

Teoria da Computação

Aula - Autómatos Finitos

Simão Melo de Sousa

Introdução à noção de autómatos

Introdução à noção de autómatos

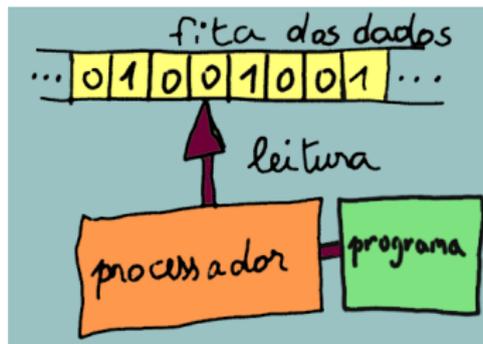
Contexto

- 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.

- 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})

- 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.

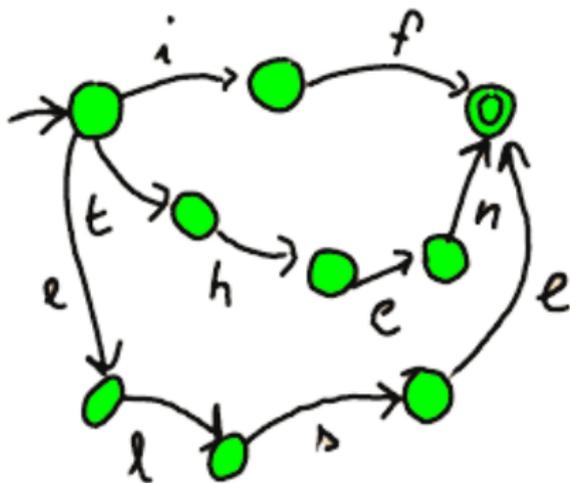
Primeira abordagem a noção de **Processo Efectivo**



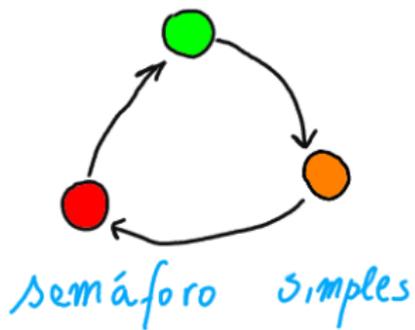
automáto (um grafo particular)	programa/algoritmo
estado/nodo do autómato	uma configuração da memória
conjunto dos vértices	relação de transição
percurso do autómato	execução
relação de transição	execuções possíveis
estado inicial	a configuração inicial da memória
fita	dados de entrada

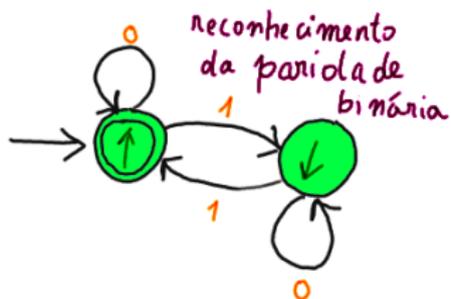
Introdução à noção de autómatos

Exemplos Introdutórios

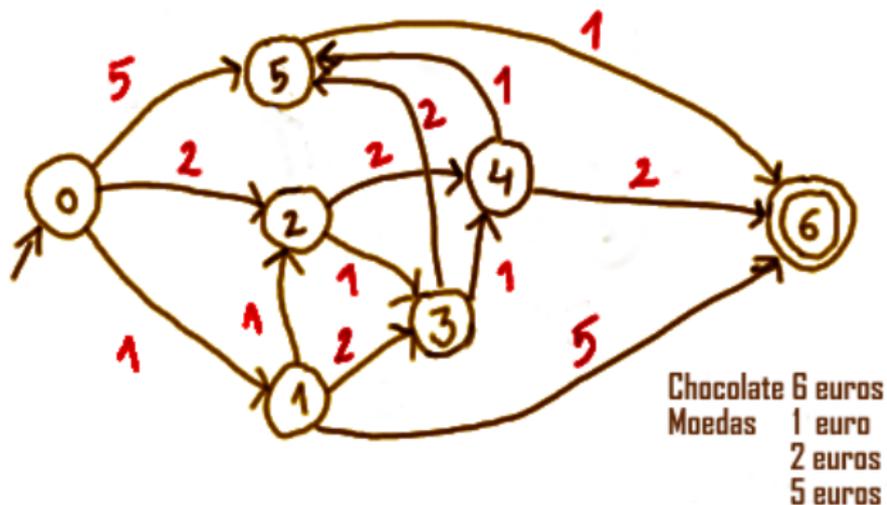


reconhecimento léxico
(if then else)

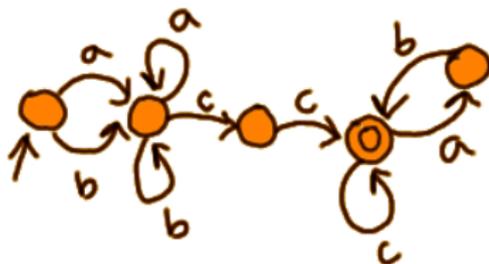




Maquina dos Chocolates em Ouro



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



O Fantástico Prémio

Introdução à noção de autómatos

Definição formal da noção de autómato

- 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**

Um autómato finito **não-determinista** (abreviatura **NDA**) 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.

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 simultaneamente 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.

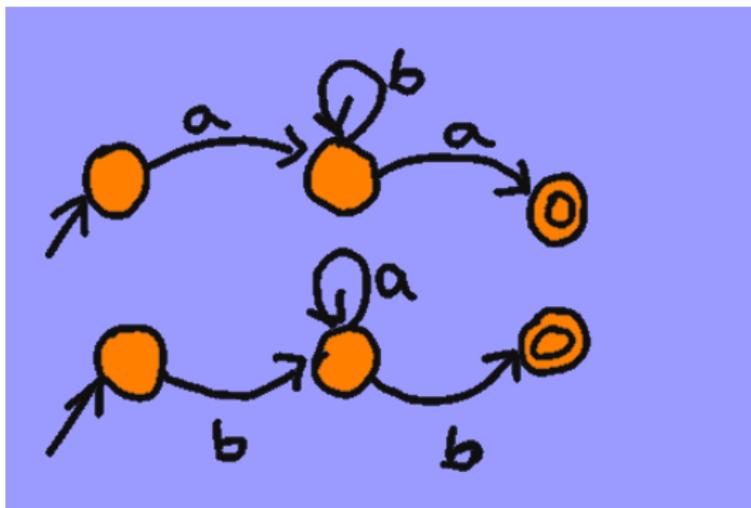


Figura: Autómato Não Determinista

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

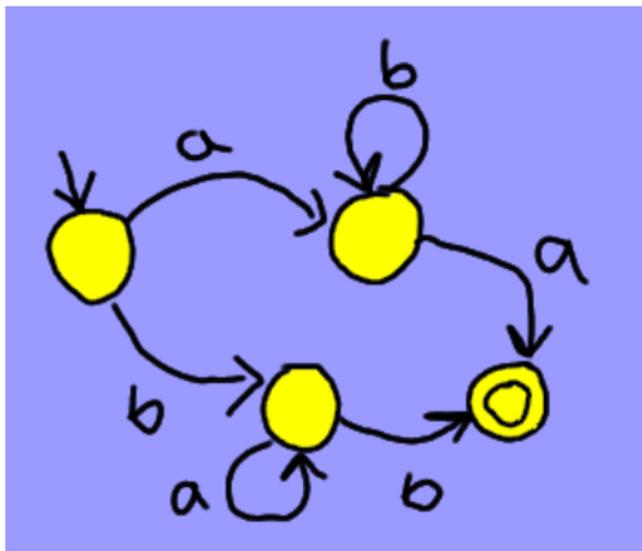


Figura: Autómato Determinista

Introdução à noção de autómatos

Exemplos ilustrativos

Especificação

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

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

• $\delta :$

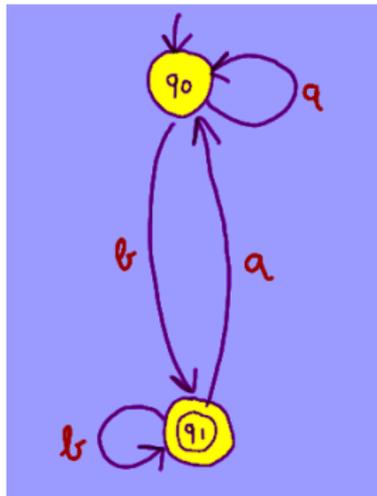
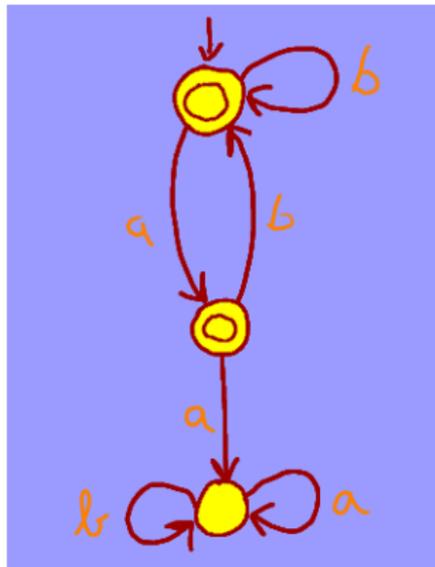


Figura: Autômato reconhecendo palavras de Σ^* que terminam por b

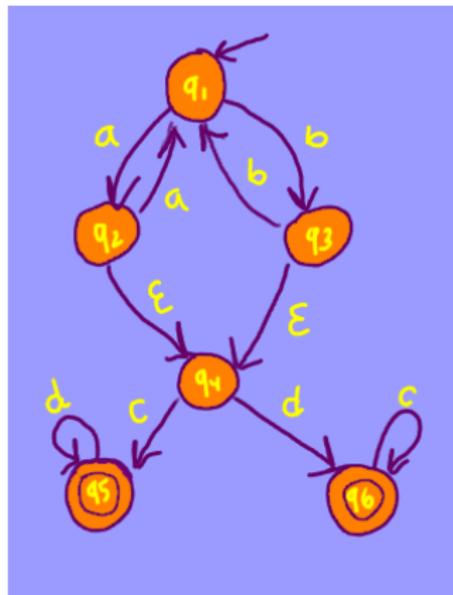
Especificação

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



Especificação

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



Introdução à noção de autómatos

execução e linguagem reconhecida por autómatos

- 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 ($(q, \epsilon.w) \vdash_M (q', w)$).

- 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): $(s, a_1 a_2 a_3 \dots a_n) \vdash_M (q_1, a_2 a_3 \dots a_n) \vdash_M (q_2, a_3 \dots a_n) \vdash_M \cdots \vdash_M (q_{n-1}, a_n) \vdash_M (q_n, \epsilon)$
- Neste contexto, diz-se que $a_1 a_2 a_3 \dots a_n$ é aceite se q_n é um estado final ($q_n \in F$).

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

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

- 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).

- No caso dos autómotos 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 autômato

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

Linguagem aceite por um autómato (segunda versão)

- Extende-se a noção de palavra **reconhecida** ou **aceite** da seguinte forma: diz-se que $a_1 a_2 a_3 \dots a_n$ 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.

- 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 estendida. Neste caso a execução é única.

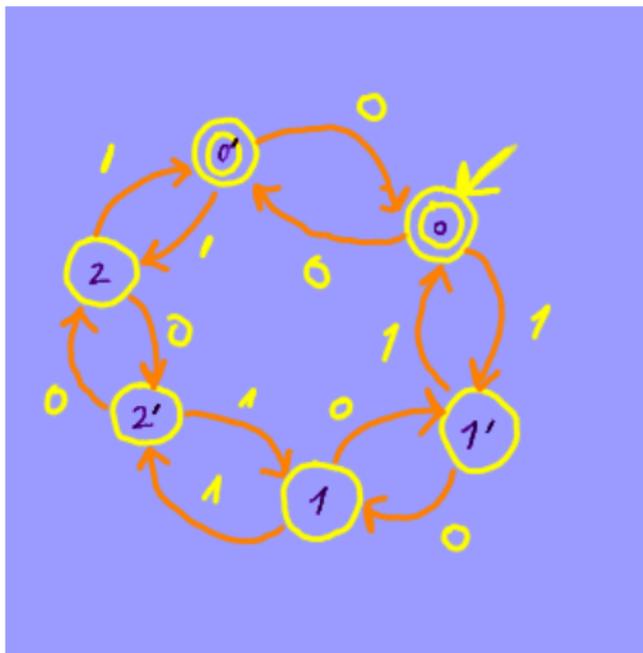


Figura: Qual é a linguagem reconhecida por este autómato?

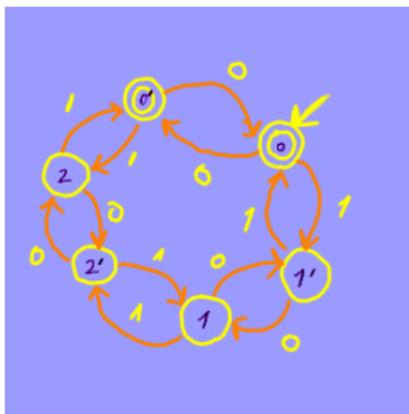


Figura: Qual é a linguagem reconhecida por este Autómato?

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 $0'$

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

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

Diz-se que c tem como **comprimento** n , que $u = a_1 a_2 \dots a_n$ 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

- Seja $M = \{Q, \Sigma, R_\delta, S, F\}$. Um caminho $c = q_0 \dots q_n$ 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 caminho 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 $Rec(A^*)$

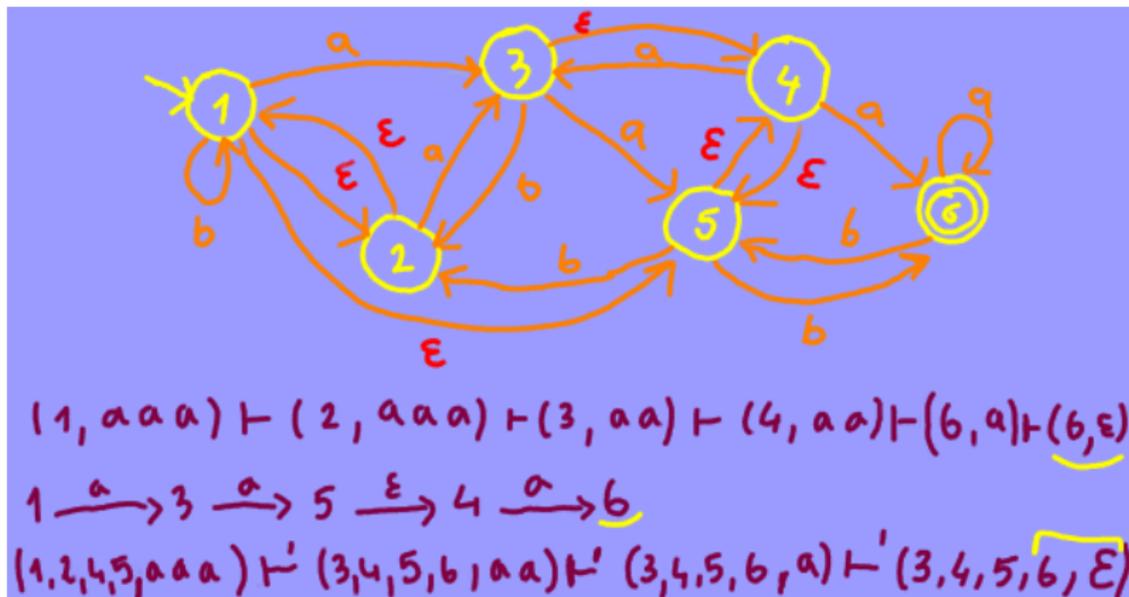


Figura: Um exemplo ilustrativo

Algoritmia dos autómatos

Algoritmia dos autómatos

Algoritmia de base

- estado poço
- remoção das transições ϵ ($NDFN_{\epsilon} \rightarrow NDFN$)
- remoção dos estados não produtivos e dos estados inacessíveis
- completção

Ver acetatos manuscritos distribuídos nas aulas

Algoritmia dos autómatos

Equivalência entre NDFA e DFA

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

Ver acetatos manuscritos distribuídos nas aulas

Teorema de Kleene

Teorema de Kleene

Propriedades de fecho das linguagens regulares

Das linguagens regulares aos autómatos

De facto podemos reforçar a implicação (das linguagens regulares aos autómatos) com a seguinte propriedade (cuja demonstração é em parte relacionada com a existência do algoritmo (e na sua correcção) que permite transformar uma expressão regular num autómatato): A classe das linguagens aceites por autómatos finitos é fechada por

- união (se L e L' regulares então $L \cup L'$ regular)
- concatenação (se L e L' regulares então $L.L'$ regular)
- fecho de Kleene/estrela (se L regular então L^* regular)
- complementação (se L e L' regulares então $\Sigma^* - L$ regular)
- intersecção (se L e L' regulares então $L \cap L'$ regular)
- diferença (se L e L' regular então $L - L'$ regular)
- espelho (se L regular então $L^r = \{c_n \cdots c_3 c_2 c_1 \mid c_1 c_2 c_3 \cdots c_n \in L(\forall n)\}$ regular)

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

- Propriedades de fecho das linguagens reconhecidas por autómatos finitos

Ver acetatos manuscritos distribuídos nas aulas

Linguagens regulares e propriedades de fecho

- O conjunto das palavras binárias divisíveis por 2 e o conjunto das palavras binárias divisíveis por 3 formam linguagens regulares
- Seja $\Sigma = \{0, 1\}$ e seja $L \subseteq \Sigma^*$ o conjunto das palavras binárias divisíveis por 2 e 3. L é regular.
- Seja $\Sigma = \{0, 1, 2, 3, \dots, 9\}$ e seja $L \subseteq \Sigma^*$ o conjunto dos inteiros naturais divisíveis por 2 e 3. L é regular.

Como demonstrar que são regulares?

- exibindo que são linguagens reconhecidas por um dado autómato finito ou
- que são linguagens que se expressam como compostas a partir de outras linguagens comprovadamente conhecidas como regulares

Ou seja

usamos as propriedades de fecho das linguagens regulares!

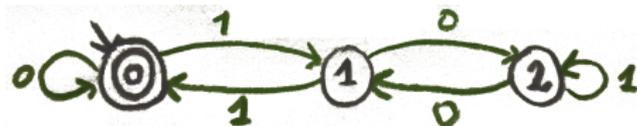
Linguagens regulares e propriedades de fecho

retomemos o exemplo da linguagem das **palavras binárias divisíveis por 2 ou 3**

seja M o autómato que reconhece a linguagem dos números binários divisíveis por 2



seja M' o autómato que reconhece a linguagem dos números binários divisíveis por 3



nestes autómatos, o nome do estado representa o resto da divisão (e.g. o estado 2 representa os valores binários para os quais o resto da divisão por 3 dá 2)

L é regular porque é $L(M) \cap L(M')$ (isto porque as linguagens regulares são fechadas por intersecção)

- Linguagens Regulares = Linguagens reconhecidas por autómatos

Ver acetatos manuscritos distribuídos nas aulas

Teorema de Kleene

Digressões sobre o teorema de Kleene

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.

Retomemos a nossa digressão sobre o teorema de Kleene.

- Designemos de *aut2regex* o primeiro algoritmo, e de *regex2aut* o segundo algoritmo.
- A demonstração relacionada com *aut2regex* 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
 $L(e) = L(\text{aut2regex}(\text{regex2aut}(e)))$ e
 $L(M) = L(\text{regex2aut}(\text{aut2regex}(M)))$ qualquer que sejam M e e .

Teorema de Kleene

Dos Autómatos às expressões regulares

vamos aqui usar um princípio de indução elaborado (e respectiva recursão)

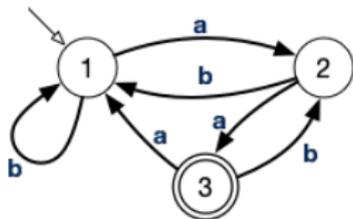
nos autómatos que vamos considerar, os (digamos n) estados são designados de 1 até n

vamos construir as expressões regulares para um conjunto ordenado particular de caminhos:

- caso de base (entre dois estados): uma simples transição ou até nenhuma
- passo indutivo: os caminhos (entre potencialmente vários estados) que nos autorizamos percorrer conforme um critério muito específico

um k -caminho é um caminho no grafo induzido pelo autómato que não passa por nenhum estado de identificação maior do que $k - 1$

os estados nas extremidades do caminho não são sujeitos a esta restrição, podem ser quaisquer



- 1-caminhos entre 2 e 3 :
caminho directo entre 2 e 3
expressão regular correspondente a
- 2-caminhos entre 2 e 3 :
caminho directo entre 2 e 3 ou usando
possivelmente o estado intermédio 1
expressão regular correspondente $a + \emptyset$,
ou seja a
- 3-caminhos entre 2 e 3 :
caminho directo entre 2 e 3 ou usando
possivelmente os estados intermédios 1 e 2
expressão regular correspondente
 $a + bb^*aa$, ou seja $(b^+a + \epsilon)a$
- 4-caminhos entre 2 e 3 : **exercício!**

Vamos considerar a expressão regular $R_{i,j}^k$ que caracteriza o conjunto das palavras constituintes dos k -caminhos do estado i ao estado j .

caso de base : $k = 1$, ou seja o conjunto (a soma $+$) das letras das transições directas entre i e j

- \emptyset se não há tais transições
- considerar adicionalmente ϵ às transições existentes se $i = j$

passo indutivo : um k -caminho do estado i para o estado j

- nunca passa pelo estado k
- ou então, passa ocasionalmente (uma ou mais vezes) por k

$$R_{ij}^{k+1} = R_{ij}^k + R_{ik}^k (R_{kk}^k)^* R_{kj}^k$$

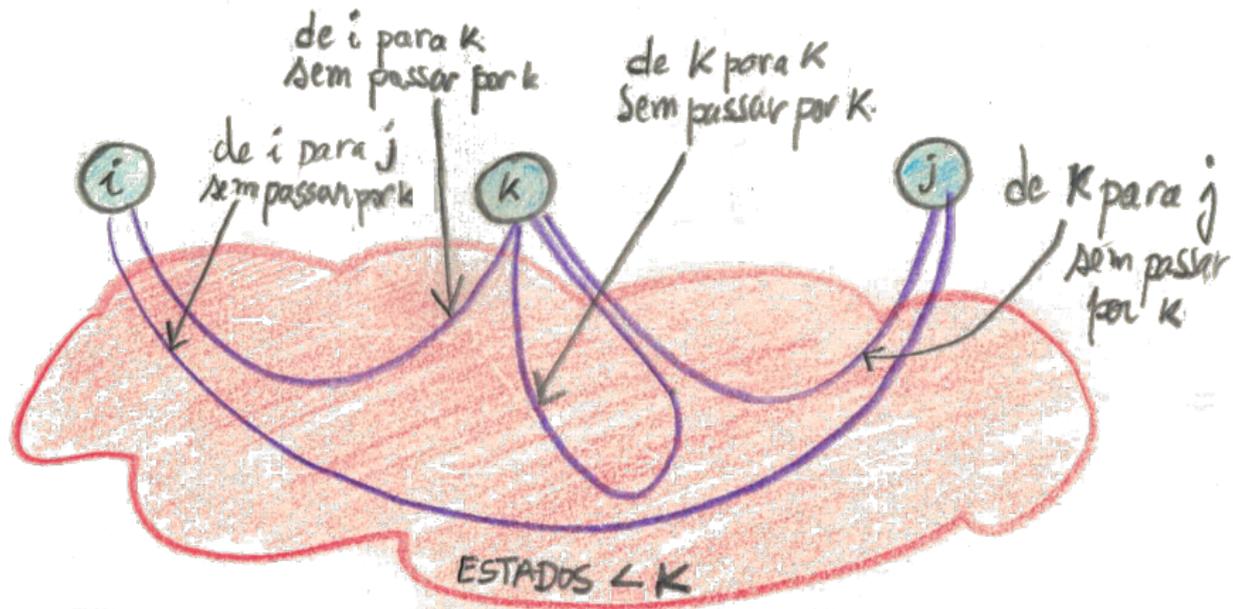
R_{ij}^k de i para j
 Sem passar por k

R_{ik}^k de i para k
 Sem passar por k (em nós intermédios)

$(R_{kk}^k)^*$ 0 ou mais vezes de k para k

R_{kj}^k de k para j

indução k - caso indutivo



Seja M um autómato, $Q = \{q_1, q_2, q_3, \dots, q_n\}$ 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 $\{q_1, \dots, q_{k-1}\}$

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

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

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

- 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**.

Leis algébricas das expressões regulares

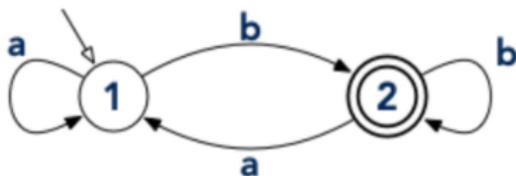
relembramos, para o uso proveitoso neste algoritmo, algumas leis algébricas (consagradas pelas álgebras de Kleene) das expressões regulares

sejam r , s e t , expressões regulares

$$\begin{array}{llll} \emptyset + r & = & r + \emptyset & = & r & (\emptyset \text{ é a identidade para } +) \\ \epsilon.r & = & r.\epsilon & = & r & (\epsilon \text{ é a identidade para } .) \\ \emptyset.r & = & r.\emptyset & = & \emptyset & (\emptyset \text{ é o elemento absorvente para } .) \\ r + (s + t) & = & (r + s) + t & & & (\text{associatividade de } +) \\ r.(s.t) & = & (r.s).t & & & (\text{associatividade de } .) \\ r + s & = & s + r & & & (\text{comutatividade de } +) \\ r.(s + t) & = & r.s + r.t & & & (\text{distributividade 1}) \\ (r + s).t & = & r.t + s.t & & & (\text{distributividade 2}) \\ r + r & = & r & & & (\text{idempotência de } +) \end{array}$$

acessoriamente (na verdade: em consequência)

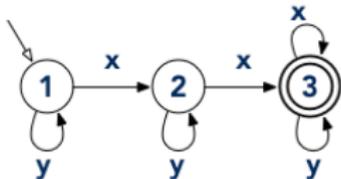
$$\begin{array}{ll} (\epsilon + r)^* & = & r^* \\ (\epsilon + r)^+ & = & r^* \\ r^{**} & = & r^* \\ s + r^*s & = & r^*s \end{array}$$



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

ou seja $(a^*b) + (a^*b)(\epsilon + a^*b)^*(\epsilon + a^*) = (a^*b)^+$



Temos $L(M) = R(1, 3, 4)$.

Calculemos estes valores por parte começando por $k = 4$ até chegar à $k = 1$

$$k = 4 \quad \text{-----}$$

$$R(1, 3, 4) = R(1, 3, 3) \cup R(1, 3, 3) \cdot R(3, 3, 3)^* \cdot R(3, 3, 3)$$

$$k = 3 \quad \text{-----}$$

$$R(1, 3, 3) = R(1, 3, 2) \cup R(1, 2, 2) \cdot R(2, 2, 2)^* \cdot R(2, 3, 2)$$

$$R(3, 3, 3) = R(3, 3, 2) \cup R(3, 2, 2) \cdot R(2, 2, 2)^* \cdot R(2, 3, 2)$$

$$k = 2 \quad \text{-----}$$

$$R(1, 3, 2) = R(1, 3, 1) \cup R(1, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 3, 1)$$

$$R(1, 2, 2) = R(1, 2, 1) \cup R(1, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 2, 1)$$

$$R(2, 2, 2) = R(2, 2, 1) \cup R(2, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 2, 1)$$

$$R(2, 3, 2) = R(2, 3, 1) \cup R(2, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 3, 1)$$

$$R(3, 3, 2) = R(3, 3, 1) \cup R(3, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 3, 1)$$

$$R(3, 2, 2) = R(3, 2, 1) \cup R(3, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 2, 1)$$

$$k = 1 \quad \text{-----}$$

$$R(1, 3, 1) = \emptyset$$

$$R(1, 1, 1) = y + \epsilon$$

$$R(1, 2, 1) = x$$

$$R(2, 2, 1) = y + \epsilon$$

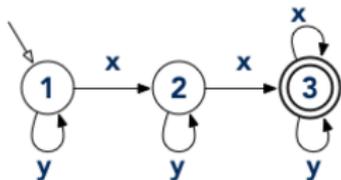
$$R(2, 1, 1) = \emptyset$$

$$R(2, 3, 1) = x$$

$$R(3, 3, 1) = x + y + \epsilon$$

$$R(3, 1, 1) = \emptyset$$

$$R(3, 2, 1) = \emptyset$$



Temos $L(M) = R(1, 3, 4)$.

Calculamos estes valores por parte começando por $k = 4$ até chegar à $k = 1$

$$k = 4 \quad \text{-----}$$

$$R(1, 3, 4) = R(1, 3, 3) \cup R(1, 3, 3).R(3, 3, 3)^*.R(3, 3, 3)$$

$$k = 3 \quad \text{-----}$$

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

$$R(3, 3, 3) = R(3, 3, 2) \cup R(3, 2, 2).R(2, 2, 2)^*.R(2, 3, 2)$$

$$k = 2 \quad \text{-----}$$

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

$$= \emptyset + (y + \epsilon)(y + \epsilon)^*\emptyset = \emptyset$$

$$R(1, 2, 2) = R(1, 2, 1) \cup R(1, 1, 1).R(1, 1, 1)^*.R(1, 2, 1)$$

$$= x + (y + \epsilon)(y + \epsilon)^*x = y^*x$$

$$R(2, 2, 2) = R(2, 2, 1) \cup R(2, 1, 1).R(1, 1, 1)^*.R(1, 2, 1)$$

$$= (y + \epsilon) + \emptyset(y + \epsilon)^*x = y + \epsilon$$

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

$$= x + \emptyset(y + \epsilon)^*\emptyset = x$$

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

$$= (x + y + \epsilon) + \emptyset(y + \epsilon)^*\emptyset = x + y + \epsilon$$

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

$$= \emptyset + \emptyset(y + \epsilon)^*x = \emptyset$$

$$k = 1 \quad \text{-----}$$

$$R(1, 3, 1) = \emptyset$$

$$R(1, 1, 1) = y + \epsilon$$

$$R(1, 2, 1) = x$$

$$R(2, 2, 1) = y + \epsilon$$

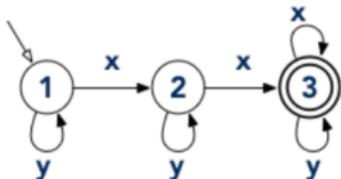
$$R(2, 1, 1) = \emptyset$$

$$R(2, 3, 1) = x$$

$$R(3, 3, 1) = x + y + \epsilon$$

$$R(3, 1, 1) = \emptyset$$

$$R(3, 2, 1) = \emptyset$$



Temos $L(M) = R(1, 3, 4)$.

Calculamos estes valores por parte começando por $k = 4$ até chegar à $k = 1$

$$k = 4 \quad \frac{R(1, 3, 4) = R(1, 3, 3) \cup R(1, 3, 3).R(3, 3, 3)^*.R(3, 3, 3)}$$

$$k = 3 \quad \frac{R(1, 3, 3) = R(1, 3, 2) \cup R(1, 2, 2).R(2, 2, 2)^*.R(2, 3, 2)}{= \emptyset + (y^*x)(y + \epsilon)^*x = y^*xy^*x}$$

$$R(3, 3, 3) = R(3, 3, 2) \cup R(3, 2, 2).R(2, 2, 2)^*.R(2, 3, 2)$$

$$= (x + y + \epsilon) + \emptyset(y^*x + \epsilon)^*x = \epsilon + x + y$$

$$k = 2 \quad \frac{R(1, 3, 2) = R(1, 3, 1) \cup R(1, 1, 1).R(1, 1, 1)^*.R(1, 3, 1)}{= \emptyset}$$

$$R(1, 2, 2) = R(1, 2, 1) \cup R(1, 1, 1).R(1, 1, 1)^*.R(1, 2, 1)$$

$$= y^*x$$

$$R(2, 2, 2) = R(2, 2, 1) \cup R(2, 1, 1).R(1, 1, 1)^*.R(1, 2, 1)$$

$$= y + \epsilon$$

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

$$= x$$

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

$$= x + y + \epsilon$$

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

$$= \emptyset$$

$$k = 1 \quad \frac{R(1, 3, 1) = \emptyset}{R(1, 1, 1) = y + \epsilon}$$

$$R(1, 2, 1) = x$$

$$R(2, 2, 1) = y + \epsilon$$

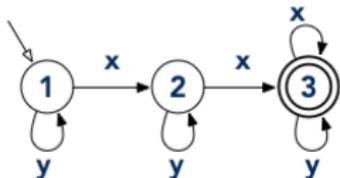
$$R(2, 1, 1) = \emptyset$$

$$R(2, 3, 1) = x$$

$$R(3, 3, 1) = x + y + \epsilon$$

$$R(3, 1, 1) = \emptyset$$

$$R(3, 2, 1) = \emptyset$$



Temos $L(M) = R(1, 3, 4)$.

Calculamos estes valores por parte começando por $k = 4$ até chegar à $k = 1$

$$\begin{aligned}
 k = 4 \quad \hline
 R(1, 3, 4) &= R(1, 3, 3) \cup R(1, 3, 3) \cdot R(3, 3, 3)^* \cdot R(3, 3, 3) = \\
 &= (y^*xy^*x) + (y^*xy^*x)(\epsilon + x + y)^*(\epsilon + x + y) \\
 &= y^*xy^*x(x + y)^*
 \end{aligned}$$

$$\begin{aligned}
 k = 3 \quad \hline
 R(1, 3, 3) &= R(1, 3, 2) \cup R(1, 2, 2) \cdot R(2, 2, 2)^* \cdot R(2, 3, 2) \\
 &= y^*xy^*x \\
 R(3, 3, 3) &= R(3, 3, 2) \cup R(3, 2, 2) \cdot R(2, 2, 2)^* \cdot R(2, 3, 2) \\
 &= \epsilon + x + y
 \end{aligned}$$

$$\begin{aligned}
 k = 2 \quad \hline
 R(1, 3, 2) &= R(1, 3, 1) \cup R(1, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 3, 1) \\
 &= \emptyset \\
 R(1, 2, 2) &= R(1, 2, 1) \cup R(1, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 2, 1) \\
 &= y^*x \\
 R(2, 2, 2) &= R(2, 2, 1) \cup R(2, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 2, 1) \\
 &= y^*x + \epsilon \\
 R(2, 3, 2) &= R(2, 3, 1) \cup R(2, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 3, 1) \\
 &= x \\
 R(3, 3, 2) &= R(3, 3, 1) \cup R(3, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 3, 1) \\
 &= x + y + \epsilon \\
 R(3, 2, 2) &= R(3, 2, 1) \cup R(3, 1, 1) \cdot R(1, 1, 1)^* \cdot R(1, 2, 1) \\
 &= y^*x + \epsilon
 \end{aligned}$$

$$\begin{aligned}
 k = 1 \quad \hline
 R(1, 3, 1) &= \emptyset \\
 R(1, 1, 1) &= y + \epsilon \\
 R(1, 2, 1) &= x \\
 R(2, 2, 1) &= y + \epsilon \\
 R(2, 1, 1) &= \epsilon \\
 R(2, 3, 1) &= x \\
 R(3, 3, 1) &= x + y + \epsilon \\
 R(3, 1, 1) &= \epsilon \\
 R(3, 2, 1) &= \epsilon
 \end{aligned}$$

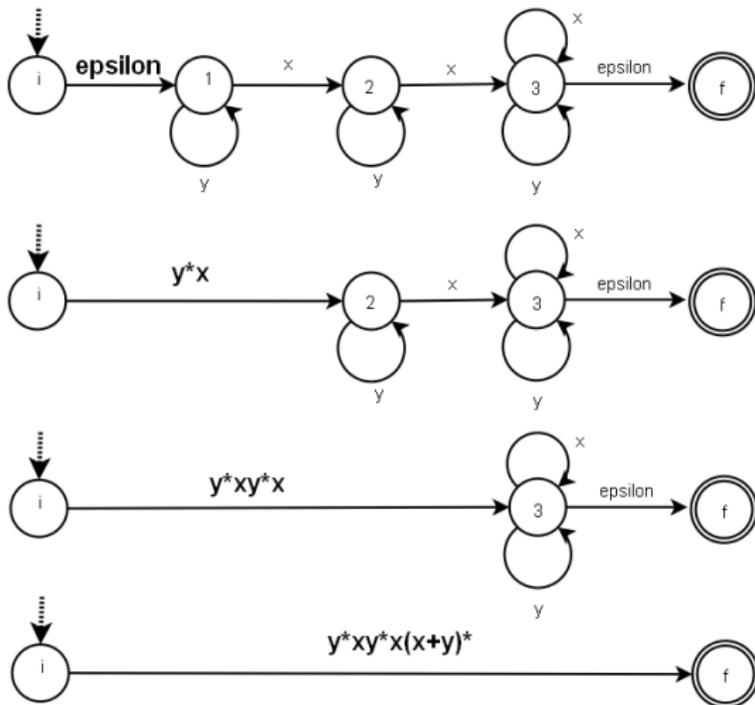
Variante do algoritmo: Técnica da eliminação de estados

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.

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 $l_{pqr} = l_{pq} \cup l_{pq} \cdot l_{qq}^* \cdot l_{qr}$. 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 .



- Conjuntos das palavras binárias divisíveis por 2 (mas também por 3 ou por k , $k \geq 4$)
- Seja $\Sigma = \{0, 1, 2, \dots, 9\}$ 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)

Autómatos e Computação

Autómatos e Computação

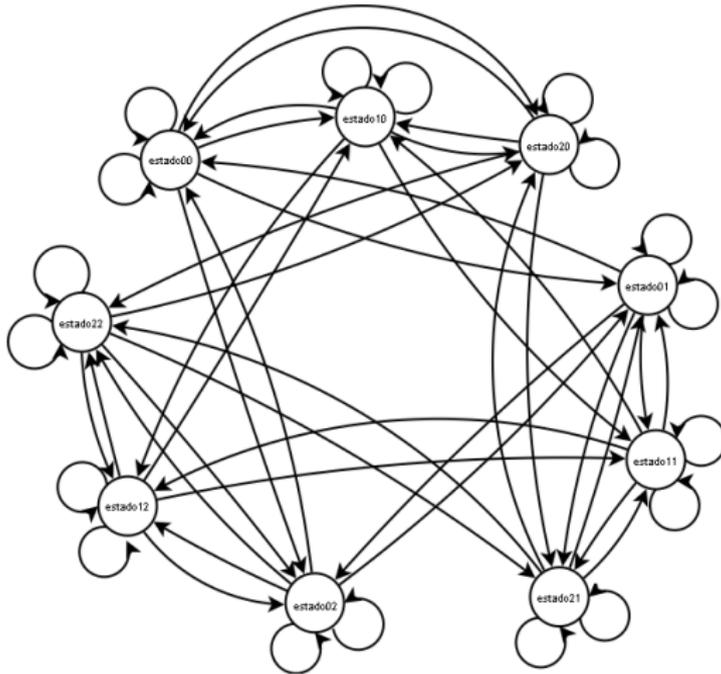
Autómatos vistos como algoritmos

- 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.

- 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 $\{0 \dots 2\}$.
- 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.

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 = ...
```



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.

Autómatos e Computação

Limites algorítmicos dos autómatos de estados finitos

1. Todas as linguagens finitas são regulares.
2. Uma linguagem não regular é assim necessariamente infinita (o inverso é falso).
3. Uma linguagem cujas palavras tem um comprimento limitado (por $p \in \mathbb{N}$, por exemplo) é necessariamente finita.
4. Se uma linguagem é infinita, então não há limite para o tamanho das palavras que contém.
5. Qualquer linguagem regular é aceita por um autômato finito, este contém um número finito de estados, digamos m .
6. Consideremos uma linguagem regular infinita L e um autômato 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 autômato M passará pelo menos duas vezes.
7. 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 $x.u^*.y$.
8. Assim espera-se que as palavras infinitas duma linguagem regular que essas apresentam um tal padrão.

Lema (de Bombeamento)

Seja L uma linguagem regular, existe um inteiro $m \geq 1$ tal que qualquer palavra w de L com $|w| \geq m$ pode ser reescrita como $w = xuy$ onde $u \neq \epsilon$, $|xu| \leq m$ 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 $x.u^n.y \in L$

outra formulação, com quantificadores (7!):

$\forall L \in \text{Reg}(\Sigma).$

$\exists m \in \mathbb{N}^*.$

$\forall w \in L. |w| \geq m \implies$

$\exists x \in \Sigma^* \exists u \in \Sigma^+ \exists y \in \Sigma^*. (w = xuy \wedge |xu| \leq m \wedge \forall n \in \mathbb{N}. xu^n y \in L)$

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:

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

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 $x = w_1 w_2 \dots w_i$, $u = w_{i+1} \dots w_j$, $y = w_{j+1} \dots w_n$. Finalmente temos $|xu^*y| = j \leq n$.

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. $L_1 = \epsilon \cup \{ab\} \cup \{aabb\} \dots \cup \{\}$ 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 para todas as palavras de tamanho $\geq m$ 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). Consideremos a palavra $a^{\lceil \frac{m}{2} \rceil} b^{\lceil \frac{m}{2} \rceil}$.
 - $u \in a^+$. Impossível porque se repetimos u mais do que uma vez temos mais a do que b .
 - $u \in b^+$. Impossível por razões semelhantes.
 - $u \in (a \cup b)^* - (a^* \cup b^*)$ e $u \neq \epsilon$. 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.

Exercício: e se tivesse escolhido a palavra $a^m b^m$?

Aplicações do Teorema do bombeamento

$L = \{a^n \mid n \text{ primo}\}$ não é regular.

Demonstração: Procedemos mais uma vez por contradição.

Hipótese de trabalho: 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 $|w| \geq m$. Logo podemos arranjar uma decomposição xuy de w tal que todos os $xu^n y$ também pertençam a L (com $n \in \mathbb{N}$).

Vejamus 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.

Vejamus agora se todos os k validam esta imposição (condição necessária).

Ora para $k = p + 2q + r + 2$, $p + kq + r = (q + 1)(p + 2q + r)$, $q + 1 > 1$, $(p + 2q + r) > 1$ logo não é primo.

Contradição. L não pode ser regular.

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

(Exercício: demonstrar)

Autómatos e Computação

Problemas e algoritmos sobre linguagens regulares

- 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.

Autómatos finitos com output

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 estender os autómatos de forma a que esses consigam fornecer respostas mais ricas: **os autómatos com saída** (**transducers** em inglês).

- 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 $f : Q \rightarrow \Omega \cup \{\epsilon\}$. 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: $g : Q \times \Sigma \rightarrow \Omega \cup \{\epsilon\}$. 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.

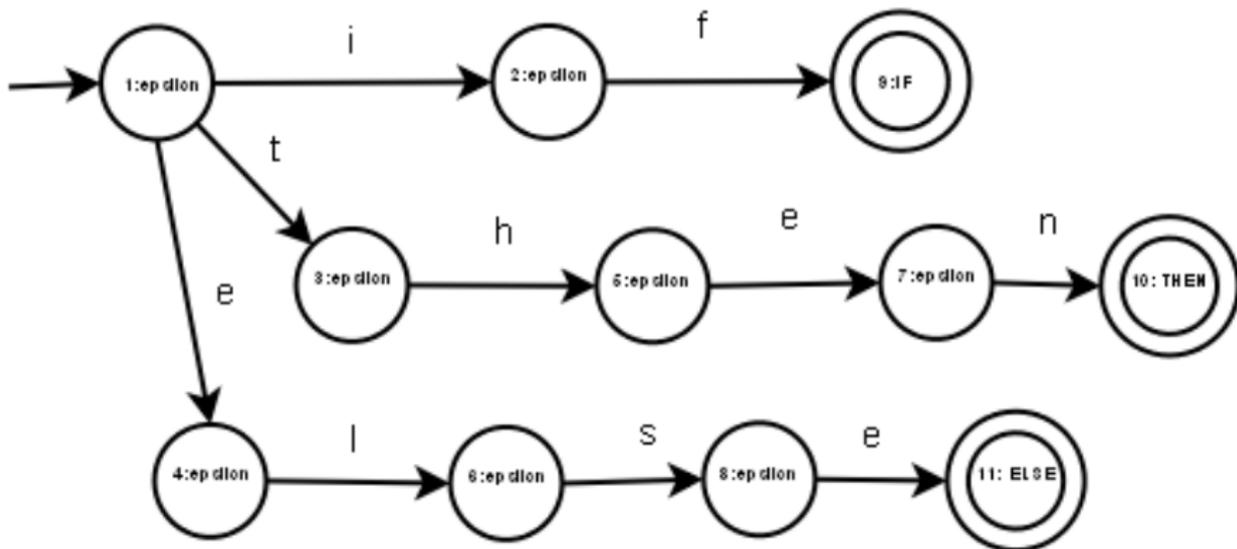


Figura: Um exemplo de autômato de Moore: um lexer simples