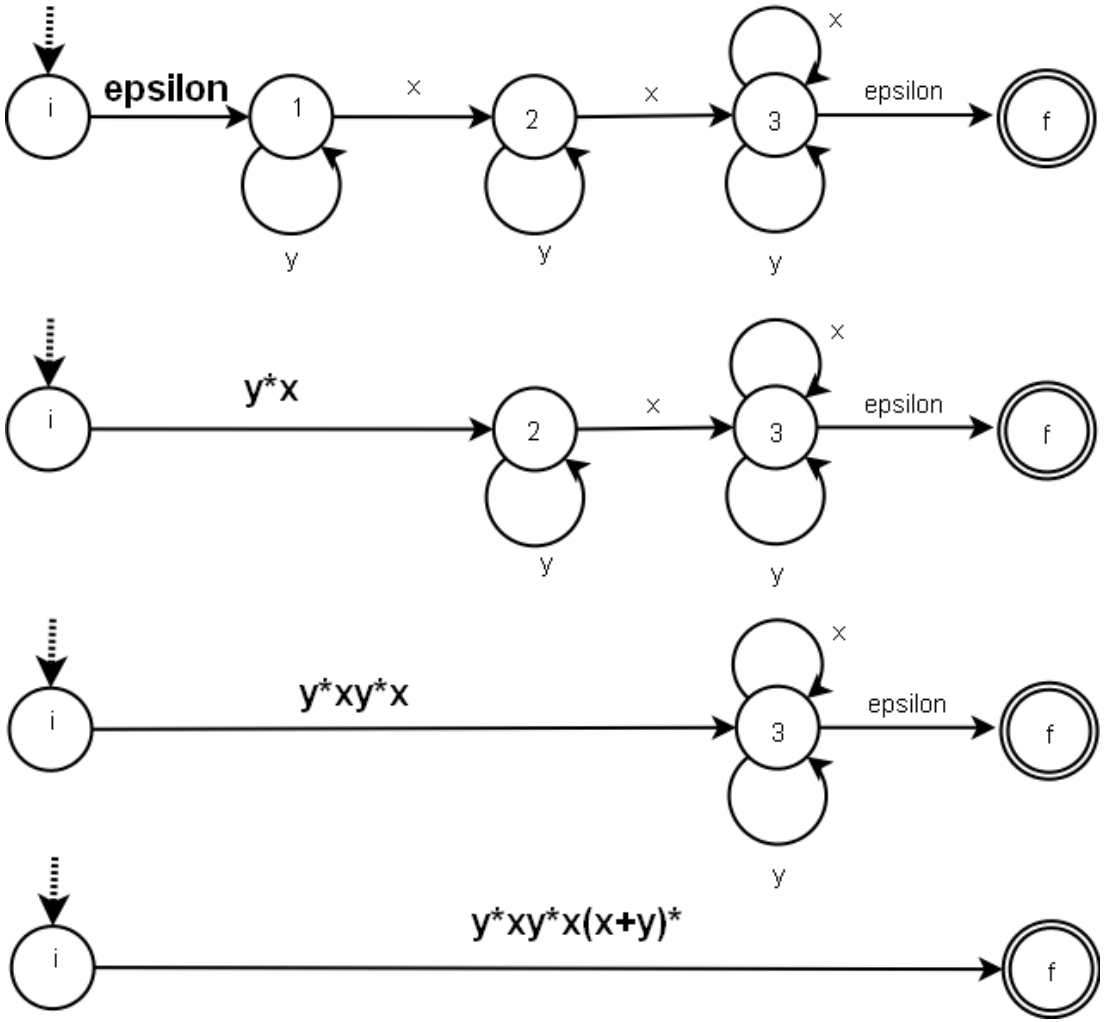
Princípio de base: permitir que as transições tenham por etiquetas expressões regulares (e não só letras).
Partindo deste pressuposto alterar convenientemente o autómato original de tal forma que sobre só dois estados e uma transição. Esta transição contém a expressão regular procurada.

Técnica da eliminação de estados

Seja $M = \{Q, \Sigma, S, F, R_\delta\}$ um autómato finito.

1. Alterar o autómato M em M' da seguinte forma: juntar dois novos estados i e f que serão respectivamente o estado inicial único (i) e o estado final único (f) de M'. Do estado i partem ϵ-transições para todos os estados iniciais de M. De todos os estados finais de M saem ϵ-transições para f.
2. Enquanto o conjunto Q de M não for considerado no seu completo (i.e. vazio) fazer o seguinte: considerar um estado q de Q. Considerar todos os estados p e r de $Q - \{q\}$ tais que existe transições directas entre p e q e entre q e r. Sejam (p, l_{pq}, q) e (q, l_{qr}, r) essas transições. Então juntar a transição (p, l_{pqr}, r) onde $l_{pqr} = l_{pq} \cup l_{pq}.l_{qq}^*.l_{qr}$. Esta fase concluída, o estado q pode ser removido de Q assim como todas as transições que partem ou chegam a q.
3. Quando a operação anterior termina só restam os dois estados i e f e uma transição entre estes dois estados. A etiqueta desta transição é a linguagem reconhecida pelo autómato M.

Outro Exemplo



Linguagens regulares

- Conjuntos das palavras binárias divisíveis por 2 (mas também por 3 ou por k, $k \geq 4$)
- Seja $\Sigma = \{0, 1, 2, \cdots, 9\}$ e seja $L \subseteq \Sigma^*$ o conjunto dos inteiros naturais divisíveis por 2 e por 3. L é regular.

Desafio: demonstrar que são regulares. Como? exibindo que são linguagens reconhecidas por um autómato finito ou que são conjuntos de palavras construídos com base em linguagens regulares conhecidas e operadores regulares (união, concatenação e fecho reflexivo-transitivo)

Plano

Autómatos vistos como algoritmos

- Já referimos que podemos ver os processos de cálculos em dispositivos com memória (assim como registos etc...) como modificadores de estado em que os estados representam configurações da memória.
- Vamos aqui reforçar esta ideia mostrando como podemos representar os autómatos nestes moldes.

Autómatos vistos como algoritmos

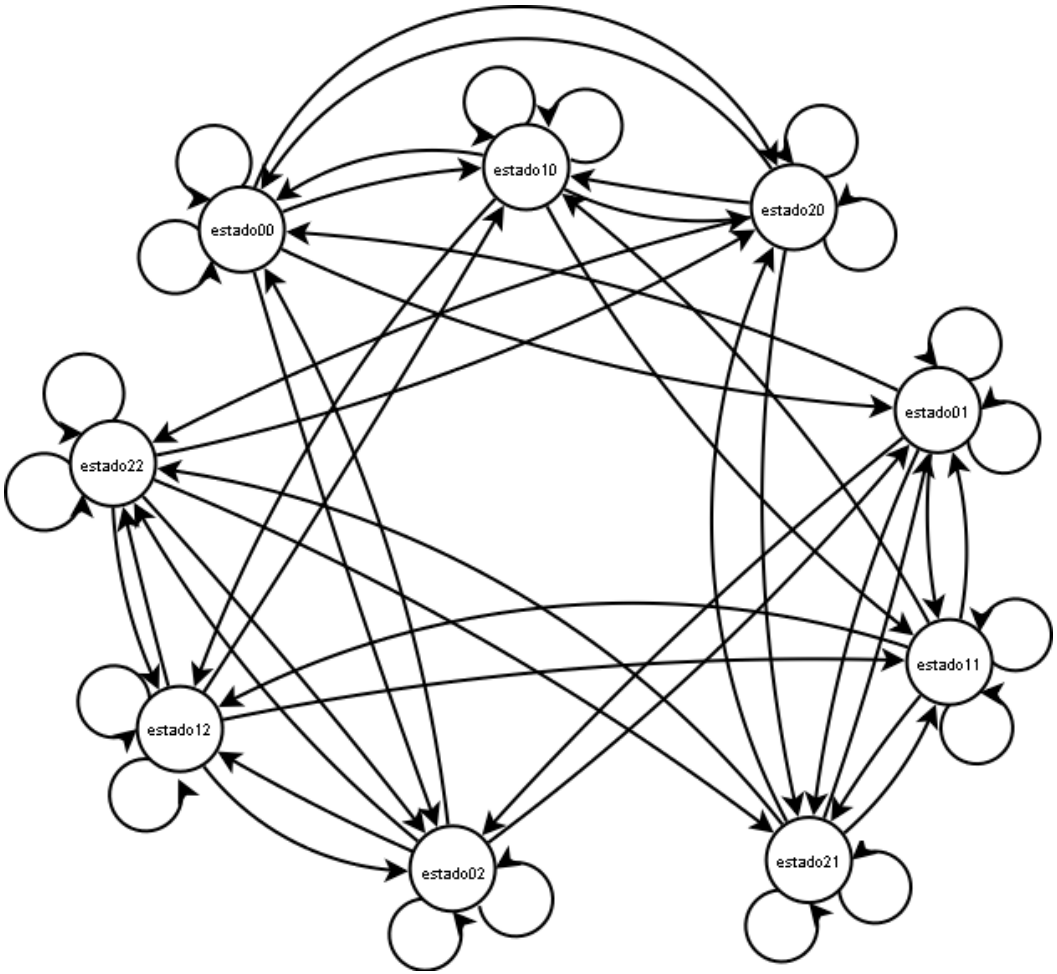
- Imaginemos que um prédio de 2 andares tenha dois elevadores e que pretendemos modelar (de forma simplista) a utilização dos dois elevadores.
- A posição de cada elevador pode ser representada por uma variável tomando valores em $\{0 \ldots 2\}$.
- Temos então $3^2 = 9$ configurações possíveis para a conjunção das duas variáveis.
- Um exemplo de configuração pode ser $\{0, 1\}$, ou seja, a primeira variável contém o valor 0 (o primeiro elevador encontra-se no rés-do-chão) e a segunda o valor 1 (o segundo elevador encontra-se no primeiro andar).
- Estas configurações possíveis formam os estados do autómato. As transições representam os pedidos dos utilizadores (por exemplo $\{a_0, a_1, a_2\}$ para o primeiro, $\{b_0, b_1, b_2\}$ para o segundo). A cada instante um só pedido é concretizado.

Autómatos vistos como algoritmos

Em termos de programa, este autómato traduz-se em: (assuma que os pedidos estão organizados numa sequência para a qual existam as funções *vazio*, *primeiro* e *resto*)

```
let config = ref (0,0) (* configuração inicial *)
let rec estado00 pedidos =
if (vazio pedidos) then ()
else
  match (primeiro pedidos) with
    a0 → config:=(0,0); estado00 (resto pedidos)
  | a1 → config:=(1,0); estado10 (resto pedidos)
  | a2 → config:=(2,0); estado20 (resto pedidos)
  | b0 → config:=(0,0); estado00 (resto pedidos)
  | b1 → config:=(0,1); estado01 (resto pedidos)
  | b2 → config:=(0,2); estado02 (resto pedidos)
and estado01 pedidos = ...
```

Numa imagem resumida



Autómatos vistos como algoritmos

- De forma genérica, os autómatos são formalismos adaptados para representar processos algorítmicos que manipulam um conjunto finito de variáveis. No exemplo do elevador, são duas variáveis.
- Esta caracterização (embora informal) é essencial: O número/tamanho dos dados manipulados deve ser perfeitamente conhecido (pelo menos o seu limite superior) para poder ser alvo duma modelização algorítmica por autómato finito. Os autómatos finitos não conseguem representar processos algorítmicos cuja necessidade em memória não tem limite conhecido.
- É essa restrição que vamos explorar e formalizar a seguir para estabelecer que o formalismo dos autómatos não é o formalismo que procuramos para descrever **todos** os algoritmos possíveis.

Algumas considerações prévias

1. Todas as linguagens finitas são regulares.
2. Uma linguagem não regular é assim necessariamente infinito (o inverso é falso).
3. Se uma linguagem é infinita, então contém palavras de tamanho infinito.
4. Qualquer linguagem regular é aceite por um autómato finito, este contém um número finito de estados.
5. Consideremos uma linguagem regular infinita L e um autómato finito M constituído de m estados. Para toda a palavra p de comprimento superior a m em entrada, existe um estado q sobre o qual a execução do autómato M passará pelo menos duas vezes.
6. Ou seja, podemos descompor a palavra p em três palavras x, u e y tais que $p = x.u.y$ e a execução passa por q ao entrar na análise de u e na análise de y. Então M aceita as palavras geradas pela expressão regular $x.u^*.y$.
7. Assim espera-se que as palavras infinitas duma linguagem regular que essas apresentam umtal padrão.

Teorema do bombeamento

- Seja L uma linguagem regular, existe um inteiro $n \geq 1$ tal que qualquer palavra w de L com $|w| \geq n$ pode ser reescrita como $w = xuy$ onde $u \neq \epsilon$, $|xu| \leq n$ e $xu^*y \in L$.
- (corolário) Seja L uma linguagem regular infinita, então $\exists x, y \in \Sigma^* \ \exists u \in \Sigma^+, \forall n \in \mathbb{N}$ tais que $x.u^n.y \in L$

Teorema do bombeamento

Demonstração. (essencialmente baseada nas observações feitas previamente) L é regular, logo existe um autómato finito determinista (minimal) M que o reconhece. Suponhamos que M tenha n estados. Seja w uma palavra de comprimento maior ou igual a n. Consideremos os n passos de execução seguintes:

$$(q_0, w_1 w_2 \dots w_n) \vdash_M (q_1, w_2 \dots w_n) \vdash_M \cdots \vdash_M (q_n, \epsilon)$$

onde $w_1, w_2, \dots, w_n$ são os primeiros caracteres de w e q_0 o estado inicial de M. Temos assim $n+1$ configurações implicadas aqui num autómato com n estados. Pelo princípio da gaiola de pombos existem i e j tais que $(0 \leq i < j \leq n)$ e $q_i = q_j$. Ou seja a palavra $u = w_i w_{i+1} \dots w_j$, não vazia (já que $i < j$), leva a execução do estado q_i ao estado q_i. Então esta palavra pode ser removida de w ou até mesmo repetida sequencialmente sem alterar a sua aceitação por M. Ou seja M aceita $xu^* y$ onde $x = w_1 w_2 \dots w_i$, $u = w_{i+1} \dots w_j$, $y = w_j \dots$. Finalmente temos $|xu| \models j \leq n$.

Aplicações do teorema do bombeamento

Dado um p inteiro, a linguagem $L_1 = \{a^n b^n \mid 0 \leq n \leq p\}$ é regular mas não a linguagem $L_2 = \{a^n b^n \mid 0 \leq n\}$.

- L_1 é regular. $L_1 = \epsilon \cup \{ab\} \cup \{aabb\} \ldots \cup \{\}$ ou seja a união finita de linguagens finitas. QED.
- Para L_2 vamos proceder via uma demonstração por absurdo. Vamos supor que L_2 é regular. Logo existe um autómato finito determinístico M com, digamos, m estados. Visto que L_2 é regular então existe uma descomposição de $a^n b^n$ em xuy tal que xu^*y também pertença a L_2. Averiguemos (basta que existe um "corte"xuy possível).
  - $u \in a^*$. Impossível porque se repetimos u mais do que uma vez temos mais a do que b.
  - $u \in b^*$. Impossível por razões semelhantes.
  - $u \in (a \cup b)^* - (a^* \cup b^*)$. Impossível visto que qualquer repetição deste padrão irá misturar a e b e quebrar o padrão "todos os a antes dos b".

  Contradição. Logo L_2 não é regular.

Aplicações do Teorema do bombeamento

$L = \{a^n \mid n \text{ primo}\}$ não é regular. Procedemos mais uma vez por contradição. L é regular. Logo existe um autómato determinista minimal com "m estados que reconhece a linguagem L. Seja w uma palavra de L tal que $|w| > m$. Logo podemos arranjar uma descomposição xuy de w tal que xu^*y também pertença igualmente a L. Vejamos esta afirmação em detalhe: $x = a^p$, $u = a^q$ e $y = a^r$ onde $p, r \geq 0$, $q > 0$ e $r + q + r > m$. Então, em particular $\forall k \in \mathbb{N}, xu^k y \in L$ ou seja $p + kq + r$ primo. Vejamos agora se todos os k validam esta imposição (condição necessária). Ora para $k = p + 2q + r + 2$,
$p + kq + r = (q + 1)(p + 2q + r)$, logo não é primo. Contradição. L não é regular.

Outras linguagens não regulares

- $\{a^n b^n c^m \mid n, m \in \mathbb{N}\}$
- $\{w \mid w \in \{a, b\}^* \wedge$ há tantos a como b em $w\}$
- Seja L uma linguagem regulares. $\{ww^{-1} \mid w \in L\}$, onde $w^{-1} = w_n w_{n-1} \ldots w_2 w_1$ se $w = w_1 w_2 \ldots w_{n-1} w_n$.
- $\{a^{n^2} \mid n \in \mathbb{N}\}$

Outras linguagens não regulares

- Sejam $D = \{0, 1\}$ e $T = D \times D \times D$. Uma multiplicação correcta de dois números binários pode ser representada como uma palavra de T^*. Por exemplo a multiplicação

$$\begin{array}{cccc} & 0 & 1 & 0 & 1 \\ \times & 0 & 1 & 1 & 0 \\ \hline & 1 & 0 & 1 & 1 \end{array}$$

pode ser descrita pela palavra de 4 letras seguinte:

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

O conjunto de todas as palavras de T^* que representam multiplicações correctas não é regular

Desafio: demonstrar que essas linguagens não são regulares.

Problemas e algoritmos

- Dada uma linguagem regular L sobre um alfabeto Σ e uma palavra $w \in \Sigma^*$. O problema de determinar se $w \in L$ é decidível.
- O problema que consiste em determinar se uma linguagem regular L é vazio (ou seja determinar se $L = \emptyset$) é decidível.
- O problema de determinar se uma linguagem regular L é universal (ou seja determinar se $L = \Sigma^*$) é decidível.
- O problema que consiste em determinar se uma linguagem regular L_1 é contida numa linguagem regular L_2 ($L_1 \subseteq L_2$) é decidível.
- O problema que consiste em determinar se duas linguagens regulares são iguais é decidível.

Plano

Objectivos

Os autómatos que vimos até este momento são designados de aceitadores (*acceptors* em inglês). Isto é, as execuções permitam dar uma resposta binária (*sim*, *não*) ao input ao quais esses autómatos são submetidos. Existem formas de extender os autómatos de forma a que esses consigam fornecer respostas mais ricas: **os autómatos com saída** (*transducers* em inglês).

Princípios de Base

- Uma acção = produção de output. Para isso é preciso definir o alfabeto de saída, digamos Ω. Executar um autómato com output produzirá assim, além de consumir a palavra de entrada, uma palavra de Ω^*.
- Duas abordagens principais: **Autómatos de Moore** e **Autómato de Mealy**.
  1. Moore: as acções são despoletadas pelos estados por onde passa a execução. Função de output $f : Q \to \Omega \cup \{\epsilon\}$. Na passagem ao estado x, o resultado de $f(x)$ é colocado no buffer de saída.
  2. Mealy: as acções são despoletadas pelas transições por onde passa a execução. Função de output: $g : Q \times \Sigma \to \Omega \cup \{\epsilon\}$. Na escolha e passagem da transição (p, a, q) o resultado da função $g(p, a)$ é colocado no buffer de saída.
- Sob certas condições é possível transformar um autómato de Mealy num autómato de Moore equivalente.

Um exemplo simples

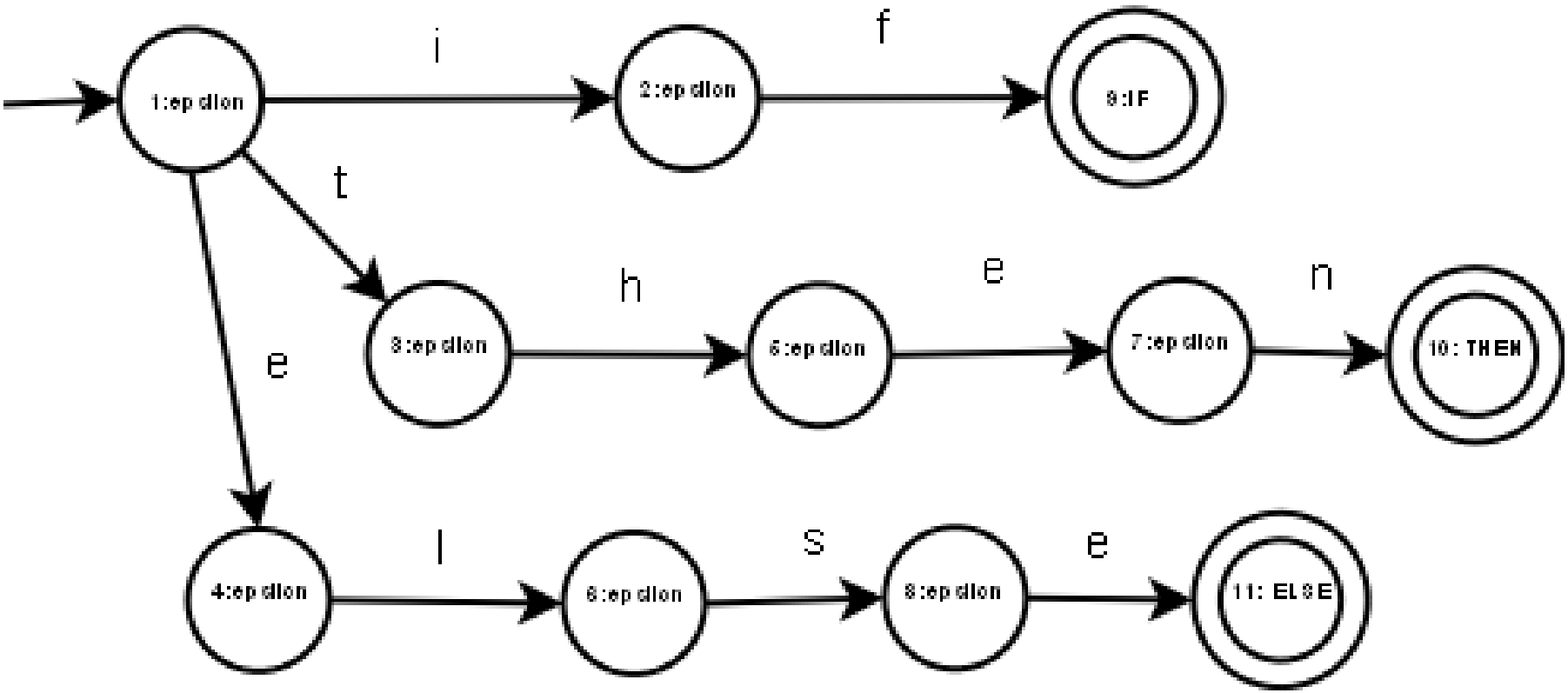


Figura : Um exemplo de autómato de Moore: um lexer simples