Universidade de Aveiro Departamento de Matemática
2011

Ana Micaela Gomes Batista

Algoritmos de Minimização de Autómatos

Universidade de Aveiro Departamento de Matemática
2011

Ana Micaela Gomes Batista

Algoritmos de Minimização de Autómatos

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações, realizada sob a orientação científica do Doutor António Ferreira Pereira, Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro.

o júri

presidente

Prof. Doutora Isabel Maria Simões Pereira
Professora auxiliar da Universidade de Aveiro

Prof. Doutora Maria Rosália Dinis Rodrigues
Professora associada da Universidade de Aveiro

Prof. Doutor António Ferreira Pereira
Professor auxiliar da Universidade de Aveiro (Orientador)

agradecimentos

Quero agradecer ao meu orientador, Professor António Pereira, pela sua disponibilidade, ajuda e paciência durante o tempo em que realizei este trabalho.
Agradeço também à minha família e amigos pelo apoio dado ao longo de todo o tempo.

palavras-chave

Autómato finito determinista, autómato finito não-determinista, relação de equivalência de estados, relação de distinguibilidade de estados, classes de equivalência da relação de equivalência de estados, algoritmos de minimização.

resumo

O presente trabalho aborda vários algoritmos de minimização de autómatos. Aqui são apresentadas algumas definições relacionadas com autómatos, como por exemplo relação de equivalência de estados, relação de distinguibilidade de estados e classes de equivalência da relação de equivalência de estados, definições que estão na base dos algoritmos apresentados. É feita uma síntese de vários algoritmos de minimização, como por exemplo o algoritmo de Hopcroft e o algoritmo de Brzozowski, e são apresentadas as respectivas ordens de complexidade.

keywords

Deterministic finite automaton, non-deterministic finite automaton, equivalence relation on states, distinguishability relation on states, equivalence classes of equivalence relation on states, minimization algorithms.

abstract

The present work shows several automaton minimization algorithms. Some definitions related to automaton are presented, for example the equivalence relation on states, the distinguishability relation on states and the equivalence classes of equivalence relation on states, which are the basis of the presented algorithms. A synthesis of several minimization algorithms is made, for example the Hopcroft algorithm and the Brzozowski algorithm, and their complexity orders are presented.

Conteúdo

Lista de Figuras

Lista de Tabelas

Introdução

Algumas das aplicações dos autómatos passam pelo reconhecimento de linguagens e construção de compiladores, nas fases de análise léxica e sintáctica. Uma vez que se utilizam autómatos no desenvolvimento de compiladores e processadores de linguagens, então durante este processo pode ser necessário optimizar o autómato.

Nesta dissertação analisam-se diferentes algoritmos para minimizar autómatos, pois nem sempre se tem um autómato mínimo ou não se sabe se é mínimo. Numa situação em que o autómato não é mínimo ou não se tem a garantia que seja, pode interessar determinar o autómato mínimo.

Existem algoritmos de minimização que se baseiam em relações de equivalência e de distinguibilidade de estados. Destes algoritmos, ainda se podem separar os que determinam estas relações pelo cálculo iterativo através da definição, e os que não determinam estas mesmas relações de forma sequencial. Consideram-se também os algoritmos que determinam as classes de equivalência da relação de equivalência de estados. Todos estes algoritmos apenas minimizam autómatos deterministas. Se o autómato for não determinista, então, antes de se utilizar qualquer um destes algoritmos, é preciso converter o autómato num autómato determinista.

Um destes algoritmos tem a particularidade, caso não se execute todas as iterações, de permitir determinar um autómato equivalente ao inicial com tamanho inferior. Contudo, este autómato pode não ser o autómato mínimo.

Um outro algoritmo considerado é o algoritmo de Brzozowski, que tem a particularidade de minimizar tanto autómatos deterministas como não-deterministas sem transições-ε.

No primeiro capítulo introduzem-se algumas definições que são utilizadas nos capítulos seguintes, como por exemplo: os diferentes tipos de autómatos, conversão de autómatos finitos não-deterministas em autómatos finitos deterministas e minimização de autómatos.

No capítulo seguinte descrevem-se vários algoritmos de minimização de autómatos que se baseiam no conceito de equivalência de estados. Para além da descrição também são determinadas as suas ordens de complexidade.

No último capítulo analisa-se o algoritmo de Brzozowski e determina-se a sua ordem de complexidade. Este algoritmo tem como principal diferença em relação aos algoritmos anteriores, o facto de minimizar tanto autómatos deterministas como autómatos não-deterministas sem transições-ε. Descreve-se também uma variante para substituir os dois últimos passos deste algoritmo e são efectuadas melhorias a esta variante do algoritmo de Brzozowski.

Capítulo 1

Conceitos Fundamentais

Neste capítulo apresentam-se algumas definições que serão utilizadas nos capítulos seguintes. Estes conceitos encontram-se na maioria dos livros introdutórios da Teoria da Computação, [2], [4], [7], [9] e [11], no entanto a notação escolhida é a utilizada em [8]. Define-se autómato finito determinista, autómato finito não-determinista com e sem transições-ε, autómato reverso, autómato co-determinista, função de transição, linguagem de um autómato, fecho-ε de um estado, conversão de autómatos finitos não-deterministas em autómatos finitos deterministas e minimização de autómatos.

Os autómatos considerados, tanto os deterministas como os não-deterministas, são sempre autómatos finitos completos, isto é, as funções de transição são totais o que é essencial para os algoritmos de minimização de autómatos apresentados nos capítulos 2 e 3.

1.1 Definições de Autómatos e Linguagens

Definição 1.1 (Autómato Finito Determinista)**.**

Um Autómato Finito Determinista (AFD) é um quíntuplo ordenado,

$A = (Q, \Sigma, \delta, q_0, F)$ onde:

- Q é um conjunto finito não vazio de estados;
- Σ é um conjunto finito de letras, ao qual se chama alfabeto;
- $\delta : Q \times \Sigma \to Q$ é a função de transição entre estados;
- $q_0 \in Q$ é o estado inicial;

- $F \subseteq Q$ é o conjunto dos estados finais ou de aceitação.

Exemplo 1.1. A figura seguinte representa graficamente o autómato finito determinista

$$A = (\{1, 2, 3, 4\}, \{0, 1\}, \delta, \{1\}, \{4\})$$

onde $\delta(1, 0) = 2$, $\delta(1, 1) = 3$, $\delta(2, 0) = 4$, $\delta(2, 1) = 3$, $\delta(3, 0) = 2$, $\delta(3, 1) = 4$, $\delta(4, 0) = 4$ e $\delta(4, 1) = 4$. O estado inicial (indicado com uma seta triangular) é o estado 1. O estado final (indicado com um circulo duplo) é o estado 4.

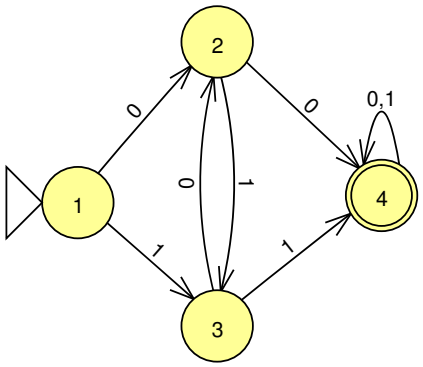

Figura 1.1: Autómato Finito Determinista

A uma sequência finita de letras de um alfabeto chama-se *palavra*. A sequência vazia chama-se *palavra vazia* e denota-se por ε.
A leitura de uma palavra é feita da esquerda para a direita. Por exemplo, para a palavra *abc*, primeiro é lida a letra a, depois a letra b e por último a letra c.

Denota-se por Σ^* o conjunto de todas as palavras que podem ser formadas com as letras do alfabeto Σ.

Considere-se o resultado das sucessivas transições, a partir de um estado, pela leitura de cada uma das letras de uma palavra. Sendo o autómato determinista, o resultado é um único estado, ao qual se chama *estado atingível*. Define-se desta forma a extensão da função de transição.

Definição 1.2 (Extensão da função de transição de um autómato finito determinista)**.**
A extensão da função de transição de um autómato finito determinista, $A = (Q, \Sigma, \delta, q_0, F)$,

$\hat{\delta} : Q \times \Sigma^* \to Q$, é definida por $\hat{\delta}(q, \varepsilon) = q$ e $\forall x \in \Sigma^*$, $\forall a \in \Sigma$, $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$.

Se $\hat{\delta}(p, w) = q$ então diz-se que por leitura da palavra w, partindo do estado p, se atinge o estado q. Uma palavra w é *reconhecida* por um autómato se, partindo do estado inicial, por leitura de w, é atingido um estado final.

Definição 1.3 (Linguagem de um autómato finito determinista)**.**
A linguagem de um autómato finito determinista, $A = (Q, \Sigma, \delta, q_0, F)$, é o conjunto de todas as palavras reconhecidas pelo autómato, ou seja, $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$.

Dado um conjunto finito X, $\mathcal{P}(X)$ representa o conjunto das partes de X, isto é, o conjunto de todos os subconjuntos de X.

Se partindo de um determinado estado pela leitura de uma letra, existem transições para mais do que um estado, então neste caso o autómato finito diz-se Não-Determinista.

Definição 1.4 (Autómato Finito Não-Determinista)**.**
Um Autómato Finito Não-Determinista (AFND) é um quíntuplo ordenado, $A = (Q, \Sigma, \delta, Q_0, F)$ onde:

- Q é um conjunto finito não vazio de estados;
- Σ é o alfabeto;
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ é a função de transição entre estados;
- $Q_0 \subseteq Q$ é o conjunto de estados iniciais;
- $F \subseteq Q$ é o conjunto dos estados finais ou de aceitação.

Exemplo 1.2. Na figura 1.2 representa-se o AFND

$$B = (\{1, 2, 3, 4\}, \{0, 1\}, \delta, \{1\}, \{4\})$$

onde $\delta(1, 0) = \{2, 3\}$, $\delta(1, 1) = \{4\}$, $\delta(2, 0) = \{3\}$, $\delta(2, 1) = \{3\}$, $\delta(3, 0) = \{2\}$, $\delta(3, 1) = \{4\}$, $\delta(4, 0) = \{4\}$ e $\delta(4, 1) = \{4\}$.

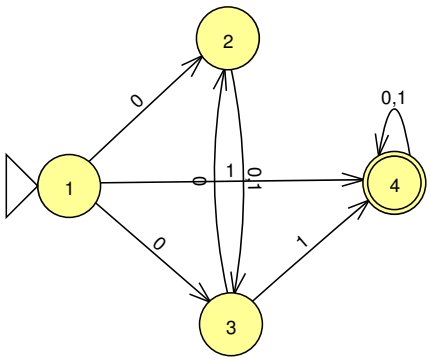

Figura 1.2: Autómato Finito Não-Determinista

À semelhança dos autómatos finitos deterministas, para os autómatos finitos não-deterministas também se podem considerar as transições a partir de um estado pela leitura de uma palavra, podendo neste caso atingir-se mais do que um estado.

Seja a uma letra, x uma palavra e q um estado. Verifica-se que partindo de q e pela leitura de x, o resultado pode ser mais do que um estado, isto é, um conjunto de estados. Por sua vez, partindo desse conjunto de estados após a leitura de a pode-se, uma vez mais, transitar para mais do que um estado, isto é, para um novo conjunto de estados. Este último conjunto de estados vai ser o resultado da transição a partir de q e após a leitura de xa. Desta forma, define-se a extensão da função transição.

Definição 1.5 (Extensão da função de transição de um autómato finito não-determinista)**.**

A extensão da função de transição de um autómato finito não-determinista, $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$, é definida por $\hat{\delta}(q, \varepsilon) = \{q\}$ e $\forall x \in \Sigma^*, \forall a \in \Sigma,\ \hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a)$.

Uma palavra $w \in \Sigma^*$ é reconhecida por um AFND quando existe $q_0 \in Q_0$ tal que o conjunto $\hat{\delta}(q_0, w)$ contém pelo menos um estado de aceitação.

Definição 1.6 (Linguagem de um autómato finito não-determinista)**.**

A Linguagem de um AFND, $A = (Q, \Sigma, \delta, Q_0, F)$, é o conjunto de todas as palavras reconhecidas pelo autómato, isto é, $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset, q_0 \in Q_0\}$.

Definição 1.7 (Autómato Finito Não-Determinista com Transições-ε)**.**

Designa-se por Autómato Finito Não-Determinista com Transições-ε (AFND-ε) um autómato

finito não determinista no qual se admitem transições de estados pela palavra vazia.

O conjunto de estados atingidos a partir de um estado por transições ε, chama-se fecho-ε desse estado. Note-se que todo o estado pertence ao seu fecho-ε.

Definição 1.8 (Fecho-ε)**.**
Seja p um estado de um AFND-ε, $A = (Q, \Sigma, \delta, Q_0, F)$. O fecho-$\varepsilon(p)$ é definido por :

- $p \in$ fecho-$\varepsilon(p)$;
- se $q \in$ fecho-$\varepsilon(p)$ e existe $r \in Q$ tal que $\delta(q, \varepsilon) = r$ então $r \in$ fecho-$\varepsilon(p)$.

Exemplo 1.3. A figura seguinte representa o AFND-ε

$$C = (\{1, 2, 3, 4\}, \{0, 1, \varepsilon\}, \delta, \{1\}, \{4\})$$

onde $\delta(1, 0) = \{2, 3\}$, $\delta(1, 1) = \{4\}$, $\delta(2, 0) = \{3\}$, $\delta(2, 1) = \{3\}$, $\delta(2, \varepsilon) = \{4\}$, $\delta(3, 0) = \{2\}$, $\delta(3, 1) = \{4\}$, $\delta(4, 0) = \{4\}$ e $\delta(4, 1) = \{4\}$.

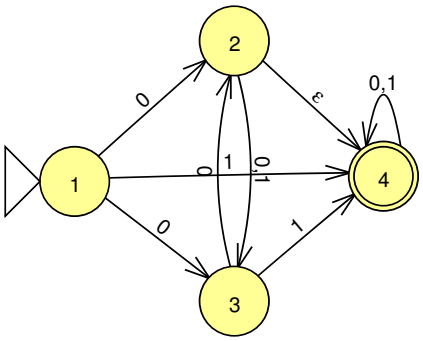

Figura 1.3: Autómato Finito Não-Determinista com Transições-ε

Os fecho-ε dos estados do autómato anterior são:

$$\text{fecho-}\varepsilon(1) = \{1\};\ \text{fecho-}\varepsilon(2) = \{2, 4\};\ \text{fecho-}\varepsilon(3) = \{3\};\ \text{fecho-}\varepsilon(4) = \{4\}$$

Pode-se definir de forma análoga à definição anterior fecho-ε de um conjunto de estados $C \subseteq Q$ por fecho-$\varepsilon(C) = \bigcup_{c \in C}$ fecho-ε(c).

No caso dos autómatos finitos não-deterministas com transições-ε, a extensão da função de transição tem em conta os efeitos das possíveis transições-ε.

Por exemplo, a função de transição extendida num estado $q \in Q$ por meio de ε é o fecho-$\varepsilon(q)$.

Definição 1.9 (Extensão da função de transição de um autómato finito não-determinista com transições-ε)**.**

A extensão da função de transição de um autómato finito não-determinista com transições-ε, $A = (Q, \Sigma, \delta, Q_0, F)$, é definida por $\forall q \in Q$, $\hat{\delta}(q, \varepsilon) = fecho - \varepsilon(q)$ e $\forall x \in \Sigma^*$, $\forall a \in \Sigma$, $\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q,x)} \text{fecho-}\varepsilon(\delta(p, a))$.

Uma palavra $w \in \Sigma^*$ é reconhecida por um AFND-ε quando o conjunto $\hat{\delta}(q_0, w)$ contém pelo menos um estado de aceitação.

Definição 1.10 (Linguagem de um autómato finito não-determinista com transições-ε)**.**

A Linguagem de um AFND-ε, $A = (Q, \Sigma, \delta, Q_0, F)$, é o conjunto de todas as palavras que são reconhecidas pelo autómato, ou seja, $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.

Note-se que a linguagem de um AFND-ε pode conter a palavra vazia mesmo quando o estado inicial é diferente do estado final, o que não acontece no caso dos autómatos deterministas e não-deterministas (sem transições-ε).

Definição 1.11 (Linguagens direitas e esquerdas de um estado num AFND-ε)**.**

A linguagem direita de um estado q é o conjunto de todas as palavras que o autómato reconhece a partir de q, isto é, $L_d(q) = \{w \in \Sigma^* \mid \hat{\delta}(q, w) \subseteq F\}$.

A linguagem esquerda de um estado q é o conjunto de todas as palavras que o autómato reconhece, partindo de um estado inicial até atingir q, isto é, $L_e(q) = \{w \in \Sigma^* \mid q \in \hat{\delta}(q_0, w),\ q_0 \in Q_0\}$.

Sejam $a \in \Sigma$, $x \in \Sigma^*$ e $w = ax$ uma palavra. Denota-se por w^R a *palavra reversa* de w, onde $w^R = x^R a$.

Definição 1.12 (Autómato reverso)**.**

Seja $A = (Q, \Sigma, \delta, Q_0, F)$ um autómato AFND-ε. Denota-se por A^R ou $r(A)$ o autómato re-

verso do autómato A, definido por $A^R = (Q, \Sigma, \delta^R, F, Q_0)$, onde para $p, q \in Q$ e $a \in \Sigma \cup \{\varepsilon\}$, se $q \in \delta(p, a)$ então $p \in \delta^R(q, a)$.

Para determinar o reverso de um autómato A é necessário:

- determinar δ^R, isto é, inverter o sentido de todas as transições em A;
- os estados iniciais do autómato reverso vão ser os estados finais de A;
- os estados finais do autómato reverso vão ser os estados iniciais de A.

Seja L a linguagem reconhecida por A. O autómato reverso de A reconhece as reversas das palavras de L, sendo esta linguagem representada por L^R.

Exemplo 1.4. A figura 1.4 representa o autómato reverso do autómato representado na figura 1.1.

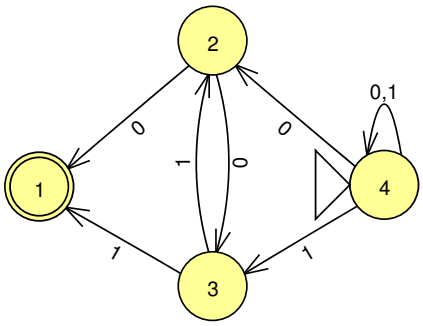

Figura 1.4: Autómato Reverso do Autómato representado na figura 1.1

O tamanho de um autómato finito, $A = (Q, \Sigma, \delta, Q_0, F)$, denota-se por $|A|$ e corresponde à cardinalidade do conjunto de estados, isto é, $|A| = |Q|$.

As definições de linguagens direita e esquerda, bem como de autómato reverso são válidas, com as adequadas adaptações, também para autómatos deterministas.

Um autómato finito determinista A é *co-determinista*[1] se o seu reverso, A^R, é determinista.

[1] Do inglês codeterministic.

Se um autómato finito não-determinista tiver mais do que um estado inicial é possível transformá-lo num autómato com um único estado inicial que reconhece a mesma linguagem. Para isto acrescenta-se um novo estado, o qual passará a ser o único estado inicial, com transições-ε para todos os estados que eram iniciais.

Definição 1.13 (Autómatos equivalentes)**.**

Diz-se que dois autómatos finitos, A e B, são equivalentes se reconhecem a mesma linguagem, ou seja, se $L(A) = L(B)$.

1.2 Conversão de autómatos finitos não-deterministas em autómatos deterministas

A partir de um AFND, $A = (Q_A, \Sigma, \delta_A, Q_0, F_A)$, é possível obter um AFD equivalente, $d(A) = B = (Q_B, \Sigma, \delta_B, q_B, F_B)$, definido por:

- $Q_B = \mathcal{P}(Q_A)$
- $q_B = Q_0$ é o conjunto de estados iniciais de A
- $F_B = \{S \in Q_B \,|\, S \cap F_A \neq \emptyset\}$
- $\forall S \in Q_B,\ \forall a \in \Sigma,\ \delta_B(S, a) = \bigcup_{p \in S} \delta_A(p, a)$

A partir de um AFND-ε, $A = (Q_A, \Sigma, \delta_A, Q_0, F_A)$, é possível obter um AFD equivalente, $d(A) = B = (Q_B, \Sigma, \delta_B, q_B, F_B)$, definido por:

- $Q_B = \mathcal{P}(Q_A)$
- $q_B =$fecho-$\varepsilon(Q_0)$
- $F_B = \{S \in Q_B \,|\, S \cap F_A \neq \emptyset\}$
- $\forall S \in Q_B,\ \forall a \in \Sigma,\ \delta_B(S, a) = \bigcup_{r \in fecho-\varepsilon(S)}$ fecho-$\varepsilon(\delta(r, a))$

Exemplo 1.5. Ilustra-se em seguida a tabela de transições do autómato finito determinista equivalente ao autómato representado na figura 1.2.

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{\mathbf{1}\}$	$\{\mathbf{2},\mathbf{3}\}$	$\{\mathbf{4}\}$
$\{\mathbf{2}\}$	$\{\mathbf{3}\}$	$\{\mathbf{3}\}$
$\{\mathbf{3}\}$	$\{\mathbf{2}\}$	$\{\mathbf{4}\}$
$*\{\mathbf{4}\}$	$\emptyset$	$\emptyset$
$\{1,2\}$	$\{2,3\}$	$\{3,4\}$
$\{1,3\}$	$\{2,3\}$	$\{4\}$
$*\{1,4\}$	$\{2,3\}$	$\{4\}$

	0	1
$\{\mathbf{2},\mathbf{3}\}$	$\{\mathbf{2},\mathbf{3}\}$	$\{\mathbf{3},\mathbf{4}\}$
$*\{2,4\}$	$\{3\}$	$\{3\}$
$*\{\mathbf{3},\mathbf{4}\}$	$\{\mathbf{2}\}$	$\{\mathbf{4}\}$
$\{1,2,3\}$	$\{2,3\}$	$\{3,4\}$
$*\{1,2,4\}$	$\{2,3\}$	$\{3,4\}$
$*\{1,3,4\}$	$\{2,3\}$	$\{4\}$
$*\{2,3,4\}$	$\{2,3\}$	$\{3,4\}$
$*\{1,2,3,4\}$	$\{2,3\}$	$\{3,4\}$

Tabela 1.1: Tabela de transições do autómato finito determinista equivalente ao autómato representado na figura 1.2

Nesta tabela verifica-se que os estados atingíveis são: $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{2,3\}$ e $\{3,4\}$. Com estes estados constroi-se o correspondente autómato finito determinista, representado na figura seguinte:

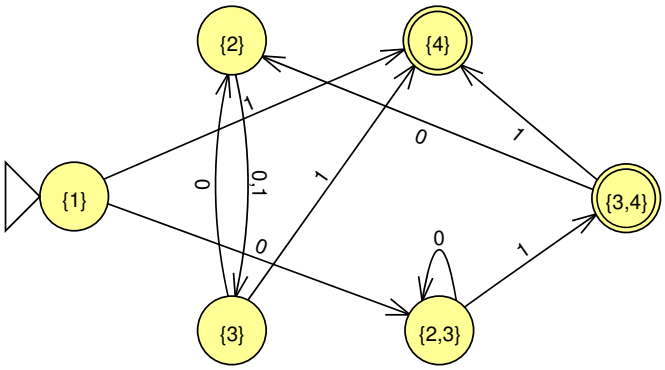

Figura 1.5: Autómato finito determinista correspondente ao autómato representado na figura 1.2

1.3 Minimização de autómatos

Seja A um autómato finito determinista. Diz-se que A é um autómato finito determinista mínimo se não existe outro AFD equivalente com menos estados.

Definição 1.14 (Isomorfismo de autómatos deterministas)**.**

Dois autómatos $A = (Q_A, \Sigma, \delta_A, q_A, F_A)$ e $B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ são isomorfos se existe

uma bijecção $g : Q_A \to Q_B$ tal que:

- se $\delta_A(p, a) = q$ então $\delta_B(g(p), a) = g(q)$;

- $q_B = g(q_A)$;

- $F_B = \{g(f) : f \in F_A\}$.

O isomorfismo de autómatos finitos é uma relação de equivalência na classe dos autómatos finitos deterministas.

Da definição de isomorfismo de autómatos conclui-se que dois autómatos isomorfos são equivalentes, mas dois autómatos equivalentes podem não ser isomorfos.

O autómato mínimo é único a menos de um isomorfismo. A demonstração deste resultado pode ser consultada em [7].

Exemplo 1.6. Na figura 1.7 representa-se o autómato finito determinista mínimo do autómato representado na figura 1.6. O autómato mínimo tem menos estados que o autómato A, mas reconhece a mesma linguagem $L(A)$.

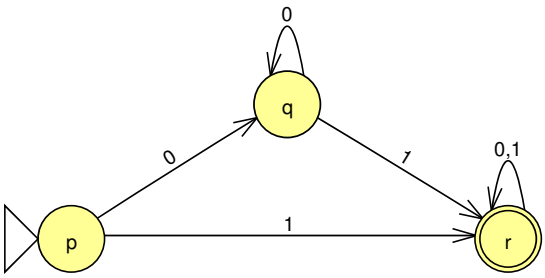

Figura 1.6: Autómato finito determinista que pode ser minimizado

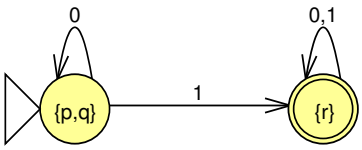

Figura 1.7: Autómato finito determinista mínimo

Quando se converte um AFND num AFD ou um AFND-ε num AFD, não existe a garantia que o autómato obtido seja o AFD mínimo. Assim, é muitas vezes importante utilizar um algoritmo de minimização para se obter o autómato mínimo. Nos capítulos seguintes analisam-se vários métodos para minimizar autómatos finitos.

Capítulo 2

Algoritmos de minimização de autómatos baseados na equivalência de estados

Neste capítulo apresentam-se vários algoritmos de minimização de autómatos que se baseiam na definição de equivalência de estados. Enquanto alguns destes algoritmos constroem as relações de equivalência e/ou de distinguibilidade, os outros constroem as classes de equivalência do autómato inicial.

O autómato mínimo obtém-se a partir das classes de equivalência do autómato inicial, onde cada classe vai corresponder a um estado do autómato mínimo, ou seja, agrupam-se os estados do autómato inicial que são equivalentes, originando um único estado do autómato mínimo.

Este capítulo baseia-se principalmente no artigo [10], o qual faz uma síntese dos diferentes algoritmos de minimização de autómatos. Aqui são apresentados e explicados esses mesmos algoritmos e também são analisadas as suas ordens de complexidade. Em alguns casos também é feita uma análise comparativa entre algoritmos.

Dois estados de um autómato dizem-se indistinguíveis ou equivalentes se, partindo quer de um quer de outro, por leitura de uma qualquer palavra, se atingem ou dois estados finais ou dois estados não finais.

Definição 2.1 (Indistinguibilidade ou equivalência de estados)**.**

Seja $A = (Q, \Sigma, \delta, q_0, F)$ um Autómato Finito Determinista. Dois estados $p, q \in Q$ são

indistinguíveis ($p \equiv q$) se,

$$\forall w \in \Sigma^*,\ \hat{\delta}(p,w) \in F \Leftrightarrow \hat{\delta}(q,w) \in F.$$

De forma análoga, dois estados p e q são equivalentes se têm linguagens direitas idênticas, $L_d(p) = L_d(q)$. De facto, se w é uma palavra de $L_d(p) = \{w \in \Sigma^* \mid \hat{\delta}(p,w) \in F\}$ e se p e q equivalentes então $\hat{\delta}(p,w) \in F$ e $\hat{\delta}(q,w) \in F$. Logo $w \in L_d(q)$.

Dois estados dizem-se distinguíveis se existe pelo menos uma palavra w que os distingue, isto é, partindo de um dos estados por leitura de w atinge-se um estado final enquanto que partindo do outro estado por leitura da mesma palavra w se atinge um estado não final.

Definição 2.2 (Distinguibilidade de estados)**.**
Seja $A = (Q, \Sigma, \delta, q_0, F)$ um Autómato Finito Determinista. Dois estados $p, q \in Q$ são distinguíveis ($p \not\equiv q$) se,

$$\exists w \in \Sigma^* : (\hat{\delta}(p,w) \in F \wedge \hat{\delta}(q,w) \notin F) \vee (\hat{\delta}(p,w) \notin F \wedge \hat{\delta}(q,w) \in F).$$

Da definição de distinguibilidade de estados, conclui-se que todo o estado final é distinguível de todo o estado não final. Note-se ainda que dois estados distinguíveis, p e q, têm linguagens direitas diferentes. De facto, se w é uma palavra que distingue os dois estados então w pertence à linguagem direita de um mas não pertence à linguagem direita de outro. Por exemplo, se $w \in L_d(p) = \{w \in \Sigma^* \mid \hat{\delta}(p,w) \in F\}$ então $\hat{\delta}(p,w) \in F$. Logo, $\hat{\delta}(q,w) \notin F$ e portanto $w \notin L_d(q)$.

Exemplo 2.1. Ilustram-se os conceitos acima definidos com o autómato representado na figura 1.6.

$$\left.\begin{array}{l} \delta(p,0) = q \notin F \wedge \delta(q,0) = q \notin F \\ \delta(p,1) = r \in F \wedge \delta(q,1) = r \in F \end{array}\right\} p \equiv q$$

Pela leitura de cada palavra, os dois estados têm ambos transições para estados finais ou têm ambos transições para estados não finais. Assim, conclui-se que p e q são equivalentes. Como $p \notin F$, $q \notin F$ e $r \in F$, conclui-se que $p \not\equiv r$ e $q \not\equiv r$, isto é, r é distinguível de p e de q. Isto significa que os estados p e q são os únicos estados equivalentes e, como se pode confirmar na figura 1.7, estes estados dão origem a um único estado no autómato mínimo.

No que se segue, denota-se por E a relação $\equiv$ que define a equivalência de dois estados e por D a relação $\not\equiv$ que define a distinguibilidade de dois estados.

Teorema 2.1. A relação E é uma relação de equivalência em Q.

Demonstração.

- E é reflexiva ($\forall p \in Q,\ (p,p) \in E$):

  qualquer estado p é equivalente a si próprio, isto é, para qualquer palavra w, $\hat{\delta}(p,w) \in F \Leftrightarrow \hat{\delta}(p,w) \in F$;

- E é simétrica ($\forall p,q \in Q,\ (p,q) \in E \Rightarrow (q,p) \in E$):

  para qualquer palavra w, se $\hat{\delta}(p,w) \in F \Leftrightarrow \hat{\delta}(q,w) \in F$ então $\hat{\delta}(q,w) \in F \Leftrightarrow \hat{\delta}(p,w) \in F$;

- E é transitiva ($\forall p,q,r \in Q,\ (p,q) \in E$ e $(q,r) \in E$ então $(p,r) \in E$):

  se $(p,q) \in E$ e $(q,r) \in E$, então pela definição de E para qualquer palavra w, $\hat{\delta}(p,w) \in F \Leftrightarrow \hat{\delta}(q,w) \in F$ e $\hat{\delta}(q,w) \in F \Leftrightarrow \hat{\delta}(r,w) \in F$. Ou seja, $\hat{\delta}(p,w) \in F \Leftrightarrow \hat{\delta}(q,w) \in F \Leftrightarrow \hat{\delta}(r,w) \in F$.

□

Note-se que a relação de distinguibilidade D não é uma relação de equivalência, embora seja simétrica.

As relações E e D podem ser determinadas pelos processos recorrentes contidos nas definições a seguir apresentadas.

Definição 2.3 (Relação E_k)**.**
Para $k = 0$, define-se $E_0 = (Q \setminus F)^2 \cup F^2$. Para $k > 0$, dados dois quaisquer estados p e q, tem-se que

$$(p,q) \in E_k \text{ se e só se } (p,q) \in E_{k-1} \text{ e } \forall a \in \Sigma,\ (\delta(p,a), \delta(q,a)) \in E_{k-1}.$$

Definição 2.4 (Relação D_k)**.**
Para $k = 0$, define-se $D_0 = ((Q \setminus F) \times F) \cup (F \times (Q \setminus F))$. Para $k > 0$, dados dois quaisquer

estados p e q, tem-se que

$$(p,q) \in D_k \text{ se } (p,q) \in D_{k-1} \text{ ou } \exists a \in \Sigma \text{ tal que } (\delta(p,a), \delta(q,a)) \in D_{k-1}.$$

Definição 2.5 (Ponto fixo de uma sequência de conjuntos)**.**

Seja $(X_k)_{k \in \mathbb{N}_0}$ uma sequência de conjuntos. Um ponto fixo de (X_k) é um elemento X_s, $s \in \mathbb{N}_0$, tal que $X_{s+1} = X_s$.

Pela definição 2.3 verifica-se que, para $k > 0$, $E_k \subseteq E_{k-1}$. Assim, existe s tal que $E_{s+1} = E_s$, isto é, E_s é um ponto fixo da sequência (E_k) e como será mostrado na proposição 2.2, $E_s = E$.

Da definição 2.4 tem-se que, para $k > 0$, $D_{k-1} \subseteq D_k \subseteq Q^2$. Assim, existe r tal que $D_{r+1} = D_r$ e neste caso, também se mostrará na proposição 2.2 que $D_r = D$.

Uma consequência directa das definições anteriores é o seguinte resultado auxiliar.

Lema 2.1.

Para todo $k \geq 0$ E_k e D_k são complementares, isto é, $E_k \cap D_k = \emptyset$ e $E_k \cup D_k = Q^2$.

Com base neste lema e na definição de ponto fixo, obtém-se a proposição seguinte, que relaciona os pontos fixos das sequências (E_k) e (D_k).

Proposição 2.1.

Sejam E_s e D_r respectivamente, os pontos fixos das sequências (E_k) e (D_k). Então $s = r$ e portanto $D_r = Q^2 \setminus E_s$ e $E_s = Q^2 \setminus D_r$.

Demonstração.

Por definição de ponto fixo, tem-se que $E_{s+1} = E_s$ e pelo lema 2.1, tem-se que

$$E_{s+1} = E_s \Leftrightarrow Q^2 \setminus D_{s+1} = Q^2 \setminus D_s \Leftrightarrow D_{s+1} = D_s.$$

Daqui conclui-se que $s \geq r$, sendo D_r o ponto fixo de D_k.

De forma análoga, tem-se que

$$D_{r+1} = D_r \Leftrightarrow Q^2 \setminus E_{r+1} = Q^2 \setminus E_r \Leftrightarrow E_{r+1} = E_r.$$

Ou seja, $r \geq s$ onde E_s é o ponto fixo de E_k. Portanto, tem-se que $s = r$.

Pelo lema 2.1, tem-se que $D_r = Q^2 \setminus E_r$. Mas como se provou que r é igual a s, então $D_r = Q^2 \setminus E_s$ e também $E_s = Q^2 \setminus D_r$.

□

O resultado seguinte serve para demonstrar que as definições 2.1 e 2.3 são equivalentes. Este resultado é usado na proposição 2.2 onde se mostra que E é o ponto fixo de (E_k).

Lema 2.2. $(p, q) \in E_k$ se e só se $\forall w \in \Sigma^*$, $|w| \leq k$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$.

Demonstração.

Seja $(p, q) \in E_0$. Pela definição de E_0 tem-se que $(p, q) \in F^2$ ou $(p, q) \in (Q \setminus F)^2$, isto é, $p \in F \Leftrightarrow q \in F$. Como $\delta(p, \varepsilon) = p$ e $\delta(q, \varepsilon) = q$ então $\delta(p, \varepsilon) \in F \Leftrightarrow \delta(q, \varepsilon) \in F$. Ou seja, $\forall w \in \Sigma^*$, $|w| \leq 0$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$.

Considere-se agora que $\forall w \in \Sigma^*$, $|w| \leq 0$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$. Daqui sai que $\delta(p, \varepsilon) \in F \Leftrightarrow \delta(q, \varepsilon) \in F$ e, como $\delta(p, \varepsilon) = p$ e $\delta(q, \varepsilon) = q$, então $p \in F \Leftrightarrow q \in F$. Portanto, pela definição de E_0, tem-se que $(p, q) \in E_0$.

Considere-se por hipótese indução que $(p, q) \in E_k$ se e só se $\forall w \in \Sigma^*$, $|w| \leq k$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$.

Seja $(p, q) \in E_{k+1}$. Pela definição de E_{k+1} tem-se que $\forall a \in \Sigma$, $(\delta(p, a), \delta(q, a)) \in E_k$. Assim, pela hipótese de indução, tem-se que $\forall z \in \Sigma^*$, $|z| \leq k$, $\hat{\delta}(\delta(p, a), z) \in F \Leftrightarrow \hat{\delta}(\delta(q, a), z) \in F$. Logo, $\forall a \in \Sigma$, $\forall z \in \Sigma^*$, $|z| \leq k$, $\hat{\delta}(p, az) \in F \Leftrightarrow \hat{\delta}(q, az) \in F$. Considerando $w = az$, conclui-se que $\forall w \in \Sigma^*$, $|w| \leq k + 1$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$.

Considere-se agora que $\forall w \in \Sigma^*$, $|w| \leq k + 1$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$. Em particular, tem-se que $\forall w \in \Sigma^*$, $|w| \leq k$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$ e portanto, pela hipótese de indução, conclui-se que $(p, q) \in E_k$.

Seja $a \in \Sigma$ e $z \in \Sigma^*$ tal que $|z| \leq k$. Então $w = az$ é tal que $|w| \leq k + 1$. Logo $\hat{\delta}(p, az) \in F \Leftrightarrow \hat{\delta}(q, az) \in F$. Daqui sai que $\hat{\delta}(\delta(p, a), z) \in F \Leftrightarrow \hat{\delta}(\delta(q, a), z) \in F$. Pela hipótese de indução conclui-se que $(\delta(p, a), \delta(q, a)) \in E_k$.

Como $(p, q) \in E_k$ e $\forall a \in \Sigma$ $(\delta(p, a), \delta(q, a)) \in E_k$, então conclui-se que $(p, q) \in E_{k+1}$.

□

A partir deste lema e pela definição de ponto fixo, mostra-se que os pontos fixos das sequências (E_k) e (D_k) correspondem às relações E e D, respectivamente. Este resultado

será utilizado nos algoritmos que determinam E e/ou D a partir das sequências (E_k) e/ou (D_k). Nesses algoritmos, em cada iteração determinam-se as relações E_k e/ou D_k, onde k corresponde ao número da iteração, até que se chega ao ponto fixo, obtendo-se dessa forma as relações E e/ou D.

Proposição 2.2. Sejam E_s e D_s os pontos fixos das sequências (E_k) e (D_k), respectivamente. Então $E_s = E$ e $D_s = D$.

Demonstração.

Seja $(p, q) \in E$. Por definição de E, tem-se que $\forall w \in \Sigma^*$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$. Em particular para w tal que $|w| \leq s$ tem-se também que $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$. Portanto $(p, q) \in E_s$, e consequentemente $E \subseteq E_s$.

Seja $(p, q) \in E_s$. Pelo lema 2.2, tem-se $\forall w \in \Sigma^*$, $|w| \leq s$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$. Seja agora $t > s$. Como s é ponto fixo de (E_k), $E_t = E_s$. Logo $(p, q) \in E_t$ e portanto $\forall w \in \Sigma^*$, $|w| \leq t$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$. Assim, $\forall w \in \Sigma^*$, $\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$, ou seja, $E_s \subseteq E$.

□

A *classe de equivalência* de um estado q por uma relação R de equivalência no conjunto Q é definida por

$$[q]_R = \{p \in Q : (p, q) \in R\}.$$

Denota-se por $[Q]_R$ o *conjunto das classes de equivalência* da relação de equivalência R no conjunto Q. Isto é, $[Q]_R = \{[q]_R : q \in Q\}$.

Por definição de E_k tem-se $E_0 \supset E_1 \supset ... \supset E_s \supseteq I_Q$, onde I_Q é a relação identidade de estados e E_s é o ponto fixo de E_k. Portanto, $|[Q]_{E_0}| < |[Q]_{E_1}| < ... < |[Q]_{E_s}| \leq |[Q]_{I_Q}|$, onde $|[Q]_{I_Q}| = |Q|$. Também pela definição de E_k, $|[Q]_{E_0}| \leq 2$.

Quando $|[Q]_{E_0}| < 2$, tem-se que os estados são todos simultaneamente finais e não finais. Por esta razão E_0 é o ponto fixo de E_k, ou seja, corresponde a E.

Quando $|[Q]_{E_0}| = 2$, tem-se $i + 2 \leq |[Q]_{E_i}|$, para qualquer i. Logo, para qualquer s, $s + 2 \leq |[Q]_{E_s}| \leq |[Q]_{I_Q}| = |Q|$, ou seja, $s \leq |Q| - 2$. Daqui conclui-se que são necessários no máximo $|Q| - 2$ iterações para determinar a relação E, isto é, $E_{|Q|-2} = E$.

Os algoritmos seguintes consideram originalmente pares ordenados de estados, mas como as relações E_k e D_k, para $k \geq 0$, são simétricas, então os exemplos destes algoritmos são apresentados utilizando pares não ordenados de estados.

2.1 Cálculo de D e E através das sequências (D_k) e (E_k)

Os algoritmos apresentados a seguir calculam as sequências (E_k) e (D_k), até serem atingidos os pontos fixos, os quais correspondem, respectivamente, às relações de equivalência e distinguibilidade.

O algoritmo seguinte calcula os estados que são distinguíveis, isto é, a relação de distinguibilidade, D.

Algoritmo 2.1.

$G := D_0$

$G' := \emptyset$

while $G \neq G'$ do

$G' := G$

$G := \{(p,q) \in G' \vee \exists a \in \Sigma : (\delta(p,a), \delta(q,a)) \in G'\}$

Neste algoritmo, parte-se de $G = D_0 = ((Q \setminus F) \times F)$ que corresponde ao conjunto formado pelos pares de estados que se sabe à partida que são distinguíveis. Em cada iteração determina-se os pares que ainda não pertencem a G mas que pela leitura de uma dada letra têm transição para estados de G e inserem-se neste mesmo conjunto. Os estados que pertencentem a G são distinguíveis. Portanto, como ao longo do algoritmo se acrescentam estados ao conjunto G, é por esta razão que G' é inicializado a $\emptyset$.

O algoritmo que se segue é semelhante ao anterior, mas em vez de determinar os estados distinguíveis, determina os equivalentes. Por outras palavras, o resultado final vai ser a relação E.

Algoritmo 2.2.

$H := E_0$

$H' := Q \times Q$

while $H \neq H'$ do

$H' := H$

$H := \{(p,q) \in H' \wedge \forall a \in \Sigma : (\delta(p,a), \delta(q,a)) \in H'\}$

Inicialmente, parte-se de $H = E_0 = (Q \setminus F)^2 \cup F^2$ que é o conjunto de todos os estados que podem ser equivalentes. Em cada iteração determina-se se existem pares de estados pertencentes a H, cujas transições são todas para elementos de H. Caso isto não se verifique, estes estados são excluidos deste conjunto. Portanto, como ao longo do algoritmo se excluem estados ao conjunto H, é por esta razão que H' é inicializado a $Q \times Q$.

Tem-se que $|H| \leq |Q|^2$, logo em cada iteração do ciclo while analisam-se no máximo $|Q|^2$ elementos. Como se viu anteriormente, são necessárias no máximo $|Q| - 2$ iterações para se determinar a relação E, isto é, o ciclo while é executado no máximo $|Q| - 2$ vezes. Assim, a complexidade deste algoritmo (bem como a do algoritmo 2.1) é $O(|Q|^2(|Q| - 2)) = O(|Q|^3)$.

O algoritmo seguinte calcula simultaneamente os estados distinguíveis e os equivalentes. Pode ser dividido em dois, um para calcular só E e outro para calcular só D.

Algoritmo 2.3.

$G := D_0$

$H := E_0$

$G' := \emptyset$

$H' := Q \times Q$

while $G \neq G'$ do

    $G' := G$

    $H' := H$

    for $(p, q) \in H'$ do

        if $(\exists a \in \Sigma : (\delta(p, a), \delta(q, a)) \in G')$ then

            $G := G \cup \{(p, q)\}$

            $H := H \setminus \{(p, q)\}$

A diferença entre este último algoritmo e o algoritmo 2.2 está na condição if. No algoritmo 2.2 verifica-se se todas as transições, a partir de todos os elementos de H' são para elementos de H'. No algoritmo 2.3 o ciclo for garante que também se analisam todos os elementos de H', contudo no if para cada elemento de H' verifica-se se tem alguma transição para algum elemento que não pertença a H'. Se esta condição se verifica, então já não analisa as restantes transições deste elemento de H'.
Como G e H são complementares, quando um dos conjuntos se mantém inalterado, o outro conjunto também se mantém igual. Logo poder-se-ia considerar o conjunto H no ciclo while que o resultado seria o mesmo.

Como $|H| \leq |Q|^2$, então o ciclo for é executado, no máximo, $|Q|^2$ vezes. Uma vez mais, como se viu anteriormente, são necessárias no máximo $|Q| - 2$ iterações para se determinar a relação E. Logo o ciclo while é executado no máximo $|Q| - 2$ vezes e assim, a ordem de complexidade deste algoritmo é a mesma dos algoritmos 2.1 e 2.2, isto é, $O(|Q|^2(|Q| - 2)) = O(|Q|^3)$.

Exemplo 2.2. Considere-se o autómato finito determinista

$$A = (\{1,2,3,4,5,6\}, \{0,1\}, \delta, \{1\}, \{6\})$$

onde $\delta(1,0) = 3$, $\delta(1,1) = 5$, $\delta(2,0) = 1$, $\delta(2,1) = 3$, $\delta(3,0) = 4$, $\delta(3,1) = 3$, $\delta(4,0) = 3$, $\delta(4,1) = 5$, $\delta(5,0) = 6$, $\delta(5,1) = 2$, $\delta(6,0) = 6$ e $\delta(6,1) = 6$.

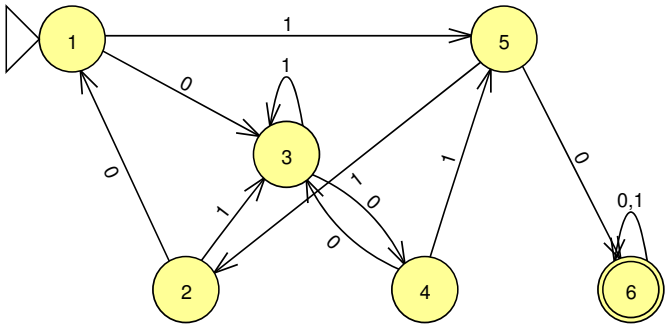

Figura 2.1: Autómato finito determinista

Este autómato, representado na figura 2.1, foi minimizado utilizando os 3 algoritmos anteriores.

No algoritmo 2.1, após três iterações obteve-se

$$G = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},\{3,5\},\{4,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}.$$

Portanto, $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $4 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$ e $3 \not\equiv 4$.

Da mesma forma no algoritmo 2.2, após três iterações obteve-se

$$H = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},\{5,5\},\{6,6\}\}.$$

Portanto, $1 \equiv 4$ e $2 \equiv 3$.

No algoritmo 2.3 também foram precisas três iterações para calcular os mesmos conjuntos

G e H.

Os resultados detalhados encontram-se na página 55 do apêndice.

2.2 Cálculo de D, E e $[Q]_E$ sem determinar as sequências (D_k) e (E_k)

O algoritmo seguinte também calcula as relações de equivalência e de distinguibilidade de estados. Difere dos algoritmos apresentados na secção 2.1 porque não calcula as sequências (E_k) e (D_k), determinando os conjuntos G e H por analise de um só par de estados em cada iteração.

Este algoritmo pode ser dividido em dois, num só se determina E e noutro só D.

Algoritmo 2.4.

$G := D_0$

$H := E_0$

while $(\exists a \in \Sigma \text{ e } (p,q) \in H : (\delta(p,a), \delta(q,a)) \in G)$ do

$G := G \cup \{(p,q)\}$

$H := H \setminus \{(p,q)\}$

Em cada iteração não há garantia de H e G serem relações de equivalência. De facto, neste algoritmo só se analisa um par de estados por iteração, podendo haver mais pares que pertencem a H e que têm transições para G. Para que H e G fossem relações de equivalência era preciso que, em cada iteração, fossem identificados todos os estados que pertencem a H e que transitam para G.

Uma das diferenças entre este algoritmo e o algoritmo 2.3 é a condição que é testada no ciclo while. No algoritmo 2.3 enquanto o conjunto G for alterado, o ciclo while é executado. Isto significa que após se determinar o conjunto G final, o ciclo while é executado uma vez mais. No algoritmo 2.4, quando se chega a este ponto, não se executam mais iterações do ciclo while porque os elementos de um dos conjuntos não têm transições para elementos do outro conjunto.

O conjunto H tem no máximo $|Q|^2$ elementos. Como o ciclo while analisa todos os elementos de H que têm transições para elementos de G, realiza no máximo executa $|Q|^2$ iterações. Portanto a complexidade deste algoritmo é $O(|Q|^2)$.

O algoritmo seguinte determina as classes de equivalência da relação de equivalência, $[Q]_E$. Este algoritmo utiliza o predicado

$$Divide(Q_0, Q_1, a) \equiv (\exists p, q \in Q_0 : (\delta(p,a) \in Q_1 \wedge \delta(q,a) \notin Q_1))$$

o qual determina se existem estados pertencentes ao conjunto Q_0 que têm transições para estados do conjunto Q_1 por leitura da letra a.

Algoritmo 2.5.

$P := [Q]_{E_0}$

while $(\exists Q_0, Q_1 \in P, \exists a \in \Sigma : Divide(Q_0, Q_1, a))$ do

$Q'_0 := \{p \in Q_0 : \delta(p, a) \in Q_1\}$

$P := P \setminus \{Q_0\} \cup \{Q_0 \setminus Q'_0, Q'_0\}$

No início deste algoritmo o conjunto P é formado pelas duas classes de equivalência de E_0: $Q \setminus F$ e F.

Em cada passo do ciclo identificam-se os estados que pertencem a um elemento de P e que pela leitura de uma dada letra têm transições para estados pertencentes a outro elemento de P. Isto significa que estes estados são distinguíveis dos restantes elementos do mesmo conjunto. Por exemplo, no primeiro passo são 'retirados' do conjunto $Q \setminus F$, os estados que pela leitura de uma dada letra transitam para um estado final. Ou seja, estes estados são equivalentes entre si e distinguíveis dos restantes elementos do conjunto $Q \setminus F$.

O conjunto P tem no máximo $|Q|$ elementos. No algoritmo testa-se se cada elemento de P tem transições para cada um dos restantes elementos de P. Existem no máximo $|Q^2| - |Q|$ possibilidades de tal ocorrer, isto é, o ciclo é executado no máximo $|Q^2| - |Q|$ vezes. Assim, a complexidade deste algoritmo é $O(|Q|^2)$.

Exemplo 2.3. Utilizam-se os algoritmos 2.4 e 2.5 para minimizar o autómato finito determinista representado na figura 2.1.

No fim da execução do algoritmo 2.4, obtiveram-se os seguintes resultados:

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

e

$$H = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},\{5,5\},\{6,6\}\}.$$

Portanto, conclui-se que $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$, $3 \not\equiv 4$ e $4 \not\equiv 5$. Também se conclui que $1 \equiv 4$ e $2 \equiv 3$.

Os resultados detalhados da execução deste algoritmo podem ser consultados na página 58.

Ilustra-se em seguida a execução do algoritmo 2.5.

$P = [Q]_{E_0} = \{Q \setminus F, F\} = \{\{1,2,3,4,5\},\{6\}\}$

1ª iteração:

$Q_0 = \{1,2,3,4,5\} \quad Q_1 = \{6\} \quad a = 0$

$Q'_0 = \{5\}$

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

2ª iteração:

$Q_0 = \{1,2,3,4\} \quad Q_1 = \{5\} \quad a = 1$

$Q'_0 = \{1,4\}$

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

Portanto, tem-se quatro classes de equivalência: $\{6\}$, $\{5\}$, $\{2,3\}$ e $\{1,4\}$. Daqui conclui-se que $1 \equiv 4$ e $2 \equiv 3$.

2.3 Cálculo mais eficiente de D e E sem determinar as sequências (D_k) e (E_k)

O algoritmo descrito em seguida também calcula as relações de equivalência e de distinguibilidade de estados. Tal como o algoritmo 2.4, também não calcula as sequências E_k e D_k, contudo é mais eficiente que aquele porque no ciclo while identifica pares de estados que pertencem a H, (p,q), e que têm transições para estados de G. Dentro deste ciclo existe um ciclo for que analisa todos os elementos de H em que um dos seus elementos é p. Assim , numa iteração do ciclo while podem ser excluídos vários elementos do conjunto H. É por esta razão que o ciclo while é executado menos vezes do que o do algoritmo 2.4.

Este algoritmo pode também ser dividido em dois, isto é, um para determinar D e outro para determinar E.

Algoritmo 2.6.

$G := D_0$

$H := E_0$

while $(\exists a \in \Sigma \text{ e } (p,q) \in H : (\delta(p,a), \delta(q,a)) \in G)$ do

    for $q : (p,q) \in H \wedge (\delta(p,a), \delta(q,a)) \in G$ do

        $G := G \cup \{(p,q)\}$

        $H := H \setminus \{(p,q)\}$

Tal como no algoritmo 2.4, no fim de cada iteração não há garantia de H e G serem relações de equivalência. Isto deve-se ao facto de em cada iteração só se analisar um par de estados, podendo haver mais pares que pertencem a H e que têm transições para G. Para que H e G fossem relações de equivalência era preciso que em cada iteração fossem identificados todos os estados que pertencem a H e que têm transições para G.

O conjunto H tem no máximo $|Q|^2$ elementos. Por isso o ciclo for é executado no máximo $|Q|$ vezes. De forma análoga, o ciclo while é executado no máximo $|Q|$ vezes, pois as transições dos restantes elementos de H foram analisadas no ciclo for. Assim, a complexidade deste algoritmo é $O(|Q|^2)$.

O algoritmo seguinte calcula as classes de equivalência da relação de equivalência, à semelhança do algoritmo 2.5.

Algoritmo 2.7.

$P := [Q]_{E_0}$

while $(\exists Q_1 \in P, \exists a \in \Sigma : (\exists Q_0 \in P : Divide(Q_0, Q_1, a)))$ do

    $P' := P$

    for $Q_0 : Q_0 \in P' \wedge Divide(Q_0, Q_1, a)$ do

        $Q_0' := \{p \in Q_0 \wedge \delta(p,a) \in Q_1\}$

        $P := P \setminus \{Q_0\} \cup \{Q_0 \setminus Q_0', Q_0'\}$

No ciclo while identificam-se dois conjuntos, Q_0 e Q_1 com a seguinte propriedade: existe uma letra a tal que em Q_0 existem estados com transições por a para estados de Q_1 e existem também estados com transições por a para estados fora de Q_1.

O ciclo for analisa todos os conjuntos de P que pela leitura de a têm transições para elementos de Q_1. Por esta razão o conjunto Q_1 é analisado numa única iteração do ciclo while, enquanto que no algoritmo 2.5 o conjunto Q_1 pode ser analisado em tantas iterações do ciclo while quantos os elementos do conjunto P.

O conjunto P tem no máximo $|Q|$ elementos, logo o ciclo for é executado no máximo $|Q|$ vezes. Do mesmo modo, o ciclo while é executado no máximo $|Q|$ vezes, uma vez que as

restantes transições foram analisadas no ciclo for. Portanto a complexidade deste algoritmo é $O(|Q|^2)$.

Exemplo 2.4. Utilizam-se os algoritmos 2.6 e 2.7 para minimizar o autómato finito determinista representado na figura 2.1.

No algoritmo 2.6, após sete iterações do ciclo while, como se pode observar na página 60 do apêndice, obteve-se

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

e

$$H = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},\{5,5\},\{6,6\}\}.$$

Portanto, conclui-se que $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$, $3 \not\equiv 4$, $4 \not\equiv 5$ e que $1 \equiv 4$ e $2 \equiv 3$.

Os resultados da execução do algoritmo 2.7 são os seguintes.

$P = [Q]_{E_0} = \{Q \setminus F, F\} = \{\{1,2,3,4,5\},\{6\}\}$

1ª iteração (while):

$Q_0 = \{1,2,3,4,5\} \quad Q_1 = \{6\} \quad a = 0$

$P' = P = \{\{1,2,3,4,5\},\{6\}\}$

1ª iteração (for):

$Q_0 = \{1,2,3,4,5\}$

$Q_0' = \{5\}$

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

2ª iteração (while):

$Q_0 = \{1,2,3,4\} \quad Q_1 = \{5\} \quad a = 1$

$P' = P = \{\{6\},\{1,2,3,4\},\{5\}\}$

1ª iteração (for):

$Q_0 = \{1, 2, 3, 4\}$

$Q'_0 = \{1, 4\}$

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

Obtiveram-se quatro classes de equivalência: $\{6\}$, $\{5\}$, $\{2, 3\}$ e $\{1, 4\}$, ou seja $1 \equiv 4$ e $2 \equiv 3$.

2.4 Algoritmo de Hopcroft e Ullman

O algoritmo seguinte também calcula a relação de distinguibilidade, D, sem calcular a sequência (D_k). Bruce Watson, em [10], atribui a sua autoria a Hopcroft e Ullman, uma vez que esse algoritmo está em [7].

Algoritmo 2.8.

for $(p, q) \in Q \times Q$ do

    $L(p, q) := \emptyset$

$G := D_0$

for $(p, q) : (p, q) \notin D_0$ do

    if $(\exists a \in \Sigma : (\delta(p, a), \delta(q, a)) \in G)$ then

        $A := \{(p, q)\}$

        $B := \emptyset$

        while $A \neq \emptyset$ do

            seja $(r, s) : (r, s) \in A$

            $G := G \cup \{(r, s)\}$

            $A := A \setminus \{(r, s)\}$

            $B := B \cup \{(r, s)\}$

            $A := A \cup (L(r, s) \setminus B)$

    else

        for $a \in \Sigma : \delta(p, a) \neq \delta(q, a)$ do

            $L(\delta(p, a), \delta(q, a)) := L(\delta(p, a), \delta(q, a)) \cup \{(p, q)\}$

Este algoritmo, à semelhança do algoritmo 2.6, começa por analisar os elementos que não pertencem a D_0 mas que têm transições para elementos de G, passando estes elementos também a pertencer a G. Contudo, aqui também se inserem em G os pares de estados que têm transições para o par pertencente a D_0 que se está a analisar. Isto acontece porque os estados que têm transições para estados distinguíveis, também são distinguíveis.

Ao contrário dos algoritmos anteriores, neste os elementos que não pertencem a D_0 e que não têm transições para elementos de G são guardados em conjuntos auxiliares associados aos estados para os quais têm transições. De facto se estes últimos forem distinguíveis, todos os estados que pertencem ao seu conjunto auxiliar também vão ser distinguíveis.

É possível alterar o algoritmo para calcular E.

O primeiro ciclo for é executado $|Q|^2$ vezes. O segundo ciclo for é executado para todos os pares de estados que não pertencem a D_0, ou seja para todos os pares de estados pertencentes a E_0. Uma vez que $|E_0| \leq |Q|^2$, este ciclo vai ser executado no máximo $|Q|^2$ vezes. O último ciclo for é executado $|\Sigma|$ vezes. Portanto a complexidade deste algoritmo é $O(|Q|^2+|Q|^2|\Sigma|) = O(|Q|^2)$.

Exemplo 2.5. Utiliza-se o algoritmo 2.8 para minimizar o autómato finito determinista representado na figura 2.1.

No fim da execução deste algoritmo, como se pode observar na página 62 do apêndice, obteve-se

$$G = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},\{3,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\},\{4,5\}\}.$$

Daqui conclui-se que $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$, $3 \not\equiv 4$ e $4 \not\equiv 5$.

2.5 Algoritmo de Hopcroft

O algoritmo de Hopcroft também determina as classes de equivalência da relação de equivalência. Este algoritmo tem a melhor ordem de complexidade dos algoritmos de minimização de autómatos. No entanto, a sua demonstração é complexa. Em [5], este algoritmo é descrito e demonstrado de forma pormenorizada.

Neste algoritmo parte-se do algoritmo 2.7, tendo em consideração que se todas as classes de equivalência já foram divididas em relação a um par (Q_1, a) então mais nenhuma classe de equivalência que venha a ser determinada, precisa de ser dividida em relação a esse mesmo par.

Algoritmo 2.9.

$P := [Q]_{E_0}$

$L := P \times \Sigma$

while $L \neq \emptyset$ do

seja $(Q_1, a) \in L$

$P' := P$

$L := L \setminus \{(Q_1, a)\}$

for $Q_0 \in P' \wedge Divide(Q_0, Q_1, a)$ do

$Q'_0 := \{p \in Q_0 \wedge \delta(p, a) \in Q_1\}$

$P := P \setminus \{Q_0\} \cup \{Q_0 \setminus Q'_0, Q'_0\}$

for $b \in \Sigma$ do

if $(Q_0, b) \in L$ then

$L := L \setminus \{(Q_0, b)\} \cup \{(Q'_0, b), (Q_0 \setminus Q'_0, b)\}$

else

$L := L \cup \{(Q'_0, b), (Q_0 \setminus Q'_0, b)\}$

Neste algoritmo o ciclo while é executado para todos os elementos de L, invocando para cada elemento uma só vez a função *Divide*. Como se referiu anteriormente, se todas as classes de equivalência foram divididas em relação a um dado par (Q_1, a), então não é preciso voltar a invocar a função $Divide(Q_0, Q_1, a)$, para qualquer Q_0. Contudo, no algoritmo 2.7 sempre que existe um conjunto, Q_0, que por leitura de uma dada letra, a, tem transições para elementos de Q_1, então invoca-se a função $Divide(Q_0, Q_1, a)$, para todos os $Q_0 \in P$.

O segundo ciclo for tem como objectivo actualizar o conjunto auxiliar L.

Segundo Hopcroft, dividir uma classe de equivalência em relação a quaisquer dois conjuntos dos seguintes - (Q_0, b), (Q'_0, b) e $(Q_0 \setminus Q'_0, b)$. É o mesmo que dividir essa classe de equivalência em relação aos três conjuntos, como é demonstrado em [10].

Utilizando este resultado e para melhorar o algoritmo, altera-se a actualização do conjunto L:

- se $(Q_0, b) \in L$ então as classes de equivalência ainda não foram divididas em relação a este par. Por isso, retira-se o elemento (Q_0, b) e adicionam-se os elementos (Q'_0, b) e $(Q_0 \setminus Q'_0, b)$ a L.

- se $(Q_0, b) \notin L$ então as classes de equivalência já foram divididas em relação a este par. Portanto, basta dividir em relação ou a (Q'_0, b) ou a $(Q_0 \setminus Q'_0, b)$, por isso escolhe-se o menor conjunto.

No início do algoritmo tem-se $P = [Q]_{E_0} = \{Q \setminus F, F\}$, isto significa que Q já foi dividido, assim é apenas necessário dividir as classes de equivalência em relação a apenas um dos conjuntos. Neste caso também se escolhe o conjunto mais pequeno.

Aplicando estas alterações ao algoritmo 2.9, obtém-se o algoritmo de Hopcroft.

Algoritmo 2.10. (Hopcroft)

$P := [Q]_{E_0}$

if $|F| \leq |Q \setminus F|$ then

    $L := \{F\} \times \Sigma$

else

    $L := \{Q \setminus F\} \times \Sigma$

while $L \neq \emptyset$ do

    seja $(Q_1, a) \in L$

    $P' := P$

    $L := L \setminus \{(Q_1, a)\}$

    for $Q_0 \in P' \wedge Divide(Q_0, Q_1, a)$ do

        $Q_0' := \{p \in Q_0 \wedge \delta(p, a) \in Q_1\}$

        $P := P \setminus \{Q_0\} \cup \{Q_0 \setminus Q_0', Q_0'\}$

        for $b \in \Sigma$ do

            if $(Q_0, b) \in L$ then

                $L := L \setminus \{(Q_0, b)\} \cup \{(Q_0', b), (Q_0 \setminus Q_0', b)\}$

            else

                if $|Q_0'| \leq |Q_0 \setminus Q_0'|$ then

                    $L := L \cup \{(Q_0', b)\}$

                else

                    $L := L \cup \{(Q_0 \setminus Q_0', b)\}$

O algoritmo de Hopcroft foi originalmente apresentado em [6] e definido usando a seguinte notação:

Algoritmo 2.11. (Hopcroft)

Passo 1: $\forall s \in Q$ e $a \in \Sigma$ construir

$$\delta^{-1}(s,a) = \{t \mid \delta(t,a) = s\}.$$

Passo 2: Construir $B(1) = F$, $B(2) = Q - F$ e para cada $a \in \Sigma$ e $1 \leq i \leq 2$ construir

$$a(i) = \{s \mid s \in B(i) \text{ e } \delta^{-1}(s,a) \neq \emptyset\}.$$

Passo 3: $k = 3$

Passo 4: $\forall a \in \Sigma$ construir

$$L(a) = \begin{cases} \{1\} & \text{se } |a(1)| \leq |a(2)| \\ \{2\} & \text{caso contrário} \end{cases}$$

Passo 5: Seleccionar $a \in \Sigma$ e $i \in L(a)$. O algoritmo termina quando $L(a) = \emptyset$ para cada $a \in \Sigma$.

Passo 6: Apagar i de $L(a)$.

Passo 7: $\forall j < k$ t.q. $\exists t \in B(j)$ com $\delta(t,a) \in a(i)$ executar os passos 7a, 7b, 7c e 7d.

Passo 7a: Dividir $B(j)$ em $B'(j) = \{t \mid \delta(t,a) \in a(i)\}$ e $B''(j) = B(j) - B'(j)$.

Passo 7b: Substituir $B(j)$ por $B'(j)$ e construir $B(k) = B''(j)$.

Construir correspondentes $a(j)$ e $a(k)$ para cada $a \in \Sigma$.

Passo 7c: $\forall a \in \Sigma$ modificar $L(a)$ como se segue.

$$L(a) = \begin{cases} L(a) \cup \{j\} & \text{se } j \notin L(a) \text{ e } 0 < |a(j)| \leq |a(k)| \\ L(a) \cup \{k\} & \text{caso contrário} \end{cases}$$

Passo 7d: $k = k + 1$

Passo 8: Voltar ao passo 5.

Paralelismo entre os algoritmos 2.10 e 2.11

Em ambos os algoritmos é criado um conjunto auxiliar L. Através deste conjunto consegue-se identificar quais são os subconjuntos de Q que ainda não foram analisados segundo uma determinada letra do alfabeto.

No algoritmo 2.10, L é um conjunto onde cada elemento é um par constituído por um subconjunto de Q e uma letra do alfabeto. Inicialmente, o subconjunto de Q que pertence a L é a classe de equivalência de E_0 que tiver menor cardinalidade.

No algoritmo 2.11, determina-se o inverso da função de transição para todas as transições. De seguida, é determinado o conjunto $a(i)$ para todo a pertencente ao alfabeto e i pode tomar o valor 1 ou 2. Este conjunto $a(i)$ vai conter os estados pertencentes a $B(i)$ que têm transições segundo a letra a. É a partir destes conjuntos $a(i)$ que se cria o conjunto L. Existe um conjunto L por cada letra do alfabeto, $L(a)$. Cada um destes conjuntos tem como elementos números inteiros que correspondem aos índices dos conjuntos B que ainda têm que ser analisados segundo as transições por a. Estes conjuntos B são subconjuntos de Q.

No algoritmo 2.10, tem-se um conjunto P cujos elementos são subconjuntos de Q. No fim do algoritmo, os elementos de P vão ser as classes de equivalência da relação de equivalência.

No algoritmo 2.11, tem-se vários conjuntos B que são subconjuntos de Q. No fim do algoritmo, estes conjuntos B serão as classes de equivalência da relação de equivalência.

A complexidade deste algoritmo é $O(|Q| \log |Q|)$, como se pode verificar em [6].

Exemplo 2.6. Utilizam-se os algoritmos 2.9 e 2.10 para minimizar o autómato finito determinista representado na figura 2.1.

No algoritmo 2.9, após onze iterações do ciclo while, como se pode observar na página 66 do apêndice, obteve-se $P = \{\{6\}, \{5\}, \{1,4\}, \{2,3\}\}$, ou seja tem-se quatro classes de equivalência: $\{6\}$, $\{5\}$, $\{1,4\}$ e $\{2,3\}$. Logo conclui-se que $1 \equiv 4$ e $2 \equiv 3$.

No algoritmo 2.10, após seis iterações do ciclo while, como também se pode observar no apêndice, na página 69, obteve-se $P = \{\{6\}, \{5\}, \{2,3\}, \{1,4\}\}$, ou seja tem-se quatro classes de equivalência: $\{6\}$, $\{5\}$, $\{2,3\}$ e $\{1,4\}$, concluindo-se que $1 \equiv 4$ e $2 \equiv 3$.

2.6 Cálculo de E determinando a equivalência de cada par de estados

O algoritmo 2.13 também determina a relação de equivalência, analisando só um par de estados em cada iteração. Os conjuntos G e H são inicializados respectivamente a $\emptyset$ e I_Q, sendo I_Q a relação identidade de estados. Em cada iteração determina-se se os estados que se estão a analisar são equivalentes ou não. Logo os elementos de G são sempre estados distinguíveis e os de H são sempre estados equivalentes. Por esta razão, qualquer resultado intermédio do algoritmo pode ser utilizado para construir um autómato finito determinista que terá tamanho menor ou igual que o inicial. Contudo, pode não ser o autómato mínimo. Por esta razão, este algoritmo pode ser usado para reduzir o tamanho de autómatos, executando-se apenas algumas iterações.

Este algoritmo utiliza uma função auxiliar, descrita no algoritmo 2.12, a qual determina a equivalência de estados de um só par de estados. Isto é, dados dois estados p e q e um número inteiro, k, o algoritmo determina se $(p, q) \in E_k$.

Algoritmo 2.12.

procedure $equiv(p, q, k)$

if $k = 0$ then

$eq := (p \in F \wedge q \in F) \vee (\neg p \in F \wedge \neg q \in F)$

else

if $(p, q) \in S$ then

$eq := true$

else

$eq := (p \in F \wedge q \in F) \vee (\neg p \in F \wedge \neg q \in F)$

$S := S \cup \{(p, q)\}$

for $a \in \Sigma$

while not eq

$equiv(\delta(p, a), \delta(q, a), k - 1)$

$S := S \setminus \{(p, q)\}$

No início tem-se $S = \emptyset$ e $eq = false$.

Se os parâmetros de entrada da função $equiv$ forem p, q e $|Q| - 2$ então o resultado vai ser se $(p, q) \in E$ ou se $(p, q) \notin E$. De facto, como se viu anteriormente, $|Q| - 2$ é o número máximo de iterações necessárias para se determinar o ponto fixo de E_k, isto é, $E_{|Q|-2} = E$.

Na primeira iteração (assumindo que não é a única, isto é, $k \neq 0$) averigua-se se p e q pertencem ambos a F ou se não pertencem ambos a F. Se esta condição se verifica, então p e q são equivalentes e eq passa a ser $true$. De seguida adiciona-se este par ao conjunto S, independentemente da veracidade da condição testada. Se eq for igual a $false$ então analisam-se as transições de p e q por todas as letras do alfabeto. Por outras palavras, para todas as letras do alfabeto evoca-se recursivamente a função $equiv$, tendo como parâmetros de entrada $\delta(p, a)$, $\delta(q, a)$ e $k - 1$, onde a é uma letra qualquer. Desta forma, pretende-se determinar se $\delta(p, a)$ e $\delta(q, a)$ são equivalentes, ou seja, se $(\delta(p, a), \delta(q, a)) \in E_{k-1}$. Este processo é repetido recursivamente. O terceiro parâmetro de entrada é $k-1$ porque se estão a analisar os estados que sucedem a p e q, ou seja, por definição de E_k, $(p, q) \in E_k$ se $(p, q) \in E_{k-1}$ e para todas as letras do alfabeto os estados que resultam das transições de p e q através dessas mesmas letras também pertencem a E_{k-1}.

Nas iterações seguintes verifica-se se é a última iteração, isto é, se k é igual a 0. Em caso

afirmativo eq só passa a $true$ se p e q pertencem ambos a F ou se não pertencem ambos a F. Se não é a última iteração ($k \neq 0$), então verifica-se se o par $(p, q) \in S$.

- Se esta condição for verdadeira, então eq passa a ser igual a $true$. De facto, $(p, q) \in S$ então este par de estados transita para si próprio através da leitura de uma determinada palavra. Por isso, conclui-se que p e q são equivalentes.

- Se esta condição for falsa, então repete-se o procedimento efectuado na primeira iteração.

Algoritmo 2.13.

$G := \emptyset$

$H := I_Q$

while $(G \cup H) \neq Q \times Q$ do

seja $(p, q) \in ((Q \times Q) \setminus (G \cup H))$

if $equiv(p, q, (|Q| - 2))$ then

$H := H \cup \{(p, q)\}$

else

$G := G \cup \{(p, q)\}$

No início do algoritmo tem-se que H é igual a I_Q porque todo o estado é equivalente a si próprio.

A função $equiv(p, q, (|Q|-2))$ tem o valor $true$ se $(p, q) \in E$, por isso, neste caso, adiciona-se o par (p, q) a H. Se a função $equiv$ for $false$, então $(p, q) \notin E$ e este par é acrescentado a G. Tem-se sempre a garantia que H só contém estados que são equivalentes. Assim, em cada passo do algoritmo, pode-se construir um autómato com base no conjunto H que terá um tamanho inferior ou igual ao autómato inicial. Mas só no fim da execução do algoritmo é que se tem a garantia que o autómato construido com base no conjunto H é o autómato mínimo.

No fim do algoritmo, tem-se $H = E$ e $G = D$ que são, respectivamente, as relações de equivalência e distinguibilidade. Este algoritmo pode ser alterado para calcular só E ou só D.

O ciclo while é executado no máximo $|Q|^2$ vezes. Portanto a complexidade deste algoritmo é $O(|Q|^2)$.

Capítulo 3

Algoritmo de Brzozowski

Em 1962 Brzozowski propôs um algoritmo de minimização de autómatos distinto dos apresentados no capítulo anterior. Este algoritmo é aplicável a autómatos finitos deterministas, mas também a autómatos finitos não-deterministas sem transições-ε. Ao contrário dos algoritmos anteriores, no fim da execução não se tem como resultado a relação de equivalência, nem a de distinguibilidade que servem de base à construção do autómato mínimo. Neste algoritmo o resultado é já o autómato mínimo. De acordo com [10], este algoritmo não utiliza o conceito de equivalência de estados, contudo em [1] é demonstrada a opinião contrária.

Se se pretender minimizar um autómato finito não-determinista sem transições-ε, utilizando um dos algoritmos apresentados anteriormente, então, antes de se utilizar o algoritmo, é necessário convertê-lo num autómato finito determinista, aumentando a ordem de complexidade da minimização.

Neste capítulo descreve-se o algoritmo de Brzozowski e apresentam-se exemplos da sua aplicação. Nestes exemplos pode-se observar alguns casos onde o algoritmo não funciona. Isto é, quando algum dos autómatos obtidos nos passos intermédios do algoritmo tem transições-ε não se pode garantir que o resultado final do algoritmo de Brzozowski seja o autómato mínimo. Mesmo eliminando estas transições, continua-se sem poder garantir o correcto funcionamento deste algoritmo.

Por fim apresenta-se uma variante às duas últimas operações deste algoritmo, assim como uma correcção a esta mesma variante.

3.1 Descrição do algoritmo

Seja A um autómato. Lembra-se que $d(A)$ é o autómato que resulta da determinização do autómato A e que $r(A)$ é o reverso do autómato A.

O algoritmo de Brzozowski baseia-se no seguinte teorema, cuja demonstração pode ser consultada em [3].

Teorema 3.1. (Brzozowski, 1962): Seja A um autómato finito sem transições-ε. Então

$$A_{min} = drdr(A)$$

é o autómato determinista mínimo de A.

Por outras palavras, como ilustrado na figura 3.1, para obter o autómato mínimo é necessário determinar o reverso do autómato inicial, $r(A)$, em seguida convertê-lo num autómato determinista, $dr(A)$, e voltar a executar estas operações. Isto é, determinar o reverso do autómato anteriormente obtido, $rdr(A)$, e por último convertê-lo num autómato determinista, $drdr(A)$.

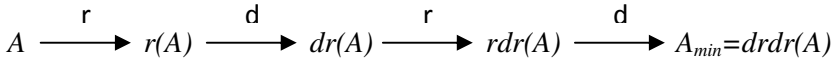

Figura 3.1: Diagrama do algoritmo de Brzozowski

A primeira determinização baseia-se nas linguagens esquerdas do autómato inicial. Ou seja, os estados que tiverem estas linguagens iguais vão ser agrupados para dar origem a um novo estado do autómato determinista. Isto é, agrupam-se os estados equivalentes. Na segunda determinização repete-se este processo, mas agora os estados que são agrupados são os estados de $dr(A)$ que têm linguagens esquerdas iguais, identificando assim os estados que são equivalentes. Portanto, no fim do algoritmo tem-se a garantia que os estados do autómato final têm todos linguagens direitas diferentes entre si, logo os estados são todos distinguíveis. Portanto, este processo assenta no conceito de equivalência de estados.

3.2 Ordem de complexidade

Embora o cálculo do reverso de um autómato seja eficiente, $O(|Q|^2)$, a ordem de complexidade da determinização é, no pior caso, $O(2^{|Q|})$.

Ainda que a complexidade teórica do algoritmo de Brzozowski seja no pior caso exponencial, na prática este algoritmo tem-se revelado bastante eficiente. De facto, as operações de determinização podem originar $2^{|Q|}$ estados, mas geralmente o autómato finito determinista equivalente possui pouco mais do que $|Q|$ estados uma vez que só se determinam os estados úteis. Quanto menor for a cardinalidade do alfabeto menor será o número de estados durante a conversão do autómato não-determinista a determinista, isto é, quanto menos transições houver, menos estados atingíveis vai haver.

Se o autómato inicial for codeterminista, a primeira determinização não vai aumentar o número de estados, e se o autómato $rdr(A)$ for um autómato determinista, então a última

operação de determinização não vai aumentar o número de estados. Logo, quando estas duas condições se verificam, a ordem de complexidade do algoritmo passa a ser linear.

3.3 Exemplos

Inicialmente ilustra-se a aplicação do algoritmo de Brzozowski a um autómato não-determinista. De seguida mostra-se a aplicação deste mesmo algoritmo a um autómato finito determinista que, durante a execução, pode originar um autómato finito não-determinista com mais do que um estado inicial. A partir destes exemplos mostra-se ainda que, as operações habituais para converter um autómato com vários estados iniciais num autómato equivalente só com um estado inicial, não se podem aplicar durante a execução deste algoritmo, pois o resultado final do algoritmo de Brzozowski é afectado, isto é, não se tem a garantia que o autómato obtido seja o mínimo. Uma possível explicação parcial para estes resultados, deve-se ao facto da introdução deste estado inicial alterar as linguagens esquerdas do autómato.

3.3.1 Minimização de um Autómato Não-Determinista

Considere-se agora o autómato não-determinista $A = (\{1,2,3,4\}, \{0,1\}, \delta, \{1\}, \{4\})$ onde $\delta(1,0) = 2$, $\delta(1,0) = 4$, $\delta(1,1) = 3$, $\delta(2,0) = 4$, $\delta(2,1) = 3$, $\delta(3,0) = 4$, $\delta(3,1) = 2$, $\delta(4,0) = 4$ e $\delta(4,1) = 4$, representado na figura seguinte:

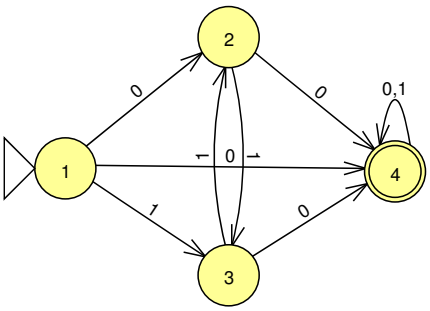

Figura 3.2: Autómato não-determinista - A

Este autómato tem como reverso o autómato seguinte:

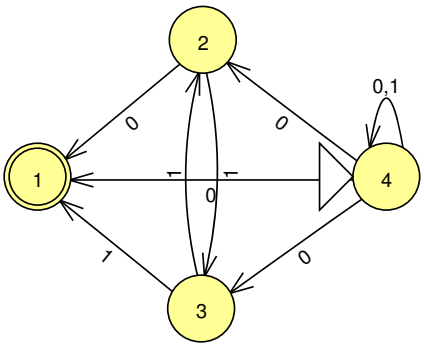

Figura 3.3: Autómato reverso do autómato da figura 3.2, A^R

A determinização de A^R conduz à seguinte tabela de transições, a qual corresponde ao autómato da figura 3.4.

		0	1
a	$\rightarrow \{4\}$	$\{1,2,3,4\}$	$\{4\}$
b	$*\{1,2,3,4\}$	$\{1,2,3,4\}$	$\{1,2,3,4\}$

Tabela 3.1: Tabela de transições da determinização do autómato A^R

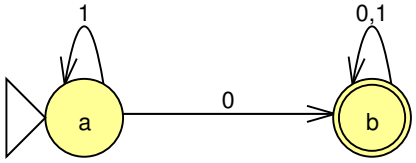

Figura 3.4: Autómato $B = dr(A)$, que reconhece a mesma linguagem que A^R

No passo seguinte determina-se o autómato reverso de B, o qual se encontra representado na figura seguinte:

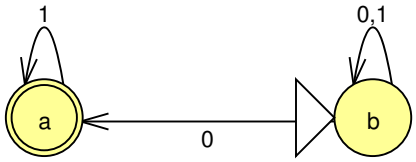

Figura 3.5: Autómato, B^R, reverso do autómato B

Na segunda determinização contrói-se a seguinte tabela de transições, a que corresponde ao autómato da figura 3.6.

	0	1
$\rightarrow \{b\}$	$\{a, b\}$	$\{b\}$
$*\{a, b\}$	$\{a, b\}$	$\{a, b\}$

Tabela 3.2: Tabela de transições da determinização do autómato B^R

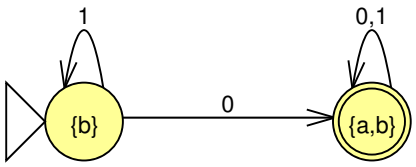

Figura 3.6: Autómato determinista equivalente a B^R

Este autómato é o autómato mínimo de A.

3.3.2 Primeira Tentativa de Minimização de um Autómato Determinista

Considere-se, novamente, o autómato A representado na figura 2.1, o qual tem o seguinte reverso:

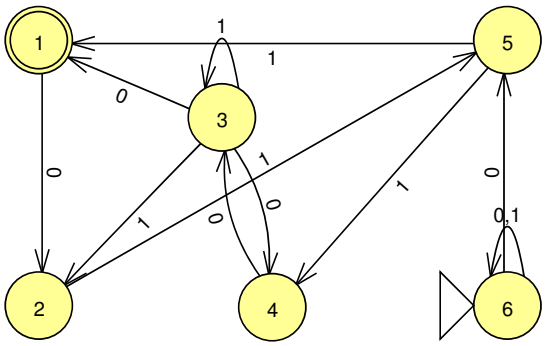

Figura 3.7: Autómato reverso do autómato da figura 2.1, A^R

A determinização do autómato A^R conduz à seguinte tabela de transições. O autómato correspondente, B, encontra-se representado na figura 3.8.

		0	1
a	$\rightarrow \{6\}$	$\{5,6\}$	$\{6\}$
b	$\{5,6\}$	$\{5,6\}$	$\{1,4,6\}$
c	$*\{1,4,6\}$	$\{2,3,5,6\}$	$\{6\}$
d	$\{2,3,5,6\}$	$\{1,4,5,6\}$	$\{1,2,3,4,5,6\}$
e	$*\{1,4,5,6\}$	$\{2,3,5,6\}$	$\{1,4,6\}$
f	$*\{1,2,3,4,5,6\}$	$\{1,2,3,4,5,6\}$	$\{1,2,3,4,5,6\}$

Tabela 3.3: Tabela de transições da determinização do autómato A^R

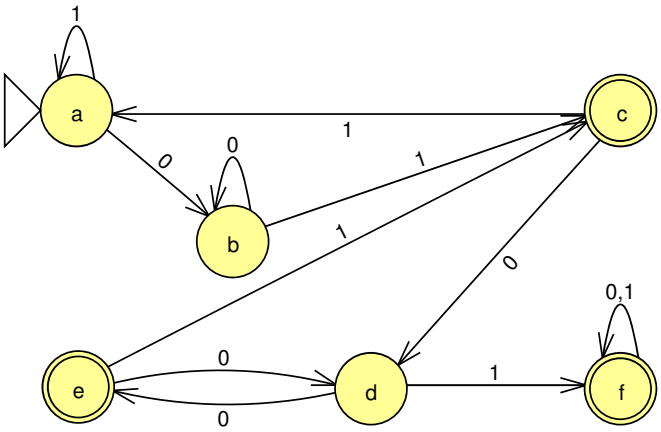

Figura 3.8: Autómato $B = dr(A)$, que reconhece a mesma linguagem que A^R

No passo seguinte determina-se o autómato reverso de B. Neste caso o autómato reverso possui três estados iniciais, c, e e f. Para garantir que o autómato ao qual se vai aplicar a determinização possui um único estado inicial, utiliza-se a técnica usual: acrescenta-se um novo estado ao autómato, o estado g, o qual passa a ser o estado inicial e transições-ε para os anteriores estados iniciais, conforme ilustrado na figura seguinte.

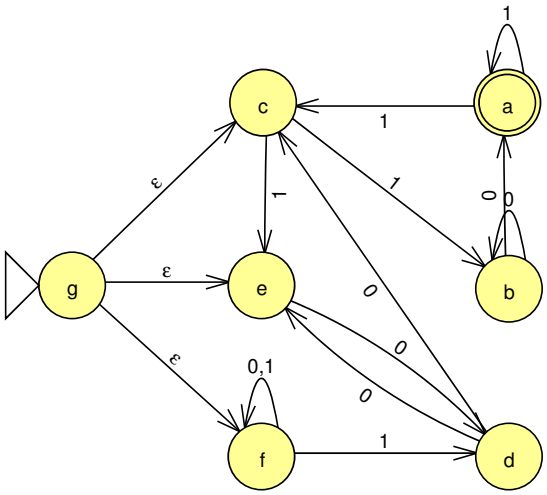

Figura 3.9: Autómato, B^R, reverso do autómato B

A segunda determinização conduz à tabela seguinte, à qual corresponde o autómato da figura 3.10.

		0	1
A	$\rightarrow \{c, e, f, g\}$	$\{d, f\}$	$\{b, d, e, f\}$
B	$\{d, f\}$	$\{c, e, f\}$	$\{d, f\}$
C	$\{b, d, e, f\}$	$\{a, b, c, d, e, f\}$	$\{d, f\}$
D	$\{c, e, f\}$	$\{d, f\}$	$\{b, d, e, f\}$
E	$*\{a, b, c, d, e, f\}$	$\{a, b, c, d, e, f\}$	$\{a, b, c, d, e, f\}$

Tabela 3.4: Tabela de transições da determinização do autómato B^R

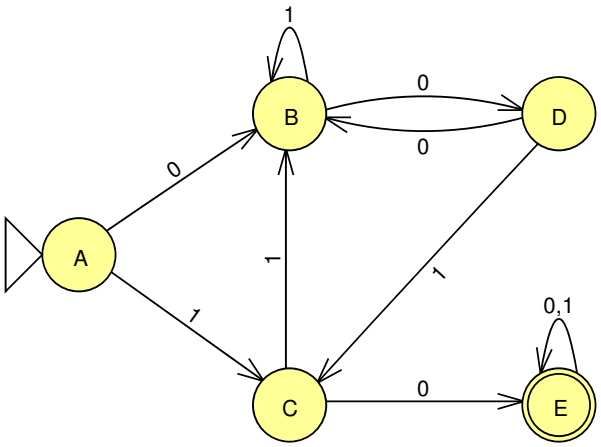

Figura 3.10: Autómato determinista equivalente a B^R

Comparando o autómato obtido com o autómato mínimo obtido no capítulo anterior verifica-se que este autómato tem mais estados, logo não é mínimo.

3.3.3 Segunda Tentativa de Minimização de um Autómato Determinista

Quando foi determinado o autómato B^R no exemplo anterior, acrescentou-se um estado para que este tivesse apenas um estado inicial e concluiu-se que dessa forma não é possível aplicar o algoritmo de Brzozowski. De facto uma das condições para a utilização deste algoritmo é que o autómato não tenha transições-ε. Assim, pode-se pensar em eliminar as transições-ε no autómato, mantendo um único estado inicial, como se ilustra na figura seguinte.

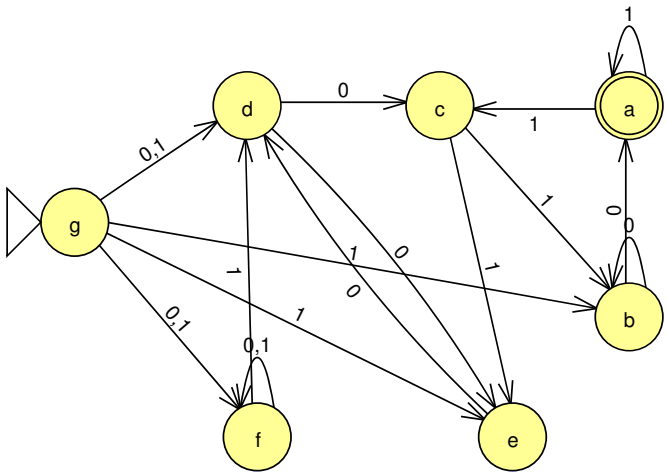

Figura 3.11: Autómato equivalente ao autómato da figura 3.9 sem transições-ε - C^R

Continuando a execução do algoritmo de Brzozowski a partir deste autómato, a determinização conduz à seguinte tabela de transições que corresponde ao autómato da figura 3.12.

		0	1
A	$\rightarrow \{g\}$	$\{d, f\}$	$\{b, d, e, f\}$
B	$\{d, f\}$	$\{c, e, f\}$	$\{d, f\}$
C	$\{b, d, e, f\}$	$\{a, b, c, d, e, f\}$	$\{d, f\}$
D	$\{c, e, f\}$	$\{d, f\}$	$\{b, d, e, f\}$
E	$*\{a, b, c, d, e, f\}$	$\{a, b, c, d, e, f\}$	$\{a, b, c, d, e, f\}$

Tabela 3.5: Tabela de transições do autómato determinista equivalente ao autómato da figura 3.11

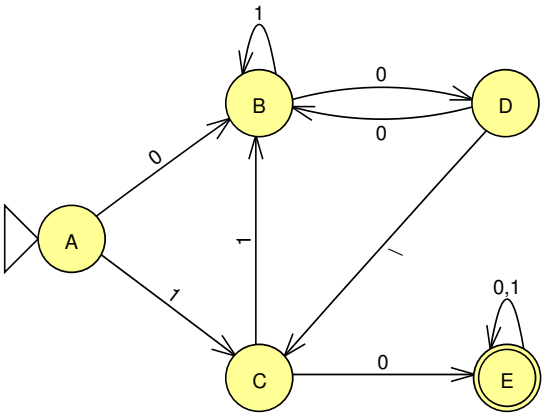

Figura 3.12: Autómato determinista que reconhece a mesma linguagem que C^R

Uma vez mais este não é o autómato mínimo pois possui mais estados do que o autómato mínimo obtido no capítulo anterior.

3.3.4 Terceira Tentativa de Minimização de um Autómato Determinista

Se permitirmos vários estados iniciais então o autómato reverso de B será D^R, representado na figura seguinte.

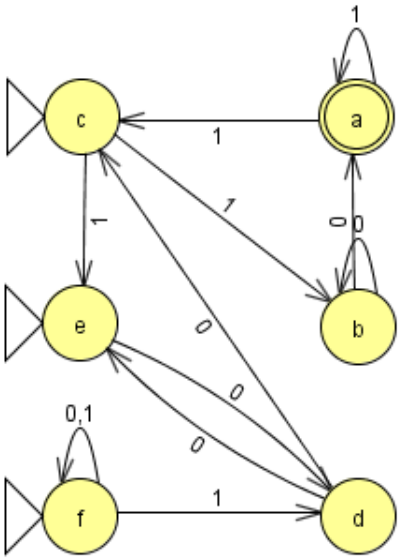

Figura 3.13: Autómato equivalente ao autómato da figura 3.9 com múltiplos estados iniciais - D^R

Continuando, uma vez mais, a execução do algoritmo de Brzozowski a partir deste autómato, determina-se a seguinte tabela de transições, que corresponde ao autómato da figura 3.14.

		0	1
A	$\rightarrow \{c, e, f\}$	$\{d, f\}$	$\{b, d, e, f\}$
B	$\{d, f\}$	$\{c, e, f\}$	$\{d, f\}$
C	$\{b, d, e, f\}$	$\{a, b, c, d, e, f\}$	$\{d, f\}$
D	$*\{a, b, c, d, e, f\}$	$\{a, b, c, d, e, f\}$	$\{a, b, c, d, e, f\}$

Tabela 3.6: Tabela de transições do autómato da figura 3.14

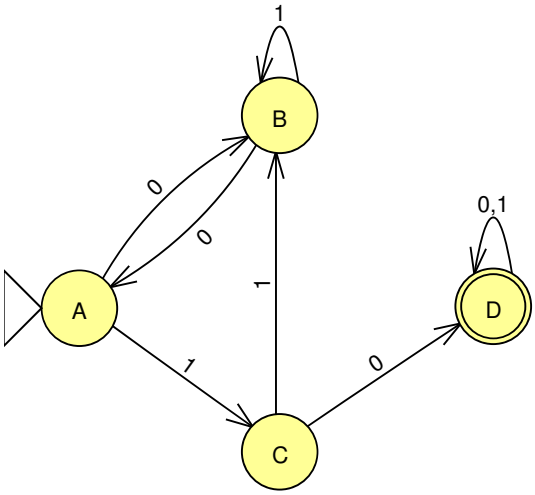

Figura 3.14: Autómato determinista que reconhece a mesma linguagem que D^R

Comparando este autómato com o obtido no capítulo anterior, verifica-se que são isomorfos. Ou seja, este é o autómato mínimo.

3.4 Variante do algoritmo

Esta variante do algoritmo de Brzozowski pode ser encontrada em [3]. O algoritmo 3.1 substitui as duas últimas operações do algoritmo de Brzozowski. Ou seja, determina-se $dr(\mathcal{A}) = (Q, \Sigma, \delta, I, F)$ e só depois é que se executa este algoritmo. Durante a sua execução determinam-se as classes de equivalência da relação de equivalência. Segundo [3], a sua ordem de complexidade é exponencial. Apresenta-se uma transcrição deste algoritmo.

Algoritmo 3.1.

Partição $= \{Q \setminus F, F\}$

Aux $= \{F\}$

while Aux $\neq \emptyset$ do

$X =$ First(Aux)

Aux = Aux$\setminus\{X\}$

Processado = Processado $\cup\{X\}$

for $a \in \Sigma$ do

$Z = \delta_r(X, a)$

if $Z \notin$ Processado then

Aux = Aux$\cup Z$

end

for $Y \in$ Partição do

$K = X \cap Y$

if $K \neq \emptyset$ then

Partição = Partição $\cup K$

if $X \not\subseteq Y$ then

Partição = Partição $\cup (X \setminus K)$

if $Y \not\subseteq X$ then

Partição = Partição $\cup (Y \setminus K)$

A função δ_r corresponde à função de transição de $r(A)$.

Embora não esteja indicado no algoritmo, no início da sua execução o conjunto Processado é inicializado a vazio.

O conjunto Partição será o conjunto das classes de equivalência do autómato mínimo. Inicialmente contém dois conjuntos, $Q \setminus F$ e F. Parte-se destes dois conjuntos e determina-se o conjunto dos estados que têm transições para elementos do outro conjunto, depois faz-se a mesma análise para este novo conjunto e assim sucessivamente. Em cada iteração determina-se a intersecção do conjunto que se está a analisar, X, com cada possível classe de equivalência, isto é, os elementos do conjunto Partição. Sendo K essa intersecção, então:

1. Se K é diferente do conjunto vazio, então K poderá ser uma classe de equivalência do autómato final, por isso adiciona-se K ao conjunto Partição.

2. Se $X \not\subseteq Y$ então adiciona-se $X \setminus K$ ao conjunto Partição. Isto é, os elementos de $X \setminus K$ não são equivalentes aos restantes elementos de X, logo vão pertencer a diferentes classes de equivalência.

3. Se $Y \not\subseteq X$ então adiciona-se $Y \setminus K$ ao conjunto Partição. Ou seja, os elementos de $Y \setminus K$ e Y não são equivalentes, logo vão pertencer a diferentes classes de equivalência.

Pode ser consultado na página 72 um exemplo da execução das duas primeiras iterações do ciclo while deste algoritmo. Com base nos resultados obtidos, verifica-se que existem estados que pertencem a mais do que uma classe de equivalência e também que foram efectuadas operações desnecessárias ao conjunto Partição, isto é, não alteraram este conjunto.

3.5 Uma proposta de melhoramento da variante do algoritmo

Apesar de o algoritmo não considerar, o último ciclo for só faz sentido ser executado se $K \neq Y$ e $K \neq \emptyset$. Quando estas condições não se verificam, Y não precisa de ser dividido. Ou

seja, não foram identificados estados pertencentes a Y que não são equivalentes aos restantes estados de Y. Sendo assim, o conjunto Partição não sofre qualquer alteração.

Quando $K \neq \emptyset$ o algoritmo apenas indica que se deve inserir o conjunto K no conjunto Partição. Contudo, ao realizar apenas esta operação, está-se a inserir um conjunto que tem estados em comum com outro elemento do conjunto. Isto é, neste caso os elementos do conjunto Partição podem não ser disjuntos, logo não correspondem às classes de equivalência do autómato mínimo pois estas têm que ser disjuntas. Para evitar esta situação, quando $K \neq \emptyset$, deve-se em primeiro lugar retirar Y do conjunto Partição e somente depois inserir outros elementos.

Alterando este algoritmo para evitar que estas situações ocorram, o último ciclo for passa a ser definido da seguinte forma:

for $Y \in$ Partição do
  $K = X \cap Y$
  if $K \neq Y$ and $K \neq \emptyset$ then
    Partição = Partição $\setminus Y$
    Partição = Partição $\cup \{Y \setminus K, K\}$

Testou-se recorrendo a exemplos, que esta alteração funciona. Os resultados das duas primeiras iterações do ciclo while podem ser consultados na página 74.

Comparando este exemplo com o exemplo da execução deste algoritmo sem alterações, verificou-se que com o mesmo número de iterações efectuaram-se menos operações ao conjunto Partição. Verificou-se também que os conjuntos pertencentes ao conjunto Partição no primeiro exemplo não são disjuntos, enquanto que no segundo exemplo são disjuntos.

Conclusão

Ao longo desta dissertação é feita uma síntese de diferentes algoritmos de minimização de autómatos. Para poder efectuar esta análise, foram apresentadas algumas definições da teoria da computação que são utilizadas no decorrer da dissertação.

Os algoritmos apresentados foram divididos em dois capítulos. Um com os algoritmos que utilizam as relações de equivalência e/ou distinguibilidade ou as classes de equivalência da relação de equivalência e outro com o algoritmo de Brzozowski.

Estes primeiros algoritmos são descritos no capítulo 2. Neste capítulo os algoritmos apresentados só minimizam autómatos deterministas. Para os utilizar com autómatos não-deterministas era necessário em primeiro lugar converter os autómatos a deterministas e só depois utilizar o algoritmo de minimização. Alguns destes algoritmos determinam as relações de equivalência e/ou distinguibilidade pelo cálculo iterativo até se obter o ponto fixo, enquanto que outros não utilizam este método para calcular as relações de equivalência. A principal diferença entre estes dois métodos está nos resultados das diferentes iterações. Pois quando se utiliza o cálculo iterativo, em todas as iterações os resultados são relações de equivalência. Neste mesmo capítulo é apresentado um algoritmo em que no fim de cada iteração, os conjuntos obtidos, G e H, contêm, respectivamente, só estados distinguíveis e equivalentes. Assim, este algoritmo pode ser usado somente para reduzir o tamanho do autómato inicial, podendo o resultado não ser o autómato mínimo. Para isto basta que não se execute o algoritmo até ao fim. Pois, só no fim da execução é que se tem a garantia que o autómato obtido é o autómato mínimo. Portanto, independentemente do método utilizado é sempre necessário analisar todos os pares de estados para obter-se o autómato mínimo. Os restantes algoritmos deste capítulo determinam as classes de equivalência da relação de equivalência.

Fazendo uma análise comparativa às diferentes ordens de complexidade destes algoritmos, confirma-se que o melhor algoritmo é o de Hopcroft, uma vez que a sua ordem de complexidade é $O(|Q| \log |Q|)$.

No último capítulo analisa-se o algoritmo de Brzozowski, que tem a particulariedade de poder ser utilizado para minimizar tanto autómatos deterministas como autómatos não-deterministas sem transições-ε. São apresentados exemplos da sua aplicação, num deles um dos autómatos obtidos durante a execução do algoritmo tem vários estados iniciais. Convertendo o autómato num só com um estado inicial, acrescentaram-se transições-ε e por isso o resultado final foi afectado, pois não se obteve o autómato mínimo. Num outro exemplo eliminaram-se estas transições mas, uma vez mais, o resultado final também não foi o

autómato mínimo. Uma possível explicação parcial, é o facto de as linguagens esquerdas do autómato terem sido alteradas. Mantendo os vários estados iniciais, o resultado final é o autómato mínimo. A partir destes exemplos conclui-se que para se garantir que o resultado final é o autómato mínimo é preciso que o autómato inicial e todos os autómatos obtidos ao longo da execução do algoritmo não tenham transições-ε. Este algoritmo tem uma ordem de complexidade pior que a do algoritmo de Hopcroft, contudo na prática consegue-se obter bons resultados, uma vez que durante as operações de determinização só se determinam os estados úteis. Por esta razão nem sempre é necessário determinar os $2^{|Q|}$ estados possíveis. Neste mesmo capítulo, é também descrita uma variante do algoritmo de Brzozowski e propõe-se uma alteração a esta variante, com vista a melhora-la. Esta variante substitui as duas últimas operações do algoritmo de Brzozowski.

Um futuro trabalho que poderia ser feito neste âmbito, seria a implementação destes algoritmos, pois desta forma seria possível obter resultados práticos para os algoritmos estudados. Assim obter-se-iam valores para os tempos de execução e poder-se-iam fazer testes para analisar estes mesmos tempos.

Apêndices

Algoritmo. 2.1

$G = D_0 = \{(Q \setminus F) \times F\} = \{\{1,2,3,4,5\} \times \{6\}\} =$

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

$G' = \emptyset$

1ª iteração:

$G' = G = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\}\}$

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

2ª iteração:

$G' = G = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},\{3,5\},\{4,5\}\}$

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

$\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}$

3ª iteração:

$G' = G = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$

$\{3,5\},\{4,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}$

$G = G'$

Portanto, $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $4 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$ e $3 \not\equiv 4$.

Algoritmo. 2.2

$H = E_0 = \{(Q \setminus F)^2 \cup F^2\} = \{\{1,2,3,4,5\}^2 \cup \{6\}^2\} =$

$= \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,2\},\{2,3\},\{2,4\},$

$\{2,5\},\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

$H' = \{Q \times Q\} =$

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

$\{3,3\},\{3,4\},\{3,5\},\{3,6\},\{4,4\},\{4,5\},\{4,6\},\{5,5\},\{5,6\},\{6,6\}\}$

1ª iteração:

$H' = H = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,2\},\{2,3\},\{2,4\},\{2,5\},$
$\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

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

2ª iteração:

$H' = H = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},\{3,3\},$
$\{3,4\},\{4,4\},\{5,5\},\{6,6\}\}$

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

3ª iteração:

$H' = H = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},\{5,5\},\{6,6\}\}$

$H = H'$

Portanto, $1 \equiv 4$ e $2 \equiv 3$.

Algoritmo. 2.3

$G = D_0 = \{(Q \setminus F) \times F\} = \{\{1,2,3,4,5\} \times \{6\}\} =$
$= \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\}\}$

$H = E_0 = \{(Q \setminus F)^2 \cup F^2\} = \{\{1,2,3,4,5\}^2 \cup \{6\}^2\} =$
$= \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,2\},\{2,3\},\{2,4\},$
$\{2,5\},\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

$G' = \emptyset$

$H' = \{Q \times Q\} = \{\{1,2,3,4,5,6\} \times \{1,2,3,4,5,6\}\} =$
$= \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{1,6\},\{2,2\},\{2,3\},\{2,4\},$
$\{2,5\},\{2,6\},\{3,3\},\{3,4\},\{3,5\},\{3,6\},\{4,4\},\{4,5\},$
$\{4,6\},\{5,5\},\{5,6\},\{6,6\}\}$

1ª iteração (while):

$G' = G$

$H' = H$

Ciclo for:

$G = G \cup \{\{1,5\},\{2,5\},\{3,5\},\{4,5\}\} =$

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

$H = H \setminus \{\{1,5\},\{2,5\},\{3,5\},\{4,5\}\} =$

$= \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},$

$\{3,3\},\{3,4\},\{4,4\},\{5,5\},\{6,6\}\}$

2ª iteração (while):

$G' = G$

$H' = H$

Ciclo for:

$G = G \cup \{\{1,2\},\{1,3\},\{2,4\},\{3,4\}\} =$

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

$\{4,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}$

$H = H \setminus \{\{1,2\},\{1,3\},\{2,4\},\{3,4\}\} =$

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

3ª iteração (while):

$G' = G$

$H' = H$

Ciclo for:

$G = G \cup \emptyset = G$

$H = H \setminus \emptyset = H$

Do conjunto G, conclui-se que $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $4 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$ e $3 \not\equiv 4$.

Do conjunto H, conclui-se que $1 \equiv 4$ e $2 \equiv 3$.

Algoritmo. 2.4

$G = D_0 = \{(Q \setminus F) \times F\} = \{\{1,2,3,4,5\} \times \{6\}\} =$

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

$H = E_0 = \{(Q \setminus F)^2 \cup F^2\} = \{\{1,2,3,4,5\}^2 \cup \{6\}^2\} =$

$= \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,2\},\{2,3\},\{2,4\},$

$\{2,5\},\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

1ª iteração:

$G = G \cup \{\{1,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\}\}$

$H = H \setminus \{\{1,5\}\} = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},\{2,5\},$

$\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

2ª iteração:

$G = G \cup \{\{2,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\}\}$

$H = H \setminus \{\{2,5\}\} = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},$

$\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

3ª iteração:

$G = G \cup \{\{3,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},\{3,5\}\}$

$H = H \setminus \{\{3,5\}\} = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},$

$\{3,3\},\{3,4\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

4ª iteração:

$G = G \cup \{\{1,2\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},\{3,5\},\{1,2\}\}$

$H = H \setminus \{\{1,2\}\} = \{\{1,1\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},$

$\{3,3\},\{3,4\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

5ª iteração:

$G = G \cup \{\{1,3\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$

$\{3,5\},\{1,2\},\{1,3\}\}$

$$H = H \setminus \{\{1,3\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},\{3,3\},$$
$$\{3,4\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$$

6ª iteração:

$$G = G \cup \{\{2,4\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$$
$$\{3,5\},\{1,2\},\{1,3\},\{2,4\}\}$$
$$H = H \setminus \{\{2,4\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{3,4\},$$
$$\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$$

7ª iteração:

$$G = G \cup \{\{3,4\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$$
$$\{3,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}$$
$$H = H \setminus \{\{3,4\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},$$
$$\{4,5\},\{5,5\},\{6,6\}\}$$

8ª iteração:

$$G = G \cup \{\{4,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$$
$$\{3,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\},\{4,5\}\}$$
$$H = H \setminus \{\{4,5\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},$$
$$\{5,5\},\{6,6\}\}$$

9ª iteração:

$$G = G \cup \emptyset = G$$
$$H = H \setminus \emptyset = H$$

Portanto, pelo conjunto G verifica-se que $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$, $3 \not\equiv 4$ e $4 \not\equiv 5$.

Pelo conjunto H, tem-se que $1 \equiv 4$ e $2 \equiv 3$.

Algoritmo. 2.6

$G = D_0 = \{(Q \setminus F) \times F\} = \{\{1,2,3,4,5\} \times \{6\}\} =$

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

$H = E_0 = \{(Q \setminus F)^2 \cup F^2\} = \{\{1,2,3,4,5\}^2 \cup \{6\}^2\} =$

$= \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,2\},\{2,3\},\{2,4\},$

$\{2,5\},\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

1ª iteração (while):

$a = 0 \quad \{p,q\} = \{1,5\}$

1ª iteração (for):

$G = G \cup \{\{1,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\}\}$

$H = H \setminus \{\{1,5\}\} = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},\{2,5\},$

$\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

2ª iteração (while):

$a = 0 \quad \{p,q\} = \{2,5\}$

1ª iteração (for):

$G = G \cup \{\{2,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\}\}$

$H = H \setminus \{\{2,5\}\} = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},$

$\{3,3\},\{3,4\},\{3,5\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

3ª iteração (while):

$a = 0 \quad \{p,q\} = \{3,5\}$

1ª iteração (for):

$G = G \cup \{\{3,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},\{3,5\}\}$

$H = H \setminus \{\{3,5\}\} = \{\{1,1\},\{1,2\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},$

$\{3,3\},\{3,4\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

4ª iteração (while):

$a = 1 \quad \{p,q\} = \{1,2\}$

$\underline{1^{\text{a}} \text{ iteração (for):}}$

$G = G \cup \{\{1,2\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$
$\{3,5\},\{1,2\}\}$

$H = H \setminus \{\{1,2\}\} = \{\{1,1\},\{1,3\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},$
$\{3,3\},\{3,4\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

$\underline{2^{\text{a}} \text{ iteração (for):}}$

$G = G \cup \{\{1,3\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$
$\{3,5\},\{1,2\},\{1,3\}\}$

$H = H \setminus \{\{1,3\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{2,4\},\{3,3\},\{3,4\},$
$\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

$\underline{5^{\text{a}} \text{ iteração (while):}}$

$a = 0 \quad \{p,q\} = \{2,4\}$

$\underline{1^{\text{a}} \text{ iteração (for):}}$

$G = G \cup \{\{2,4\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$
$\{3,5\},\{1,2\},\{1,3\},\{2,4\}\}$

$H = H \setminus \{\{2,4\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{3,4\},\{4,4\},$
$\{4,5\},\{5,5\},\{6,6\}\}$

$\underline{6^{\text{a}} \text{ iteração (while):}}$

$a = 1 \quad \{p,q\} = \{3,4\}$

$\underline{1^{\text{a}} \text{ iteração (for):}}$

$G = G \cup \{\{3,4\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$
$\{3,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\}\}$

$H = H \setminus \{\{3,4\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},\{4,5\},\{5,5\},\{6,6\}\}$

$\underline{7^{\text{a}} \text{ iteração (while):}}$

$a = 0 \quad \{p,q\} = \{4,5\}$

$\underline{1^{\text{a}} \text{ iteração (for):}}$

$G = G \cup \{\{4,5\}\} = \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\},\{1,5\},\{2,5\},$

$$\{3,5\},\{1,2\},\{1,3\},\{2,4\},\{3,4\},\{4,5\}\}$$

$$H = H \setminus \{\{4,5\}\} = \{\{1,1\},\{1,4\},\{2,2\},\{2,3\},\{3,3\},\{4,4\},\{5,5\},\{6,6\}\}$$

Portanto, conclui-se que $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$, $3 \not\equiv 4$, $4 \not\equiv 5$ e que $1 \equiv 4$ e $2 \equiv 3$.

Algoritmo. 2.8

1º Ciclo for:

$L(1,1) = \emptyset, L(1,2) = \emptyset, L(1,3) = \emptyset, ..., L(6,6) = \emptyset$

$G = D_0 = \{(Q \setminus F) \times F\} = \{\{1,2,3,4,5\} \times \{6\}\} =$
$= \{\{1,6\},\{2,6\},\{3,6\},\{4,6\},\{5,6\}\}$

1ª iteração (2º ciclo for):

$\{p,q\} = \{1,1\}$

2ª iteração (2º ciclo for):

$\{p,q\} = \{1,2\}$

1ª iteração (3º ciclo for):

$L(\{\delta(1,0),\delta(2,0)\}) = L(\{3,1\}) \cup \{\{1,2\}\} = \{\{1,2\}\}$

2ª iteração (3º ciclo for):

$L(\{\delta(1,1),\delta(2,1)\}) = L(\{5,3\}) \cup \{\{1,2\}\} = \{\{1,2\}\}$

3ª iteração (2º ciclo for):

$\{p,q\} = \{1,3\}$

1ª iteração (3º ciclo for):

$L(\{\delta(1,0),\delta(3,0)\}) = L(\{3,4\}) \cup \{\{1,3\}\} = \{\{1,3\}\}$

2ª iteração (3º ciclo for):

$L(\{\delta(1,1),\delta(3,1)\}) = L(\{5,3\}) \cup \{\{1,3\}\} = \{\{1,2\},\{1,3\}\}$

4ª iteração (2º ciclo for):

$\{p, q\} = \{1, 4\}$

5ª iteração (2º ciclo for):

$\{p, q\} = \{1, 5\}$

$A = \{\{1, 5\}\}$

$B = \emptyset$

1ª iteração (while):

$\{r, s\} = \{1, 5\}$

$G = G \cup \{\{1, 5\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}\}$

$A = A \setminus \{\{1, 5\}\} = \emptyset$

$B = B \cup \{\{1, 5\}\} = \{\{1, 5\}\}$

$A = A \cup (L(\{1, 5\}) \setminus B) = \emptyset$

6ª iteração (2º ciclo for):

$\{p, q\} = \{2, 2\}$

7ª iteração (2º ciclo for):

$\{p, q\} = \{2, 3\}$

1ª iteração (3º ciclo for):

$L(\{\delta(2, 0), \delta(3, 0)\}) = L(\{1, 4\}) \cup \{\{2, 3\}\} = \{\{2, 3\}\}$

8ª iteração (2º ciclo for):

$\{p, q\} = \{2, 4\}$

1ª iteração (3º ciclo for):

$L(\{\delta(2, 0), \delta(4, 0)\}) = L(\{1, 3\}) \cup \{\{2, 4\}\} = \{\{1, 2\}, \{2, 4\}\}$

2ª iteração (3º ciclo for):

$L(\{\delta(2, 1), \delta(4, 1)\}) = L(\{3, 5\}) \cup \{\{2, 4\}\} = \{\{1, 2\}, \{1, 3\}, \{2, 4\}\}$

9ª iteração (2º ciclo for):

$\{p, q\} = \{2, 5\}$

$A = \{\{2, 5\}\}$

$B = \emptyset$

1ª iteração (while):

$\{r, s\} = \{2, 5\}$

$G = G \cup \{\{2, 5\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}\}$

$A = A \setminus \{\{2, 5\}\} = \emptyset$

$B = B \cup \{\{2, 5\}\} = \{\{2, 5\}\}$

$A = A \cup (L(\{2, 5\}) \setminus B) = \emptyset$

10ª iteração (2º ciclo for):

$\{p, q\} = \{3, 3\}$

11ª iteração (2º ciclo for):

$\{p, q\} = \{3, 4\}$

1ª iteração (3º ciclo for):

$L(\{\delta(3, 0), \delta(4, 0)\}) = L(\{4, 3\}) \cup \{\{3, 4\}\} = \{\{1, 3\}, \{3, 4\}\}$

2ª iteração (3º ciclo for):

$L(\{\delta(3, 1), \delta(4, 1)\}) = L(\{3, 5\}) \cup \{\{3, 4\}\} = \{\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}\}$

12ª iteração (2º ciclo for):

$\{p, q\} = \{3, 5\}$

$A = \{\{3, 5\}\}$

$B = \emptyset$

1ª iteração (while):

$\{r, s\} = \{3, 5\}$

$G = G \cup \{\{3, 5\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}, \{3, 5\}\}$

$A = A \setminus \{\{3, 5\}\} = \emptyset$

$B = B \cup \{\{3, 5\}\} = \{\{3, 5\}\}$

$A = A \cup (L(\{3, 5\}) \setminus B) = \{\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}\}$

2ª iteração (while):

$\{r, s\} = \{1, 2\}$

$G = G \cup \{\{1, 2\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}, \{3, 5\}, \{1, 2\}\}$

$A = A \setminus \{\{1, 2\}\} = \{\{1, 3\}, \{2, 4\}, \{3, 4\}\}$

$B = B \cup \{\{1, 2\}\} = \{\{3, 5\}, \{1, 2\}\}$

$A = A \cup (L(\{1, 2\}) \setminus B) = \{\{1, 3\}, \{2, 4\}, \{3, 4\}\}$

3ª iteração (while):

$\{r, s\} = \{1, 3\}$

$G = G \cup \{\{1, 3\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}, \{3, 5\},$
$\{1, 2\}, \{1, 3\}\}$

$A = A \setminus \{\{1, 3\}\} = \{\{2, 4\}, \{3, 4\}\}$

$B = B \cup \{\{1, 3\}\} = \{\{3, 5\}, \{1, 2\}, \{1, 3\}\}$

$A = A \cup (L(\{1, 3\}) \setminus B) = \{\{2, 4\}, \{3, 4\}\}$

4ª iteração (while):

$\{r, s\} = \{2, 4\}$

$G = G \cup \{\{2, 4\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}, \{3, 5\},$
$\{1, 2\}, \{1, 3\}, \{2, 4\}\}$

$A = A \setminus \{\{2, 4\}\} = \{\{3, 4\}\}$

$B = B \cup \{\{2, 4\}\} = \{\{3, 5\}, \{1, 2\}, \{1, 3\}, \{2, 4\}\}$

$A = A \cup (L(\{2, 4\}) \setminus B) = \{\{3, 4\}\}$

5ª iteração (while):

$\{r, s\} = \{3, 4\}$

$G = G \cup \{\{3, 4\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}, \{3, 5\},$
$\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}\}$

$A = A \setminus \{\{3, 4\}\} = \emptyset$

$B = B \cup \{\{3, 4\}\} = \{\{3, 5\}, \{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}\}$

$A = A \cup (L(\{3, 4\}) \setminus B) = \emptyset$

13ª iteração (2º ciclo for):

$\{p, q\} = \{4, 4\}$

14ª iteração (2º ciclo for):

$\{p, q\} = \{4, 5\}$

$A = \{\{4, 5\}\}$

$B = \emptyset$

1ª iteração (while):

$\{r, s\} = \{4, 5\}$

$G = G \cup \{\{4, 5\}\} = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}, \{3, 5\},$
$\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$

$A = A \setminus \{\{4, 5\}\} = \emptyset$

$B = B \cup \{\{4, 5\}\} = \{\{4, 5\}\}$

$A = A \cup (L(\{4, 5\}) \setminus B) = \emptyset$

15ª iteração (2º ciclo for):

$\{p, q\} = \{5, 5\}$

16ª iteração (2º ciclo for):

$\{p, q\} = \{6, 6\}$

No fim da execução do algoritmo tem-se

$$G = \{\{1, 6\}, \{2, 6\}, \{3, 6\}, \{4, 6\}, \{5, 6\}, \{1, 5\}, \{2, 5\}, \{3, 5\}, \{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}.$$

Daqui conclui-se que $1 \not\equiv 6$, $2 \not\equiv 6$, $3 \not\equiv 6$, $4 \not\equiv 6$, $5 \not\equiv 6$, $1 \not\equiv 5$, $2 \not\equiv 5$, $3 \not\equiv 5$, $1 \not\equiv 2$, $1 \not\equiv 3$, $2 \not\equiv 4$, $3 \not\equiv 4$ e $4 \not\equiv 5$.

Algoritmo. 2.9

$P = [Q]_{E_0} = \{Q \setminus F, F\} = \{\{1, 2, 3, 4, 5\}, \{6\}\}$

$L = \{(\{1, 2, 3, 4, 5\}, 0), (\{1, 2, 3, 4, 5\}, 1), (\{6\}, 0), (\{6\}, 1)\}$

1ª iteração (while):

$(Q_1, a) = (\{1,2,3,4,5\}, 0)$

$P' = P = \{\{1,2,3,4,5\}, \{6\}\}$

$L = L \setminus \{(\{1,2,3,4,5\}, 0)\} = \{(\{1,2,3,4,5\}, 1), (\{6\}, 0), (\{6\}, 1)\}$

1ª iteração (1º ciclo for):

$Q_0 = \{1,2,3,4,5\}$

$Q_0' = \{1,2,3,4\}$

$P = P \setminus \{\{1,2,3,4,5\}\} \cup \{\{5\}, \{1,2,3,4\}\} = \{\{6\}, \{5\}, \{1,2,3,4\}\}$

1ª iteração (2º ciclo for):

$(Q_0, b) = (\{1,2,3,4,5\}, 0)$

$L = L \cup \{(\{1,2,3,4\}, 0), (\{5\}, 0)\} =$

$= \{(\{1,2,3,4,5\}, 1), (\{6\}, 0), (\{6\}, 1), (\{1,2,3,4\}, 0), (\{5\}, 0)\}$

2ª iteração (2º ciclo for):

$(Q_0, b) = (\{1,2,3,4,5\}, 1)$

$L = L \setminus \{(\{1,2,3,4,5\}, 1)\} \cup \{(\{1,2,3,4\}, 1), (\{5\}, 1)\} =$

$= \{(\{6\}, 0), (\{6\}, 1), (\{1,2,3,4\}, 0), (\{5\}, 0), (\{1,2,3,4\}, 1), (\{5\}, 1)\}$

2ª iteração (while):

$(Q_1, a) = (\{6\}, 0)$

$P' = P = \{\{6\}, \{5\}, \{1,2,3,4\}\}$

$L = L \setminus \{(\{6\}, 0)\} = \{(\{6\}, 1), (\{1,2,3,4\}, 0), (\{5\}, 0), (\{1,2,3,4\}, 1), (\{5\}, 1)\}$

3ª iteração (while):

$(Q_1, a) = (\{6\}, 1)$

$P' = P = \{\{6\}, \{5\}, \{1,2,3,4\}\}$

$L = L \setminus \{(\{6\}, 1)\} = \{(\{1,2,3,4\}, 0), (\{5\}, 0), (\{1,2,3,4\}, 1), (\{5\}, 1)\}$

4ª iteração (while):

$(Q_1, a) = (\{1,2,3,4\}, 0)$

$P' = P = \{\{6\}, \{5\}, \{1,2,3,4\}\}$

$L = L \setminus \{(\{1,2,3,4\},0)\} = \{(\{5\},0),(\{1,2,3,4\},1),(\{5\},1)\}$

5ª iteração (while):

$(Q_1, a) = (\{5\},0)$

$P' = P = \{\{6\},\{5\},\{1,2,3,4\}\}$

$L = L \setminus \{(\{5\},0)\} = \{(\{1,2,3,4\},1),(\{5\},1)\}$

6ª iteração (while):

$(Q_1, a) = (\{1,2,3,4\},1)$

$P' = P = \{\{6\},\{5\},\{1,2,3,4\}\}$

$L = L \setminus \{(\{1,2,3,4\},1)\} = \{(\{5\},1)\}$

1ª iteração (1º ciclo for):

$Q_0 = \{1,2,3,4\}$

$Q_0' = \{2,3\}$

$P = P \setminus \{\{1,2,3,4\}\} \cup \{\{1,4\},\{2,3\}\} = \{\{6\},\{5\},\{1,4\},\{2,3\}\}$

1ª iteração (2º ciclo for):

$(Q_0, b) = (\{1,2,3,4\},0)$

$L = L \cup \{(\{2,3\},0),(\{1,4\},0)\} =$

$= \{(\{5\},1),(\{2,3\},0),(\{1,4\},0)\}$

2ª iteração (2º ciclo for):

$(Q_0, b) = (\{1,2,3,4\},1)$

$L = L \cup \{(\{2,3\},1),(\{1,4\},1)\} =$

$= \{(\{5\},1),(\{2,3\},0),(\{1,4\},0),(\{2,3\},1),(\{1,4\},1)\}$

7ª iteração (while):

$(Q_1, a) = (\{5\},1)$

$P' = P = \{\{6\},\{5\},\{1,4\},\{2,3\}\}$

$L = L \setminus \{(\{5\},1)\} = \{(\{2,3\},0),(\{1,4\},0),(\{2,3\},1),(\{1,4\},1)\}$

8ª iteração (while):

$(Q_1, a) = (\{2,3\}, 0)$

$P' = P = \{\{6\}, \{5\}, \{1,4\}, \{2,3\}\}$

$L = L \setminus \{(\{2,3\}, 0)\} = \{(\{1,4\}, 0), (\{2,3\}, 1), (\{1,4\}, 1)\}$

9ª iteração (while):

$(Q_1, a) = (\{1,4\}, 0)$

$P' = P = \{\{6\}, \{5\}, \{1,4\}, \{2,3\}\}$

$L = L \setminus \{(\{1,4\}, 0)\} = \{(\{2,3\}, 1), (\{1,4\}, 1)\}$

10ª iteração (while):

$(Q_1, a) = (\{2,3\}, 1)$

$P' = P = \{\{6\}, \{5\}, \{1,4\}, \{2,3\}\}$

$L = L \setminus \{(\{2,3\}, 1)\} = \{(\{1,4\}, 1)\}$

11ª iteração (while):

$(Q_1, a) = (\{1,4\}, 1)$

$P' = P = \{\{6\}, \{5\}, \{1,4\}, \{2,3\}\}$

$L = L \setminus \{(\{1,4\}, 1)\} = \emptyset$

No fim da execução do algoritmo tem-se $P = \{\{6\}, \{5\}, \{2,3\}, \{1,4\}\}$, ou seja tem-se quatro classes de equivalência: $\{6\}$, $\{5\}$, $\{2,3\}$ e $\{1,4\}$. Logo conclui-se que $1 \equiv 4$ e $2 \equiv 3$.

Algoritmo. 2.10

$P = [Q]_{E_0} = \{Q \setminus F, F\} = \{\{1,2,3,4,5\}, \{6\}\}$

Como $|F| = |\{6\}| \leq |Q \setminus F| = |\{1,2,3,4,5\}|$ então $L = \{(\{6\}, 0), (\{6\}, 1)\}$

1ª iteração (while):

$(Q_1, a) = (\{6\}, 0)$

$P' = P = \{\{1,2,3,4,5\}, \{6\}\}$

$L = L \setminus \{(\{6\}, 0)\} = \{(\{6\}, 1)\}$

$\underline{1^{\text{a}} \text{ iteração } (1^{\text{o}} \text{ ciclo for}):}$

$Q_0 = \{1, 2, 3, 4, 5\}$

$Q_0' = \{5\}$

$P = P \setminus \{\{1, 2, 3, 4, 5\}\} \cup \{\{1, 2, 3, 4\}, \{5\}\} = \{\{6\}, \{1, 2, 3, 4\}, \{5\}\}$

$\underline{1^{\text{a}} \text{ iteração } (2^{\text{o}} \text{ ciclo for}):}$

$(Q_0, b) = (\{1, 2, 3, 4, 5\}, 0)$

Como $|Q_0'| = |\{5\}| \leq |Q_0 \setminus Q_0'| = |\{1, 2, 3, 4\}|$ então

$L = L \cup \{(\{5\}, 0)\} = \{(\{6\}, 1), (\{5\}, 0)\}$

$\underline{2^{\text{a}} \text{ iteração } (2^{\text{o}} \text{ ciclo for}):}$

$(Q_0, b) = (\{1, 2, 3, 4, 5\}, 1)$

Como $|Q_0'| = |\{5\}| \leq |Q_0 \setminus Q_0'| = |\{1, 2, 3, 4\}|$ então

$L = L \cup \{(\{5\}, 1)\} = \{(\{6\}, 1), (\{5\}, 0), (\{5\}, 1)\}$

$\underline{2^{\text{a}} \text{ iteração (while)}:}$

$(Q_1, a) = (\{6\}, 1)$

$P' = P = \{\{6\}, \{1, 2, 3, 4\}, \{5\}\}$

$L = L \setminus \{(\{6\}, 1)\} = \{(\{5\}, 0), (\{5\}, 1)\}$

$\underline{3^{\text{a}} \text{ iteração (while)}:}$

$(Q_1, a) = (\{5\}, 0)$

$P' = P = \{\{6\}, \{1, 2, 3, 4\}, \{5\}\}$

$L = L \setminus \{(\{5\}, 0)\} = \{(\{5\}, 1)\}$

$\underline{4^{\text{a}} \text{ iteração (while)}:}$

$(Q_1, a) = (\{5\}, 1)$

$P' = P = \{\{6\}, \{1, 2, 3, 4\}, \{5\}\}$

$L = L \setminus \{(\{5\}, 1)\} = \emptyset$

$\underline{1^{\text{a}} \text{ iteração } (1^{\text{o}} \text{ ciclo for}):}$

$Q_0 = \{1, 2, 3, 4\}$

$Q'_0 = \{1,4\}$

$P = P \setminus \{\{1,2,3,4\}\} \cup \{\{2,3\},\{1,4\}\} = \{\{6\},\{5\},\{2,3\},\{1,4\}\}$

$\underline{1^{\text{a}} \text{ iteração } (2^{\text{o}} \text{ ciclo for}):}$

$(Q_0, b) = (\{1,2,3,4\}, 0)$

Como $|Q'_0| = |\{1,4\}| \leq |Q_0 \setminus Q'_0| = |\{2,3\}|$ então

$L = L \cup \{(\{1,4\},0)\} = \{(\{1,4\},0)\}$

$\underline{2^{\text{a}} \text{ iteração } (2^{\text{o}} \text{ ciclo for}):}$

$(Q_0, b) = (\{1,2,3,4\}, 1)$

Como $|Q'_0| = |\{1,4\}| \leq |Q_0 \setminus Q'_0| = |\{2,3\}|$ então

$L = L \cup \{(\{1,4\},1)\} = \{(\{1,4\},0),(\{1,4\},1)\}$

$\underline{5^{\text{a}} \text{ iteração (while)}:}$

$(Q_1, a) = (\{1,4\}, 0)$

$P' = P = \{\{6\},\{5\},\{2,3\},\{1,4\}\}$

$L = L \setminus \{(\{1,4\},0)\} = \{(\{1,4\},1)\}$

$\underline{6^{\text{a}} \text{ iteração (while)}:}$

$(Q_1, a) = (\{1,4\}, 1)$

$P' = P = \{\{6\},\{5\},\{2,3\},\{1,4\}\}$

$L = L \setminus \{(\{1,4\},1)\} = \emptyset$

No fim da execução do algoritmo tem-se $P = \{\{6\},\{5\},\{2,3\},\{1,4\}\}$, ou seja tem-se quatro classes de equivalência: $\{6\}$, $\{5\}$, $\{2,3\}$ e $\{1,4\}$. Logo conclui-se que $1 \equiv 4$ e $2 \equiv 3$.

Exemplo da variante do algoritmo de Brzozowski

Este algoritmo é executado utilizando o autómato representado na figura 3.8. Para simplificar a notação utiliza-se a seguinte correspondência:

- a - corresponde ao estado $\{6\}$
- b - corresponde ao estado $\{5,6\}$
- c - corresponde ao estado $\{1,4,6\}$

- d - corresponde ao estado $\{2, 3, 5, 6\}$

- e - corresponde ao estado $\{1, 4, 5, 6\}$

- f - corresponde ao estado $\{1, 2, 3, 4, 5, 6\}$

Algoritmo. 3.1

Partição $= \{Q \setminus F, F\} = \{\{a, b, d\}, \{c, e, f\}\}$

Aux $= \{F\} = \{\{c, e, f\}\}$

1ª iteração (while):

$X = \{c, e, f\}$

Aux $= \emptyset$

Processado $= \{\{c, e, f\}\}$

1ª iteração (1º ciclo for):

$a = 0$

$Z = \delta_r(\{c, e, f\}, 0) = \{d, f\}$

$Z \notin$ Processado então Aux $= \{\{d, f\}\}$

2ª iteração (1º ciclo for):

$a = 1$

$Z = \delta_r(\{c, e, f\}, 1) = \{b, d, e, f\}$

$Z \notin$ Processado então Aux $= \{\{d, f\}, \{b, d, e, f\}\}$

1ª iteração (2º ciclo for):

$Y = \{a, b, d\}$

$K = X \cap Y = \emptyset$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}\}$

$Y \not\subseteq X$ então Partição $= \{\{a, b, d\}, \{c, e, f\}\}$

2ª iteração (2º ciclo for):

$Y = \{c, e, f\}$

$K = X \cap Y = \{c, e, f\}$

$K \neq \emptyset$ então Partição $= \{\{a, b, d\}, \{c, e, f\}\}$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}\}$

$Y \not\subseteq X$ então Partição $= \{\{a, b, d\}, \{c, e, f\}\}$

2ª iteração (while):

$X = \{d, f\}$

Aux $= \{\{b, d, e, f\}\}$

Processado $= \{\{c, e, f\}, \{d, f\}\}$

1ª iteração (1º ciclo for):

$a = 0$

$Z = \delta_r(\{d, f\}, 0) = \{c, e, f\}$

$Z \in$ Processado

2ª iteração (1º ciclo for):

$a = 1$

$Z = \delta_r(\{d, f\}, 1) = \{d, f\}$

$Z \in$ Processado

1ª iteração (2º ciclo for):

$Y = \{a, b, d\}$

$K = X \cap Y = \{d\}$

$K \neq \emptyset$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}\}$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}\}$

$Y \not\subseteq X$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}\}$

2ª iteração (2º ciclo for):

$Y = \{c, e, f\}$

$K = X \cap Y = \{f\}$

$K \neq \emptyset$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}\}$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}\}$

$Y \not\subseteq X$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}\}$

3ª iteração (2º ciclo for):

$Y = \{d\}$

$K = X \cap Y = \{d\}$

$K \neq \emptyset$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}\}$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}\}$

$Y \subseteq X$

4ª iteração (2º ciclo for):

$Y = \{f\}$

$K = X \cap Y = \{f\}$

$K \neq \emptyset$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}\}$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}\}$

$Y \subseteq X$

5ª iteração (2º ciclo for):

$Y = \{a, b\}$

$K = X \cap Y = \emptyset$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}, \{d, f\}\}$

$Y \not\subseteq X$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}, \{d, f\}\}$

6ª iteração (2º ciclo for):

$Y = \{c, e\}$

$K = X \cap Y = \emptyset$

$X \not\subseteq Y$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}, \{d, f\}\}$

$Y \not\subseteq X$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}, \{d, f\}\}$

7ª iteração (2º ciclo for):

$Y = \{d, f\}$

$K = X \cap Y = \{d, f\}$

$K \neq \emptyset$ então Partição $= \{\{a, b, d\}, \{c, e, f\}, \{d\}, \{f\}, \{a, b\}, \{c, e\}, \{d, f\}\}$

$X \subseteq Y$

$Y \subseteq X$

Exemplo da variante alterada do algoritmo de Brzozowski

Partição $= \{Q \setminus F, F\} = \{\{a, b, d\}, \{c, e, f\}\}$

Aux $= \{F\} = \{\{c, e, f\}\}$

1ª iteração (while):

$X = \{c, e, f\}$

Aux $= \emptyset$

Processado $= \{\{c, e, f\}\}$

1ª iteração (1º ciclo for):

$a = 0$

$Z = \delta_r(\{c, e, f\}, 0) = \{d, f\}$

$Z \notin$ Processado então Aux $= \{\{d, f\}\}$

2ª iteração (1º ciclo for):

$a = 1$

$Z = \delta_r(\{c, e, f\}, 1) = \{b, d, e, f\}$

$Z \notin$ Processado então Aux $= \{\{d, f\}, \{b, d, e, f\}\}$

1ª iteração (2º ciclo for):

$Y = \{a, b, d\}$

$K = X \cap Y = \emptyset$

2ª iteração (2º ciclo for):

$Y = \{c, e, f\}$

$K = X \cap Y = \{c, e, f\}$

2ª iteração (while):

$X = \{d, f\}$

Aux $= \{\{b, d, e, f\}\}$

Processado $= \{\{c, e, f\}, \{d, f\}\}$

1ª iteração (1º ciclo for):

$a = 0$

$Z = \delta_r(\{d, f\}, 0) = \{c, e, f\}$

$Z \in$ Processado

2ª iteração (1º ciclo for):

$a = 1$

$Z = \delta_r(\{d, f\}, 1) = \{d, f\}$

$Z \in$ Processado

1ª iteração (2º ciclo for):

$Y = \{a, b, d\}$

$K = X \cap Y = \{d\}$

Partição $= \{\{c, e, f\}\}$

Partição $= \{\{c, e, f\}, \{a, b\}, \{d\}\}$

2ª iteração (2º ciclo for):

$Y = \{c, e, f\}$

$K = X \cap Y = \{f\}$

Partição $= \{\{a, b\}, \{d\}\}$

Partição $= \{\{a, b\}, \{d\}, \{c, e\}, \{f\}\}$

3ª iteração (2º ciclo for):

$Y = \{a, b\}$

$K = X \cap Y = \emptyset$

4ª iteração (2º ciclo for):

$Y = \{d\}$

$K = X \cap Y = \{d\}$

5ª iteração (2º ciclo for):

$Y = \{c, e\}$

$K = X \cap Y = \emptyset$

6ª iteração (2º ciclo for):

$Y = \{f\}$

$K = X \cap Y = \{f\}$

Bibliografia

[1] M. Almeida, N. Moreira, and R. Reis. On the performance of automata minimization algorithms. *In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, CiE 2008: Abstracts and extended abstracts of unpublished papers*, 2008.

[2] M. J. Atallah. *Algorithms and Theory of Computation Handbook*. CRC Press LLC, 1999.

[3] J.-M. Champarnaud, A. Khorsi, and T. Paranthoën. Split and join for minimizing: Brzozowski's algorithm. Technical report, 2002.

[4] R. Greenlaw and H. J. Hoover. *Fundamentals of the Theory of Computation Principles and Practice*. Morgan Kaufmann Publishers, Inc., 1998.

[5] D. Gries. Describing an algorithm by hopcroft. *Acta Informática Vol. 2 Num. 2*, pages 97–109, 1972.

[6] J. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford University, 1971.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[8] M. R. Rodrigues. *Acetatos das aulas de Teoria da Computação*. Universidade de Aveiro, 2006/2007.

[9] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[10] B. W. Watson. A taxonomy of finite automata minimization algorithms. Technical report, Eindhoven University of Technology, 1994.

[11] D. Wood. *Theory of Computation*. John Wiley & Sons, Inc, 1987.