Teoria da Computação **FCT-UNL, 2º semestre 2023-2024**

Notas 6: Minimização de AFDs

Autor: João Ribeiro

Introdução

Nestas notas estudamos formas de minimizar o número de estados de um AFD que reconhece uma dada linguagem. O acesso ao AFD "minimal" para uma linguagem é útil na prática, pois leva a um menor esforço computacional em várias tarefas que usam AFDs, tal como a pesquisa de padrões em texto. Surpreendentemente, existem algoritmos de minimização de AFDs eficientes.

6.1 Optimização de AFDs

O número de estados de um AFD controla o tamanho da sua descrição em memória, bem como a quantidade de memória usada durante a computação do AFD. Certamente já terão notado que existem muitos (de facto, um número infinito de) AFDs diferentes que aceitam a mesma linguagem. Como AFDs são usados como componentes de muitos algoritmos, é importante dispormos de métodos eficientes que minimizam o número de estados do AFD que realiza uma dada tarefa (isto é, que aceita uma dada linguagem).

Mais precisamente, o nosso objectivo é o seguinte: Dado um AFD M, construir um AFD M' equivalente a M com um número *mínimo de estados*. Este AFD minimal (que é único) é obtido através de duas optimizações.

6.1.1 Estados inúteis

A primeira, e mais simples, optimização consiste na remoção de *estados inúteis*. Um estado q de um AFD M diz-se *inútil* se não é acessível a partir do estado inicial de M, ou se o estado final de M não é acessível a partir de q.

Por exemplo, os estados u e v do AFD da Figura 6.1 são inúteis, pois u não é acessível a partir do estado inicial s, e o estado final t não é acessível a partir de v.

Claramente, o AFD M' obtido a partir de M através da remoção de todos os estados inúteis é equivalente a M. Mais precisamente, se $M = (S, \Sigma, \delta, s, F)$ e $q \in S$ é inútil, construímos o AFD equivalente $M' = (S', \Sigma, \delta', s, F)$ com conjunto de estados $S' = S \setminus \{q\}$ e função de transição $\delta' : S' \times \Sigma \to S'$ dada por

$$\delta'(t, a) = \begin{cases} \delta(t, a), & \text{se } \delta(t, a) \neq q, \\ \perp, & \text{se } \delta(t, a) = q. \end{cases}$$

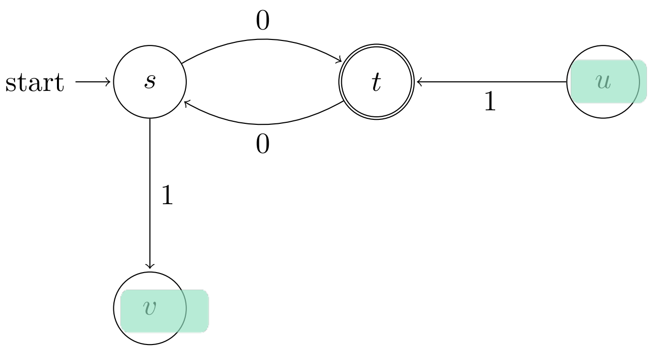


Figure 6.1: Diagrama de estados de um AFD com estados inúteis.

A aplicação desta operação ao AFD da Figura 6.1 resulta no AFD equivalente descrito na Figura 6.2.

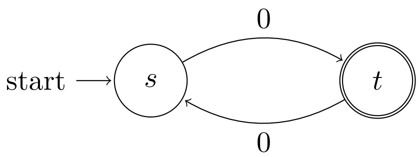


Figure 6.2: Diagrama de estados de AFD obtido pela remoção de estados inúteis do AFD da Figura 6.1.

6.1.2 Estados equivalentes

A segunda, e mais importante, optimização consiste na combinação de *estados equivalentes*. Dado um AFD $M = (S, \Sigma, \delta, s, F)$, dois estados t e u dizem-se *equivalentes* se para qualquer $w \in \Sigma^*$ temos

$$\delta^*(t, w) \in F \iff \delta^*(u, w) \in F.$$

Por palavras, dois estados t e u de um AFD M são equivalentes se a execução de qualquer input w a partir de t leva ao mesmo resultado que a execução de w a partir de u. Quando dois estados t e u de M são equivalentes, podemos escrever $t \equiv_M u$.

Quando dois estados t e u não são equivalentes, dizemos que estes são *distinguíveis*, e escrevemos $t \not\equiv_M u$. Se t e u são distinguíveis, isto significa que tem de existir uma string $w \in \Sigma^*$ tal que ou $\delta^*(t, w) \in F$ e $\delta^*(u, w) \notin F$, ou $\delta^*(t, w) \notin F$ e $\delta^*(u, w) \in F$. Neste caso, dizemos que t e u são distinguíveis por w.

Por exemplo, os estados t e u do AFD descrito na Figura 6.3 são equivalentes. A aplicação da operação acima a este par de estados resulta no AFD equivalente com menos um estado descrito na Figura 6.4.

A noção de equivalência de estados de um AFD é importante pela seguinte razão: Se t e u são estados equivalentes de M, então podemos "combiná-los" num só estado, eliminando, por exemplo, u e redireccionando todas as transições de M que levam a u para t. O AFD M' resultante é equivalente a M e tem menos um estado que M.

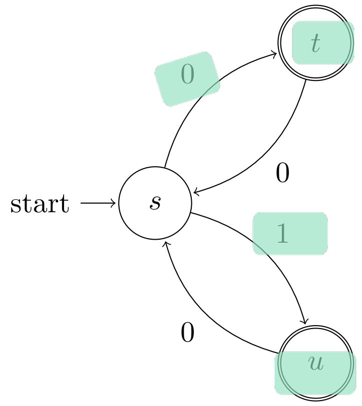


Figure 6.3: Diagrama de estados de um AFD com estados equivalentes.

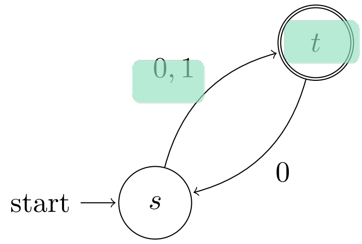


Figure 6.4: Diagrama de estados do AFD obtido por combinação dos estados equivalentes do AFD da Figura 6.3.

Mais formalmente, se $M = (S, \Sigma, \delta, s, F)$ e $t \equiv_M u$ para alguns estados $t, u \in S$ com $u \neq s$ (i.e., u não é o estado inicial de M), construímos $M' = (S', \Sigma, \delta', s, F')$ com conjunto de estados $S' = S \setminus \{u\}$, conjunto de estados finais $F' = F \setminus \{u\}$, e função de transição $\delta' : S' \times \Sigma \to S'$ dada por

$$\delta'(v, a) = \begin{cases} \delta(v, a), & \text{se } \delta(v, a) \neq u, \\ t, & \text{se } \delta(v, a) = u. \end{cases}$$

Relações de equivalência associadas a linguagens. A noção de equivalência entre estados de um AFD que discutimos nesta secção está intimamente relacionada com outra relação de equivalência fundamental no estudo das linguagens regulares. Dada uma linguagem regular $L \subseteq \Sigma^*$, dizemos que duas strings x e y são indistinguíveis por L se para qualquer $z \in \Sigma^*$ temos que $xz \in L \iff yz \in L$, o que denotamos por $x \equiv_L y$.

A relação $\equiv_L$ é uma relação de equivalência sobre Σ^*, e portanto particiona Σ^* em *classes de equivalência* disjuntas, i.e., existem conjuntos disjuntos $S_1, S_2, \ldots, S_k$ (as classes de equivalência) tal que $\Sigma^* = S_1 \cup S_2 \cup \cdots \cup S_k$ e $x \equiv_L y$ se e só se x e y pertencem à mesma classe de equivalência, i.e., se e só se existe i tal que x e y pertencem ao mesmo conjunto S_i.

As ideias presentes no algoritmo de minimização de AFDs que discutimos aqui permitem também mostrar uma relação próxima entre $\equiv_L$ e o AFD minimal que aceita L. Em particular, o número de classes de equivalência da relação $\equiv_L$ corresponde exactamente ao número de estados do AFD minimal. Este é o celebrado teorema de Myhill-Nerode [Ner58]. Como o nosso tempo é limitado, não estudaremos este teorema em aula. No entanto, a sua demonstração e exemplos da sua utilidade serão lançados como desafios aos alunos.

6.2 Algoritmo de minimização

Tendo em conta a discussão acima, um algoritmo de minimização para AFDs tem dois passos, dado um AFD M:

1. Primeiro, remover todos os estados inúteis de M, resultando num AFD equivalente M' sem estados inúteis;
2. Segundo, encontrar todos os pares de estados equivalentes de M' e combiná-los. O AFD resultante é o AFD minimal equivalente a M.

O primeiro passo é simples de implementar, e portanto focamo-nos no segundo passo. Assumimos que M não tem estados inúteis, e discutimos um procedimento eficiente (com respeito ao número de estados de M) para encontrar todos os pares de estados equivalentes de M. Este procedimento baseia-se na seguinte observação.

Lema 6.1 *Seja $M = (S, \Sigma, \delta, s, F)$ um AFD, (t, u) um par de estados distinguíveis de M, e (q, r) um par de estados tal que*

$$\delta(q, a) = t \qquad e \qquad \delta(r, a) = u$$

para algum $a \in \Sigma$. Então q e r são distinguíveis.

Demonstração: Por hipótese, t e u são distinguíveis por algum $w \in \Sigma^*$. Basta agora observar que q e r são distinguíveis pela string aw, pois

$$\delta^*(q, aw) = \delta^*(\delta(q, a), w) = \delta^*(t, w)$$

e

$$\delta^*(r, aw) = \delta^*(\delta(r, a), w) = \delta^*(u, w).$$

??

■

O algoritmo procede da seguinte forma:

1. **Inicialização:** Inicializamos o conjunto de pares de estados distinguíveis D com os pares de estados "obviamente" distinguíveis. Estes são os pares (t, u) com $t \in F$ e $u \notin F$, que são distinguíveis pela string vazia ε, e também os pares (t, u) que não têm as mesmas transições definidas, ou seja, para os quais existe $a \in \Sigma$ tal que $\delta(t, a) = \perp$ e $\delta(u, a) \neq \perp$.
2. **Actualização do conjunto D:** Procuramos todos os pares de estados (q, r) para os quais existe um símbolo $a \in \Sigma$ e um par de estados $(t, u) \in D$ tal que $\delta(q, a) = t$ e $\delta(r, a) = u$. Adicionamos cada um destes pares (q, r) ao conjunto D.

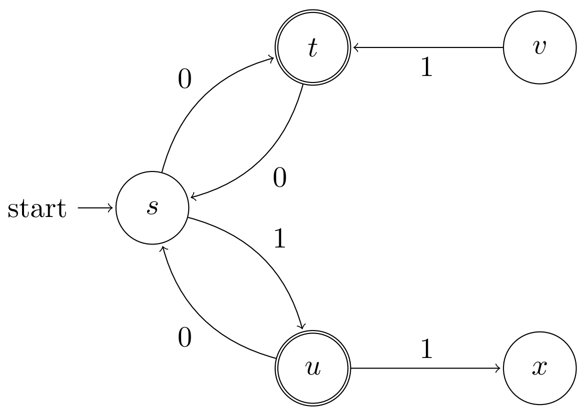
3. Se o Passo 2 adicionou novos pares de estados a D, repetimos o Passo 2. Caso contrário, o algoritmo pára. Quando isto acontece, sabemos que dois estados t e u são equivalentes se e só se $(t, u) \notin D$. Para cada par de estados equivalentes (t, u) usamos o método da Secção 6.1.2 para os combinar num só estado.

Vamos analisar a correcção deste algoritmo. Primeiro, notamos que o Lema 6.1 garante que todos os pares (q, r) adicionados a D nas várias iterações do Passo 2 são distinguíveis.

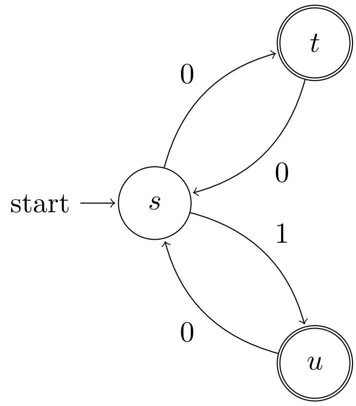
Resta argumentar que este processo adiciona, eventualmente, *todos* os pares de estados distinguíveis a D. Para simplificar a nossa discussão, focamo-nos no caso especial em que o AFD M é *completo*, i.e., a sua função de transição δ é total. O argumento que apresentamos agora pode ser facilmente estendido ao caso mais geral. Quando o AFD M é completo, o Passo 1 (inicialização) apenas adiciona a D os pares (t, u) com $t \in F$ e $u \notin F$. Tomamos uma perspectiva alternativa útil: A inicialização do algoritmo adiciona a D todos os pares de estados distinguíveis por uma string de tamanho 0 (a string vazia ε). Sob esta perspectiva, concluímos que a primeira actualização do conjunto D no Passo 2 adiciona todos os pares de estados distinguíveis por strings de tamanho 1. Seguindo este raciocínio, concluímos que, após n repetições do Passo 2, adicionámos a D todos os pares de estados distinguíveis por strings de tamanho menor ou igual a n. Como cada par de estados distinguíveis é distinguível por alguma string de tamanho finito, concluímos que todos os pares serão eventualmente adicionados a D, e portanto, no final do algoritmo, D contém exactamente os pares de estados distinguíveis.

6.2.1 Exemplos de minimização de AFDs

Exemplo 6.1 Vamos minimizar o AFD abaixo usando o algoritmo que discutimos acima.



Primeiro, começamos por identificar e remover os estados inúteis. O estado v não é acessível a partir do estado inicial e nenhum estado final é acessível a partir do estado x. Ao removermos estes estados obtemos o AFD equivalente abaixo, que não tem estados inúteis.



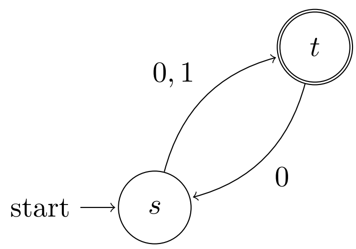
Queremos agora encontrar o conjunto D dos pares de estados distinguíveis (o que implica que também encontramos todos os pares de estados indistinguíveis). Seguindo Passo 1 do algoritmo acima, inicializamos D como

$$D = \{(s,t), (s,u)\},$$

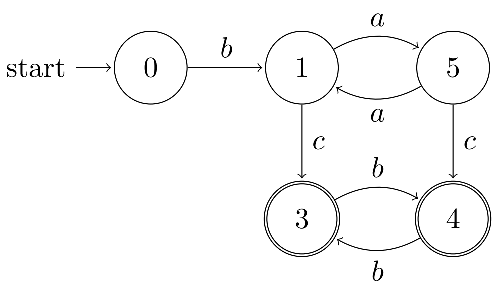
pois t e u são finais mas s não é. De seguida, temos de encontrar todos os pares de estados que não pertencem a D e que se transformam num par de D através de alguma transição comum. Ora, o único par de estados que ainda não está em D é (t,u). Se realizarmos uma transição-0 de t e u, obtemos o par de estados

$$\begin{pmatrix} t \\ u \end{pmatrix} \xrightarrow{0} \begin{pmatrix} s \\ s \end{pmatrix}.$$

Como não há transições-1 definidas a partir de t e u e como $(s,s) \notin D$, não adicionamos (t,u) a D. Como o conjunto D não se modificou por aplicação deste passo, o algoritmo termina e concluímos que (t,u) é um par de estados indistinguíveis. Combinamos os estados t e u para obter o AFD equivalente minimal abaixo.



Exemplo 6.2 Vamos minimizar o AFD abaixo.



Começamos por observar que não existem estados inúteis neste AFD. Portanto, passamos para a procura do conjunto D de pares de estados distinguíveis.

No primeiro passo doalgoritmo inicializamos D como

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

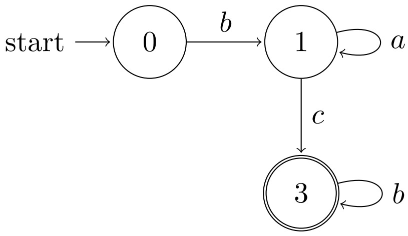
Observamos que só existem dois pares de estados que actualmente não pertencem a D: estes são $(1,5)$ e $(3,4)$. Vamos actualizar D pelo Passo 2 do algoritmo. Para o par $(1,5)$ temos as seguintes transições definidas:

$$\begin{pmatrix}1\\5\end{pmatrix} \xrightarrow{a} \begin{pmatrix}5\\1\end{pmatrix}$$
$$\begin{pmatrix}1\\5\end{pmatrix} \xrightarrow{c} \begin{pmatrix}3\\4\end{pmatrix}.$$

Como nenhum dos pares resultantes pertence a D, não adicionamos $(1,5)$ a D. Para o par $(3,4)$ temos a seguinte transição definida:

$$\begin{pmatrix}3\\4\end{pmatrix} \xrightarrow{b} \begin{pmatrix}4\\3\end{pmatrix}.$$

Portanto, também não adicionamos o par $(3,4)$ a D. Como D não mudou depois da actualização, o algoritmo termina e concluímos que $(1,5)$ e $(3,4)$ são pares de estados indistinguíveis. Ao combinar cada um destes pares obtemos o AFD equivalente minimal abaixo.



6.2.2 E se quisermos minimizar AFNs?

Vimos acima como podemos minimizar de forma eficiente o número de estados de um AFD para uma dada linguagem. É natural questionarmo-nos se o mesmo é possível no contexto de AFNs:

Dado um AFN M que aceita uma dada linguagem L, é possível construir eficientemente o AFN minimal de L?

Notavelmente, não conhecemos tal procedimento para AFNs. Mais ainda, a existência de um algoritmo de minimização eficiente para AFNs falsificaria conjecturas importantes de teoria da complexidade (que muitos investigadores acreditam serem verdadeiras, e que sem as quais muita da nossa criptografia seria quebrável) [JR93].

6.3 Para explorar

O primeiro algoritmo de minimização de AFDs foi introduzido por Moore [Moo56] (aplicado ao conceito de máquinas de Moore). Mais tarde, Hopcroft [Hop71] desenhou e analisou um algoritmo de minimização de AFDs mais eficiente.

Uma discussão mais aprofundada sobre a minimização de AFDs é apresentada em [LP97, Section 2.5].

References

[Hop71] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971. Preliminary technical report available at `http://i.stanford.edu/pub/cstr/reports/cs/tr/71/190/CS-TR-71-190.pdf`.

[JR93] Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[LP97] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, USA, 2nd edition, 1997.

[Moo56] Edward F. Moore. Gedanken-experiments on sequential machines. *Automata Studies*, 34:129–153, 1956.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.