1

Universidade do Minho

Escola de Engenharia

Departamento de Informática

Largo do Paço - 4709 BRAGA Codex- PORTUGAL

Tel +351-53-604470- Fax +351-53-612954

Elementos de Teoria dos Autómatos

Luís Soares Barbosa

Monografia
DI-LSB-96:5:1

Maio 1996

Elementos de Teoria dos Autómatos

Luís Soares Barbosa
lsb@di.uminho.pt

Departamento de Informática
Universidade do Minho
Largo do Paço
4709 BRAGA Codex
PORTUGAL

Maio 1996

[1]Parte deste texto foi escrito para apoiar a disciplina de LINGUAGENS DE PROGRAMAÇÃO, da LIG, em 1993. Agradeço reconhecidamente a colaboração do J. E. Pina Miranda na sua leccionação e os seus comentários ao texto que, ao correr das aulas, ia sendo escrito.

Índice

Capítulo 1

Introdução

1.1 De Que Falamos?

O objectivo de uma linguagem é estabelecer um padrão de interacção entre objectos distintos. A noção é-nos familiar: a interacção humana com a realidade é mediada por um vasto conjunto de linguagens (*eg*, verbais, musicais, corporais, matemáticas, etc.) e esta (re-)faz-se precisamente na medida em que é codificada num suporte simbólico determinado. Tal codificação é, no caso das linguagens especificamente humanas, um processo complexo, de natureza social, onde forma e significado se entretecem numa interacção dinâmica e essencialmente *frágil*. A ambiguidade inerente a uma linguagem natural — como o português — radica nessa fragilidade, que é, paradoxalmente, a fonte do seu poder expressivo[1].

Abordaremos aqui um tipo muito restrito de linguagens para as quais *forma* e *significado* podem ser completamente definidos — através de regras análogas às de um jogo — estabelecendo com clareza as formas de interação possível. Classificaremos tais linguagens como *formais*, no sentido em que qualquer controvérsia sobre a forma ou o sentido de uma frase pode ser elucidada por recurso a um seu modelo formal (matemático)[2]. A interação com (e entre) dispositivos físicos — por exemplo, um controlador de semáforos, uma rede de dados ou um computador — baseia-se em linguagens deste tipo e será esse o âmbito do nosso interesse.

O estudo das linguagens formais faz-se a dois níveis: sintático e semântico. Por *sintaxe* entendemos a especificação das sequências de símbolos que constiuem as palavras e frases válidas da linguagem. No que às linguagens de programação concerne, esta especificação é feita de dois modos: por um documento escrito, para garantir a efectiva interacção com as pessoas, e por um interpretador ou compilador, para assegurar a correcta interacção com a máquina. Muito mais complexa é a definição da *semântica*, que lida com o significado dessas frases (ditas sintaticamente válidas ou bem formadas). Neste texto, porém, versaremos exclusivamente os aspectos sintáticos.

O problema básico associado à sintaxe das linguagens é o do seu *reconhecimento* por uma

[1] *A inexcedível clareza das linguagens não poéticas é de uma substancial negrura.* Eduardo Lourenço, in *Tempo e Poesia*, 1957.

[2] O assunto não é por isso menos "nobre": *O binómio de Newton é tão belo como a Vénus de Milo. O que há é menos gente para dar por isso.* Álvaro de Campos, in *Poesias*, 1935.

máquina, do qual depende a efectiva construção de tradutores (*eg*, interpretadores ou compiladores). O que nos leva directamente à Teoria dos Autómatos.

A *Teoria dos Autómatos* definiu-se, na sua génese, como o estudo do comportamento macroscópico dos sistemas discretos (*cf.*, por exemplo, [Boo67]). O seu objectivo é a concepção de *modelos matemáticos de máquinas abstractas* que descrevam processos de transformação (*i.é*, computacionais) observados em diferentes áreas do conhecimento. Por meados deste século, a "visão cibernética" [Tur50, McC65] teve um impacto enorme em muitas áreas de investigação fundamental e aplicada. Não apenas na génese da computação digital, mas também na biologia, na psicologia e na bioquímica. Questões do tipo *Como se organizam as redes nervosas?*, *De que forma uma célula se auto-controla?* ou *Como crescem e se organizam os seres vivos?* foram abordadas, com maior ou menor sucesso, pela formulação de modelos matemáticos dos mecanismos que se suponham envolvidos.

A noção de *autómato* é, contudo, muito mais antiga e tem raízes nas doutrinas do determinismo que dominaram a filosofia das ciências nos séculos XVII e XVIII. Na época, a progressiva compreensão da fisíca, construiu uma visão dos processos naturais, do comportamento dos animais e do homem e da própria dinâmica do Universo, essencialmente mecanicista. Se um particular comportamento é a expressão macroscópica de um mecanismo interno bem determinado, então podemos compreendê-lo construindo modelos (abstractos) desse mecanismo. Tais modelos, pensava-se, podiam guiar a reprodução mecânica das "máquinas" da Natureza. Tal era, por exemplo, a visão cartesiana.

Apesar dos argumentos usados serem discutíveis (o determinismo é largamento contestado quer no comportamento humano quer na própria realidade fisíca [Pop56]), a noção de autómato revelou-se fundamental na concepção dos computadores e na compreensão dos limites da computabilidade. De facto, o conceito tornou-se muito mais relevante para as Ciências da Computação (e para a própria Matemática) do que para a biologia ou a psicologia.

No contexto em que nos situamos, um autómato é um modelo matemático de uma máquina abstracta que reconhece um conjunto de palavras sobre um determinado alfabeto, conjunto esse que designamos por *linguagem* aceite ou reconhecida pelo autómato. Neste sentido, um autómato constitui ua descrição finita e rigorosa de um, eventualmente não finito, conjunto de palavras — *i.é*, de uma linguagem.

Numerosos processos computacionais são baseados na noção simples de *autómato finito*, assim chamado porque se assume a finitude dos estados internos que a máquina pode alcançar. Citemos, por exemplo, os algoritmos para a identificação de palavras num editor de texto, os verificadores de ortografia e os analisadores léxicos usados nos tradutores de linguagens de programção. Processos (ou linguagens) mais elaborados exigem generalizações do conceito base — tipicamente a progressiva estruturação do espaço de estados.

Referimos a associação entre a noção de autómato e a de linguagem. Numa perspectiva complementar, contudo, os autómatos surgem como objectos matemáticos que justificam um estudo próprio. Tendo sido um dos primeiros modelos da computação, estão, a nosso ver, na gênese de dois conceitos fundamentais em ciências da computação: a de *estrutura de dados* (em que o autómato é codificado num determinado universo algébrico, por exemplo, na categoria dos conjuntos) e a de *comportamento* (tal como é entendida no estudo da semântica da concorrência [WN93] e, em particular, nas álgebras de processos [Hen88, Mil80])

Embora a perspectiva dominante neste texto seja o estudo do reconhecimento e classificação das linguagens formais, procuraremos, no último capítulo, reler a definição de autómato enquanto modelo semântico do comportamento dos sistemas. Para isso iremos relaxar algumas cláusulas da definição e, por outro lado, sofisticar as estruturas de observação associadas (que, até lá, identificaremos exclusivamente com a de linguagem reconhecida). Chegaremos, assim, à definição de *sistema de transição* [WN93, Rut95], com a qual podemos estabelecer uma ponte entre o estudo da teoria clássica dos autómatos e das linguagens e os cálculos de processos.

1.2 Como Falamos?

Neste texto são considerados, de forma necessariamente abreviada, os conceitos e resultados fundamentais da teoria dos autómatos de estado finito (Capítulo 2) e de estado recorrente (Capítulo 4), que constituem os modelos *cognitivos* básicos de um vasto conjunto de linguagens formais. A qualificação de cognitivo indica que a especificação da linguagem é feita pela construção de um seu reconhecedor.

No Capítulo 3 introduziremos a noção de *gramática* como um modelo alternativo para a especificação de linguagens. O modelo é, agora, de natureza *generativa* — a linguagem é especificada por uma descrição do processo de geração das palavras e frases válidas. A noção de gramática surge nas Ciências da Computação por transferência de categorias usadas no estudo das línguas naturais a partir do anos 50, nomeadamente dos trabalhos pioneiros de N. Chomsky [Cho56, Cho69]. Procuraremos, pois, analisar a equivalência entre autómatos e gramáticas.

No Capítulo 5 é introduzida a noção de máquina de Turing, a sua relação com o problema da computabilidade e o estudo da complexidade da computação ([Pap94] é uma introdução excelente a estas questões). A teoria das máquinas de Turing, juntamente com as visões complementares do λ-calculus ([Bar80] é a referência enciclopédica; [Han94] uma introdução muito agradável) e da teoria das funções recursivas (o capítulo 4 de [Joh82] introduz as definições básicas), ocupa um lugar central nos fundamentos da computação. Este capítulo pretende ser apenas uma motivação para o seu aprofundamento.

Por fim, no Capítulo 6, estuda-se a referida generalização da noção clássica de autómato à de *sistema de transição*. De uma forma simples, podemos caracterizar estas estruturas como autómatos onde não se especificam estados finais e sobre os quais é possível definir noções de *observação* mais elaboradas do que a clássica "*linguagem aceite*".

Inicialmente concebido para fins didáticos, o texto não reclama qualquer originalidade, a não ser, talvez, na selecção, organização e apresentação das matérias. Deve, pois, ser tomado como uma introdução ao tema e um guia para a abordagem de obras mais elaboradas. Entre os muitos livros de texto publicados nesta área, salientamos [Eil74], [HU79], [Hol82], [CL89] e [How91]. Os três primeiros são monografias clássicas. A obra de Carrol e Long é essencialmente didática, com muitos exemplos e a preocupação por efectuar paulatinamente as demonstrações dos principais resultados. A última referência é uma obra concisa e agradável, mas mais aconselhável ao leitor com alguma experiência na leitura de texto matemático.

Capítulo 2

Autómatos Finitos

2.1 Linguagens

Comecemos por precisar a noção de linguagem e definir algumas operações associadas.

Definição 2.1.1. Dado um conjunto Σ finito e não vazio de símbolos — que designamos por **alfabeto** — diz-se que qualquer sequência de símbolos $s_1 s_2 ... s_n$, tal que cada $s_i \in \Sigma$, constitui uma **palavra sobre** Σ. Designa-se por **palavra vazia** e representa-se por λ, a sequência constituida por zero símbolos de Σ.

Definição 2.1.2. Designa-se por **comprimento** de uma palavra ω sobre um alfabeto Σ^*, e representa-se por $|\omega|$, o número total de ocorrências de cada símbolo em ω. O número de ocorrências de um símbolo particular $a \in \Sigma$ em ω é representado por $|\omega|_a$.

Podemos, agora, definir, para cada alfabeto Σ, o conjunto das palavras com um comprimento determinado. Formalmente

$$\Sigma^k \triangleq \{\omega \in \Sigma^* \mid |\omega| = k\}$$

Assim como o conjunto das palavras com comprimento positivo,

$$\Sigma^+ \triangleq \bigcup_{n \geq 1} \Sigma^n$$

e, obviamente,

$$\Sigma^* \triangleq \Sigma^+ \cup \{\lambda\}$$

Concluimos, pois, que o conjunto de todas as palavras sobre Σ corresponde ao conjunto Σ^* das sequências finitas e não finitas de símbolos de Σ.

A operação binária fundamental sobre Σ^* é a concatenação de palavras.

Definição 2.1.3. Dadas duas palavras $\omega_1 = x_1x_2...x_n$ e $\omega_2 = y_1y_2...y_m$ sobre Σ, define-se a sua **concatenação**, representada por $\omega_1 \cdot \omega_2$ ou, simplesmente por, $\omega_1\omega_2$, como a sequência formada pela justaposição dos símbolos em ω_1 e ω_2, *i.é*,

$$\omega_1 \cdot \omega_2 \;=\; x_1x_2...x_ny_1y_2...y_m$$

Facto 2.1.1. A estrutura $(\Sigma^*; \cdot, \lambda)$ é um monoide não comutativo.

Uma linguagem é um conjunto, possivelmente não finito, de palavras sobre um alfabeto. Mais precisamente,

Definição 2.1.4. Dado um alfabeto Σ, qualquer subconjunto $\mathcal{L}$ de Σ^* é dito uma **linguagem sobre** Σ.

Da definição acima se infere que todas as operações usuais sobre conjuntos são igualmente operações sobre linguagens — das quais a mais importante será sem dúvida a reunião. Por outro lado, a estrutura monoidal de Σ^* justifica a introdução de duas novas operações. Note-se que ambas se definem à custa da concatenação de palavras dada acima.

Definição 2.1.5. Dadas duas linguagens $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$, define-se o **produto** de $\mathcal{L}_1$ por $\mathcal{L}_2$ como o conjunto

$$\mathcal{L}_1\mathcal{L}_2 \;\triangleq\; \{\omega_1\omega_2 |\, \omega_1 \in \mathcal{L}_1 \,\wedge\, \omega_2 \in \mathcal{L}_2\}$$

De acordo com esta definição cada palavra de $\mathcal{L}_1\mathcal{L}_2$ é formada por uma palavra de $\mathcal{L}_1$ seguida de uma palavra de $\mathcal{L}_2$, pelo que o produto é também, por vezes, referido como concatenação de linguagens. A segunda operação que consideramos consiste igualmente na concatenação de palavras, mas, agora, oriundas da mesma linguagem e em número arbitrário. Formalmente,

Definição 2.1.6. Dada a lingugem $\mathcal{L} \subseteq \Sigma^*$, define-se o **fecho de Kleene** de $\mathcal{L}$, como o conjunto

$$\mathcal{L}^* \;\triangleq\; \{\omega_1\omega_2\omega_3...\omega_n |\, n \geq 0 \,\wedge\, \omega_1, \omega_2, \omega_3, ..., \omega_n \in \mathcal{L}\}$$

Note-se que λ pertence sempre ao fecho de Kleene de uma qualquer linguagem $\mathcal{L}$ — mesmo quando se tem $\lambda \notin \mathcal{L}$. Genericamente, $\mathcal{L}^*$ é a reunião de produtos sucessivos de $\mathcal{L}$ consigo própria. Usando a notação familar da exponenciação de conjuntos, podemos representar $\{\lambda\}$ por $\mathcal{L}^0$, $\mathcal{L}\mathcal{L}$ por $\mathcal{L}^2$ e assim sucessivamente. Então,

$$\mathcal{L}^* \;=\; \mathcal{L}^0 \cup \mathcal{L} \cup \mathcal{L}^2 \cup \mathcal{L}^3 \cup ...$$

De facto, Σ^* pode ser visto como o fecho de Kleene da linguagem formada pelas palavras compostas por um único $x \in \Sigma$, para todo o $x \in \Sigma$. Tal linguagem é, obviamente, um conjunto isomorfo a Σ, pelo que não usaremos uma notação distinta para nos referirmos ao símbolo x de Σ e à palavra x de Σ^*.

2.2 Automatos Determinísticos e Linguagens Regulares

A noção de autómato será introduzida por um pequeno exemplo, que ilustrará, também, a notação diagramática com que vulgarmente se faz a sua descrição.

Exemplo 2.2.1. Consideremos um sistema de reconhecimento de palavras de acesso constituidas por combinações dos caracteres a, b e c e que mantém, de momento, informação sobre 4 utilizadores, de acordo com a seguinte tabela.

Utilizador	Acesso
ut1	aab
ut2	bcb
ut3	acb
ut4	ba

Os estímulos que chegam ao sistema são precisamente os símbolos do alfabeto $\Sigma = \{a, b, c\}$. Este, por outro lado, deve memorizar um número (finito) de situações. Em particular, manterá um registo dos caracteres premidos até cada instante. Tais níveis de conhecimento constituem os estados do sistema, que se representam por pequenos círculos. Inicialmente o sistema estará no estado 1 que representa o facto de nenhuma tecla ter sido premida. Diagramaticamente, este estado inicial representa-se por um círculo com uma seta incidente livre (*i.é*, sem origem). Ao ser premida uma tecla correspondente a um símbolo válido a, o sistema transita para o estado 2. Este modela a situação “foi reconhecido o símbolo a”. À transição corresponde, no diagrama, um arco dirigido e etiquetado com o estímulo que a activa. O estado correspondente ao reconhecimento de uma palavra de acesso válida é dito final e representado no diagrama por um duplo círculo. Qualquer palavra que conduz o sistema a um estado final, a partir do estado inicial, diz-se aceite pelo sistema. O conjunto de palavras aceites constitui a linguagem aceite (ou reconhecida). Neste caso será, como esperado, o conjunto $\{aab, acb, bcb, ba\}$. Na Figura 2.2.1 representa-se o diagrama que descreve o sistema que temos vindo a discutir. É a este tipo de sistemas que chamaremos *autómatos*.

Que elementos podemos retirar deste exemplo como constituintes básicos de um autómato?

Antes de mais assume-se um alfabeto Σ sobre o qual se define a lingugem a reconhecer e um conjunto de estados S (necessáriamente finito) correspondentes à progressão do processo de reconhecimento. Cada estado representa uma etapa nesse processo. A passagem de um estado a outro é etiquetada por um ou mais símbolos do alfabeto, significando que é efectuada perante o reconhecimento de um desses símbolos. Note-se que o estado alcançado pelo reconhecimento de um símbolo $x \in \Sigma$ é determinado univocamente por x e pelo estado de partida. Por essa determinação ser única, dizemos que o autómato é *determinístico*.

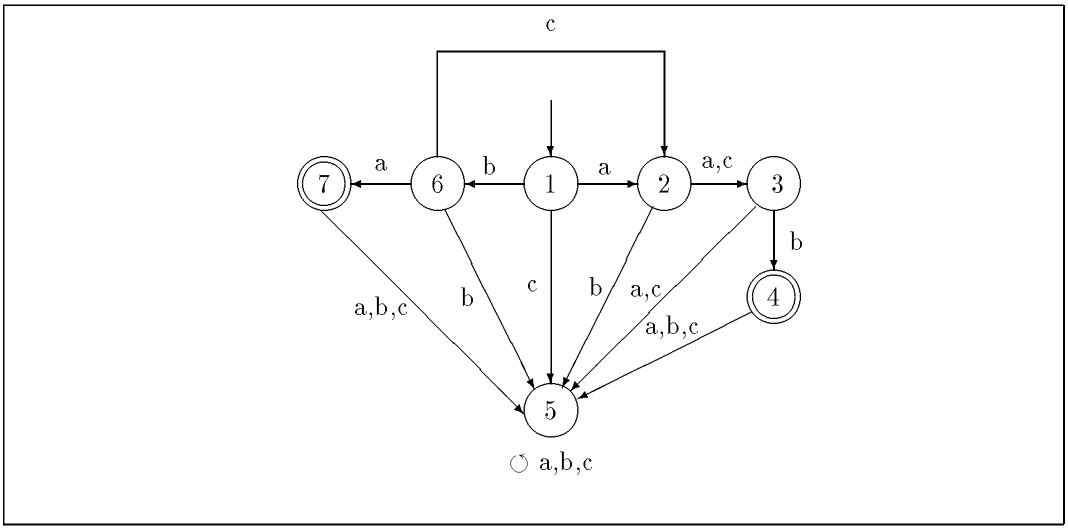

Figura 2.1: Reconhecedor de palavras de acesso

Naturalmente, a entidade matemática adequada para modelar este tipo de transição é o espaço das funções do produto cartesiano $\Sigma \times S$ para S.

Distinguimos, ainda, um estado onde o processo de reconhecimento se inicia. Estando disponível antes do processamento de qualquer símbolo, podemos dizer que é alcançado pela palavra vazia — λ. Identificamos, também, determinados estados em que esse processo termina com sucesso e que nos indicam as palavras aceites pelo autómato. É o conjunto dos estado finais. Podemos sumariar estas características na seguinte definição.

Definição 2.2.1. Um **Autómato Finito Determinístico** é um tuplo $(\Sigma, S, s_0, \delta, F)$ onde Σ é um alfabeto, S um conjunto finito e não vazio de estados, $s_0 \in S$ o estado inicial, $\delta : S \times \Sigma \to S$ a função de transição de estados e $F \subseteq S$ o subconjunto, eventualmente vazio, dos estados finais.

Como vimos no exemplo anterior, um autómato é vulgarmente descrito por um grafo dirigido onde o conjunto dos vértices coincide com S e os ramos são setas ligando estados, etiquetadas por símbolos do alfabeto. Outra representação possível é a tabular, conforme se ilustra no seguinte exemplo.

Exemplo 2.2.2. A tabela correspondente à função de transição do exemplo anterior é

	a	b	c
1	2	6	5
2	3	5	3
3	5	4	5
4	5	5	5
5	5	5	5
6	7	5	3
7	5	5	5

Interessa-nos, agora, estender o domínio da função de transição de modo a exprimir a evolução do autómato face ao processamento de sequências de símbolos.

Definição 2.2.2. Dado um autómato finito determinístico $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ define-se recursivamente uma função $\overline{\delta} : S \times \Sigma^* \to S$, dita a **extensão de** δ **a** Σ^* da seguinte forma

$$\begin{array}{rcl}\overline{\delta}(s,\lambda) & = & s \\ \overline{\delta}(s,a\omega) & = & \overline{\delta}(\delta(s,a),\omega)\end{array}$$

para todo o $s \in S$, $a \in \Sigma$ e $\omega \in \Sigma^*$.

Note-se que a palavra a, para cada $a \in \Sigma$, equivale a $a\lambda$, pelo que $\overline{\delta}(s,a) = \delta(s,a)$. A recursividade na definição de $\overline{\delta}$ ocorre sobre o final da sequência de símbolos a processar, de modo que um "olhar" algorítmico identificará logo um esquema iterativo de cálculo. É fácil provar o seguinte resultado:

Teorema 2.2.1. *Seja $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$. Então, para todo o $s \in S$ e $\omega, \upsilon \in \Sigma^*$,*

$$\overline{\delta}(s,\omega\upsilon) = \overline{\delta}(\overline{\delta}(s,\omega),\upsilon)$$

Prova: Procedamos por indução sobre ω. Para $\omega = \lambda$ tem-se

$$\begin{array}{rcl}\overline{\delta}(\overline{\delta}(s,\omega),\upsilon) & = & \overline{\delta}(\overline{\delta}(s,\lambda),\upsilon) \\ & = & \overline{\delta}(s,\upsilon) \\ & = & \overline{\delta}(s,\lambda\upsilon) \\ & = & \overline{\delta}(s,\omega\upsilon)\end{array}$$

Suponhamos, agora, que o resultado se verifica para toda a palavra de comprimento $n-1$ (*i.é*, $\omega \in \Sigma^{n-1}$). Mostremos que, nessas condições ainda é válido para qualquer $\omega \in \Sigma^n$. Repare-se que ω pode ser escrita na forma $\omega = a\omega'$, para um $a \in \Sigma$. Assim,

$$\begin{array}{rcll}\overline{\delta}(\overline{\delta}(s,\omega),\upsilon) & = & \overline{\delta}(\overline{\delta}(s,a\omega'),\upsilon) & \\ & = & \overline{\delta}(\overline{\delta}(\delta(s,a),\omega'),\upsilon) & \\ & = & \overline{\delta}(\overline{\delta}(t,\omega'),\upsilon) & \text{com } t = \delta(s,a) \\ & = & \overline{\delta}(t,\omega'\upsilon) & \text{por hipótese de indução} \\ & = & \overline{\delta}(\delta(s,a),\omega'\upsilon) & \\ & = & \overline{\delta}(s,a\omega'\upsilon) & \\ & = & \overline{\delta}(s,\omega\upsilon) & \end{array}$$

❏

Referimos já que as palavras que conduzem o autómato a um estado final são aquelas que este efectivamente aceita. Um autómato traça uma separação entre os elementos de Σ^* e, nesse sentido, define uma linguagem (*cf.*, Definição 2.1.4). Formalmente diremos

Definição 2.2.3. Dada uma palavra $\omega \in \Sigma^*$ e um autómato $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$, diz-se que $\mathcal{A}$ reconhece ω sse $\overline{\delta}(s_0,\omega) \in F$. Caso contrário, ω constitui uma palavra rejeitada por $\mathcal{A}$.

Prosseguindo,

Definição 2.2.4. Dado um autómato $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$, a **linguagem reconhecida por** $\mathcal{A}$, que se representa por $\mathcal{L}(\mathcal{A})$, é o subconjunto de Σ^* dado por

$$\mathcal{L}(\mathcal{A}) \triangleq \{\omega \in \Sigma^* \mid \overline{\delta}(s_0, \omega) \in F\}$$

Conversamente,

Definição 2.2.5. Uma linguagem $\mathcal{L} \subseteq \Sigma^*$ é dita **regular** ou **reconhecível por um autómato finito determinístico** sse existir um $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ tal que $\mathcal{L} = \mathcal{L}(\mathcal{A})$.

Exemplo 2.2.3. Seja $\Sigma = \{a, b\}$. É fácil mostrar que a linguagem formada pelas palavras com um número ímpar de bs é regular. O autómato finito determinístico seguinte é, efectivamente, um seu reconhecedor.

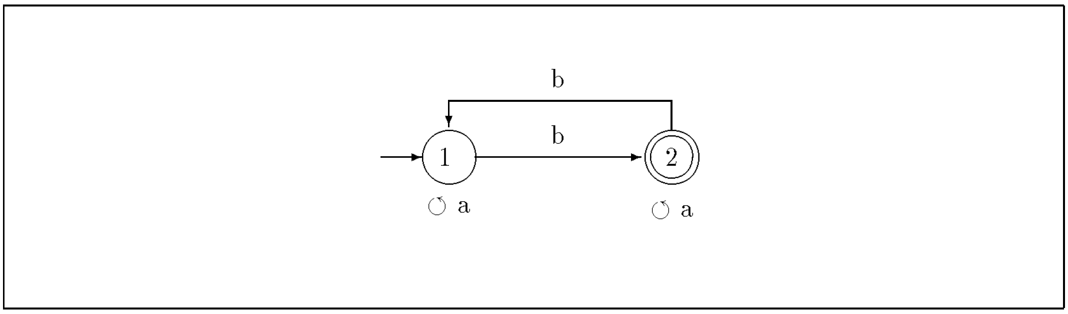

Figura 2.2: $\mathcal{L} = \{\omega \in \Sigma^* \mid (|\omega|_b mod_2) \neq 0\}$

O mesmo se passa para a linguagem das palavras com números pares de ocorrências de cada um dos símbolos do alfabeto (*cf.*, Figura 2.2.3). A identificação dos estados, neste exemplo, é sugestiva do seu papel de memorizadores das etapas do reconhecimento (porquê?).

Veremos, em breve, que nem todas as linguagens são especificáveis por um autómato finito determinístico — tal é o caso, por exemplo de uma linguagem onde a cada ocorrência do símbolo "(" corresponda uma ocorrência de ")" — e procuraremos formular critérios para a averiguação da regularidade de uma linguagem.

Antes, porém, será útil convencionar uma simplificação da notação usada, que consiste na omissão de δ e $\overline{\delta}$ sempre que tal não comprometa a clareza do texto. Assim, repersentaremos o efeito de um estímulo $a \in \Sigma$ ou de uma sequência de estímulos $\omega \in \Sigma^*$ sobre um estado $s \in S$ por

$$sa \quad \text{e} \quad s\omega$$

em vez de $\delta(s, a)$ e $\overline{\delta}(s, \omega)$, respectivamente.

Esta notação simplifica consideravelmente a expressão e a prova dos principais resultado em que estamos interessados. Por exemplo, a equação no enunciado do teorema 2.2.1 passa a escrever-se mais concisamente como

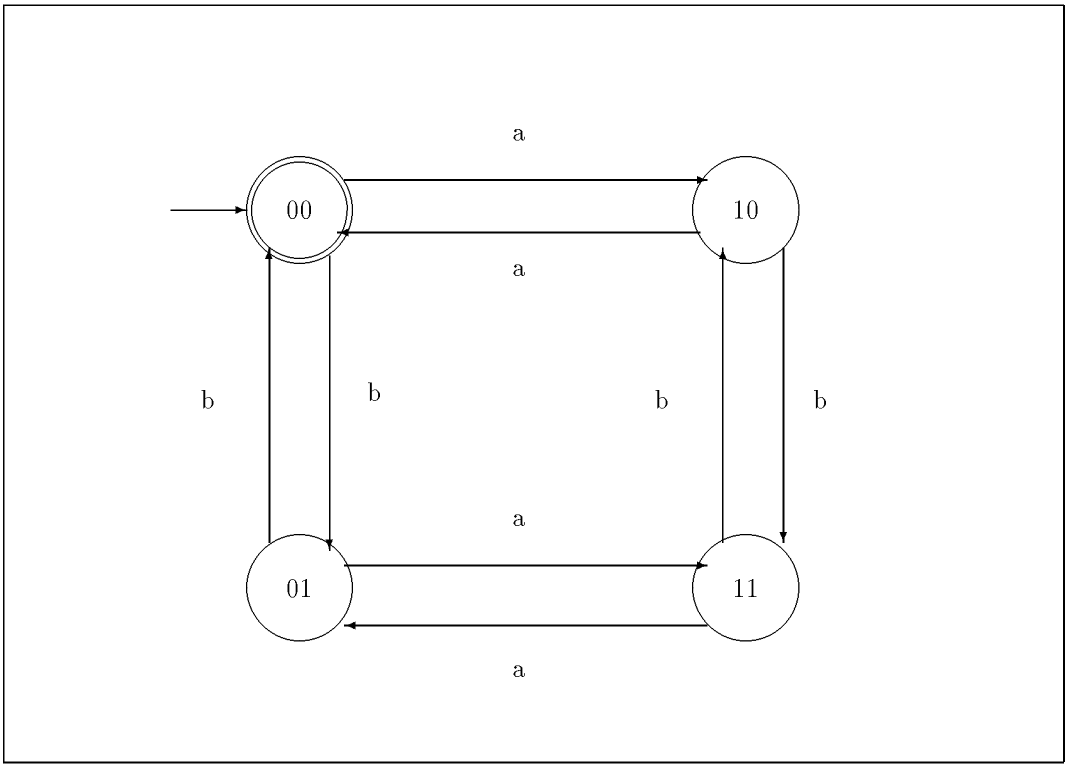

Figura 2.3: $\mathcal{L} \; = \; \{\omega \in \Sigma^* | \; (|\omega|_a mod_2) = 0 \; \wedge \; (|\omega|_b mod_2) = 0\}$

$$s(\omega v) \; = \; (s\omega)v$$

Estamos agora, em condições de abordar a questão acima enunciada: *como decidir sobre a regularidade de uma linguagem arbitrária?* Começaremos por analisar, com detalhe, alguns exemplos, ilustrando o tipo de raciocínio a seguir na averiguação de propriedades de linguagens e autómatos finitos. Introduziremos, depois, um teorema que fornece um método geral para identificar linguagens regulares, embora o seu uso seja mais simples (e mais comum) nas provas negativas. Acontece que esta é uma questão central na teoria dos autómato, pelo que a ela voltaremos mais tarde, em particular na Secção 2.5 e no próximo capítulo. Mas comecemos a "desbravar terreno"!

Exemplo 2.2.4. Seja $\mathcal{A}$ o autómato finito determinístico representado no diagrama da Figura 2.2.4. É fácil verificar que $\omega = b^2a^3b^2a^4$ pertence a $\mathcal{L}(\mathcal{A})$. De facto

$$\begin{aligned}
1\omega &= 1(ba^3b^2a^4) \\
&= 1(a^3b^2a^4) \\
&= 2(a^2b^2a^4) \\
&= \ldots \\
&= 3(ba^4) \\
&= 1a^4 \\
&= 2a^3 \\
&= 2 \in F
\end{aligned}$$

Já a palavra $\omega = aba$ não é reconhecida por $\mathcal{A}$. De facto, $1\omega \ = \ 2(ba) \ = \ 3a \ = \ 0$ e $0 \notin F$.

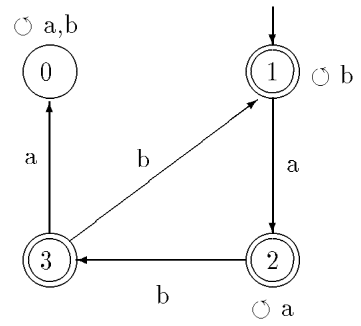

Figura 2.4: $\mathcal{A}$

Poderemos caracterizar, de forma geral, $\mathcal{L}(\mathcal{A})$ para o autómato do exemplo anterior? A prova da proposição seguinte é ilustrativa de um tipo comum de raciocínio.

Proposição 2.2.1. *O autómato $\mathcal{A}$ do exemplo anterior reconhece qualquer palavra sobre $\{a, b\}^*$ que não contenha aba como subsequência.*

Prova: Seja $\omega \in \{a, b\}^*$ e provemos por indução sobre $|\omega|$. Se $|\omega| = 1$, então $\omega = a$, e tem-se $1a = 2 \in F$, ou $\omega = b$, e verifica-se $1b = 1 \in F$.

Suponhamos, agora, que $|\omega| = n > 1$ e que ω não contém a subsequência aba (vimos no Exemplo 2.2.4 que se aba estiver contida em ω, $\omega \notin \mathcal{L}(\mathcal{A})$). Como hipótse de indução vamos admitir que toda a palavra $v \in \{a, b\}^*$, não contendo aba e com $|v| < n$, pertence a $\mathcal{L}(\mathcal{A})$. Então, $\omega = va$ ou $\omega = vb$, e, por hipótese de indução, $1v \in \{1, 2, 3\}$. Consideremos os dois casos.

Seja $\omega = va$. Neste caso $1v \in \{1, 2\}$. De facto, se $1v = 3$, v terminaria forçosamente em ab, pelo que ω conteria aba. Assim,

$$1\omega \ = \ 1va \ \in \{1a, 2a\} = \{2\} \ \subseteq \ F$$

Seja, agora, $\omega = vb$. Então,

$$1\omega \ = \ 1vb \ \in \{1b, 2b, 3b\} = \{1, 3\} \ \subseteq \ F$$

❏

Vejamos, agora, um exemplo de verificação da não regularidade de uma linguagem.

Proposição 2.2.2. *O subconjunto $\mathcal{L} = \{a^n b^n \mid n \geq 1\}$ de $\{a, b\}^*$ não é regular.*

Prova: Suponhamos que existe um autómato $\mathcal{A} = (\{a, b\}, S, s_0, \delta, F)$ que reconhece $\mathcal{L}$. Seja $s_n = s_0 a^n$ o estado correspondente ao reconhecimento sucessivo de n símbolos a. Então, $s_n b^n \in F$, para todo o n. Como S é finito, existe forçosamente um valor para n tal que o estado s_n é idêntico a um estado s_m, com $m \neq n$. De facto, a finitude de S obriga à re-utilização de estados no reconhecimento de palavras arbitráriamente longas. Temos, então,

$$s_m b^n \;=\; s_n b^n \;\in F$$

mas também,

$$s_m b^n \;=\; s_0 a^m b^n$$

ou seja, a palavra $a^m b^n \in \mathcal{L}$, para $m \neq n$, o que contradiz a hipótese.

❏

O argumento da "re-utilização" de estados usado na última demonstração, prepara convenientemente o terreno para o seguinte resultado geral.

Teorema 2.2.2. *Seja $\mathcal{L}$ uma linguagem regular infinita sobre um alfabeto Σ. Então, existe um número natural positivo N tal que toda a palavra $\omega \in \mathcal{L}$, com $|\omega| \geq N$, pode ser factorizada em $\omega = \upsilon\nu\sigma$ de tal forma que*

i) $\upsilon, \sigma \in \Sigma^*$ *e* $\nu \in \Sigma^+$

ii) $|\upsilon, \nu| \leq N$

iii) $\upsilon\nu^i\sigma \;\in \mathcal{L}$ *, para todo o* $i \geq 0$

Prova: A intuição subjacente a este resultado é que, se um autómato finito determinístico pode procesar uma palavra de comprimento arbitrário, então deve, em algum momento do processo de reconhecimento, visitar pelo menos um estado mais do que uma vez. Assim $\mathcal{L}$ será regular se a palavra $\upsilon\nu\sigma$ puder ser "insuflada" com tantas cópias de ν quantas se quizer [1].

Seja $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ e $\mathcal{L} = \mathcal{L}(\mathcal{A})$. Escolhamos $N = |S|$ e tomemos uma palavra $\omega = a_1 a_2 ... a_k$, onde cada $a_i \in \Sigma$ e $k > N$ ($i.é$, $|\omega| > N$).

No percurso

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \xrightarrow{a_k} s_k$$

tem de existir pelo menos uma repetição entre os $k+1$ estados envolvidos. Fixemos s_r como o primeiro estado que se repete na sequência $s_0 s_1 s_2 ... s_k$ e s_{r+s} como a sua primeira repetição. Note-se que $r \geq 0$ e $s > 0$. Por outro lado, $r + s \leq N$.

[1] Sugestivamente este teorema é referido na literatura anglo-saxónica como o *pumping lema*.

Podemos, então, factorizar ω em $\omega = \upsilon\nu\sigma$ onde $\upsilon(\in \Sigma^*)$, $\nu(\in \Sigma^+)$ e $\sigma(\in \Sigma^*)$ etiquetam os sub-percursos seguintes:

$$\begin{array}{lcl} \upsilon & : & s_0 \longrightarrow \ldots \longrightarrow s_r \\ \nu & : & s_r \longrightarrow \ldots \longrightarrow s_{r+s} \\ \sigma & : & s_{r+s} \longrightarrow \ldots \longrightarrow s_N \longrightarrow \ldots \longrightarrow s_k \end{array}$$

o que prova **i)**. Note-se, por outro lado, que o sub-percurso etiquetado por ν é um ciclo no percurso global (*cf.*, coincidência de s_r e s_{r+s}). Nestas condições podemos obter novos percursos de s_0 a s_k iterando o ciclo um número arbitrário de vezes ou, simplesmente, omitindo-o. I.é, $\upsilon\nu^i\sigma \in \mathcal{L}$, o que prova **iii)**. Finalmente, tem-se $|\upsilon\nu| = r + s \leq N$ porque todos os estados visitados até s_{r+s} são distintos, o que verifica **ii)** e completa a prova.

❏

Como dissemos acima este teorema é geralmente utilizado para mostrar que uma linguagem dada *não* é regular. Ilustremos a aplicação repetindo a demonstração da Proposição 2.2.2.

Prova: Se $\mathcal{L}$ é regular então existe um N nas condições do teorema anterior. Tomemos, pois, uma palavra $\omega = a^n b^n$, com $n > N$. O teorema implica que ω pode ser escrita na forma $\upsilon\nu\sigma$, com $|\upsilon\nu| \leq N < n$, de modo que $\upsilon\nu^i\sigma \in \mathcal{L}$, para todo o $i \geq 0$. A condição $|\upsilon\nu| \leq N < n$ significa que tanto υ como ν são subsequências do bloco inicial de as em ω. Temos, então,

$$\begin{array}{lcl} \upsilon & = & a^p \\ \nu & = & a^q \\ \sigma & = & a^r b^n \end{array}$$

para $p, r \geq 0$, $q > 0$ e $p + q + r = n$. Mas isto significa que

$$\upsilon\nu^0\sigma = a^{n-q}b^n \in \mathcal{L}$$

o que contradiz a hipótese.

❏

2.3 Não Determinismo e Transições Espontâneas

2.3.1 Autómatos Não Determinísticos

Apresentamos na secção anterior a noção de autómato finito determinístico e de linguagem por ele reconhecida (que classificamos de regular). A questão que agora nos interessa é a de saber em que sentido a definição de autómato finito pode ser generalizada e quais as vantagens dessa generalização.

A primeira generalização consiste em tomar δ na definição 2.2.1 como uma função *parcial*. Isto significa que, a partir de um dado estado, nem todas as transições serão possíveis. Tais autómatos são designados por *incompletos*. Se, ao processar uma palavra o autómato alcança um estado a partir do qual não existe nenhuma transição etiquetada com o próximo símbolo a processar, o reconhecimento termina. Esta terminação sem sucesso toma o lugar do *estado de erro* encontrado nos autómatos completos para rejeição das palavras que não pertencem à sua linguagem [2].

Um autómato incompleto tem, geralmente, uma estrutura simples que, omitindo os estados de erro, nos permite porventura conjecturar mais rapidamente $\mathcal{L}(\mathcal{A})$. No entanto, o seu poder expressivo é similar ao dos autómatos completos, *i.é*, as linguagens reconhecidas por autómatos finitos determinísticos completos e incompletos são exactamente as mesmas, como em breve provaremos.

Uma generalização mais profunda consiste em relaxar a condição de determinação. Neste caso a resposta do autómato a um estímulo $a \in \Sigma$ poderá ser ambígua. Em vez de um *próximo estado* determinado univocamente, existirá um *conjunto de próximos estados* simultaneamente activados por uma mesma transição. A entidade adequada à modelação de δ é, agora a *relação binária*

$$\delta \subseteq S \times \Sigma \times S$$

ou, equivalentemente,

$$\delta : S \times \Sigma \rightarrow 2^S$$

Podemos ainda supôr que o autómato possui diversos pontos de activação, *i.é*, não um mas vários estados iniciais. Chegamos, assim, à seguinte generalização da Definição 2.2.1.

Definição 2.3.1. Um **Autómato Finito Não Determinístico** é um tuplo $(\Sigma, S, S_0, \delta, F)$ onde Σ é um alfabeto, S um conjunto finito e não vazio de estados, $S_0 \subseteq S$ o conjunto dos estados iniciais, $\delta : S \times \Sigma \rightarrow 2^S$ a relação de transição de estados e $F \subseteq S$ o subconjunto, eventualmente vazio, dos estados finais.

A extensão de δ a Σ^* sofre a generalização adequada.

Definição 2.3.2. Dado um autómato finito não determinístico $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ define-se recursivamente uma função $\overline{\delta} : S \times \Sigma^* \rightarrow 2^S$, dita a **extensão de** δ **a** Σ^* da seguinte forma

$$\begin{array}{rcl} \overline{\delta}(s, \lambda) & = & \{s\} \\ \overline{\delta}(s, a\omega) & = & \bigcup_{q \in \delta(s,a)} \overline{\delta}(q, \omega) \end{array}$$

[2] É o caso do estado 5 no autómato do Exemplo 2.2.1.

ou, na notação abreviada que vimos utilizando,

$$\begin{array}{lcl} s\lambda & = & \{s\} \\ s(a\omega) & = & \bigcup_{q \in sa} q\omega \end{array}$$

para todo o $s \in S$, $a \in \Sigma$ e $\omega \in \Sigma^{*}$[3].

O teorema 2.2.1 permanece válido, embora envolvendo agora a igualdade, não de elementos, mas de subconjuntos de S. Por outro lado, a definição de linguagem aceite deve ser ligeiramente modificada: as palavras aceites serão aquelas que etiquetam *pelo menos um percurso* no autómato que termina num estado de F. Formalmente,

Definição 2.3.3. Dado um autómato $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$, a **linguagem reconhecida por** $\mathcal{A}$, que se representa por $\mathcal{L}(\mathcal{A})$, é o subconjunto de Σ^* dado por

$$\mathcal{L}(\mathcal{A}) \triangleq \{\omega \in \Sigma^* | \bigcup_{s \in S_0} s\omega \cap F \neq \emptyset\}$$

De novo, e um tanto paradoxalmente, deparamos com a equivalência entre o poder expressivo de um autómato finito não determinístico e um autómato finito determinístico. O teorema seguinte formaliza esse resultado. Chamamos a atenção do leitor para a sua demonstração que segue uma técnica que se repetirá várias vezes neste texto. A ideia é construir explicitamente, para cada autómato finito não determinístico (eventualmente incompleto), um autómato finito determinístico completo correspondente e mostrar que a linguagem reconhecida por um e por outro é a mesma [4].

Teorema 2.3.1. *Seja $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ um autómato finito não determinístico, eventualmente incompleto. É sempre possível definir um autómato finito determinístico completo $\mathcal{A}'$ tal que $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Prova: A prova procede por construção de $\mathcal{A}' = (\Sigma, S', s_0', \delta', F')$ determinístico e completo. A determinação de δ' requer que o espaço de estados passe a ser o conjunto de todos os subconjuntos de S, *i.é*, $S' = 2^S$. A função δ' define-se da maneira óbvia,

$$\delta'(Q, a) = \bigcup_{q \in Q} \delta(q, a)$$

ou, abreviadamente,

$$Qa = \bigcup_{q \in Q} qa$$

[3] Convencionando que, para um subconjunto $Q \subseteq S$, a notação $Q\omega$ significa $\bigcup_{q \in Q} q\omega$, o caso geral da definição de $\overline{\delta}$ reescreve em $s(a\omega) = (sa)\omega$, análoga à correspondente condição para o caso determinístico.

[4] Trata-se, de uma prova *construtiva* em que o objecto cuja existência se pretende mostrar é explicitamente definido, verificando-se depois que as propriedades requeridas são satisfeitas.

para $Q \in 2^S$ e $a \in \Sigma$.

Finalmente, $s'_0 = S_0$ e $F' = \{Q \in 2^S | \, (Q \cap F) \neq \emptyset\}$.

Comecemos por mostrar que, para todo o $Q \in S'$ e $\omega \in \Sigma^*$, se tem $Q\omega = \bigcup_{q \in Q} q\omega$ (*i.é*, $\overline{\delta}'(Q, \omega) = \bigcup_{q \in Q} \overline{\delta}(q, \omega)$). Para $|\omega| = 1$ o enunciado reduz-se à própia definição de δ'. Para $|\omega| \geq 1$, consideremos $\omega = va$ (com $v \in \Sigma^*$ e $a \in \Sigma$) e suponhamos que o resultado se verifica para todo o v. Então,

$$\begin{array}{rcl} Q\omega & = & Q(av) \\ & = & (Qa)v \\ & = & (\bigcup_{q \in Q} qa)v \\ & = & \bigcup_{q \in Q} (qa)v \\ & = & \bigcup_{q \in Q} q\omega \end{array}$$

Vejamos, agora, que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

$$\begin{array}{rcl} \omega \in \mathcal{L}(\mathcal{A}) & sse & \bigcup_{s \in S_0} s\omega \cap F \neq \emptyset \\ & sse & S_0\omega \cap F \neq \emptyset \\ & sse & S_0\omega \in F' \\ & sse & \omega \in \mathcal{L}(\mathcal{A}') \end{array}$$

❏

Exemplo 2.3.1. Seja $\mathcal{A}$ o autómato finito não determinístico representado no diagrama da Figura 2.3.1. Aplicando o processo usado na prova do teorema 2.3.1 facilmente se chega ao autómato finito determinístico $\mathcal{A}'$ representado na Figura 2.3.1, cujo espaço de estados é $S' = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$.

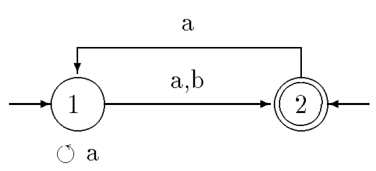

Figura 2.5: $\mathcal{A}$

2.3.2 Autómatos com Transições Espontâneas

Uma outra generalização da definição de autómato procede pela consideração da existência de transições entre os estados que ocorrem *espontaneamente* (ou, se preferirmos, *invisivelmente*), sem que nenhum estímulo externo seja assinalado. A situação é usual na prática,

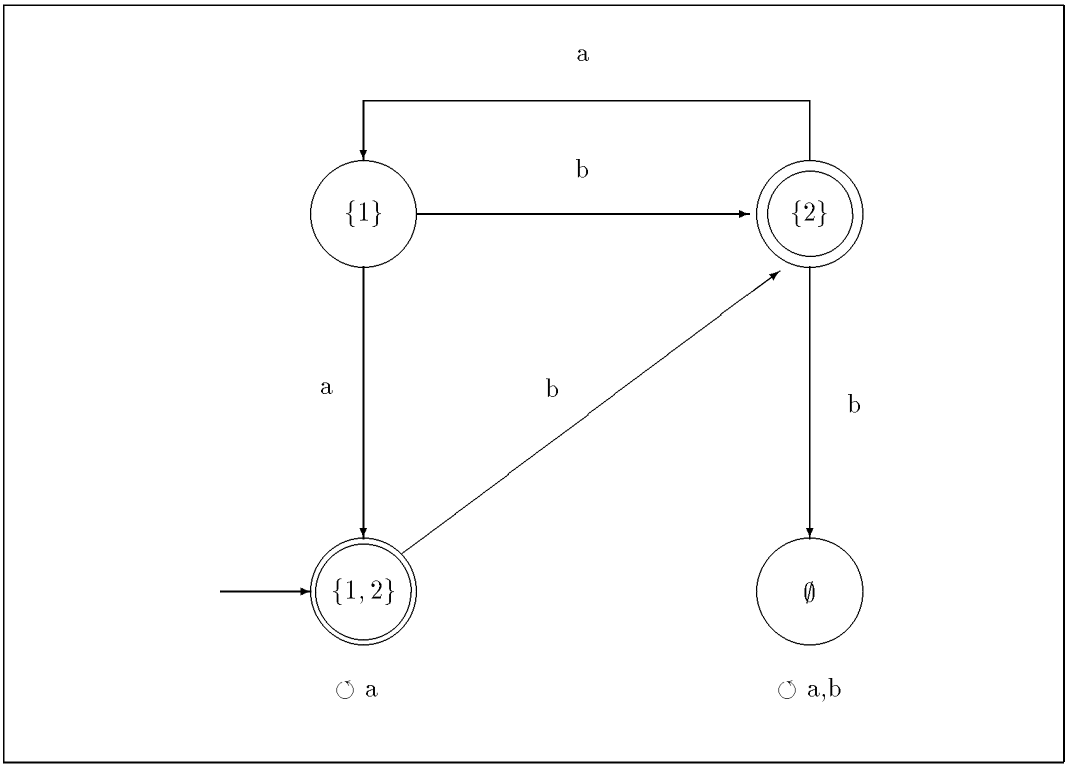

Figura 2.6: $\mathcal{A}'$

em particular quando pretendemos modelar por um autómato uma vista parcelar de um sistema[5]. Tais transições são etiquetadas com a palavra vazia e, por isso mesmo, recebem o nome de *transições-λ* ou *transições espontâneas*.

Definição 2.3.4. Um **Autómato com Transições Espontâneas** é um autómato finito não determinístico $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ em que a função de transição de estados tem a seguinte assinatura:

$$\delta : \ S \times (\Sigma \cup \{\lambda\}) \rightarrow 2^S$$

É curioso pensar na extensão de δ a Σ^*. Suponha-se que o autómato está num estado s e transita para outro estado s' sob acção do estímulo $a \in \Sigma$. Se de s' fôr possível uma transição espontânea para um outro estado s'', por exemplo, torna-se evidente que s'' (tal com s') pode ser alcançado de s pelo reconhecimento de a. Este facto é registado na definição de $\overline{\delta}$ da seguinte forma: o conjunto dos estados acessíveis a partir de s pelo reconhecimento de $\omega \in \Sigma^*$ inclui não apenas os estados directamente acessíveis, mas também todos aqueles que são espontaneamente acessíveis a partir dos primeiros. Designaremos o conjunto dos estados acessíveis por uma transição espontânea a partir de s por $Ac(s)$ — o λ-fecho do estado s. Obviamente, $s \in Ac(s)$. Então,

Definição 2.3.5. Dado um autómato finito não determinístico com transições espontâneas $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ define-se $\overline{\delta} : S \times \Sigma^* \rightarrow 2^S$, dita a **extensão de δ a Σ^*** da seguinte forma:

[5]Pense, por exemplo, no sub-sistema *interface com o utente* das caixas automáticas do sistema Multibanco.

$$\begin{array}{lcl}\overline{\delta}(s,\lambda) & = & Ac(\{s\}) \\ \overline{\delta}(s,\omega a) & = & Ac(\bigcup_{q\in\overline{\delta}(s,\omega)}\delta(q,a))\end{array}$$

Naturalmente $\overline{\delta}(s,a) = \overline{\delta}(s,\lambda a) = Ac(\bigcup_{q\in\overline{\delta}(s,\lambda)}\delta(q,a)) = Ac(\bigcup_{q\in Ac(s)}\delta(q,a))$. Note-se que, por outro lado, $\overline{\delta}(s,a) \neq \delta(s,a)$, para $s \in S$ e $a \in \Sigma$, no caso geral.

Vejamos uma utilização prática deste tipo de autómatos.

Exemplo 2.3.2. O autómato finito não determinístico representado na Figura2.3.2 reconhece a linguagem $\mathcal{L}$ sobre $\Sigma = \{a, b, c\}$, cujas palavras contém exactamente um b seguido de um c.

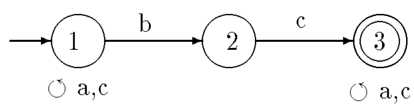

Figura 2.7: Utilização das transições-λ

Suponhamos, agora, que se pretende um autómato para reconhecer a linguagem $\mathcal{L}_1 = \mathcal{L}^+$. Intuitivamente se verifica que $\mathcal{L}_1$ contém todas as palavras sobre Σ com pelo menos um b e com a propriedade de que qualquer b é imediatamente seguido por um c. Tal pode ser conseguido de forma muito simples: basta adicionar uma transição espontânea do estado final 3 para o estado inicial 1.

Como exemplificamos, a introdução de transições espontâneas entre estados, tal como a introdução do não determinismo, facilita muito a construção de autómatos finitos para o reconhecimento de determinadas linguagens. Contudo, o seu poder expressivo é análogo ao dos outros modelos referidos, de acordo com o teorema seguinte.

Teorema 2.3.2. *Seja $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ um autómato finito não determinístico com transições espontâneas. É sempre possível definir um autómato finito não determinístico $\mathcal{A}'$ tal que $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Prova: A prova procede, de novo, por construção de $\mathcal{A}' = (\Sigma, S', S_0', \delta', F')$. A ideia base é juntar ao resultado de uma transição com origem em s e etiquetada por a em $\mathcal{A}$ o λ-fecho de cada $s' \in \delta(s,a)$. Apenas um cuidado suplementar deve ser tomado. De facto, é fácil verificar que o λ-fecho dos estados em S inviabiliza a inclusão de λ na linguagem reconhecida por $\mathcal{A}'$, apesar de λ poder pertencer efectivamente a $\mathcal{L}(\mathcal{A})$. Para prevenir esta possibilidade inclui-se um novo estado inicial em $\mathcal{A}'$ — que designaremos por q_0 — e que surge desligado de todos os outros. Caso $\lambda \in \mathcal{L}(\mathcal{A})$, q_0 será marcado como um estado final, de forma que λ pertencerá igualmente a $\mathcal{L}(\mathcal{A}')$. Temos, então,

$$\begin{aligned} S' &= S \cup \{q_0\} \\ S'_0 &= S_0 \cup \{q_0\} \\ \delta'(q_0, a) &= \emptyset \\ \delta'(s, a) &= \overline{\delta}(s, a) \end{aligned}$$

e, finalmente,

$$F' = \begin{cases} F & \text{se} \lambda \notin \mathcal{L}(\mathcal{A}) \\ F \cup \{q_0\} & \text{se} \lambda \in \mathcal{L}(\mathcal{A}) \end{cases}$$

Consideremos, agora, uma palavra $\omega \in \mathcal{L}(\mathcal{A})$. Se $\omega = \lambda$ temos que $q_0 \in F'$, logo $\omega \in \mathcal{L}(\mathcal{A}')$. Se $\omega \neq \lambda$ vem

$$\begin{array}{lll} \omega \in \mathcal{L} & sse & \bigcup_{s \in S_0} \overline{\delta}(s, \omega) \cap F \neq \emptyset \\ & sse & \bigcup_{s \in S_0} \overline{\delta}'(s, \omega) \cap F \neq \emptyset \\ & sse & \bigcup_{s \in S_0} \overline{\delta}'(s, \omega) \cap F' \neq \emptyset \\ & sse & \omega \in \mathcal{L}(\mathcal{A}') \end{array}$$

Note-se que a substituição de F por F' no terceiro passo decorre da definição de F e do facto de $\overline{\delta}'(s, \omega)$ não conter $\{q_0\}$. Já o segundo passo na demonstração deve ser justificado mostrando-se que, para todo o $s \in S$ e $\omega \in \Sigma^+$, $\overline{\delta}'(s, \omega) = \overline{\delta}(s, \omega)$, o que se faz por indução sobe $|\omega|$. O caso de base — $|\omega| = 1$ — é trivial. Suponhamos, agora, que $\omega = \omega' a$ e que $\overline{\delta}'(s, \omega') = \overline{\delta}(s, \omega')$. Então,

$$\begin{aligned} \overline{\delta}'(s, \omega) &= Ac(\bigcup_{q \in \overline{\delta}'(s, \omega')} \delta'(q, a)) \\ &= Ac(\bigcup_{q \in \overline{\delta}'(s, \omega')} \overline{\delta}(q, a)) \\ &= Ac(\bigcup_{q \in \overline{\delta}(s, \omega')} \overline{\delta}(q, a)) \\ &= \overline{\delta}(s, \omega' a) \\ &= \overline{\delta}(s, \omega) \end{aligned}$$

❏

2.4 Minimização e Equivalência

2.4.1 Minimização de Autómatos Finitos

Um problema com que frequentemente se depara ao desenhar autómatos para reconhecer uma determinada linguagem é o de avaliar a sua simplicidade, em particular a existência de estados redundantes. Nesta secção identificaremos os diferentes modos em que um estado pode ser redundante e investigaremos a construção do autómato correspondente com o menor número de estados. Este tipo de transformações que permitem simplificar a estrutura de um autómato, sem afectar o seu comportamento, têm um enorme interesse prático. De facto, autómatos mais simples originam reconhecedores de concepção mais simples e execução mais eficiente.

Um ponto que emergirá desta discussão é a clarificação do que se entende por autómatos *equivalentes* e autómatos *idênticos*. A definição das noções de equivalência e igualdade, das quais já temos uma ideia intuitiva, torna-se fundamental sempre que se introduz uma nova estrutura matemática, como aqui vimos fazendo com a noção de autómato. De uma forma mais geral ainda, iremos definir *homomorfismo entre autómatos*, *i.é*, um tipo de transformação (funcional) de autómatos que preserva a sua estrutura. De facto, definir essas transformações numa dada classe de estruturas é tanto, ou mais, importante que definir a própria estrutura (*cf.*, [ML86] para uma discussão intuitiva). No nosso caso irá permitir raciocinar sobre colecções de autómatos, relaciona-los e agrega-los de diferentes modos. É o que faremos no Capítulo6, no contexto, ligeiramente mais geral, dos sistemas de transição.

Neste programa o primeiro critério com que deparamos é o de *acessibilidade*.

Definição 2.4.1. Um autómato $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ diz-se **acessível** se, para todo o $s \in S$, existir uma palavra $\omega \in \Sigma^*$ tal que $s_0\omega = s$.

Como o comportamento de um autómato é descrito em termos dos percursos válidos de s_0 para os elementos de F e apenas os estados acessíveis podem ser visitados, os restantes podem ser removidos sem afectar o comportamento do autómato.

Na prática, dado um autómato finito não determinístico podemos construir o determinístico correspondente aplicando em simultâneo os processos de *determinação* e *acessibilização*. Começa-se por considerar os subconjuntos S_0a de S, para cada $a \in \Sigma$. Depois calculam-se os $S_0\omega$ para $|\omega| = 2$, e assim sucessivamente, até não serem obtidos novos subconjuntos de 2^S. O que resta de 2^S "está a mais", *i.é*, não é acessível. Isso mesmo se ilustra no seguinte exemplo.

Exemplo 2.4.1. Consideremos o autómato descrito na Figura 2.4.1 e calculemos

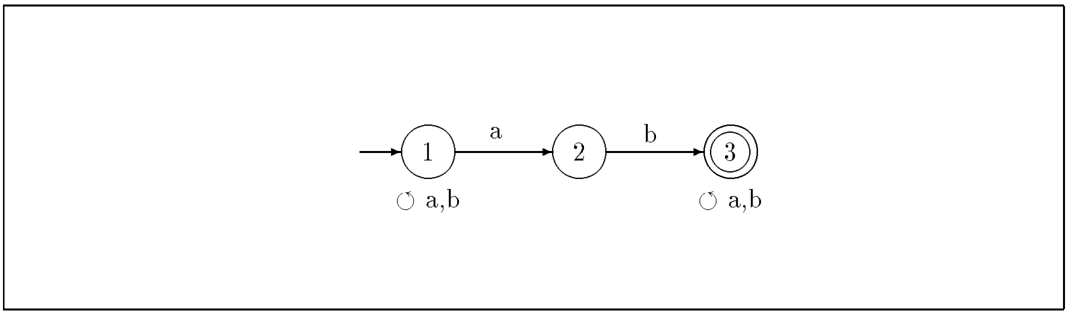

Figura 2.8: Autómato a minimizar

$$\begin{array}{lcl}
\{1\}a & = & \{1,2\} \\
\{1\}b & = & \{1\} \\
\{1,2\}a & = & \{1,2\} \\
\{1,2\}b & = & \{1,3\} \\
\{1,3\}a & = & \{1,2,3\} \\
\{1,3\}b & = & \{1,3\} \\
\{1,2,3\}a & = & \{1,2,3\} \\
\{1,2,3\}b & = & \{1,3\}
\end{array}$$

Fixado o espaço de estados e a função de transição, podemos construir rapidamente o autómato determinístico e acessível, correspondente. Tal construção está representada na Figura 2.4.1.

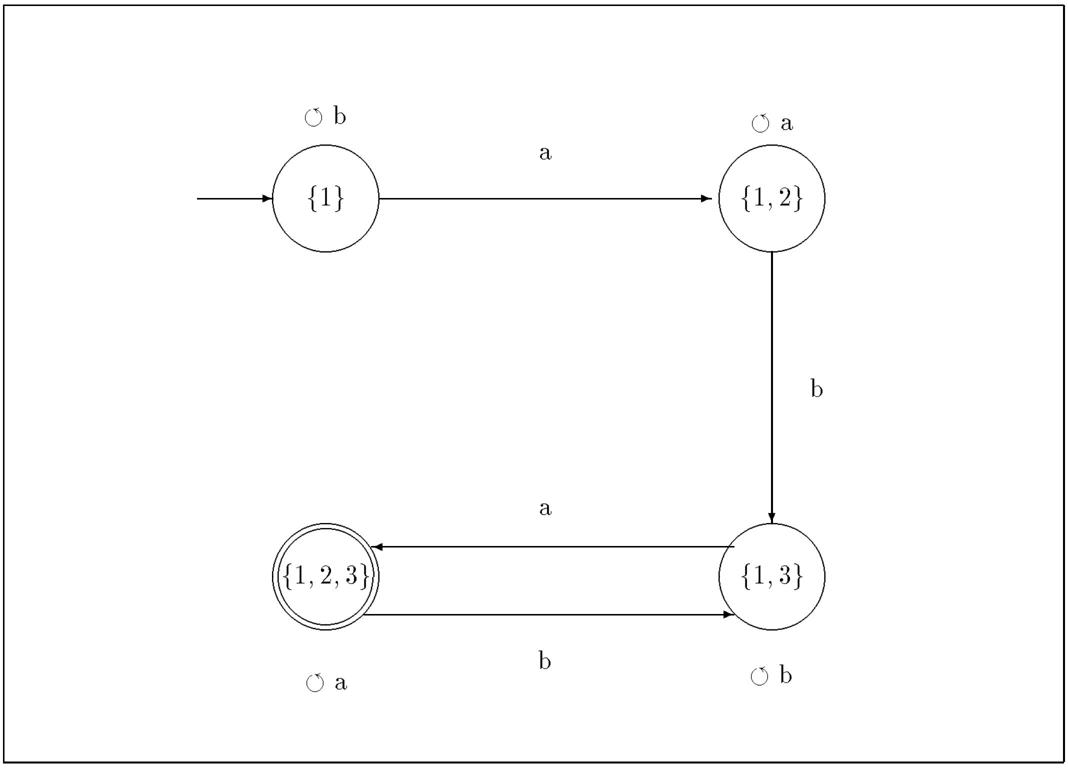

Figura 2.9: Resultado da minimização

Consideremos, agora, um autómato finito determinístico acessível. Existirão ainda estados redundantes em $\mathcal{A}$? A resposta é sim: possivelmente existirão estados que, sendo acessíveis, permitem reconhecer o mesmo conjunto de palavras. Ora, numa versão reduzida de $\mathcal{A}$, bastaria tomar um desses estados. É, basicamente, o que faremos.

Comecemos por caracterizar a noção de redundância acima referida. O conjunto de palavras reconhecidas por um autómato a partir de um dado estado s é usualmente referido como o *conjunto terminal* desse estado, de acordo com a definição seguinte.

Definição 2.4.2. Num autómato $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ define-se **conjunto terminal de** s **em** $\mathcal{A}$, e representa-se por $T(s, \mathcal{A})$, o subconjunto de Σ^* dado por

$$T(s,\mathcal{A}) \triangleq \{\omega \in \Sigma^* |\ s\omega \in F\}$$

Numa notação abreviada é usual escrever-se simplesmente

$$T(s,\mathcal{A}) \triangleq s^{-1}F$$

onde $s^{-1}X \triangleq \{\omega \in \Sigma^* |\ s\omega \in X\}$ para $s \in S$ e $X \subseteq \Sigma^*$. Note-se que $\mathcal{L}(\mathcal{A}) = s_0^{-1}F = T(s_0, \mathcal{A})$.

Podemos, agora, definir uma relação entre os estados de um autómato que identifique os estados com idênticos conjuntos terminais — que serão redundantes de acordo com o critério que acima enunciamos. Assim,

Definição 2.4.3. Num autómato $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ define-se **relação de redundância em** S como a relação binária $\rho \subseteq S \times S$ dada por, para todo o $s_1, s_2 \in S$,

$$(s_1, s_2) \in \rho \ \ sse \ \ T(s_1, \mathcal{A}) = T(s_2, \mathcal{A})$$

É imediato verificar que

Facto 2.4.1. Num autómato $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$, a relação ρ que acabamos de definir é uma relação de equivalência sobre S.

Podemos, a partir daqui, formalizar a noção de *versão reduzida de* $\mathcal{A}$.

Definição 2.4.4. Um autómato $\mathcal{A}^m = (\Sigma, S^m, s_0^m, \delta^m, F^m)$ diz-se uma **versão reduzida de A** se $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^m)$ e a relação de redundância ρ sobre os seus estados coincidir com a identidade em S^m.

De facto, se, num autómato $\mathcal{A}$, ρ coincidir com a identidade, haverá apenas um único estado a partir do qual é possível reconher um particular subconjunto de Σ^* — nenhum estado "está a mais". Se $\mathcal{A}$ fôr acessível, a condição de minimização enunciada estará satisfeita. Neste contexto, é possível construir uma versão reduzida de $\mathcal{A}$ tomando como espaço de estados as classes de equivalência definidas por ρ sobre S. Mais precisamente,

Definição 2.4.5. Dado um autómato $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ determinístico e acessível, define-se o autómato $\mathcal{A}^m = (\Sigma, S^m, s_0^m, \delta^m, F^m)$ da forma seguinte:

$$\begin{aligned}
S^m &= S/\rho = \{[s]_\rho |\ s \in S\} \\
s_0^m &= [s_0]_\rho \\
F^m &= \{[t]_\rho |\ t \in F\} \\
\delta^m([s]_\rho, a) &= [\delta(s,a)]_\rho \quad para\ a \in \Sigma, s \in S^6
\end{aligned}$$

O leitor poderá comprovar a adequação desta formulação mostrando que, para todo o $s \in S$ e $a \in \Sigma$, se tem

$$\begin{array}{ll} \textbf{i)} & ([s]_\rho = [s']_\rho) \;\Rightarrow\; ([s]_\rho a = [s']_\rho a) \\ \textbf{ii)} & F^m \;=\; \{[s]_\rho \in S^m \mid [s]_\rho \cap F \neq \emptyset\} \end{array}$$

Podemos, agora, formular o teorema esperado.

Teorema 2.4.1. *Seja $\mathcal{A}$ um autómato finito determinístico e acessível. Então, o autómato $\mathcal{A}^m$ introduzido pela definição 2.4.5 é uma versão reduzida de $\mathcal{A}$.*

Prova: Verifiquemos, primeiro, que ρ coincide com a identidade sobre S^m. Sejam $[s]_\rho, [s]_\rho \in S^m$ e suponhamos que $([s]_\rho)^{-1}F^m = ([s']_\rho)^{-1}F^m$. Então,

$$\begin{array}{ll} & \{\omega \in \Sigma^* \mid [s]_\rho\omega \in F^m\} \;=\; \{\omega \in \Sigma^* \mid [s']_\rho\omega \in F^m\} \\ i.é, & \\ & \{\omega \in \Sigma^* \mid [s\omega]_\rho \in F^m\} \;=\; \{\omega \in \Sigma^* \mid [s'\omega]_\rho \in F^m\} \\ i.é, & \\ & \{\omega \in \Sigma^* \mid s\omega \in F\} \;=\; \{\omega \in \Sigma^* \mid s'\omega \in F\} \\ i.é, & \\ & s^{-1}F \;=\; s'^{-1}F \end{array}$$

Logo $[s]_\rho = [s']_\rho$, como pretendíamos.

Mostremos, agora, a equivalência entre as linguagens reconhecidas por $\mathcal{A}$ e $\mathcal{A}^m$.

$$\begin{array}{lll} \omega \in \mathcal{L}(\mathcal{A}^m) & sse & [s_0]_\rho\omega \in F^m \\ & sse & s_0\omega \in F \\ & sse & \omega \in \mathcal{L}(\mathcal{A}) \end{array}$$

❏

2.4.2 Autómatos Canónicos

O caminho que seguimos na construção do autómato mínimo que reconhece uma dada linguagem regular resume-se nos seguintes passos. Conjecturamos um autómato $\mathcal{A}$ que reconhece $\mathcal{L}$, transformamo-lo posteriormente num autómato determinístico e acessível e, por fim, calculamos a sua versão reduzida. Existe, porém, uma resposta mais directa a este problema. Veremos, de seguida, como, a partir de $\mathcal{L}$, é possível gerar, num só passo, o autómato minímo que a reconhece.

Comecemos por conjecturar a sua existência e representemo-lo por

$$\mathcal{A}_\mathcal{L} \;=\; (\Sigma, S_\mathcal{L}, s_{0_\mathcal{L}}, \delta_\mathcal{L}, F_\mathcal{L})$$

Se o objectivo de $\mathcal{A}_{\mathcal{L}}$ é reconhecer $\mathcal{L}$, uma escolha sensata para o seu espaço de estados serão os conjuntos de palavras sobre Σ que representam as sub-linguagens aceites em cada um dos estádios de reconhecimento. Temos, assim, que $s \in S_{\mathcal{L}}$ estará contido em Σ^*. Escolhamos $s_{0_{\mathcal{L}}} = \mathcal{L}$, o que intuitivamente significa que, no estado inicial, a sub-linguagem a reconhecer é o próprio $\mathcal{L}$. Reagindo a $a \in \Sigma$, a partir de $s_{0_{\mathcal{L}}}$, $\mathcal{A}_{\mathcal{L}}$ evolui para um estado que contém todos os sufixos das palavras de $\mathcal{L}$ iniciadas por a. De certa forma, cada estado representa aquilo que falta reconhecer. Neste sentido, um estado será final se contiver a sequência vazia — λ — que é sufixo de uma palavra de $\mathcal{L}$ completamente reconhecida.

Usaremos uma notação emprestada da álgebra universal para designar o conjunto dos sufixos das palavras de $\mathcal{L}$ após a aceitação de a, de acordo com a seguinte definição.

Definição 2.4.6. Seja M um monoide, $X \subseteq M$ e $a \in M$. Define-se $a^{-1}X = \{x \in M \mid ax \in X\}$, onde se representa por justaposição a operação binária do monoide.

Mostra-se facilmente que, se 1 fôr o elemento unidade do monoide, se tem

$$\begin{aligned} 1^{-1}X &= X \\ (xy)^{-1}X &= y^{-1}(x^{-1}X) \end{aligned}$$

Com base nesta notação, e notando de novo que o monoide particular em que estamos interessado é $(\Sigma^*; \cdot, \lambda)$, podemos formalizar a construção de $\mathcal{A}_{\mathcal{L}}$.

Definição 2.4.7. A cada linguagem regular $\mathcal{L}$ é possível associar um autómato — dito **canónico** — $\mathcal{A}_{\mathcal{L}}$ definido por, para todo $s \in S_{\mathcal{L}}, \omega \in \Sigma^*$,

$$\begin{aligned} S_{\mathcal{L}} &= \{\omega^{-1}\mathcal{L} \mid \omega \in \Sigma^*\} \\ s_{0_{\mathcal{L}}} &= \lambda^{-1}\mathcal{L} = \mathcal{L} \\ F_{\mathcal{L}} &= \{\omega^{-1}\mathcal{L} \mid \omega \in \mathcal{L}\} \\ \delta_{\mathcal{L}}(s, \omega) &= \omega^{-1}s \end{aligned}$$

Note-se que $F_{\mathcal{L}}$ pode ser definido alternativamente como $F_{\mathcal{L}} = \{s \in S_{\mathcal{L}} \mid \lambda \in s\}$, uma vez que $\lambda \in s$ sse $\lambda^{-1}\omega \in \mathcal{L}$ sse $\omega \in \mathcal{L}$.

Esta definição de canonicidade é restrita às linguagens regulares, não sendo transponível para os tipos mais elaborados de linguagens que trataremos no próximo capítulo. A razão é simples: $S_{\mathcal{L}}$ só é finito, e, portanto, $\mathcal{A}_{\mathcal{L}}$ só define um autómato, se $\mathcal{L}$ fôr regular, como se mostra no seguinte teorema.

Teorema 2.4.2. *Na definição de autómato canónico, $S_{\mathcal{L}}$ é finito sse $\mathcal{L}$ fôr regular.*

Prova: Se $S_{\mathcal{L}}$ é finito, $\mathcal{A}_{\mathcal{L}}$ é um autómato que reconhece $\mathcal{L}$. De facto, tem-se

$$\begin{aligned} \delta(s_{0_{\mathcal{L}}}, \omega) \in F_{\mathcal{L}} \quad & sse \quad \omega^{-1}\mathcal{L} \in F_{\mathcal{L}} \\ & sse \quad \omega \in \mathcal{L} \end{aligned}$$

Conversamente, suponhamos $\mathcal{L}$ regular. Logo existe um autómato finito determinístico e acessível $\mathcal{A}$ tal que $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. Defina-se uma função $\phi : S \to 2^{\Sigma^*}$ por

$$\phi(s) \triangleq \{\omega \in \Sigma^* |\ s\omega \in F\}$$

Tem-se que para todo o $s \in S$, $\phi(s) \in S_{\mathcal{L}}$. Então,

$$\begin{array}{lll} \omega \in \phi(s) & sse & s\omega \in F \\ & sse & s_0 v\omega \in F \\ & sse & v\omega \in \mathcal{L} \\ & sse & \omega \in v^{-1}\mathcal{L} \end{array}$$

Assim, ϕ pode ser visto como uma função de S para $S_{\mathcal{L}}$. Mais ainda, trata-se duma função sobrejectiva: para qualquer $v^{-1}\mathcal{L} \in S_{\mathcal{L}}$ podemos considerar o estado $s = s_0 v$ de $\mathcal{A}$ e verificar que

$$\begin{array}{lll} \phi(s) & = & \{\omega \in \Sigma^* |\ s_0 v\omega \in F\} \\ & = & \{\omega \in \Sigma^* |\ v\omega \in \mathcal{L}\} \\ & = & v^{-1}\mathcal{L} \end{array}$$

Sendo S finito (por definição de $\mathcal{A}$) e ϕ sobrejectiva (como acabamos de mostrar), $S_{\mathcal{L}}$ é necessariamente finito.

❏

Em geral, a construção de $\mathcal{A}_{\mathcal{L}}$, sendo matematicamente elegante, é frequentemente difícil, envolvendo conceitos que estão fora do âmbito deste texto. Ilustremo-la, contudo, num caso de aplicação simples.

Exemplo 2.4.2. Seja $\mathcal{L}$ a linguagem sobre $\Sigma = \{a, b\}$ cujas palavras contém pelos menos uma ocorrência da sequência ab. Calculemos $\mathcal{A}_{\mathcal{L}}$. De acordo com a definição 2.4.7, temos $s_{0_{\mathcal{L}}} = \mathcal{L}$. A partir desse estado, vejamos quais os valores de $\delta_{\mathcal{L}}$ para os dois símbolos do alfabeto.

$$\begin{array}{lllllllll} \delta_{\mathcal{L}}(\mathcal{L}, a) & = & a^{-1}\mathcal{L} & = & \{\omega \in \Sigma^* |\ a\omega \in \mathcal{L}\} & = & \mathcal{L} \cup \{b\}\Sigma^* & = & \mathcal{L}_1 \\ \delta_{\mathcal{L}}(\mathcal{L}, b) & = & b^{-1}\mathcal{L} & = & \{\omega \in \Sigma^* |\ b\omega \in \mathcal{L}\} & = & \mathcal{L} & & \end{array}$$

Um estado distinto de $s_{0_{\mathcal{L}}}$, e que designamos por $\mathcal{L}_1$, foi introduzido. É, pois, necessário repetir os cálculos a partir dele. Assim

$$\begin{array}{lllllllll} \delta_{\mathcal{L}}(\mathcal{L}_1, a) & = & a^{-1}\mathcal{L}_1 & = & \{\omega \in \Sigma^* |\ a\omega \in \mathcal{L}_1\} & = & \{\omega \in \Sigma^* |\ a^2\omega \in \mathcal{L}\} & = & \mathcal{L} \cup \{b\}\Sigma^* \\ & = & \mathcal{L}_1 & & & & & & \\ \delta_{\mathcal{L}}(\mathcal{L}_1, b) & = & b^{-1}\mathcal{L}_1 & = & \{\omega \in \Sigma^* |\ b\omega \in \mathcal{L}_1\} & = & \{\omega \in \Sigma^* |\ ab\omega \in \mathcal{L}\} & = & \Sigma^* \\ & = & \mathcal{L}_2 & & & & & & \end{array}$$

e, ainda,

$$\begin{array}{rclcl} \delta_{\mathcal{L}}(\mathcal{L}_2, a) & = & a^{-1}\mathcal{L}_2 & = & \mathcal{L}_2 \\ \delta_{\mathcal{L}}(\mathcal{L}_2, b) & = & b^{-1}\mathcal{L}_2 & = & \mathcal{L}_2 \end{array}$$

Como, nesta etapa, nenhum novo estado foi gerado, o processo termina com a identificação de $S_{\mathcal{L}} = \{\mathcal{L}, \mathcal{L}_1, \mathcal{L}_2\}$. Temos, por fim, $F_{\mathcal{L}} = \{s \in S_{\mathcal{L}} | \ \lambda \in s\} = \{\mathcal{L}_2\}$. A Figura 2.4.2 representa o autómato canónico que acabamos de construir.

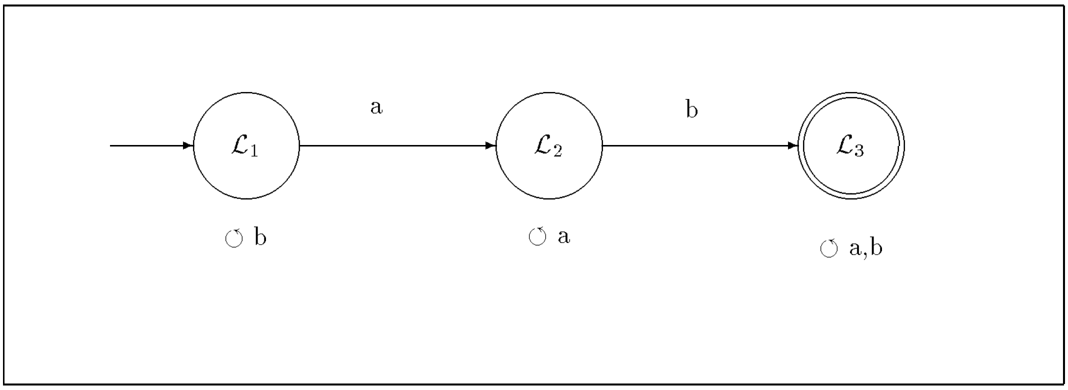

Figura 2.10: Construção do autómato canónico

2.4.3 Equivalência e Isomorfismo

Que relação existe entre um autómato $\mathcal{A}$ que reconhece uma linguagem $\mathcal{L}$, a sua versão reduzida $\mathcal{A}^m$ e o autómato canónico correspondente a $\mathcal{L}$? Vimos já que se tratam de autómatos equivalentes entre si na medida em que reconhecem a mesma linguagem. Vale a pena registar essa noção de equivalência.

Definição 2.4.8. Diz-se que dois autómatos $\mathcal{A}_1$ e $\mathcal{A}_2$ são **equivalentes**, e representa-se por $\mathcal{A}_1 \equiv \mathcal{A}_2$ sse $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.

Entre $\mathcal{A}^m$ e $\mathcal{A}_{\mathcal{L}}$, porém, é possível estabelecer uma relação muito mais forte. Tratam-se de autómatos *estruturalmente análogos* ou, para usar o termo que em álgebra capta este tipo de propriedade, *isomorfos*. Precisemos essa noção.

Definição 2.4.9. Dados dois autómatos sobre um mesmo alfabeto, $\mathcal{A}_1$ e $\mathcal{A}_2$, diz-se que uma função $h : S_1 \rightarrow S_2$ é um **homomorfismo entre autómatos** se verificar as seguintes condições[7],

$$\begin{array}{ll} \textbf{i)} & h(s_{0_1}) \ = \ s_{0_2} \\ \textbf{ii)} & s \in F_1 \ \Rightarrow \ h(s) \in F_2 \text{ para todo } s \in S_1 \\ \textbf{iii)} & h(sa) \ = \ h(s)a \text{ para todo } s \in S_1, a \in \Sigma \end{array}$$

Definição 2.4.10. Nas condições da Definição 2.4.9, h diz-se um **isomorfismo** entre $\mathcal{A}_1$ e $\mathcal{A}_2$ se fôr, adicionalmente, uma função bijectiva[8]. Neste caso os dois autómatos são ditos isomorfos, escrevendo-se $\mathcal{A}_1 \cong \mathcal{A}_2$.

[7] Pela sua observação se diz que h *preserva a estrutura* de $\mathcal{A}_1$ ou que a *reflecte* em $\mathcal{A}_2$.

[8] Caso em que a estrutura dos autómatos é respeitada nos dois sentidos.

O seguinte teorema estabelece a relação anunciada.

Teorema 2.4.3. *Seja $\mathcal{A}^m$ a versão reduzida de um autómato que reconhece $\mathcal{L}$. Então $\mathcal{A}^m \cong \mathcal{A}_{\mathcal{L}}$.*

Prova: Seja $h : S^m \longrightarrow S_{\mathcal{L}}$ a função dada por $h(s) \triangleq s^{-1}F^m$ para todo o $s \in S^m$, *i.é*, a cada estado h faz corresponder o seu conjunto terminal, que é, como facilmente se comprova, um estado de $S_{\mathcal{L}}$. A injectividade de h decorre da própria definição de S^m e a verificação da sobrejectividade é proposta como exercício ao leitor. Verifiquemos, aqui, que h constitui um homomorfismo, analisando as condições da Definição 2.4.9.

$$\begin{array}{llll}
\textbf{i)} & h(s_0^m) & = & (s_0^m)^{-1}F^m \\
 & & = & \{\omega \in \Sigma^* |\ s_0^m\omega \in F^m\} \\
 & & = & \mathcal{L} \\
 & & = & s_{0_{\mathcal{L}}} \\
 & & & \\
\textbf{ii)} & s \in F^m & sse & \lambda \in s^{-1}F^m \\
 & & sse & s^{-1}F^m \in F_{\mathcal{L}} \\
 & & sse & h(s) \in F_{\mathcal{L}} \\
 & & & \\
\textbf{iii)} & h(sa) & = & (sa)^{-1}F^m \\
 & & = & a^{-1}(s^{-1}F^m) \\
 & & = & h(s^{-1}F^m)a
\end{array}$$

❏

2.5 As Linguagens Regulares Revisitadas

Definimos linguagem regular como a linguagem reconhecida por umautómato finito. Ou, dito de outro modo, como o objecto formal que representa o tipo comum de observação do comportamento de um autómato finito. Vamos, nesta secção, introduzir uma caracterização puramente algébrica (*i.é*, "*machine-independent*") deste tipo de linguagens. A sua correcção foi estabelecida por Kleene, em 1956, num teorema que recebeu o seu nome.

A caracterização algébrica das linguagens regulares vai permitir-nos definir um formalismo, alternativo aos autómatos, para a sua especificação. Trata-se da notação das *expressões regulares* que o leitor, ao longo do curso, já terá certamente encontrado.

2.5.1 O Teorema de Kleene

Comecemos pelo seguinte resultado.

Teorema 2.5.1. *Todo o subconjunto finito de Σ^* é regular.*

Prova: O conjunto vazio $\emptyset$ e os conjuntos singulares constituem os subconjuntos finitos primitivos de Σ^*. Todos os outros são contruidos, a partir destes, por aplicação do operador $\cup$ um número arbitrário, mas finito, de vezes. Podemos, então, dividir a prova em 3 partes, mostrando que:

i) $\mathcal{L} = \emptyset$ é regular. Basta considerar o autómato finito não determinístico da Figura 2.5.1.

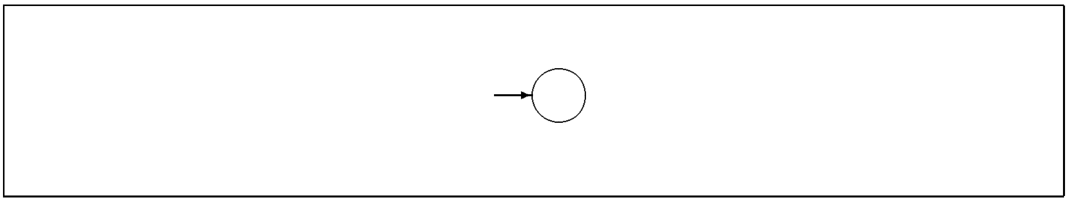

Figura 2.11: $\mathcal{L} = \emptyset$

ii) $\mathcal{L} = \{x\}$, para $x \in \Sigma$ ou $x = \lambda$, é regular. De novo é fácil construir os autómatos indicados (Figura 2.5.1).

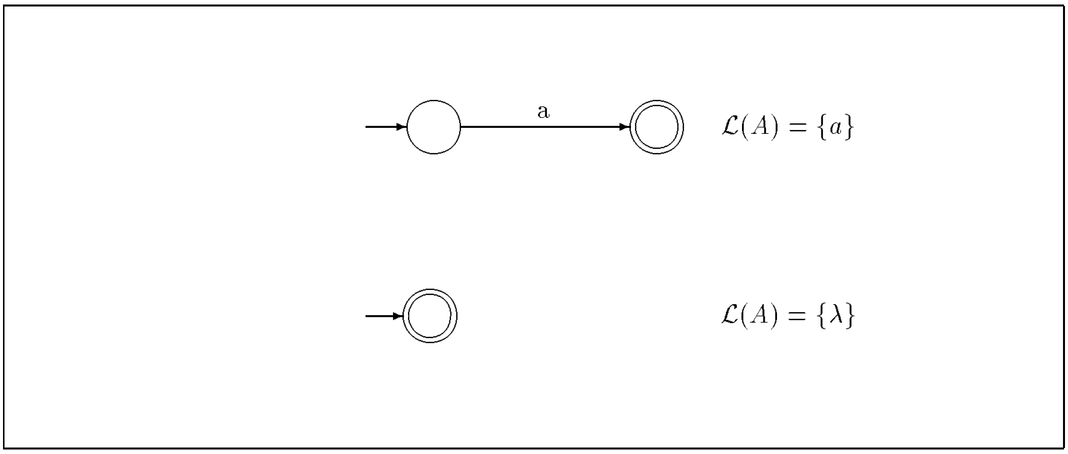

Figura 2.12: $\mathcal{L} = \{a\}$ e $\mathcal{L} = \{\lambda\}$

iii) Se $\mathcal{L}_1, \mathcal{L}_2$ são regulares, $\mathcal{L}_1 \cup \mathcal{L}_2$ também o é. Sejam $\mathcal{A}_1 = (\Sigma, S_1, s_{0_1}, \delta_1, F_1)$ e $\mathcal{A}_2 = (\Sigma, S_2, s_{0_2}, \delta_2, F_2)$ tais que $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}_1$ e $\mathcal{L}(\mathcal{A}_2) = \mathcal{L}_2$. Supondo, sem perda de generalidade, que $S_1 \cap S_2 = \emptyset$, definamos $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ de acordo com

$$\begin{array}{l} S = S_1 \cup S_2 \\ S_0 = \{s_{0_1}, s_{0_2}\} \\ F = F_1 \cup F_2 \\ \delta(s,a) = \left\{ \begin{array}{ll} \delta_1(s,a) & \Leftarrow s \in S_1 \\ \delta_2(s,a) & \Leftarrow s \in S_2 \end{array} \right. \end{array}$$

Temos, então,

$$\begin{array}{rll} \omega \in \mathcal{L}(\mathcal{A}) & sse & \exists_{s \in S_0}.\ s\omega \cap F \neq \emptyset \\ & sse & \exists_{s \in \{s_{0_1}, s_{0_2}\}}.\ s\omega \cap F \neq \emptyset \\ & sse & \exists_{s \in s_{0_1}}.\ s\omega \cap F \neq \emptyset \ \vee \ \exists_{s \in s_{0_2}}.\ s\omega \cap F \neq \emptyset \\ & sse & \exists_{s \in s_{0_1}}.\ s\omega \cap F_1 \neq \emptyset \ \vee \ \exists_{s \in s_{0_2}}.\ s\omega \cap F_2 \neq \emptyset \\ & sse & \omega \in \mathcal{L}(\mathcal{A}_1) \ \vee \ \omega \in \mathcal{L}(\mathcal{A}_2) \\ & sse & \omega \in \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) \end{array}$$

É, ainda, necessário provar a propriedade que acabamos de usar para recorrer a $\overline{\delta}_1$ e $\overline{\delta}_2$ em substituição de $\overline{\delta}$. O facto

$$\overline{\delta}(s,\omega) = \begin{cases} \overline{\delta}_1(s,\omega) & \Leftarrow s \in S_1 \\ \overline{\delta}_2(s,\omega) & \Leftarrow s \in S_2 \end{cases}$$

decorre, por indução sobre o comprimento de ω, da definição de δ.

❏

A definição apresentada para o autómato que reconhece $\mathcal{L}_1 \cup \mathcal{L}_2$ não é, por certo, única, pelo que existem outros caminhos para provar a última parte do teorema anterior. Uma alternativa toma como espaço de estados o produto cartesiano dos espaços de estados parcelares. Define-se, assim, $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$, com

$$\begin{array}{l} S = S_1 \times S_2 \\ S_0 = \{s_{0_1}\} \times \{s_{0_2}\} \\ F = (F_1 \times S_2) \cup (S_1 \times F_2) \\ \delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a)) \end{array}$$

Seja $\omega \in \mathcal{L}_1$. Então, $s_{0_1}\omega \in F_1$ e $(s_{0_1}, s_{0_2})\omega \in F_1 \times S_2 \subseteq F$. Logo, $\mathcal{L}_1 \subseteq \mathcal{L}(\mathcal{A})$ e, por um argumento análogo, $\mathcal{L}_2 \subseteq \mathcal{L}(\mathcal{A})$. I.é, $\mathcal{L}_1 \cup \mathcal{L}_2 \subseteq \mathcal{L}(\mathcal{A})$. Conversamente, se $\omega \in \mathcal{L}(\mathcal{A})$ então $(s_{0_1}, s_{0_2})\omega \in F$, ou seja, $(s_{0_1}, s_{0_2})\omega$ pertence a $F_1 \times S_2$ ou $S_1 \times F_2$. Donde, $s_{0_1}\omega \in F_1$ ou $s_{0_1}\omega \in F_2$, *i.é*, $\omega \in \mathcal{L}_1$ ou $\omega \in \mathcal{L}_2$. Logo $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_1 \cup \mathcal{L}_2$. Note-se, todavia, que $\mathcal{A}$ terá, regra geral, um número de estados muito superior ao do autómato baseado na reunião de S_1 e S_2 que atrás discutimos.

O último resultado do teorema 2.5.1 é, por vezes, referida como o fecho para a reunião das linguagens regulares. Com a palavra *fecho* quer-se precisamente dizer que uma dada propriedade (neste caso, a regularidade de uma linguagem) que é verificada pelos argumentos de uma operação (neste caso, a reunião de conjuntos), o é igualmente pelo seu resultado. Trata-se de uma situação com deparamos frequentemente em matemática[9]. De facto, a utilidade de um conceito formal é, não raro, função da sua preservação por determinadas operações. O leitor é convidado a mostrar que o conjunto das linguagens regulares é fechado para outros operadores usuais entre conjuntos — por exemplo, a intersecção e a complementação — fazendo-o quer construtivamente quer por combinação de outros resultados.

Por agora, porém, interessa-nos analisar o fecho destas linguagens para as operações produto e fecho de Kleene.

Teorema 2.5.2. *Sejam $\mathcal{L}_1$ e $\mathcal{L}_2$ duas linguagens regulares. Então, $\mathcal{L}_1\mathcal{L}_2$ também o é.*

Prova: Sejam $\mathcal{A}_1 = (\Sigma, S_1, s_{0_1}, \delta_1, F_1)$ e $\mathcal{A}_2 = (\Sigma, S_2, s_{0_2}, \delta_2, F_2)$ tais que $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}_1$ e $\mathcal{L}(\mathcal{A}_2) = \mathcal{L}_2$. Suponhamos, sem perda de generalidade, que $\mathcal{A}_1$ e $\mathcal{A}_2$ são determinísticos e completos e que $S_1 \cap S_2 = \emptyset$. O autómato correspondente a $\mathcal{L}_1\mathcal{L}_2$ traduzirá a ideia intuitiva de uma *ligação em série* dos dois autómatos. O espaço de estados reunirá S_1 e S_2, o estado inicial coincidirá com o correspondente em $\mathcal{A}_1$ (recorde-se que uma palavra de $\mathcal{L}_1\mathcal{L}_2$ é sempre iniciada por uma palavra de $\mathcal{L}_1$) e os estados finais serão os de $\mathcal{A}_2$ (toda a palavra termina numa palavra de $\mathcal{L}_2$). A função de transição deverá articular δ_1 e δ_2. Mais concretamente, δ_1 será aplicada sempre que esteja em causa uma transição iniciada num estado $s \in S_1$ e δ_2

[9] Por exemplo, $\mathbb{N}$ é fechado para a multiplicação mas não para a radiciação.

sempre que $s \in S_2$. A ligação entre as duas partes do autómato é estabelecida pela adição do estado inicial de $\mathcal{A}_2$ ao resultado de toda a transição que conduza a um estado final de $\mathcal{A}_1$, permitindo, desse modo, o prosseguimento do processo de reconhecimento. Definamos, assim, $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ de acordo com

$$
\begin{aligned}
&S = 2^{S_1 \cup S_2} \\
&S_0 = \{s_{0_1}\} \\
&F = F_2 \\
&\delta(s,a) = \begin{cases} \{\delta_1(s,a)\} & \Leftarrow s \in S_1 \ \wedge \ \delta_1(s,a) \notin F_1 \\ \{\delta_1(s,a), s_{0_2}\} & \Leftarrow s \in S_1 \ \wedge \ \delta_1(s,a) \in F_1 \\ \{\delta_2(s,a)\} & \Leftarrow s \in S_2 \end{cases}
\end{aligned}
$$

Mostremos, agora, que $\mathcal{L}(\mathcal{A}) = \mathcal{L}_1\mathcal{L}_2$.

Seja $\omega = \omega_1\omega_2$, onde $\omega_1 \in \mathcal{L}_1$ e $\omega_2 \in \mathcal{L}_2$. Então, $\overline{\delta}_1(s_{0_1}, \omega_1) \in F_1$ e, atendendo à definição de δ, $s_{0_2} \in \overline{\delta}(s_{0_1}, \omega_1)$. Segue-se que $\overline{\delta}(s_0, \omega)$ cotém $\overline{\delta}_2(s_{0_2}, \omega_2)$ e, como $\omega_2 \in \mathcal{L}_2$, $\overline{\delta}(s_0, \omega)$ tem uma intersecção não vazia com $F = F_2$. Logo, $\omega \in \mathcal{L}(\mathcal{A})$. I.é, $\mathcal{L}_1\mathcal{L}_2 \subseteq \mathcal{L}(\mathcal{A})$.

Conversamente, suponhamos que $\omega \in \mathcal{L}(\mathcal{A})$, o que significa que $\overline{\delta}(s_0, \omega) \cap F \neq \emptyset$. No reconhecimento de ω em $\mathcal{A}$, iniciado em $s_{0_1} \in S_0$, as transições de $\mathcal{A}_2$ apenas são activadas se existir pelo menos um prefixo ω_1 de ω tal que $\overline{\delta}_1(s_{0_1}, \omega_1) \in F_1$, *i.é*, $\omega_1 \in \mathcal{L}_1$. Neste caso, $s_{0_2} \in \overline{\delta}(s_0, \omega_1)$ e o "resto" de ω (*i.é*, ω_2 tal que $\omega = \omega_1\omega_2$) é reconhecido a partir de s_{0_2}. Donde, $\overline{\delta}(s_{0_2}, \omega_2) \cap F \neq \emptyset$ o que equivale a $\omega_2 \in \mathcal{L}_2$. Logo, $\omega \in \mathcal{L}_1\mathcal{L}_2$ mostrando que $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_1\mathcal{L}_2$, o que nos permite concluir a igualdade procurada.

❏

Teorema 2.5.3. *Seja $\mathcal{L}$ uma linguagem regular. Então, $\mathcal{L}^*$ também o é.*

Prova: Organizaremos a prova de acordo com a seguinte cadeia de implicações:

$$
\begin{array}{ccc}
\mathcal{L} \text{ regular} & \Rightarrow & \mathcal{L} - \{\lambda\} \text{ regular} \\
& & \Downarrow \\
& & (\mathcal{L} - \{\lambda\})^* \text{ regular} \\
& & \Downarrow \\
& & \mathcal{L}^* \text{ regular}
\end{array}
$$

A última implicação justifica-se trivialmente: $(\mathcal{L} - \{\lambda\})^* = \mathcal{L}^*$. As outras duas demonstram-se por construção. Assim

i) $\mathcal{L}$ regular $\Rightarrow \mathcal{L} - \{\lambda\}$ regular

Seja $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ o autómato determinístico que reconhece $\mathcal{L}$. Define-se um novo autómato $\mathcal{A}'$ que impossibilita o reconhecimento de λ recorrendo a uma etiquetação dos estados. Assim,

$$\mathcal{A}' = (\Sigma, S \times \{0,1\}, (s_0, 0), \delta', F \times \{1\})$$

com $\delta'((s,0),a) = \delta'((s,1),a) = (\delta(s,a),1)$, onde 0 etiqueta apenas o estado inicial que queremos garantir não poder ser terminal (caso em que $\lambda \in \mathcal{L}(\mathcal{A}')$). A extensão de

δ' a Σ^* força $\overline{\delta}'((s_0,0),\lambda) = (s_0,0)$. Logo $\lambda \notin \mathcal{L}(\mathcal{A}')$. Por outro lado, $\overline{\delta}'((s_0,0),\omega) \in F \times \{1\}$ sse $\overline{\delta}(s_0,\omega) \in F$. Donde, $\omega \in \mathcal{L}(\mathcal{A}')$ sse $\omega \neq \lambda$ e $\omega \in \mathcal{L}$.

ii) $\mathcal{L} - \{\lambda\}$ regular $\Rightarrow (\mathcal{L} - \{\lambda\})^*$ regular

Esta implicação reduz-se a $\mathcal{L} - \{\lambda\}$ regular $\Rightarrow$ $(\mathcal{L} - \{\lambda\})^+$ regular, considerando $\mathcal{L}^* = \mathcal{L}^+ \cup \{\lambda\}$ e aplicando o teorema 2.5.1.

Para simplificar a notação, representaremos $\mathcal{L} - \{\lambda\}$ por $\mathcal{L}$, supondo sempre que $\lambda \notin \mathcal{L}$. Seja, então $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ o autómato determinístico que reconhece $\mathcal{L}$. O autómato correspondente a $\mathcal{L}^+$ será estruturalmente análogo a $\mathcal{A}$ com uma única diferença: de todo o estado final de $\mathcal{A}$ será possível retornar ao estado inicial, o que capta a presença intuitiva de um ciclo na definição de $\mathcal{L}^+$. Assim, definimos

$$\mathcal{A}' = (\Sigma, 2^S, \{s_0\}, \delta', F)$$

com

$$\delta'(s,a) = \begin{cases} \{\delta(s,a)\} & \Leftarrow \delta(s,a) \notin F \\ \{\delta(s,a), s_0\} & \Leftarrow \delta(s,a) \in F \end{cases}$$

Seja $\omega \in \mathcal{L}^+$. Se $\omega \in \mathcal{L}$ então $\overline{\delta}'(s,\omega) \cap F \neq \emptyset$ e, logo, $\omega \in \mathcal{L}(\mathcal{A}')$. Consideremos, agora, o caso geral em que $\omega \notin \mathcal{L}$. Por definição de $\mathcal{L}^+$, ω tem a forma geral $\omega = \omega_1\omega_2...\omega_n$ com $n > 0$ e $\omega_1, \omega_2, ..., \omega_n \in \mathcal{L}$. Suponhamos que $\omega' = \omega_1\omega_2...\omega_{n-1} \in \mathcal{L}(\mathcal{A}')$. Então, $\overline{\delta}'(s,\omega') \cap F \neq \emptyset$ e $s_0 \in \overline{\delta}'(s,\omega')$. Temos, assim,

$$\overline{\delta}'(\overline{\delta}'(s_0,\omega'),\omega_n) = \bigcup_{s \in \overline{\delta}'(s_0,\omega')} \overline{\delta}'(s,\omega_n) = ... \cup \overline{\delta}'(s_0,\omega_n) \cup ...$$

Como $\omega_n \in \mathcal{L}$, $\overline{\delta}'(s_0,\omega_n) \cap F \neq \emptyset$ o que nos permite concluir que $\overline{\delta}'(\overline{\delta}'(s_0,\omega'),\omega_n) \cap F \neq \emptyset$. Ou seja, $\overline{\delta}'(s_0,\omega'\omega_n) \cap F \neq \emptyset$. Logo, $\omega = \omega'\omega_n \in \mathcal{L}(\mathcal{A}')$, *i.é*, $\mathcal{L}^+ \subseteq \mathcal{L}(\mathcal{A}')$.

Conversamente, suponhamos que $\omega \in \mathcal{L}(\mathcal{A}')$, *i.é*, $\overline{\delta}'(s_0,\omega) \cap F \neq \emptyset$. Seja ω_1 o menor prefixo de ω tal que $\overline{\delta}'(s_0,\omega_1) \cap F \neq \emptyset$ (tal prefixo existe sempre, nem que coincida com ω). Então,

$$\overline{\delta}'(s_0,\omega_1) = \{\overline{\delta}(s_0,\omega_1), s_0\}$$

e $\omega_1 \in \mathcal{L}$. Se $\omega = \omega_1$, $\omega \in \mathcal{L} \subseteq \mathcal{L}^+$. Se $\omega \neq \omega_1$, $\omega = \omega_1\omega'$. Tomemos, de novo, o menor prefixo de ω' — ω_2 — tal que $\overline{\delta}'(s_0,\omega_1\omega_2)$ contém um elemento de F. Então

$$\overline{\delta}'(s_0,\omega_1\omega_2) = \{\overline{\delta}(s_0,\omega_1\omega_2), \overline{\delta}(s_0,\omega_2), s_0\}$$

e, necessariamente, $\omega_1\omega_2$ ou ω_2 pertencem a $\mathcal{L}$. Em qualquer caso, $\omega_1\omega_2$ e ω_2 pertencem a $\mathcal{L}^+$. Se $\omega = \omega_1\omega_2$ podemos concluir. Senão, $\omega = \omega_1\omega_2\omega''$ e o processo referido pode continuar. Como o comprimento de ω é finito, o processo terminará garantidamente e concluimos que $\omega \in \mathcal{L}^+$. Logo, $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}^+$.

❏

Os teoremas 2.5.1, 2.5.2 e 2.5.3 estabelecem como regulares, *i.é*, reconhec1veis por um autómato finito, os subconjuntos vazio e singulares de Σ^* assim como todos os outros que são obtidos por aplicação finita da reunião, produto e fecho de Kleene. Tais subconjuntos de Σ^* são, por isso mesmo, designados por *conjuntos regulares*.

Exemplo 2.5.1. Os seguintes subconjuntos de $\{a, b\}^*$ são regulares:

$$\{ab, ba\}^*$$
$$\{a\}^* \cup \{b\}^*$$
$$\{a, b\}^+ \text{ (porque } = \{a\}\{a, b\}^* \cup \{b\}\{a, b\}^* \text{)}$$

Para completar a caracterização algébrica das linguagens regulares que empreendemos, faltanos demonstrar o resultado converso. Assim,

Teorema 2.5.4. *Toda a linguagem regular sobre um alfabeto Σ é um subconjunto regular de Σ^**

Prova: Seja $\mathcal{L}$ uma linguagem regular e $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ o autómato finito determinístico que a reconhece. Então, $\mathcal{L} = \mathcal{L}(\mathcal{A}) = \{\omega \in \Sigma^* | \ s_0\omega \in F\}$. Sem dificuldade se verifica que $\mathcal{L}$ pode ser escrito como a reunião de um número finito de conjuntos de palavras cada um deles colectando as palavras de $\mathcal{L}$ reconhecidas em cada estado final $f \in F$. I.é,

$$\mathcal{L} = \bigcup_{f \in F} L(s_0, f)$$

com $L(s_0, f) = \{\omega \in \Sigma^* | \ s_0\omega = f\}$.

É, então, suficiente mostrar que cada $L(s_0, f)$ forma um conjunto regular. Fa-lo-emos provando um resultado um pouco mais geral: para todo o $s, t \in S$ $L(s, t)$ é regular. Note-se que cada $L(s, t)$ é o conjunto de palavras que etiquetam todos os percursos em $\mathcal{A}$ de s a t. Se nos restringirmos a um subconjunto $R \subseteq S$, podemos ainda definir os conjuntos

$$L(s, R, t) = \{\omega \in \Sigma^* | \ s\omega = t \ \wedge \ s\omega' \in R \text{ para todos os prefixos } \omega' \text{ de } \omega\}$$

i.é, o conjunto de palavras que etiquetam todos os percursos em $\mathcal{A}$ de s a t visitando apenas estados em R. Igualmente se define

$$K(s, R, t) = \{\omega \in \Sigma^* | \ s\omega = t \ \wedge \ s\omega' \in R \text{ para todos os prefixos estritos } \omega' \text{ de } \omega\}$$

onde *prefixo estrito* significa que nunca se tem $\omega' = \omega$. Obviamente, neste caso, $s, t \notin R$.

O nosso objectivo será mostrar que $L(s, t)$ $(= L(s, S, t))$ é um conjunto regular. De novo provaremos um resultado mais geral: $L(s, R, t)$ e $K(s, T, t)$, para R contendo s e t e T não contendo s e t, são regulares. A prova procede por indução sobre o cardinal de R e T.

No caso de base deparamos com as seguintes possibilidades:

$|R| = 1$ Esta situação ocorre sse $s = t$ (note-se que nunca se tem $|R| = 0$). Neste caso, $L(s, \{s\}, s) = (\Sigma(s, s))^*$, onde $\Sigma(s, t) = \{a \in \Sigma | \ sa = t\}$. Como $\Sigma(s, t)$ é finito, logo regular, também o é $(\Sigma(s, t))^*$.

$|T| = 0$ Neste caso $K(s, \emptyset, t) = (\Sigma(s,t))^+$, logo regular.

$|T| = 1$ Agora $T = \{v\}$, para um $v \in S$ e distinto de s e t. Então $K(s, \{v\}, t) = \Sigma(s,v)(\Sigma(v,v))^*\Sigma(v,t)$ que também é regular.

Seja, agora, $|R|, |T| \geq 2$. Suponhamos, por hipótese de indução, que $L(s, R, t)$ e $K(s, T, t)$ são regulares para $\{s,t\} \subset R$ e $s, t \notin T$ e, ainda, $|R|, |T| < n$. Consideremos $L(s, R, t)$ com $|R| = n$ e seja $\omega \in L(s, R, t)$. Podemos supôr que o percurso de s a t etiquetado por ω visita s diversas vezes. I.é,

$$\omega = \omega_1\omega_2...\omega_k\omega'$$

para algum $k \geq 0$, e onde $\omega_1\omega_2...\omega_k \in K(s, R - \{s\}, t)$. Se $s = t$ então $\omega' = \lambda$ e temos $L(s, R, t) = (K(s, R - \{s\}, t))^*$ que, por hipótese de indução, é regular. Se, porém, $s \neq t$ é sempre possível escolher um prefixo v_1 de ω' tal que $sv_1 = t$. Então, $v_1 \in K(s, R - \{s,t\}, t)$ e $\omega = \omega_1\omega_2...\omega_k v_1 v_2$ (para $\omega' = v_1 v_2$). Como $v_2 \in L(s, R - \{s\}, t)$, concluimos que

$$L(s, R, t) \;=\; (K(s, R - \{s\}, t))^* K(s, R - \{s,t\}, t) L(s, R - \{s\}, t)$$

$L(s, R, t)$ é, pois, um conjunto obtido pelo produto e fecho de Kleene de conjuntos que, por hipótese de indução, são regulares. Logo é igualmente regular.

Mostremos, também, que $K(s, T, t)$, com $|T| = n$ e $s, t \notin T$ é regular. Estamos a considerar as etiquetas do percurso

$$s \xrightarrow{a} r \longrightarrow ... \longrightarrow v \xrightarrow{b} t$$

com $r, v \in T$, que têm a forma geral $\omega = a\omega' b$. Certamente, $\omega' \in L(s, T, t)$. Assim,

$$K(s, T, t) \;=\; \bigcup_{r,v \in T} (\Sigma(s,r) L(r, T, v) \Sigma(v,t))$$

Dado que $\Sigma(s, s')$ é sempre regular, para qualquer $s, s' \in S$, e, como acima provamos, $L(r, T, v)$ também o é, segue-se que $K(s, T, t)$ é ainda um conjunto regular, o que completa a demonstração.

❏

Estamos, por fim, em condições de enunciar o resultado que sumaria a caracterização algébrica das linguagens regulares.

Teorema 2.5.5 (Kleene, 1956). *Uma linguagem sobre um alfabeto Σ é regular,* i.é, *reconhecida por um autómato finito determinístico, sse formar um subconjunto regular de Σ^*.*

Prova: Os teoremas 2.5.1, 2.5.2 e 2.5.3 estabelecem a implicação num sentido (*i.é*, os subconjuntos regulares de Σ^* são linguagens reconhecidas por autómatos finitos). O teorema 2.5.4 estabelece-a no outro (*i.é*, as linguagens reconhecidas por autómatos finitos são subconjuntos regulares de Σ^*). A sua conjunção constitui a prova deste resultado.

❏

2.5.2 Expressões Regulares

Até aqui recorremos ao conceito de autómato finito, nas suas múltipla variantes, como veículo de especificação de linguagens regulares. Neste contexto, a especificação de uma linguagem é feita por construção do modelo matemático da máquina abstracta que o reconhece — o *autómato*. Este modelo, se torna simples a averiguação da pertença de uma particular palavra à linguagem, tem uma desvantagem com que já deparamos na prática. De facto, nem sempre é simples, a partir de um autómato, descrever completamente o conjunto das palavras aceites, *i.é*, $\mathcal{L}(\mathcal{A})$.

Que alternativas existem? Se a linguagem é finita podemos optar pela simples enumeração das palavras que a formam (o que, não obstante, pode requer várias toneladas de papel ...). E se o não fôr?

A caracterização algébrica das linguagens regulares proporcionada pelo Teorema de Kleene oferece-nos uma alternativa com um grande espectro de utilização prática — a saber, a linguagem das *expressões regulares*. Trata-se de uma notação específica para descrever conjuntos regulares do tipo dos que introduzimos atrás.

Definição 2.5.1. Uma **expressão regular** sobre um alfabeto Σ é uma sequência de símbolos formada pela aplicação repetida das regras seguintes.

1. $\emptyset$ é uma expressão regular que denota o conjunto vazio.
2. ε é uma expressão regular que denota o conjunto $\{\lambda\}$.
3. Para todo o $a \in \Sigma$, a é uma expressão regular que denota o conjunto $\{a\}$.
4. Se E_1 e E_2 forem expressões regulares, representando os conjuntos regulares $\mathcal{L}_1$ e $\mathcal{L}_2$, então (E_1E_2) é a expressão regular correspondente ao conjunto $\mathcal{L}_1\mathcal{L}_2$, $(E_1 + E_2)$ é a expressão regular correspondente ao conjunto $\mathcal{L}_1 \cup \mathcal{L}_2$ e E_1^* é a expressão regular correspondente ao conjunto $\mathcal{L}_1^*$.

Para facilitar a escrita, estabelece-se uma hierarquia de precedências que ordena os operadores fecho de Kleene, produto e reunião da maior para a menor precedência, o que permite minimizar a escrita de parentesis.

Exemplo 2.5.2. Os conjuntos regulares $\{\lambda\}$, $\{b\} \cup \{c\}$ e $(\{a\} \cup \{\lambda\})\{b\}^*$ são representados pelas expressões regulares ε, $b + c$ e $(a + \varepsilon)b^*$, respectivamente.

Note-se que a representação de uma linguagem por uma expressão regular não é única. De facto, por exemplo, $a(b+c)$ e $ab+ac$ ou b e $b\varepsilon$, especificam a mesma linguagem — $\{ab, ac\}$ e

$\{b\}$, respectivamente. Quer isto dizer que, ao representar uma linguagem por um conjunto regular (escrito, ususalmente, na forma mais agradável de uma expressão regular) fazemo-lo sempre a menos da igualdade em 2^{Σ^*}. Um conjunto de igualdades permite-nos, de resto, transformar uma expressão regular noutra, sempre que conveniente. Temos assim,

Teorema 2.5.6. *Sejam E, E_1, E_2 e E_3 expressões regulares. Então são válidas os seguintes factos:*

1. $(2^{\Sigma^*}; +, \emptyset)$ *forma um monoide comutativo e idempotente.*
2. $(2^{\Sigma^*}; \_, \varepsilon)$ *forma um monoide não comutativo.*
3. $E_1(E_2 + E_3) \ = \ E_1E_2 + E_1E_3$
4. $(E_2 + E_3)E_1 \ = \ E_2E_1 + E_3E_1$
5. $\varepsilon^* \ = \ \emptyset^* \ = \ \varepsilon$
6. $(E^*)^* \ = \ E^*$
7. $E^*E^* \ = \ E^*$
8. $(E_1 + E_2)^* \ = \ (E_1^* + E_2^*)^* \ = \ (E_1^*E_2^*)^*$

Prova: A verificação de cada uma destas igualdddades baseia-se nas propriedades dos operadores usuais sobre conjuntos acrescidos do *produto* e *fecho de Kleene* que introduzimos nas Definições 2.1.5 e 2.1.6. A prova é rotineira, sendo proposta como exercício ao leitor.

❏

As expressões regulares são úteis para descrever a sintaxe das entidades básicas de uma linguagem de programação, como se pode ver pelo exemplo seguinte. De facto, uma das utilizações mais comuns da teoria dos autómatos finitos é, precisamente, o processamento da análise léxica das linguagens de programação — um dos andares fundamentais na estrutura de um tradutor.

Exemplo 2.5.3.

$$
\begin{array}{lcl}
\textit{Inteiro} & = & \textit{Sinal Digito} \\
\textit{Sinal} & = & \varepsilon + \texttt{+} - \texttt{+} \\
\textit{Digito} & = & (\texttt{0} + \texttt{1} + \texttt{2} + \ldots + \texttt{9})^+ \\
\textit{PalavraReservada} & = & \texttt{begin} + \texttt{end} + \texttt{if} + \texttt{then} + \texttt{else} + \ldots \\
\ldots & = & \ldots
\end{array}
$$

Constituindo autómatos finitos e expressões regulares duas formas alternativas de especificação das linguagens que vimos discutindo, interessa-nos analisar as regras da sua mútua conversão.

Vimos já que é relativamente simples, dada uma expressão regular, construir o autómato que reconhece a linguagem por ela definida. Basicamente, trata-se de uma combinação dos autómatos associados aos construtores básicos de conjuntos regulares tal como os descrevemos nas demonstrações dos teoremas 2.5.1, 2.5.2 e 2.5.3. A resposta à questão inversa é menos imediata.

Interessa-nos determinar $\mathcal{L}(\mathcal{A})$, para um autómato $\mathcal{A}$ dado. De acordo com a definição 2.2.4, $\mathcal{L}(\mathcal{A}) = \{\omega \in \Sigma^* |\ s_0\omega \in F\}$, ou, usando a notação atrás introduzida, $\mathcal{L}(\mathcal{A}) = s_0^{-1}F$. I.é, $\mathcal{L}(\mathcal{A})$ é o conjunto terminal correspondente ao estado inicial — $T(s_0, \mathcal{A})$[10] Reparemos, agora, que as relações entre os diversos conjuntos terminais de um autómato podem ser descritas por um sistema de equações sobre 2^{Σ^*}.

Suponhamos que, num autómato com 3 estados, s é final e pode apenas ser alcançado a partir dos estados s' e s'' por transições etiquetadas com os símbolos a e b, respectivamente. Por hipótese, as transições iniciadas em s conduzem, por b a s', por c a s'' e por a de novo a s. Nesta situação o conjunto terminal de s — $T(s, \mathcal{A})$ — pode ser igualado à reunião das palavras que constituem $T(s', \mathcal{A})$ prefixadas pelo símbolo que provoca a transição de s para s' — b, neste caso — com as palavras de $T(s'', \mathcal{A})$ prefixadas por c e as de $T(s, \mathcal{A})$ prefixadas por a. I.é,

$$T(s, \mathcal{A}) \;=\; bT(s', \mathcal{A}) + cT(s'', \mathcal{A}) + aT(s, \mathcal{A}) + \varepsilon$$

A inclusão de ε é necessária uma vez que λ pertence sempre ao conjunto terminal de um estado final. Note-se, porém, que no cálculo de $T(s', \mathcal{A})$ intervém também $T(s, \mathcal{A})$. Em particular,

$$T(s', \mathcal{A}) \;=\; ... + aT(s, \mathcal{A}) + ...$$

Este exemplo sugere-nos, como acima referimos, que as relações entre os diversos conjuntos terminais de um autómato podem ser descritas por um sistema de equações sobre 2^{Σ^*}. No caso geral, considerando um autómato com n estados, numerados de 1 a n, teremos o seguinte sistema de n equações a n incógnitas, onde, para simplificar a escrita, se representa $T(i, \mathcal{A})$ por X_i.

$$\begin{array}{lcl}
X_1 & = & E_1 + A_{11}X_1 + A_{12}X_2 + ... + A_{1n}X_n \\
X_2 & = & E_2 + A_{21}X_1 + A_{22}X_2 + ... + A_{2n}X_n \\
... & & ... \\
X_i & = & E_i + A_{i1}X_1 + A_{i2}X_2 + ... + A_{in}X_n \\
... & & ... \\
X_n & = & E_n + A_{n1}X_1 + A_{n2}X_2 + ... + A_{nn}X_n
\end{array}$$

onde o coeficiente A_{ij} representa o conjunto de símbolos que provocam uma transição do estado i para o estado j. Formalmente,

$$A_{ij} \;=\; \bigcup_{a \in \{x \in \Sigma |\ ix = j\}} \{a\}$$

O termo independente E_i capta a referida necessidade de incluir λ em $T(s, \mathcal{A})$ sempre que $s \in F$. Assim,

$$E_i \;=\; \begin{cases} \varepsilon & \text{se } i \in F \\ \emptyset & \text{se } i \notin F \end{cases}$$

[10] *cf.*, Definição 2.4.2.

Exemplo 2.5.4. Consideremos o autómato finito determinístico representado na Figura 2.5.4.

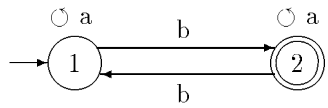

Figura 2.13: $\mathcal{A}$

Seguindo a construção introduzida, a relação entre os seus conjuntos terminais é expressa pelo seguinte conjunto de equações:

$$\begin{aligned} X_1 &= \emptyset + aX_1 + bX_2 \\ X_2 &= \varepsilon + bX_1 + aX_2 \end{aligned}$$

Obviamente $\mathcal{L}(\mathcal{A}) = X_1$.

A resolução deste tipo de sistemas determina uma expressão regular para cada um dos conjuntos terminais do autómato, e, em particular, para $T(s_0, \mathcal{A})$, como queríamos. A questão que resta pôr é de natureza técnica: como resolver efectivamente tais sistemas?

O resultado fundamental para responder a esta questão encontra-se no seguinte teorema.

Teorema 2.5.7. *Dado um alfabeto Σ e dois subconjuntos A e E de Σ^*, a equação $X = E + AX$ admite como solução mínima $X = A^*E$. Adicionalmente, se $\lambda \notin X$, a solução indicada é única.*

Prova: Comecemos por notar que A^*E é, de facto, uma solução da equação dada, usando o teorema 2.5.6.

$$\begin{aligned} E + A(A^*E) &= E + (AA^*)E \\ &= E + A^+E \\ &= (\varepsilon + A^+)E \\ &= A^*E \end{aligned}$$

Suponhamos, agora, que Y é outra solução da equação. Então, $Y = E + AY$ o que implica que $E \subseteq Y$ e $AY \subseteq Y$. Como, para qualquer R, S e T, se tem $R \subseteq T \Rightarrow SR \subseteq ST$, temos que $AE \subseteq AY$ e $AAY \subseteq AY$. Aplicando a mesma propriedade à primeira desigualdade vem $AAE \subseteq AAY$, o que nos leva a $AAE \subseteq AAY \subseteq AY \subseteq Y$. Admitamos, agora, como hipótese de indução, que para todo o $n \in \mathbb{N}$ se verifica $A^{n-1}E \subseteq A^{n-1}Y \subseteq AY \subseteq Y$. Então, $A^nE = AA^{n-1}E \subseteq AY \subseteq Y$. Por definição de A^* decorre então que $A^*E \subseteq Y$. I.é, qualquer solução da equação $X = E + AX$ contém A^*E, que, neste sentido, é mínima[11]. A prova da última parte do teorema é deixada como exercício para o leitor.

❏

Este resultado, combinado com as técnicas de substituição inversa usuais, permite-nos resolver o sistema que modela as relações entre os conjuntos terminais do autómato e chegar a uma expressão regular que descreva a linguagem por ele reconhecida. Como sempre, uma escolha judiciosa da ordem de resolução pode simplificar muito o processo. Note-se que a solução deste sistema de equações é única, atendendo ao teorema 2.5.7 e ao facto de nenhum dos conjuntos A_{ij} conter λ. O procedimento é ilustrado no exemplo seguinte.

Exemplo 2.5.5. Consideremos o sistema obtido no Exemplo 2.5.4. Reparemos que as equações consideradas têm a forma geral $X = E + AX$ (aplique a propriedade distributiva para se convencer disso). Teremos, então

$$\begin{array}{lcl}
X_1 & = & \emptyset + aX_1 + bX_2 \\
X_2 & = & \varepsilon + bX_1 + aX_2 \\
 & & \\
\equiv & & \\
 & & \\
X_1 & = & \emptyset + aX_1 + bX_2 \\
X_2 & = & (\varepsilon + bX_1) + aX_2 \\
 & & \\
\equiv & & \\
 & & \\
X_1 & = & \emptyset + aX_1 + bX_2 \\
X_2 & = & a^*(\varepsilon + bX_1) \\
 & & \\
\equiv & & \\
 & & \\
X_1 & = & \emptyset + aX_1 + ba^*(\varepsilon + bX_1) \\
X_2 & = & a^*(\varepsilon + bX_1) \\
 & & \\
\equiv & & \\
 & & \\
X_1 & = & \emptyset + aX_1 + ba^*\varepsilon + ba^*bX_1 \\
X_2 & = & a^*(\varepsilon + bX_1) \\
 & & \\
\equiv & & \\
 & & \\
X_1 & = & ba^*\varepsilon + aX_1 + ba^*bX_1 \\
X_2 & = & a^*(\varepsilon + bX_1)
\end{array}$$

[11] Em rigor, A^*E é o menor dos pontos fixos da equação, podendo ser calculado tomando o limite da cadeia de Kleene no recticulado $(2^{\Sigma^*}; \subseteq)$.

$$\equiv$$

$$\begin{array}{lcl} X_1 & = & ba^*\varepsilon + (a + ba^*b)X_1 \\ X_2 & = & a^*(\varepsilon + bX_1) \end{array}$$

$$\equiv$$

$$\begin{array}{lcl} X_1 & = & (a + ba^*b)^*ba^* \\ X_2 & = & a(\varepsilon + b(a + ba^*b)^*ba^*) \end{array}$$

Logo, $\mathcal{L}(\mathcal{A}) = (a + ba^*b)^*ba^*$.

O processo que acabamos de introduzir para a determinação de $\mathcal{L}(\mathcal{A})$, expressa por uma expresão regular, a partir de $\mathcal{A}$, é também aplicável aos autómatos não determinísticos. Uma pequena diferença surge, no entanto, na definição dos conjuntos A_{ij}, reflectindo a diferença entre as funções de transição determinística e não determinística. Assim

$$A_{ij} = \bigcup_{a \in \{x \in \Sigma |\ j \in ix\}} \{a\}$$

Se, porém, $\mathcal{A}$ possuir transições espontâneas é possível que pelo menos algum A_{ij} contenha λ. O sistema deixa, então de ter uma solução única e o processo descrito gerará apenas uma solução "mínima". Uma alternativa será converter primeiro $\mathcal{A}$ a um autómato $\mathcal{A}'$ sem transições espontâneas, de acordo com a construção discutida na prova do teorema 2.3.2, e calcular, pelo processo indicado, $\mathcal{L}(\mathcal{A}')$, que é obviamente igual a $\mathcal{L}(\mathcal{A})$.

2.6 Autómatos Reactivos

Uma elaboração importante do conceito de autómato é a noção de *autómato reactivo*. Trata-se de um modelo de uma máquina abstracta que, à medida que reconhece uma linguagem $\mathcal{L}$, é capaz de a traduzir para outra ou, mais genericamente, de realizar determinadas acções. Num sentido que ficará claro no Capítulo 6, diremos que na passagem da noção de autómato para a de autómato reactivo ocorre uma complexificação da estrutura de observação associada.

O que agora importa reter é o facto de um autómato reactivo constituir o modelo mínimo necessário para efectivamente emular um tradutor. É claro, porém, que da mesma forma que um autómato finito é incapaz de reconhecer linguagens com o nível de elaboração das linguagens de programação usuais, também um autómato reactivo será incapaz de modelar um interpretador desse tipo de linguagens. Tem, contudo, numerosas aplicações: um sistema de semáforos, um evelador ou um controlador de processos de fabrico respondendo a um conjunto finito de estímulos, caem facilmante sob a sua alçada.

Conceptualmente temos de lidar com dois alfabetos: um alfabeto Σ que especifica os símbolos recebidos (*i.é*, os estímulos a processar) e um alfabeto Γ que enumera os símbolos que o autómato produz em resposta a esses estímulos. É, também, essencial especificar a regra que decide a resposta a dar a um dado estímulo. No modelo (simplificado) que consideraremos esta será função do símbolo correntemente reconhecido e do estado presente do autómato. Por fim, e uma vez que o objectivo é descrever um processo de tradução e não um processo de reconhecimento, a noção de conjunto de estados finais deixa de ser relevante. Podemos sumariar estas características na seguinte definição.

Definição 2.6.1. Um **Autómato Reactivo**, também designado por **máquina de Mealy**, é um tuplo $(\Sigma, \Gamma, S, s_0, \delta, f)$ onde Σ é um alfabeto de entrada, Γ um alfabeto de saída, S um conjunto finito e não vazio de estados, $s_0 \in S$ o estado inicial, $\delta : S \times \Sigma \to S$ a função de transição de estados e $f : S \times \Sigma \to \Gamma$ a função resposta.

A representação diagramática de um autómato reactivo é análoga à que já utilizavamos com a diferença que cada transição passa a ser etiquetada por um par (a, x) com $a \in \Sigma$ e $x \in \Gamma$, devendo ler-se como "*o autómato transita do estado s para o estado s′ mediante o reconhecimento do símbolo a e produz, nessa transição, o símbolo x*".

Exemplo 2.6.1. Suponhamos um autómato reactivo $\mathcal{A}$ que realiza a monotoria de um determinado processo de fabrico recebendo dois tipos de estímulos — a e b — da linha de produção. Pretende-se que o autómato sinalize uma situação crítica correspondente à detecção da sequência de estímulos $\underline{aab}$, escrevendo no écran do terminal associado a palavra *ALARME*. Quando não fôr esse o caso, deve limitar-se a escrever um ponto. Identificamos, assim, $\Sigma = \{a, b\}$ e $\Gamma = \{., ALARME\}$. O diagrama da Figura 2.6.1 descreve o autómato pretendido.

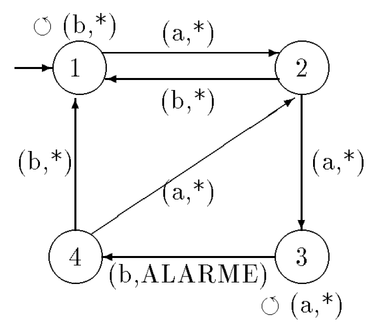

Figura 2.14: Autómato reactivo

Da mesma forma que estendemos δ a Σ^*, será conveniente realizar a correspondente extensão de f. Formalmente,

Definição 2.6.2. Dado um autómato reactivo $\mathcal{A} = (\Sigma, \Gamma, S, s_0, \delta, f)$ define-se recursivamente uma função $\overline{f} : S \times \Sigma^* \to \Gamma^*$, dita a **extensão de f a Σ^*** da seguinte forma

$$\begin{aligned} \overline{f}(s, \lambda) &= \lambda \\ \overline{f}(s, a\omega) &= f(s, a)\overline{f}(\delta(s, a), \omega) \end{aligned}$$

para todo o $s \in S$, $a \in \Sigma$ e $\omega \in \Sigma^*$.

Exemplo 2.6.2. Considerando, de novo, o autómato do Exemplo 2.6.1, tem-se

$$\overline{f}(1, bbaabab) \;=\; ....ALARME..$$

Um resultado análogo ao teorema 2.2.1 verifica-se igualmente para a extensão da função resposta de um autómato reactivo.

Teorema 2.6.1. *Seja* $\mathcal{A} = (\Sigma, \Gamma, S, s_0, \delta, f)$. *Então, para todo o* $s \in S$ *e* $\omega, \upsilon \in \Sigma^*$,

$$\overline{f}(s, \omega\upsilon) \;=\; \overline{f}(s, \omega)\, \overline{f}(\overline{\delta}(s, \omega), \upsilon)$$

Prova: A demonstração, por indução sobre $|\upsilon|$, é deixada como exercício.

❏

A definição de autómato reactivo que formulamos é determinística. Isso significa que a cada palavra $\omega \in \Sigma^*$ reconhecida corresponde uma única resposta $\upsilon \in \Gamma^*$ produzida. De facto, a relação que associa palavras sobre Σ^* às correspondentes palavras sobre Γ^* é uma função que designamos por *função de tradução*. Note-se que esta função preserva o comprimento da palavra de entrada: se ω fôr constituida por n símbolos de Σ, a sua tradução terá exactamente n símbolos de Γ^*. Formalmente,

Definição 2.6.3. Dado um autómato reactivo $\mathcal{A} = (\Sigma, \Gamma, S, s_0, \delta, f)$ define-se **função de tradução** e representa-se por $f_{\mathcal{A}}$, a função $f_{\mathcal{A}} : \Sigma^* \longrightarrow \Gamma^*$ dada por

$$f_{\mathcal{A}}(\omega) \;=\; \overline{f}(s_0, \omega)$$

para todo o $\omega \in \Sigma^*$.

A definição de função de tradução proporciona um critério para discutir a equivalência entre autómatos reactivos sobre os mesmos alfabetos Σ e Γ.

Definição 2.6.4. Dois autómatos reactivos, $\mathcal{A}_1$ e $\mathcal{A}_2$, sobre os mesmos alfabetos Σ e Γ, são ditos **equivalentes**, escrevendo-se $\mathcal{A}_1 \equiv \mathcal{A}_2$, sse $f_{\mathcal{A}_1} = f_{\mathcal{A}_2}$.

Note-se, por outro lado, que nem todas as funções $h : \Sigma^* \longrightarrow \Gamma^*$ são modeláveis por autómatos reactivos do tipo que acabamos de introduzir. É o caso, por exemplo, da função $h : \{a, b, c\}^* \longrightarrow \{0, 1\}^*$ que substitui os símbolos de entrada por 0 a menos que o próximo símbolo a processar seja c, caso em que produz 1. Genericamente, dizemos que

Definição 2.6.5. Uma função $h : \Sigma^* \longrightarrow \Gamma^*$ é modelável por um autómato reactivo sse existir um desses autómatos $\mathcal{A}$ tal que $h = f_{\mathcal{A}}$[12].

[12] Compare esta definição com a de linguagem regular dada atrás.

Procuramos nesta secção dar uma ideia do que são autómatos reactivos, por vezes justamente designados por *autómatos finitos com acções semânticas*, e começar a construir a teoria associada. Diversas transformações definidas sobre os autómatos finitos (*eg*, minimizações e transformações homomórficas) são também, formuláveis neste caso, assim como se estudam numerosas elaborações da Definição 2.6.1. O leitor interessado é referido para obras mais especializadas, em particular, [CL89].

Capítulo 3

Linguagens e Gramáticas

Este Capítulo trata de uma "*mudança de agulhas*" e de uma "*fuga para a frente*". A primeira consiste na introdução de um novo modelo formal, de natureza generativa, da noção de linguagem — a *gramática*. Estudaremos com detalhe as gramáticas das linguagens regulares, obtendo assim uma sua caracterização complementar à discutida no capítulo anterior. Com este novo modelo investigaremos, de seguida, uma classe de linguagens, ditas de *contexto livre*, muito mais rica que a das linguagens regulares — será a "*fuga para a frente*". No próximo capítulo procederemos à revisão da noção de autómato de forma a acomodar o reconhecimento desse tipo de linguagens.

3.1 Estrutura e Gramática

Definimos linguagem como um qualquer subconjunto de sequências de símbolos sobre um dado alfabeto. Este conjunto e as sequências que o formam não são arbitrários mas exibem, geralmente, um determinado nível de estruturação com o qual, até aqui, não nos preocupamos. Por exemplo, uma frase comum em português é construida pela especificação de um sujeito, um predicado e um complemento. O sujeito inclui, tipicamente, um artigo seguido de um substantivo, eventualmente qualificado por um adjectivo. E assim por diante. Da mesma forma um programa numa linguagem de programação estruturada compreende um bloco de declarações e um bloco de comandos. Uma declaração consiste num identificador de tipo seguido de uma expressão de tipo ou num identificador de variável seguido de uma expressão de tipo e, opcionalmente, uma expressão de inicialização. As noções de sujeito, predicado, artigo, declaração, expressão de inicialização, etc., representam *classes sintáticas* que, não fazendo parte da linguagem que se está a definir, permitem impôr nela uma determinada estrutura. Por exemplo, podemos dizer que uma frase válida em portugês se inicia por uma sequência de símbolos do alfabeto[1] pertencente à classe sintática *sujeito*, e não a nenhuma outra, a que se seguem entidades pertencentes às classes sintáticas *predicado* e *complemento*.

Este tipo de relação entre as classes sintáticas que qualificam as diferentes partes de uma palavra da linguagem é usualmente estabelecida por *regras de produção*. Estas definem o modo

[1] Note-se que neste exemplo o alfabeto é o conjunto das palavras da língua portuguesa, pelo que uma palavra da linguagem coincide com a noção intuitiva de frase.

como as palavras válidas (ou sub-palavras relevantes) são geradas a partir da especificação da classe sintática a que pertencem. Por exemplo,

$$\begin{array}{lll} Frase & \longrightarrow & Suj\ Pred\ Comp \\ Frase & \longrightarrow & \ldots \\ Suj & \longrightarrow & Art\ Subs\ Qual \\ Art & \longrightarrow & o \end{array}$$

$$\begin{array}{lll} Art & \longrightarrow & um \\ Art & \longrightarrow & \ldots \\ Subs & \longrightarrow & copo \\ Subs & \longrightarrow & miguel \\ Subs & \longrightarrow & \ldots \\ Qual & \longrightarrow & Adj \\ Qual & \longrightarrow & \varepsilon \\ Adj & \longrightarrow & cheio \\ Adj & \longrightarrow & vazio \\ Pred & \longrightarrow & ficou \\ Pred & \longrightarrow & pertence \\ Pred & \longrightarrow & \ldots \\ Comp & \longrightarrow & Adj \\ Comp & \longrightarrow & \ldots \\ Comp & \longrightarrow & \varepsilon \\ \ldots & \ldots & \ldots \end{array}$$

Reparemos que algumas destas regras especificam como determinados símbolos do alfabeto constituem determinadas classes sintáticas. Por exemplo, *cadeira* pertence à classe $Subs$. Regras do tipo $T \longrightarrow \varepsilon$ indicam que a palavra vazia (λ) pode ser considerada um elemento da classe T, o que no nosso exemplo é usado para traduzir o caracter opcional da presença de um complemento numa frase e de um qualificador num sujeito. Com é óbvio, nenhum símbolo do alfabeto aparece no lado esquerdo de uma regra de produção, o que marca bem a distinção entre estes e os outros símbolos que usamos para designar as classes sintáticas. De facto, só os últimos são susceptíveis de expansão.

Reparemos ainda que, expandindo progressivamente estes símbolos vamos contruindo passso a passo uma palavra da linguagem, respeitando a estrutura que lhe impusemos. Por exemplo,

$$\begin{array}{llllll} Frase & \longrightarrow & Suj\ Pred\ Comp & \longrightarrow & Art\ Subs\ Qual\ Pred\ Comp & \longrightarrow \\ & \longrightarrow & o\ copo\ Pred\ Comp & \longrightarrow & \ldots & \longrightarrow \\ & \longrightarrow & \ldots & \longrightarrow & o\ copo\ ficou\ vazio & \end{array}$$

Este tipo de regras de produção ocupam um lugar fundamental na definição sintática de uma linguagem não trivial. Veremos mais tarde que, na construção de compiladores, se tornam essenciais, não apenas para determinar a correcção sintática de um programa[2], mas também para guiar a sua tradução.

[2] Um programa não é mais do que uma palavra sobre o alfabeto base da linguagem de programação em que é escrito — tipicamente $\Sigma = \{\texttt{begin}, \texttt{end}, \texttt{if}, \texttt{<=}, (,), \ldots\}$.

A especificação estrutural de linguagens através de um processo generativo do tipo que (informalmente) acabamos de descrever, baseia-se no conceito de *gramática* de que nos ocuparemos neste capítulo e que tem raízes na Teoria das Gramáticas Formais formulada por Chomsky em 1959.

Para definir uma gramática necessitamos de especificar um alfabeto a partir do qual são geradas as palavras de linguagem, um conjunto de símbolos que identificam as classes sintáticas relevantes e um conjunto de regras de produção conforme acima indicamos. Torna-se, ainda, necessário escolher a classe sintática a que pertencem as palavras válidas da linguagem (*eg*, a classe $Frase$ no exemplo anterior). Formalmente somos conduzidos à seguinte definição.

Definição 3.1.1. Uma **gramática** é um tuplo

$$\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$$

onde Ω é o alfabeto dos identificadores das classes sintáticas, Σ é o alfabeto sobre o qual a linguagem especificada por $\mathcal{G}$ se define, $\mathcal{P}$ é uma relação binária finita — dita de **derivação imediata** — sobre $\Omega^+ \times (\Omega \cup \Sigma)^*$ e $S \in \Omega$ é dito o **símbolo inicial** (ou axioma) de $\mathcal{G}$. Os elementos de Σ e Ω são referidos como os símbolos **terminais** e **não terminais** de $\mathcal{G}$, repectivamente. A sua reunião constitui o **vocabulário** da gramática e representa-se pelo símbolo $\mathcal{V}$. Os pares $(\alpha, \beta) \in \mathcal{P}$ representam as regras de produção de $\mathcal{G}$, que passaremos a denominar por **produções**, e escrevem-se geralmente na forma $\alpha \longrightarrow \beta$ (leia-se "*α deriva em β*").

A noção fundamental associada às produções de uma gramática é a de *derivação*, que se define recursivamente a partir da relação de derivação imediata $\mathcal{P}$.

Definição 3.1.2. Dada uma gramática $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$, define-se uma relação dita de **derivação** sobre $\Omega^+ \times (\Omega \cup \Sigma)^*$, e representa-se por $\longrightarrow^*$ da seguinte forma:

$$\begin{array}{lll} \alpha \longrightarrow^* \beta & \text{sse} & \alpha = \beta \\ & \text{ou} & \\ & & \exists_{\gamma \in \mathcal{V}^*}.\ \alpha \longrightarrow \gamma\ \wedge\ \gamma \longrightarrow^* \beta \end{array}$$

As sequências sobre $\mathcal{V}^*$ envolvendo símbolos não terminais, que ocorrem num processo de derivação, designam-se por *palavras parciais*. A linguagem especificada por uma gramática $\mathcal{G}$ é, como habitualmente, um subconjunto de Σ^*, formado pelas palavras que podem ser geradas a partir do símbolo inicial S por aplicação sucessiva das produções em $\mathcal{P}$. Formalmente,

Definição 3.1.3. Dada uma gramática $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$, a **linguagem gerada por** $\mathcal{G}$ é o subconjunto de Σ^* dado por

$$\mathcal{L}(\mathcal{G}) = \{\omega \in \Sigma^* \mid S \longrightarrow^* \omega\}$$

Antes de prosseguirmos convirá anotar uma convenção usual na escrita de gramáticas que consiste em agregar as produções com um mesmo lado esquerdo, separando os lados direitos por uma barra. Assim, por exemplo, em vez de escrevermos $A \longrightarrow aB$ e $A \longrightarrow baC$, escreveremos simplesmente $A \longrightarrow aB|baC$, para $A, B, C \in \Omega$ e $a, b \in \Sigma$.

É intuitivo verificar que com a noção de gramática podemos especificar uma classe de linguagens, que inclui, mas não se restringe, à das linguagens regulares discutida atrás. Nem todas essas classes são relevantes para o domínio que nos interessa — o das linguagens de programação — pelo que será necessário restringir a definição 3.1.1 para particularizar o nosso estudo. Esta restrição incide sobre o conjunto de definição da relação $\mathcal{P}$, ou seja, sobre a forma das produções admissíveis[3]. Neste capítulo consideraremos duas importantes classes de gramáticas: as *gramáticas regulares* e as *gramáticas de contexto livre*, também designadas por *gramáticas independentes do contexto*.

3.2 Gramáticas Regulares

3.2.1 Definições e Exemplos

A restrição mais séria que faremos à definição 3.1.1 conduz-nos à identificação da classe das gramáticas *regulares* que, como o nome indica, geram linguagens regulares. Estas gramáticas constituem, portanto, uma ferramenta de especificação de linguagens equivalente em poder expressivo aos *autómatos finitos* e às *expressões regulares* que discutimos no capítulo anterior.

Definição 3.2.1. Uma **gramática** $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ diz-se **regular** (à direita) sse $\mathcal{P}$ se define sobre $\Omega \times (\Sigma^* \cup \Sigma^*\Omega)$. Isto significa que as suas produções são da forma

$$A \longrightarrow vB$$

ou

$$A \longrightarrow v$$

para $A, B \in \Omega$ e $v \in \Sigma^*$.

Exemplo 3.2.1. Seja $\mathcal{G}$ a gramática regular definida sobre $\Sigma = \{a, b\}$ e $\Omega = \{S, T, V\}$, com as seguintes produções,

$$\begin{array}{lcl} S & \longrightarrow & aS|bS|aT \\ T & \longrightarrow & bV \\ V & \longrightarrow & \varepsilon \end{array}$$

Uma derivação típica será

[3] Note-se que a própria definição 3.1.1 é já uma restrição de uma definição mais geral que permitisse a presença de símbolos terminais no lado direito das produções, *i.é*, onde $\mathcal{P}$ se definisse sobre $\mathcal{V}^+ \times \mathcal{V}^*$.

$$\begin{array}{lllllllll} S & \longrightarrow & aS & \longrightarrow & a^2S & \longrightarrow & a^2bS & \longrightarrow & a^2b^2S & \longrightarrow \\ & \longrightarrow & a^2b^3S & \longrightarrow & a^2b^3aT & \longrightarrow & a^2b^3abV & \longrightarrow & a^2b^3ab \end{array}$$

Outra derivação possível será

$$\begin{array}{lllllllll} S & \longrightarrow & bS & \longrightarrow & baS & \longrightarrow & babS & \longrightarrow & bab^2S & \longrightarrow \\ & \longrightarrow & bab^2aS & \longrightarrow & bab^2baT & \longrightarrow & bab^2babV & \longrightarrow & bab^2bab \end{array}$$

É fácil concluir que a linguagem $\mathcal{L}$ expressa pela expressão regular $(a+b)^*ab$ coincide com $\mathcal{L}(\mathcal{G})$. De facto, para toda a palavra $\omega \in \Sigma^*$, existe uma derivação $S \longrightarrow^* \omega S$ que pode ser continuada por $\omega S \longrightarrow \omega aT \longrightarrow \omega abV \longrightarrow \omega ab$. Logo $\mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$. Conversamente, seja $\omega \in \mathcal{L}(\mathcal{G})$, *i.é*, $S \longrightarrow^* \omega$. Uma vez que a única produção que não envolve um símbolo não terminal é $V \longrightarrow \varepsilon$, verifica-se $S \longrightarrow^* \omega V \longrightarrow \omega$. A forma regular das produções em $\mathcal{G}$ força que toda a palavra parcial que ocorra numa derivação tenha, no máximo, um único símbolo não terminal e que este apareça sempre na sua terminação. Uma vez que $T \longrightarrow bV$ é a única produção em que V aparece no lado direito, teremos

$$S \longrightarrow^* \omega_1 bV \longrightarrow \omega_1 b$$

Por um argumento semelhante, chegamos a

$$S \longrightarrow^* \omega_2 aT \longrightarrow \omega_2 abV \longrightarrow \omega_2 ab$$

I.é, $\omega_1 = \omega_2 a$. Logo, qualquer palavra de $\mathcal{L}(\mathcal{G})$ deve terminar em ab, o que é suficiente para mostrar que $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}$.

Seguindo um raciocínio análogo, o leitor poderá verificar que uma gramática com as porduções

$$\begin{array}{lll} S & \longrightarrow & aS|bT \\ T & \longrightarrow & aa \end{array}$$

gera a linguagem regular $\{a^n baa|\ n \geq 0\}$ que, por indução sobre n, se mostra coincidir com a^*baa.

Exemplo 3.2.2. Seja, agora, a gramática $\mathcal{G}$ definida sobre $\Sigma = \{0, 1, .\}$ e $\Omega = \{S, T\}$, com as seguintes produções,

$$\begin{array}{lll} S & \longrightarrow & 0S|1S|0.T|1.T \\ T & \longrightarrow & \varepsilon|0T|1T \end{array}$$

Trata-se de uma gramática regular que define a linguagem dos números binários. Facilmente se verifica que 0.11 e 1101100.1010 pertencem a $\mathcal{L}(\mathcal{G})$.

Tal como acontecia com outras ferramentas de especificação, uma mesma linguagem pode ser gerada por diferentes gramáticas. Em particular, mostra-se que que toda a gramática regular à direita corresponde uma gramática cujas produções têm a forma

$$A \longrightarrow Bv$$

ou

$$A \longrightarrow v$$

para $A, B \in \Omega$ e $v \in \Sigma^*$, que, muito adequadamente, se designa por *regular à esquerda*.

Exemplo 3.2.3. A linguagem descrita pela gramática regular à direita do exemplo 3.2.1, é igualmente gerada por uma gramática regular à esquerda cujas produções são[4]

$$\begin{array}{lcl} S & \longrightarrow & Ta \\ T & \longrightarrow & Vb \\ V & \longrightarrow & \varepsilon|Va|Vb \end{array}$$

Em geral, diremos que

Definição 3.2.2. Duas gramáticas $\mathcal{G}_1$ e $\mathcal{G}_2$, regulares ou não, mas definidas sobre o mesmo alfabeto terminal Σ, são **equivalentes** sse $\mathcal{L}(\mathcal{G}_1) = \mathcal{L}(\mathcal{G}_2)$.

À definição de uma classe de gramáticas é usual associar determinadas forma canónicas que, dotadas do mesmo poder expressivo, colocam restrições adicionais no tipo de produções que podem ser usadas. O seu objectivo é facilitar o raciocínio sobre as linguagens geradas e a verificação de determinadas propriedades. No caso das gramáticas regulares define-se

Definição 3.2.3. Uma **gramática** $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ diz-se uma **forma canónica regular** (também designada por **simplesmente regular** ou **hiper-regular**) sse as produções em $\mathcal{P}$ são exclusivamente da forma

$$A \longrightarrow aB$$

ou

$$A \longrightarrow \varepsilon$$

para $A, B \in \Omega$ e $a \in \Sigma$.

[4] Ao contrário do que este exemplo possa sugerir, nem sempre existe um procedimento óbvio e sistemático para transformar uma gramática regular à direita numa regular à esquerda equivalente. O leitor é convidado a investigar os seus próprios exemplos.

Teorema 3.2.1. *Toda a gramática regular admite uma forma canónica de acordo com a definição 3.2.3.*

Prova: Seja $\mathcal{L} = \mathcal{L}(\mathcal{G})$ onde $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ é uma gramática regular. Admitamos, sem perda de generalidade, que $\mathcal{P} \subseteq (\Omega \times (\Sigma^* \cup \Sigma^+\Omega))$, *i.é*, que não existem em $\mathcal{G}$ produções da forma $T \longrightarrow R$, para $T, R \in \Omega$. Provaremos mais tarde, num contexto um pouco mais geral, que produções deste tipo são redundantes (Teorema 3.3.4). Vamos procurar definir uma forma canónica $\mathcal{G}'$ para $\mathcal{G}$ e mostrar que $\mathcal{L} = \mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$.

Definamos $\mathcal{G}' = (\Omega', \Sigma, \mathcal{P}', S)$ onde Ω' e $\mathcal{P}'$ são construidos, a partir de Ω e $\mathcal{P}$, da seguinte forma:

- Cada produção em $\mathcal{P}$ da forma $A \longrightarrow a_1a_2...a_nB$, com $a_1, a_2, ..., a_n \in \Sigma$, $B \in \Omega$ $n > 1$, é substituida em $\mathcal{P}'$ por n produções

$$\begin{array}{lcl} A & \longrightarrow & a_1A_1 \\ A_1 & \longrightarrow & a_2A_2 \\ \ldots & \ldots & \ldots \\ A_{n-1} & \longrightarrow & a_nB \end{array}$$

- Cada produção em $\mathcal{P}$ da forma $A \longrightarrow a_1a_2...a_n$, com $a_1, a_2, ..., a_n \in \Sigma$ e $n > 1$, é substituida em $\mathcal{P}'$ por n produções

$$\begin{array}{lcl} A & \longrightarrow & a_1A_1 \\ A_1 & \longrightarrow & a_2A_2 \\ \ldots & \ldots & \ldots \\ A_{n-1} & \longrightarrow & a_nA_n \\ A_n & \longrightarrow & \varepsilon \end{array}$$

Os símbolos A_i introduzidos constituem novos não terminais cuja reunião com Ω origina Ω'.

$\mathcal{G}'$ é uma forma canónica regular e resulta óbvio que toda a derivação $S \longrightarrow^* \omega$ em $\mathcal{G}$ pode ser simulada por uma derivação $S \longrightarrow^* \omega$ em $\mathcal{G}'$, usualmente mais extensa. Logo $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}')$. Para mostrar a inclusão inversa suponhamos que $\omega \in \mathcal{L}(\mathcal{G}')$ e consideremos uma nova gramática $\mathcal{G}'' = (\Omega' \cup \Omega, \Sigma, \mathcal{P}' \cup \mathcal{P}, S)$. Certamente existe uma derivação de ω em $\mathcal{G}''$. Mostremos, agora, que existe uma derivação de ω em $\mathcal{G}$ por indução sobre o número de símbolos não terminais pertencentes a Ω' que surgem na referida derivação em $\mathcal{G}$. Suponhamos que A_1 é a primeira ocorrência desses símbolos. A sua introdução na derivação em causa é baseada numa produção da forma $A \longrightarrow a_1A_1$, com $A \in \Omega$, introduzida em $\mathcal{P}'$ no processo de substituição de uma produção de $\mathcal{G}$ da forma $A \longrightarrow a_1a_2...a_nB$ ou $A \longrightarrow a_1a_2...a_n$. Como $\mathcal{P}''$ não tem produções da forma $X \longrightarrow v$, para $X \in \Omega$ e $v \in \Sigma^*$, a eliminação de A_1 na derivação em estudo deve proceder pela sua substituição por a_2A_2, $a_2a_3A_3$ e assim sucessivamente. Isto significa que a cadeia de derivação

$$A \longrightarrow a_1A_1 \longrightarrow a_1a_2A_2 \longrightarrow ... \longrightarrow a_1a_2...a_{n-1}A_{n-1} \longrightarrow a_1a_2...a_{n-1}a_nB$$

ou

$$A \longrightarrow a_1A_1 \longrightarrow a_1a_2A_2 \longrightarrow ... \longrightarrow a_1a_2...a_{n-1}A_{n-1} \longrightarrow a_1a_2...a_{n-1}a_n$$

pode ser substituida por uma única transição

$$A \longrightarrow a_1a_2...a_nB$$

ou

$$A \longrightarrow a_1 a_2 ... a_n$$

que não contém elementos de Ω'. O número destes símbolos na derivação de ω é, assim, reduzido em uma unidade. Por um argumento indutivo segue-se que $\mathcal{L}(\mathcal{G}'') \subseteq \mathcal{L}(\mathcal{G})$ o que implica que $\mathcal{L}(\mathcal{G}') \subseteq \mathcal{L}(\mathcal{G})$.

❏

3.2.2 Gramáticas Regulares e Autómatos Finitos

O leitor atento terá já nesta altura a suspeita de que qualquer linguagem gerada por uma gramática regular pode ser reconhecida por um autómato finito. Esta secção é dedicada à análise dessa equivalência. De facto, os conceitos atrás introduzidos permitem-nos provar o seguinte resultado fundamental.

Teorema 3.2.2. *O conjunto das linguagens regulares e o das geradas por gramáticas regulares coincidem.*

Prova: Seja $\mathcal{L}$ uma linguagem regular e $\mathcal{A} = (\Sigma, S, s_0, \delta, F)$ o autómato finito determinístico que a reconhece. Comecemos por definir uma gramática regular $\mathcal{G}$ da seguinte forma:

$$\mathcal{G} = (S \cup \Sigma, \Sigma, \mathcal{P}, s_0)$$

onde $\mathcal{P}$ é composto pelas produções

$$\begin{array}{rcll} s & \longrightarrow & a\ \delta(s,a) & \text{para todo } s \in S, a \in \Sigma \\ s & \longrightarrow & \varepsilon & \text{para todo } s \in F \end{array}$$

Mostremos, agora, que $\mathcal{L}(\mathcal{G}) = \mathcal{L}$. Seja $\omega = a_1, a_2, ..., a_n \in \mathcal{L}(\mathcal{G})$, o que significa que existe uma derivação em $\mathcal{G}$ da forma $s_0 \longrightarrow^* a_1, a_2, ..., a_n$. Esta, de acordo com a definição das produções em $\mathcal{G}$ será da forma

$$s_0 \longrightarrow a_1 s_1 \longrightarrow a_1 a_2 s_2 \longrightarrow a_1 a_2 ... a_n s_n \longrightarrow a_1 a_2 ... a_n$$

onde $s_0 \longrightarrow a_1 s_1$, $s_1 \longrightarrow a_2 s_2$, ..., $s_n \longrightarrow \varepsilon$ são produções em $\mathcal{P}$. Nesse caso podemos concluir que em $\mathcal{A}$ se tem $s_0 a_1 = s_1$, $s_1 a_2 = s_2$, ..., $s_{n-1} a_n = s_n$ e $s_n \in F$. Logo $s_0 \omega \in F$ e $\omega \in \mathcal{L}(\mathcal{A}) = \mathcal{L}$.

Conversamente, seja $\omega = a_1, a_2, ..., a_n \in \mathcal{L}(\mathcal{A})$. Então $s_0 \omega \in F$ e existem em $\mathcal{A}$ transições elementares que conduzem o autómato de s_0 a um estado final mediante o reconhecimento de $a_1, a_2, ..., a_n$. I.é, temos $s_0 a_1 = s_1$, $s_1 a_2 = s_2$, ..., $s_{n-1} a_n = s_n$ e $s_n \in F$. Mas, sendo assim, também existem em $\mathcal{G}$ as produções $s_0 \longrightarrow a_1 s_1$, $s_1 \longrightarrow a_2 s_2$, ..., $s_n \longrightarrow \varepsilon$ que possibilitam a derivação $s_0 \longrightarrow^* \omega$. Logo $\omega \in \mathcal{L}(\mathcal{G})$.

Concluimos, pois, que toda a linguagem regular é gerada por uma gramática regular. Mostremos, agora, a implicação inversa (*i.é*, que toda a linguagem gerada por uma gramática regular é regular).

Seja $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ uma gramática regular e $\mathcal{L} = \mathcal{L}(\mathcal{G})$. Pelo teorema 3.2.1, qualquer gramática regular pode ser convertida a uma forma canónica regular. Admitamos, portanto, e sem perda de generalidade, que $\mathcal{G}$ é já uma forma canónica. Definamos o autómato finito não determinístico

$$\mathcal{A} = (\Omega, \Sigma, \{S\}, \delta, F)$$

onde $Y \in \delta(X, a)$ sse $(X \longrightarrow aY) \in \mathcal{P}$ e $F = \{X \in \Omega | \ X \longrightarrow \varepsilon \in \mathcal{P}\}$. Seja $\omega = a_1, a_2, ..., a_n \in \mathcal{L}(\mathcal{G})$. A sua derivação será da forma

$$S \longrightarrow a_1 A_1 \longrightarrow a_1 a_2 A_2 \longrightarrow ... \longrightarrow a_1 a_2 ... a_n A_n \longrightarrow a_1 a_2 ... a_n$$

com $S \longrightarrow a_1 A_1$, $A_1 \longrightarrow a_2 A_2$, ..., $A_{n-1} \longrightarrow a_n A_n$ e $A_n \longrightarrow \varepsilon$ em $\mathcal{P}$. Então, $A_n \in F$ e são válidos em $\mathcal{A}$ os seguintes factos: $A_1 \in Sa_1$, $A_2 \in A_1 a_2$, ..., $A_n \in A_{n-1} a_n$. Logo $\omega \in \mathcal{L}(\mathcal{A})$. I.é, $\mathcal{L} \subseteq \mathcal{L}(\mathcal{A})$.

Conversamente, suponhamos que $\omega = a_1, a_2, ..., a_n \in \mathcal{L}(\mathcal{A})$. Então, em $\mathcal{A}$ temos $A_1 \in Sa_1$, $A_2 \in A_1 a_2$, ..., $A_n \in A_{n-1} a_n$ e $A_n \in F$. Por construção de $\mathcal{A}$, tais transições correspondem às produções $S \longrightarrow a_1 A_1$, $A_1 \longrightarrow a_2 A_2$, ..., $A_{n-1} \longrightarrow a_n A_n$ e $A_n \longrightarrow \varepsilon$ em $\mathcal{P}$, que tornam possível a derivação $S \longrightarrow^* \omega$ em $\mathcal{G}$. Logo $\omega \in \mathcal{L}(\mathcal{G})$ e, portanto, $\mathcal{L}(\mathcal{A}) = \mathcal{L} \subseteq \mathcal{L}(\mathcal{G})$.

❏

Com base neste teorema podemos afirmar a equivalência, em termos de poder expressivo, das três entidades formais que introduzimos para especificar linguagens regulares, a saber, *autómatos de estado finito*, *expressões regulares* e, finalmente, *gramáticas regulares*. A mútua conversão entre as duas primeiras foi abordada no final do capítulo anterior. Na prova do teorema 3.2.2 indicamos como construir um autómato que reconhecesse uma linguagem gerada por uma gramáticar regular e vice-versa. Aplicando o procedimento estudado na secção 2.5.2 podemos daí chegar à expressão regular correspondente.

Uma forma muito mais directa de obter o mesmo resultado consiste em construir uma sistema de equações, a resolver em ordem à incógnita que representa a linguagem, a partir das produções da gramática. Consideremos uma gramática $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$. Para cada não terminal $S_i \in \Omega$ define-se o conjunto $X_{S_i} \in 2^{\Sigma^*}$ compreendendo todas as palavras sobre Σ que podem ser derivadas de S_i usando $\mathcal{P}$. Obviamente $X_S = \mathcal{L}(\mathcal{G})$. Estes conjuntos satisfazem equações do tipo

$$X_{S_i} \ = \ E_i + A_{i1} X_{S_1} + A_{i2} X_{S_2} + ... + A_{in} X_{S_n}$$

onde E_i é a união de todas as palavras $v \in \Sigma^*$ que surgem em produções da forma $S_i \longrightarrow v$, e cada A_{ij} é a união de todas as palavras $v \in \Sigma^*$ que surgem em produções da forma $S_i \longrightarrow vS_j$. I.é,

$$\begin{array}{rcl} E_i & = & \{v \in \Sigma^* | \ S_i \longrightarrow v \ \in \ \mathcal{P}\} \\ A_{ij} & = & \{v \in \Sigma^* | \ S_i \longrightarrow vS_j \ \in \ \mathcal{P}\} \end{array}$$

Este sistema de equações modela as relações que existem entre os não terminais de $\mathcal{G}$. A sua resolução não deve oferecer ao leitor qualquer dificuldade.

Exemplo 3.2.4. Considere a gramática $\mathcal{G}$ com as seguintes produções

$$\begin{array}{lcl} S & \longrightarrow & aS|bT \\ T & \longrightarrow & aa \end{array}$$

Define-se, então, o sistema

$$\begin{array}{lcl} X_S & = & \emptyset + aX_S + bX_T \\ X_T & = & aa + \emptyset X_S + \emptyset X_T \end{array}$$

cuja resolução prossegue como se ilustra

$$\begin{array}{lcl} X_S & = & \emptyset + aX_S + bX_T \\ X_T & = & aa \\ \\ \equiv \\ \\ X_S & = & aX_S + baa \\ X_T & = & aa \\ \\ \equiv \\ \\ X_S & = & a^*baa \\ X_T & = & aa \end{array}$$

Assim, concluimos que $\mathcal{L}(\mathcal{G}) = a^*baa$.

Podemos, igualmente abordar o problema inverso: dada uma expressão regular, construir uma gramática regular correspondente. Basicamente, o problema reduz-se à determinação das gramáticas correspondentes aos contrutores de expressões regulares introduzidos na secção 2.5.2[5]. É o que fazemos na tabela seguinte onde se supõe que $\mathcal{G}$, $\mathcal{G}_1$ e $\mathcal{G}_2$ são gramáticas regulares que geram as linguagens representadas pelas expressões regulares E, E_1 e E_2.

[5] O argumento tem uma estrutura análoga ao usado na determinação do autómato equivalente aí discutido.

Exp. Regular	Gramática Regular
$\emptyset$	$(\{S\}, \Sigma, \emptyset, S)$
ε	$(\{S\}, \Sigma, \{S \longrightarrow \varepsilon\}, S)$
$a \in \Sigma$	$(\{S\}, \Sigma, \{S \longrightarrow a\}, S)$
$E_1 + E_2$	$(\Omega_1 \cup \Omega_2 \cup \{Z\}, \Sigma, \mathcal{P}_1 \cup \mathcal{P}_2 \cup \{Z \longrightarrow S_1, Z \longrightarrow S_2\}, Z)$
$E_1 E_2$	$(\Omega_1 \cup \Omega_2, \Sigma, \mathcal{P}_1' \cup \mathcal{P}_2, S_1)$ com $\mathcal{P}_1' = \{X \longrightarrow vY \mid X \longrightarrow vY \in \mathcal{P}_1, v \in \Sigma^*, X, Y \in \Omega\} \cup$ $\{X \longrightarrow vS_2 \mid X \longrightarrow v \in \mathcal{P}_1 \ \wedge \ v \in \Sigma^*\}$
E^*	$(\Omega \cup \{Z\}, \Sigma, \mathcal{P}', Z)$ com $\mathcal{P}' = \{Z \longrightarrow S, Z \longrightarrow \varepsilon\} \cup$ $\{X \longrightarrow vY \mid X \longrightarrow vY \in \mathcal{P}, v \in \Sigma^*, X, Y \in \Omega\} \cup$ $\{X \longrightarrow vZ \mid X \longrightarrow v \in \mathcal{P}, v \in \Sigma^*, X \in \Omega\}$

O leitor é convidado a mostrar a correcção das construções propostas verificando que, efectivamente, as linguagens geradas por gramáticas regulares são fechadas para a reunião, concatenação e fecho de Kleene.

3.3 Gramáticas de Contexto Livre

3.3.1 Definições e Exemplos

Consideraremos, nesta secção, uma classe de linguagens suficientemente ampla para acomodar a definição de linguagens de programação. Iremos aborda-la a partir da noção de gramática, embora sejam igualmente especificáveis, numa perspectiva cognitiva, através de uma generalização da definição de autómato[6].

Definição 3.3.1. Uma **gramática** $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ diz-se **de contexto livre** ou **independente do contexto** sse as suas produções forem da forma

$$A \longrightarrow \alpha$$

para $A \in \Omega$ e $\alpha \in \mathcal{V}^*$.

Independência do contexto significa que, no decurso de uma derivação, todo o não terminal pode ser substituido sem consideração do contexto em que ocorre. De facto, se se admitissem produções da forma $\zeta A \eta \longrightarrow \alpha$, com $\alpha, \zeta, \eta \in \mathcal{V}^*$, a substituição de A só poderia dar-se quando a sua ocorrência fosse enquadrada pelas sequências ζ e η e a gramática seria, por isso, sensível ao contexto.

Exemplo 3.3.1. Considere a definição por uma gramática de contexto livre $\mathcal{G}$ da linguagem cujas palavras são os números reais representados numa linguagem do tipo do PASCAL.

[6] Trata-se da noção de *autómato recorrente* (também dito *de estado variável*, ou, na terminologia anglo-saxónica, *pushdown automata*) que será introduzida no próximo capítulo.

Nesta representação um real contém uma parte inteira e uma parte decimal (opcionais mas não mutuamente ausentes) e, ainda, um expoente opcional.

$$
\begin{array}{lcl}
S & \longrightarrow & Sinal\ Mantissa\ Exp \\
Sinal & \longrightarrow & \varepsilon \mid + \mid - \\
Mantissa & \longrightarrow & Digitos\ Ponto\ Digitos \\
Digitos & \longrightarrow & \varepsilon \mid Digito\ Digitos \\
Ponto & \longrightarrow & Digito\ . \mid .\ Digito \\
Digito & \longrightarrow & \mathtt{0} \mid \mathtt{1} \mid \ldots \mid \mathtt{9} \\
Exp & \longrightarrow & \varepsilon \mid \mathtt{E}\ Sinal\ Digito\ Digitos
\end{array}
$$

O leitor poderá verificar que $\mathcal{L}(\mathcal{G})$ é, de facto, uma linguagem regular. A gramática regular correspondente tem um número de produções significativamente maior e é de leitura menos intuitiva.

Este exemplo ilustra uma potencial fonte de confusão: para mostrar que uma linguagem não é regular não basta exibir uma gramática de contexto livre que a gere. É de facto necessário provar que *não* pode ser gerada por nenhuma gramática regular. Vejamos, agora, alguns exemplos de linguagens de contexto livre que não são regulares.

Exemplo 3.3.2. Verificamos na proposição 2.2.2 a não regularidade da linguagem $\mathcal{L} = \{a^n b^n \mid n \geq 1\}$. Contudo $\mathcal{L}$ é gerada pela seguinte gramática de contexto livre.

$$
\begin{array}{lcl}
S & \longrightarrow & aTb \\
T & \longrightarrow & \varepsilon | aTb
\end{array}
$$

Exemplo 3.3.3. Outra linguagem que, por aplicação do teorema 2.2.2, se mostra não regular é $\mathcal{L} = \{\omega \in \Sigma^* \mid \omega = \overline{\omega}\}$. No entanto, $\mathcal{L} = \mathcal{L}(\mathcal{G})$ se $\mathcal{G}$ fôr a gramática de contexto livre com as produções seguintes.

$$
S \longrightarrow \varepsilon|a|b|aSa|bSb
$$

Estudemos, agora, com um pouco mais de detalhe, a relação de derivação induzida por uma gramática de contexto livre. A forma das produções neste tipo de gramática possibilita a representação de cada derivação por uma estrutura arbórea. Estas estruturas designam-se por *árvores de derivação* (*parse trees*, na terminologia anglo-saxónica) e jogam um papel muito importante na análise das linguagens de contexto livre e, em particular, na construção de algoritmos de reconhecimento sintático. Formalmente,

Definição 3.3.2. Dada uma gramática de contexto livre $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ e uma palavra $\omega \in \mathcal{L}(\mathcal{G})$ define-se **árvore de derivação de ω em $\mathcal{G}$** como uma árvore n-ária, ordenada, cuja raíz é etiquetada por S e as n sub-árvores de um nodo etiquetado por um não terminal A conduzem a nodos etiquetados por v_1 a $v_n \in \mathcal{V}$ sse $A \longrightarrow v_1 ... v_n \in \mathcal{P}$. Se $A \longrightarrow \varepsilon \in \mathcal{P}$, A terá um nodo descendente na árvore etiquetado por ε. Nodos etiquetados por símbolos terminais ou por ε não têm descendentes.

Cada árvore de derivação exibe uma sequência particular de substituições de símbolos não terminais permitidas pelas produções da gramática. Concatenando as suas folhas da esquerda para a direita obtem-se uma palvra de $\mathcal{L}(\mathcal{G})$. Os exemplos seguintes ajudarão a consolidar esta definição.

Exemplo 3.3.4. Seja $\mathcal{G}$ a gramática de contexto livre discutida no Exemplo 3.3.2. A árvore de derivação correspondente a $aabb \in \mathcal{L}(\mathcal{G})$ é representada na Figura 3.3.4.

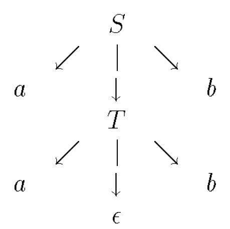

Figura 3.1: Árvore de derivação para $aabb$

Exemplo 3.3.5. Considere a seguinte gramática para uma linguagem de expressões booleanas que, como facilmente poderá concluir, não é regular.

$$
\begin{array}{lcl}
\Omega & = & \{ExB, Var\} \\
\Sigma & = & \{0, 1, \wedge, \vee, \neg, (,), p, q, r\} \\
S & = & ExB \\
\mathcal{P} & = &
\end{array}
$$

$$
\begin{array}{lcl}
ExB & \longrightarrow & ExB \wedge ExB \mid ExB \vee ExB \mid \neg\, ExB \mid (\ ExB\ ) \mid 0 \mid 1 \mid Var \\
Var & \longrightarrow & p \mid q \mid r
\end{array}
$$

A árvore de derivação correspondente a $p \wedge (q \vee r)$ representa-se na Figura 3.3.5.

Dada uma determinada árvore de derivação para uma palavra $\omega \in \mathcal{L}(\mathcal{G})$ existe alguma liberdade na escolha da ordem pela qual as produções associadas são aplicadas.

Definição 3.3.3. Uma derivação é dita pela **esquerda** (respectivamente, pela **direita**) se, em cada passo, o não terminal situado mais à esquerda (respectivamente, mais à direita) na palavra parcial em causa fôr expandido para produzir o passo seguinte.

Exemplo 3.3.6. A derivação pela esquerda da palavra $p \wedge (q \vee r)$ gerada pela gramática introduzida no Exemplo 3.3.5 é

$$
\begin{array}{lclclcl}
ExB & \longrightarrow & ExB \wedge ExB & \longrightarrow & Var \wedge ExB & \longrightarrow \\
p \wedge ExB & \longrightarrow & p \wedge (\ ExB\ ) & \longrightarrow & p \wedge (\ ExB \vee ExB\ ) & \longrightarrow \\
p \wedge (\ Var \vee ExB\ ) & \longrightarrow & p \wedge (\ q \vee ExB\ ) & \longrightarrow & p \wedge (\ q \vee Var\ ) & \longrightarrow \\
p \wedge (\ q \vee r\ ) & & & & &
\end{array}
$$

Abrimos aqui um parentesis para salientar que uma das tarefas típicas de um compilador de uma linguagem descrita por uma gramática de contexto livre $\mathcal{G}$ consiste em processar uma

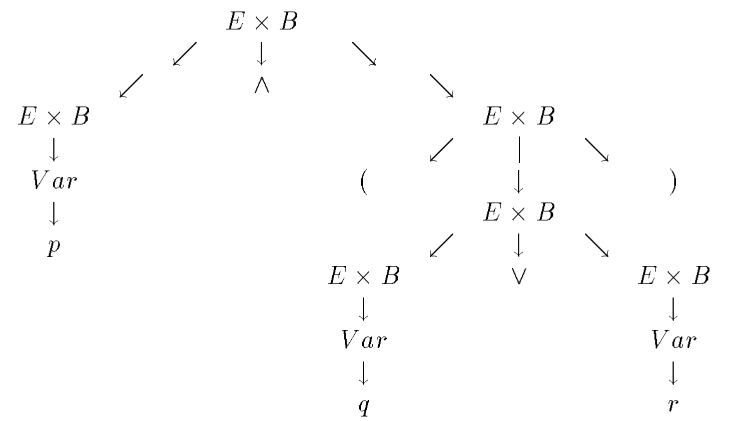

Figura 3.2: Árvore de derivação para $p \wedge (q \vee r)$

palavra ω e construir, se possível (*i.é*, se efectivamente $\omega \in \mathcal{L}(\mathcal{G})$) a árvore de derivação associada. Numa segunda fase, após a determinação das árvores de derivação, o compilador usa-as para gerar as parcelas de código máquina (ou, mais genericamente, para desencadear as acções semânticas) correspondentes a cada palavra reconhecida. Acontece que, se as produções da gramática forem construidas respeitando determinadas condições, o algoritmo requerido é muito simplificado — mas essas são já questões de cariz tecnológico.

As árvores de derivação constituem um meio privilegiado para analisar um problema típico em teoria das linguagens de contexto livre: o da *ambiguidade*. Genericamente, dizemos que

Definição 3.3.4. Uma gramática de contexto livre $\mathcal{G}$ é **ambígua** sse existe alguma palavra $\omega \in \mathcal{L}(\mathcal{G})$ à qual correspondem, pelo menos, duas árvores de derivação distintas[7].

Numa gramática que especifique uma linguagem de programação a presença de ambiguidades pode causar problemas sérios. Por exemplo, suponha que existem duas árvores de derivação para a expressão em PASCAL `i-j-k`, uma associando à direita e outra à esquerda.

A remoção de ambiguidades é muitas vezes possível, geralmente à custa da expansão de Ω e $\mathcal{P}$, para forçar a introdução de algumas regras de precedência e associatividade entre os operadores. Note-se, contudo, que existem linguagens de contexto livre que são *inerentemente ambíguas*, *i.é*, para as quais não existem gramáticas de contexto livre não ambíguas susceptíveis de as gerarem. Por exemplo, $\mathcal{L} = \{a^n b^n c^m d^m \mid n, m \in \mathbb{N}\} \cup \{a^o b^p c^p d^o \mid o, p \in \mathbb{N}\}$. Trata-se de um resultado que não provaremos aqui mas para o qual o leitor deve ser alertado. Por outro lado, é possível mostrar que nenhuma linguagem regular é inerentemente ambígua.

Terminaremos esta análise das derivações em gramáticas de contexto livre formulando dois

[7] O leitor não deve confundir essa situação com a possibilidade, muito comum, de em $\mathcal{G}$ existirem diferentes derivações para uma mesma palavra.

teoremas que serão muito úteis na verificação de algumas propriedades deste tipo de linguagens na secção seguinte.

Teorema 3.3.1. *Seja $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ uma gramática de contexto livre e $\alpha_1\alpha_2 \longrightarrow^* \beta$ uma derivação em $\mathcal{G}$, com $\alpha_1, \alpha_2, \beta \in \mathcal{V}^*$. Então, β é exprimível na forma $\beta_1\beta_2$ de modo que existem em $\mathcal{G}$ as derivações $\alpha_1 \longrightarrow^* \beta_1$ e $\alpha_2 \longrightarrow^* \beta_2$, cada uma das quais tem um comprimento igual ou inferior ao da derivação original.*

Prova: Procedamos por indução sobre o comprimento da derivação $\alpha_1\alpha_2 \longrightarrow^* \beta$. Se esta é feita num único passo será necessariamente baseada numa produção $A \longrightarrow \eta$, para $A \in \Omega$ e $\eta \in \mathcal{V}^*$, e A deve pertencer a α_1 ou a α_2. Suponhamos, sem perda de generalidade, que $\alpha_1 = \xi_1 A \xi_2$, com $\xi_1, \xi_2 \in \mathcal{V}^*$. Então, a derivação em causa será $\alpha_1\alpha_2 = \xi_1 A \xi_2 \alpha_2 \longrightarrow \xi_1 \eta \xi_2 \alpha_2 = \beta$. Logo, $\beta = \beta_1\beta_2$ com $\beta_1 = \xi_1\eta\xi_2$ e $\beta_2 = \alpha_2$ e, claramente, existem em $\mathcal{G}$ as derivações $\alpha_1 \longrightarrow^* \beta_1$ e $\alpha_2 \longrightarrow^* \beta_2$ de comprimento 1 e 0, respectivamente.

No caso geral o comprimento de $\alpha_1\alpha_2 \longrightarrow^* \beta$ é n e podemos supôr, novamente sem perda de generalidade, que o seu primeiro passo é $\alpha_1\alpha_2 \longrightarrow \alpha_1'\alpha_2$, onde $\alpha_1 = \xi_1 A \xi_2$, $\alpha_1' = \xi_1\eta\xi_2$ e $A \longrightarrow \eta \in \mathcal{P}$. Por hipótese de indução podemos assumir que $\beta = \beta_1\beta_2$ e que existem em $\mathcal{G}$ as derivações $\alpha_1' \longrightarrow^* \beta_1$ e $\alpha_2 \longrightarrow^* \beta_2$ cada uma das quais de comprimento igual ou inferior a $n-1$. Então, a derivação $\alpha_1 \longrightarrow \alpha_1' \longrightarrow^* \beta_1$, com comprimento igual ou inferior a n, está definida em $\mathcal{G}$, como queríamos mostrar.

❏

Teorema 3.3.2. *Seja $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ uma gramática de contexto livre e $\alpha_1 A \alpha_2 \longrightarrow^* \alpha_1' A \alpha_2'$, com $A \in \Omega$ e $\alpha_1, \alpha_2, \alpha_1', \alpha_2' \in \mathcal{V}^*$, uma derivação em $\mathcal{G}$ que apenas recorre a produções nas quais A não ocorre no lado direito. Então existem em $\mathcal{G}$ as derivações $\alpha_1 \longrightarrow^* \alpha_1'$ e $\alpha_2 \longrightarrow^* \alpha_2'$.*

Prova: O resultado é um corolário praticamente directo do teorema 3.3.1.

❏

3.3.2 Normalização e Propriedades

Tal como acontecia com as gramáticas regulares, também as de contexto livre podem ser escritas em determinadas formas canónicas, que facilitam a sua manipulação e a verificação de certas propriedades. Iniciaremos esta discussão mostrando que é sempre possível eliminar as produções triviais de uma gramática e, em quase todos os casos, as produções da forma $A \longrightarrow \varepsilon$, que designamos por produções-ε.

Teorema 3.3.3. *Qualquer linguagem $\mathcal{L}$ de contexto livre que não contenha a palavra vazia ($\lambda \notin \mathcal{L}$) pode ser gerada por uma gramática de contexto livre sem produções-ε.*

Prova: Seja $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ uma gramática de contexto livre tal que $\mathcal{L} = \mathcal{L}(\mathcal{G})$. Comecemos por definir uma nova gramática $\mathcal{G}'$ cuja relação de derivação imediata $\mathcal{P}'$ é obtida de $\mathcal{G}$ por *remoção consistente* das suas produções-ε. Clarifiquemos o que se entende por remoção consistente. Seja $A \longrightarrow \alpha$ uma produção de $\mathcal{G}$ onde α contém $k \geq 1$ ocorrências de símbolos para os quais existe uma derivação em ε. Isto é, α é da forma

$$\alpha \ = \ \beta_1\alpha_1\beta_2\alpha_2...\beta_k\alpha_k\beta_{k+1}$$

onde, para todo o i, $\alpha_i \in \Omega$, $\beta_i \in \mathcal{V}^*$ e $\alpha_i \longrightarrow^* \varepsilon$. Seja Y um qualquer subconjunto de $\{1, 2, 3, ..., k\}$ e definamos α_Y como a sequência obtida de α por remoção de cada α_i, para $i \in Y$ [8]. Cada produção em $\mathcal{P}$ nestas condições vai ser substituida em $\mathcal{P}'$ por um máximo de 2^k produções da forma $A \longrightarrow \alpha_Y$, para todo o $Y \subseteq \{1, 2, 3, ..., k\}$. Obviamente não incluiremos em $\mathcal{P}'$ nenhuma das produções-ε de $\mathcal{P}$ nem as que resultam deste processo de remoção quando $\alpha_Y = \varepsilon$ (o que apenas acontece quando α consiste num não terminal B tal que $B \longrightarrow^* \varepsilon$). Note-se que a produção original — $A \longrightarrow \alpha$ — permanece em $\mathcal{P}'$ (porquê?).

Qualquer derivação em $\mathcal{G}'$ tem uma correspondente derivação em $\mathcal{G}$. De facto, sempre que nela fôr usada uma produção $A \longrightarrow \alpha_Y$, o seu uso pode ser substituido pelo recurso a $A \longrightarrow \alpha$ e $\alpha_i \longrightarrow^* \varepsilon$, para todo o $i \in Y$. Logo, $\mathcal{L}(\mathcal{G}') \subseteq \mathcal{L}(\mathcal{G})$.

Conversamente, suponhamos que $\omega \in \mathcal{L}(\mathcal{G})$ e consideremos a derivação $S \longrightarrow^* \omega$. Se esta não recorrer a produções-ε é ainda uma derivação possível em $\mathcal{G}'$. Suponhamos, agora, e no caso geral, que a derivação referida envolve n produções-ε. Vamos admitir, como hipótese de indução, que toda a palavra $\upsilon \in \mathcal{L}(\mathcal{G})$ cuja derivação recorre a um número de produções-ε inferior a n, pertence também a $\mathcal{L}(\mathcal{G}')$.

Consideremos o último passo na derivação $S \longrightarrow^* \omega$ que envolve uma produção-ε e seja ela $V \longrightarrow \varepsilon$. Podemos retroceder na derivação até ao ponto em que, pela última vez, se usou uma produção em que V surge no lado direito. Seja $T \longrightarrow \eta_1 V \eta_2$ a produção em causa. Então a derivação de ω em $\mathcal{G}$ tem a forma

$$S \ \longrightarrow^* \ \tau_1 T \tau_2 \ \longrightarrow \ \tau_1\eta_1 V \eta_2\tau_2 \ \longrightarrow^* \ \tau_1'\eta_1' V \eta_2'\tau_2' \ \longrightarrow \ \tau_1'\eta_1'\eta_2'\tau_2' \ \longrightarrow^* \ \omega_1\upsilon_1\upsilon_2\omega_2 \ = \ \omega$$

com $\tau_1, \tau_1', \tau_2, \tau_2', \eta_1, \eta_1', \eta_2, \eta_2' \in \Omega$ e $\omega_1, \omega_2, \upsilon_1, \upsilon_2 \in \Sigma^*$.

Pelo teorema 3.3.2 existem em $\mathcal{G}$ as derivações

$$\begin{array}{lllll} \tau_1 & \longrightarrow^* & \tau_1' & \longrightarrow^* & \omega_1 \\ \tau_2 & \longrightarrow^* & \tau_2' & \longrightarrow^* & \omega_2 \\ \eta_1 & \longrightarrow^* & \eta_1' & \longrightarrow^* & \upsilon_1 \\ \eta_2 & \longrightarrow^* & \eta_2' & \longrightarrow^* & \upsilon_2 \end{array}$$

Todas estas derivações, tal como $S \longrightarrow^* \tau_1 T \tau_2$, envolvem menos que n aplicações de produções-ε podendo, por hipótese de indução, ser substituidas por derivações em $\mathcal{G}'$. Como em $\mathcal{G}'$ existe a produção $T \longrightarrow \eta_1\eta_2$ (porquê?), à derivação $S \longrightarrow^* \omega$ em $\mathcal{G}$ irá corresponder, em $\mathcal{G}'$, a derivação

$$S \ \longrightarrow^* \ \tau_1 T \tau_2 \ \longrightarrow \ \tau_1\eta_1\eta_2\tau_2 \ \longrightarrow^* \ \tau_1'\eta_1'\eta_2'\tau_2' \ \longrightarrow^* \ \omega_1\upsilon_1\upsilon_2\omega_2 \ = \ \omega$$

Note-se, porém, que o argumento apresentado falha se $\eta_1\eta_2 = \lambda$. Neste caso particular tem-se $T \longrightarrow V \ \in \mathcal{P}$ e nenhuma produção nova foi incluida em $\mathcal{P}'$ no processo de remoção

[8] Note que $\alpha_\emptyset = \alpha$ e que $\alpha_{\{1,2,3,...,k\}} = \beta_1\beta_2...\beta_k\beta_{k+1}$.

consistente indicado acima. Podemos, contudo, retroceder na derivação $S \longrightarrow^* \omega$ até ao último passo baseado numa produção em que T surge no lado direito. Seja ela $R \longrightarrow \eta_1'' T \eta_2''$. De novo a derivação pode ser decomposta em

$$S \longrightarrow^* \tau_1' R \tau_2' \longrightarrow \tau_1' \eta_1'' T \eta_2'' \tau_2' \longrightarrow^* \tau_1 \eta_1 T \eta_2 \tau_2 \longrightarrow \tau_1 \eta_1 V \eta_2 \tau_2 \longrightarrow^* \omega$$

e substituida em $\mathcal{G}'$ por

$$S \longrightarrow^* \tau_1' R \tau_2' \longrightarrow \tau_1' \eta_1'' \eta_2'' \tau_2' \longrightarrow^* \omega$$

Se $\eta_1'' \eta_2'' = \lambda$ o mesmo processo deve ser repetido procurando-se a última utilização de uma produção com R no lado direito. Contudo, como $\omega \neq \lambda$, este processo não pode continuar indefinidamente, o que completa a demonstração de que $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}')$.

❑

Teorema 3.3.4. *Toda a linguagem de contexto livre pode ser gerada por uma gramática de contexto livre que não contenha produções triviais.*

Prova: Seja $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ uma gramática de contexto livre tal que $\mathcal{L} = \mathcal{L}(\mathcal{G})$. Vamos por definir uma nova gramática $\mathcal{G}'$ cuja relação de derivação imediata $\mathcal{P}'$ é obtida de $\mathcal{G}$ por remoção das produções triviais e pela aplicação da regra seguinte. Seja $A \longrightarrow \alpha$, com $\alpha \in \mathcal{V}^+$, uma produção de $\mathcal{G}$. Para todo o $X \in \Omega$ susceptível de ser ligado a A por uma sequência finita de aplicações de produções triviais, *i.é*,

$$X \longrightarrow A_1 \longrightarrow A_2 \longrightarrow A_3 \longrightarrow \ldots \longrightarrow A_n \longrightarrow A$$

com $n \geq 0$, incluir em $\mathcal{P}'$ uma nova produção $X \longrightarrow \alpha$.

A igualdade $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{G}')$ é estabelecida trivialmente.

❑

Estamos agora em condições de definir o que se entende por normalização de gramáticas de contexto livre. A definição de forma canónica a seguir apresentada, introduzida por Chomsky e, muitas vezes, referida pelo seu nome, não sendo a única possível, é suficiente para os propósitos do nosso estudo.

Definição 3.3.5. Uma gramática de contexto livre $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ diz-se estritamente normalizada sse as suas produções forem exclusivamente da forma

$$\begin{array}{llll} & A & \longrightarrow & A_1 A_2 \quad \text{com } A, A_1, A_2 \in \Omega \\ \text{ou} & A & \longrightarrow & a \qquad\qquad \text{com } a \in \Sigma \end{array}$$

A gramática diz-se normalizada se todas as suas produções respeitarem a forma acima indicada a menos da possibilidade de incluir, no máximo, uma produção-ε e uma produção trivial para o símbolo inicial.

Teorema 3.3.5. *Qualquer linguagem $\mathcal{L}$ de contexto livre que não contenha a palavra vazia ($\lambda \notin \mathcal{L}$) pode ser gerada por uma gramática de contexto livre estritamente normalizada.*

Prova: Uma linguagem nas condições do enunciado pode ser gerada por uma gramática de contexto livre sem produções-ε nem produções triviais, de acordo com os teoremas 3.3.3 e 3.3.4. Seja $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ a gramática em causa e mostremos que existe uma versão estritamente normalizada equivalente. A construção será feita em dois passos.

As produções em $\mathcal{P}$ são da forma $A \longrightarrow a$, com $a \in \Sigma, A \in \Omega$, ou $A \longrightarrow \alpha$, com $A \in \Omega, \alpha \in \mathcal{V}^+$ e $\alpha \notin \Omega$. Definamos um novo conjunto de produções $\mathcal{P}'$ reunindo

i) Todas as produções de $\mathcal{P}$ do primeiro tipo

ii) Para toda a produção de $\mathcal{P}$ do segundo tipo, uma produção $A \longrightarrow \alpha'$, onde α' é obtido de α por substituição de toda a ocorrência de um símbolo terminal a por um novo não terminal N_a

iii) Para cada símbolo terminal a, uma produção $N_a \longrightarrow a$.

A gramática $\mathcal{G}' = (\Omega \cup \{N_a |\ a \in \Sigma\}, \Sigma, \mathcal{P}', S)$ é equivalente a $\mathcal{G}$ (como facilmente poderá verificar) e as suas produções são da forma

$$\begin{array}{lcll} & A & \longrightarrow & \alpha & \text{com } A \in \Omega, \alpha \in \Omega^+ \\ \text{ou} & A & \longrightarrow & a & \text{com } A \in \Omega, a \in \Sigma \end{array}$$

Podemos, agora, convertê-la à forma estritamente normal substituindo cada produção da forma

$$A \longrightarrow A_1 A_2 A_3 ... A_n$$

com $n \geq 3$ por um conjunto de produções $A \longrightarrow A_1 X_1$, $X_1 \longrightarrow A_2 X_2$, $X_2 \longrightarrow A_3 X_4$, ..., $X_{n-2} \longrightarrow A_{n-1} A_n$, onde $X_1, X_2, ..., X_{n-2}$ representam novos símbolos não terminais.

❏

Teorema 3.3.6. *Qualquer linguagem $\mathcal{L}$ de contexto livre pode ser gerada por uma gramática de contexto livre normalizada.*

Prova: Se $\lambda \notin \mathcal{L}$ o teorema 3.3.5 estabelece o resultado pretendido. Consideremos, agora, o caso geral em que $\lambda \in \mathcal{L}$. Seja $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ a gramática estritamente normalizada que gera $\mathcal{L} - \{\lambda\}$. Então a gramática

$$\mathcal{G}' = (\Omega \cup \{Z\}, \Sigma, \mathcal{P} \cup \{Z \longrightarrow S, Z \longrightarrow \varepsilon\}, Z)$$

é normalizada e, claramente, $\mathcal{L} = \mathcal{L}(\mathcal{G}')$.

❏

Uma gramática $\mathcal{G}$ normalizada gera, para qualquer palavra $\omega \in \mathcal{L}(\mathcal{G})$, uma árvore de derivação binária. Além disso, pode mostrar-se que existe uma relação exacta entre $|\omega|$ e a profundidade da árvore de derivação respectiva. Assim, se $|\omega| = 2^n$, a profundidade da árvore será n. Por profundidade de uma árvore de derivação entende-se o comprimento, medido em número de símbolos não terminais, do seu ramo mais extenso.

Vejamos, agora, um resultado que é a versão para as gramáticas de contexto livre do teorema 2.2.2 para linguagens regulares, desempenhando, a este nível, funções de classificação análogas.

Teorema 3.3.7. *Seja $\mathcal{L}$ uma linguagem de contexto livre. Então existe um $n \in \mathbb{N}$ tal que toda a palavra $\omega \in \mathcal{L}(\mathcal{G})$, com $|\omega| > n$, pode ser factorizada em $\omega = uvzxy$ de forma que*

$$\begin{array}{ll} i) & |vx| \geq 1 \\ ii) & |vzx| \leq n \\ iii) & \forall_{m \geq 0}.\ uv^m zx^m y \in \mathcal{L} \end{array}$$

Prova: Seja $\mathcal{L}$ uma linguagem de contexto livre e $\mathcal{G} = (\Omega, \Sigma, \mathcal{P}, S)$ a gramática estritamente normalizada tal que $\mathcal{L}(\mathcal{G}) = \mathcal{L} - \{\lambda\}$, cuja existência é garantida pelo teorema 3.3.5. A árvore de derivação gerada por $\mathcal{G}$ para qualquer palavra $\omega \in \mathcal{L} - \{\lambda\}$ é uma árvore binária cujas folhas correspondem a distintas ocorrências de símbolos de Σ. Fixemos $n = 2^{k+1}$, onde $k = |\Omega|$, e tomemos uma palavra $\omega \in \mathcal{L}(\mathcal{G})$ tal que $|\omega| > n$. O teorema é vacuosamente verdadeiro se não existir nenhuma palavra nessas condições. A profundidade da árvore de derivação de ω é pelo menos igual a $k+1$, de acordo com a observação acima, o que significa que nela existe pelo menos um caminho com pelo menos $k+2$ nodos, dos quais os primeiros $k+1$ são não terminais e que não podem ser, portanto, todos distintos. Seja $A \in \Omega$ o símbolo não terminal que se repete nesse caminho. Então existe uma derivação em $\mathcal{G}$

$$S \longrightarrow^* uAy \longrightarrow^* uvAxy \longrightarrow^* uvzxy = \omega$$

com $u, v, z, x, y \in \Sigma^*$, de acordo com a figura seguinte

Isto significa que existem produções em $\mathcal{P}$ que permitem as derivações $S \longrightarrow^* uAy$, $A \longrightarrow^* vAx$ e $A \longrightarrow^* z$. Logo, $uzy \in \mathcal{L}(\mathcal{G})$. As mesmas produções permitem concluir que $uvzxy \in \mathcal{L}(\mathcal{G})$ e, indutivamente, que, para todo o $m \geq 0$, $uv^m zx^m y \in \mathcal{L}(\mathcal{G})$. Como as duas ocorrências de A consideradas surgem em pontos distintos da árvore de derivação, pelo menos uma produção de $\mathcal{P}$ foi usada na derivação $uAy \longrightarrow^* uvAxy$. Como $\mathcal{G}$ não contém produções-ε nem produções triviais, u e x não podem ser simultaneamente vazias, donde $|vx| \geq 1$. Por outro lado, a sub-árvore com raíz na primeira ocorrência de A tem uma profundidade que não excede $k+1$, logo a palavra por ela gerada — *i.é*, vzx — não pode ter mais que 2^{k+1} símbolos, donde $|vzx| \leq n$, o que completa a demonstração.

❑

Como o número de símbolos não terminais numa gramática é finito e qualquer gramática de contexto livre pode ser normalizada, o teorema indica que é sempre possível escolher um número natural $n \in \mathbb{N}$ tal que a existência em $\mathcal{L}$ de palavras de comprimento igual ou superior a n, implica a existência em $\mathcal{L}$ de uma sequência infinita de palavras distintas (uma vez que v e x não são simultaneamente vazias).

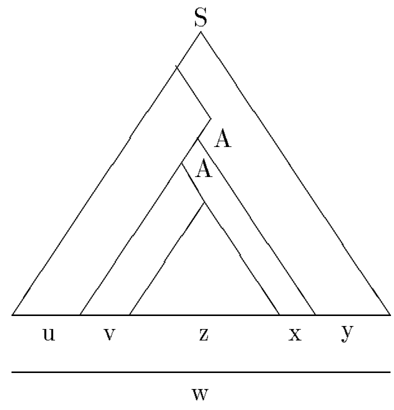

Figura 3.3: Esquema de derivação

Tal como acontecia com o resultado correspondente para as linguagens regulares, este teorema é usualmente usado negativamente, *i.é*, para mostrar que uma dada linguagem não é de contexto livre. É o que se ilustra no seguinte exemplo.

Exemplo 3.3.7. Pretende-se indagar se a linguagem

$$\mathcal{L} \;=\; \{a^k b^k a^k \mid k \geq 1\}$$

é de contexto livre. Suponhamos a afirmativa. Então, de acordo com o teorema 3.3.7, para um $n \in \mathbb{N}$ suficientemente grande, é possível encontrar $u, v, z, x, y \in \{a, b\}^*$ tal que $\omega = a^k b^k a^k = uvzxy$ e $\mathcal{L}$ contém $uv^m zx^m y$, para todo o $m \geq 0$.

Note-se que todas as palavras de $\mathcal{L}$ apresentam sempre duas fronteiras, definindo-se fronteira como a passagem de uma sub-sequência potência de um dado elemento de Σ a uma sub-sequência que é potência de um outro. No caso das palavras de $\mathcal{L}$ existe uma fronteira que separa o bloco da primeira potência de a do bloco gerado por b, e outra que separa este último do segundo bloco gerado por a.

Claramente, na factorização de ω, v e x não podem incluir nenhuma fronteira, pois, nesse caso, a palavra uv^2zx^2y teria mais de duas fronteiras e, consequentemente, não pertenceria a $\mathcal{L}$. Logo, v e x são potências de um único elemento de Σ. Suponhamos que pertencem a dois blocos distintos de ω. Então, à medida que aumentamos m, o comprimento desses dois blocos cresce enquanto o do terceiro permanece inalterado. Se, por outro lado, v e x pertencem ao mesmo bloco, com o aumento de m o comprimento desse bloco aumentará ao passo que o dos dois restantes permanece inalterado. Em qualquer caso, para $m \geq 1$, $uv^m zx^m y \notin \mathcal{L}$, pelo que a factorização indicada é impossível, o que contradiz a hipótese de que partimos. Logo, $\mathcal{L}$ não é de contexto livre.

O leitor é convidado a exlporar a utilização do teorema para verificar que as linguagens $\mathcal{L} = \{a^m b^n a^m b^n \mid m, n \geq 1\}$ e $\mathcal{Q} = \{a^n b^n c^n \mid n \geq 1\}$ não são, igualmente, de contexto livre.

Finalizaremos este estudo com a análise de algumas propriedades das linguagens de contexto livre relativas ao seu fecho para os operadores usuais sobre 2^{Σ^*}.

Teorema 3.3.8. *O conjunto das linguagens de contexto livre é fechado para a reunião, concatenação e fecho de Kleene.*

Prova: Sejam $\mathcal{L}_1$ e $\mathcal{L}_2$ linguagens de contexto livre e $\mathcal{G}_1$, $\mathcal{G}_2$ as gramáticas que as geram. Suponhamos, ainda, sem perda de generalidade que $\Omega_1 \cap \Omega_2 = \emptyset$. Verifiquemos, então as três afirmações do teorema.

i) Definamos uma nova gramática

$$\mathcal{G} \ = \ (\Omega_1 \cup \Omega_2 \cup \{Z\}, \Sigma, \mathcal{P}, Z)$$

onde $Z \notin \Omega_1 \cup \Omega_2$ e $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \{Z \longrightarrow S_1, Z \longrightarrow S_2\}$.

Se $\omega \in \mathcal{L}_1$ então existe uma derivação $S_1 \longrightarrow^* \omega$ em $\mathcal{G}_1$ a que corresponde a derivação $Z \longrightarrow S_1 \longrightarrow^* \omega$ em $\mathcal{G}$, e, similarmente, para $\omega \in \mathcal{L}_2$. Logo, $\mathcal{L}_1 \cup \mathcal{L}_2 \subseteq \mathcal{L}(\mathcal{G})$.

Conversamente, seja $\omega \in \mathcal{L}(\mathcal{G})$. Então existe uma derivação $Z \longrightarrow^* \omega$ cujo primeiro passo é necessariamente $Z \longrightarrow S_1$ ou $Z \longrightarrow S_2$. Conforme os casos a derivação prossegue usando produções em $\mathcal{P}_1$ ou $\mathcal{P}_2$, respectivamente, atendendo a que Ω_1 e Ω_2 são disjuntos. Logo, existem as derivações $S_1 \longrightarrow^* \omega$ em $\mathcal{G}_1$ ou $S_2 \longrightarrow^* \omega$ em $\mathcal{G}_2$. Donde $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}_1 \cup \mathcal{L}_2$.

ii) Definamos agora

$$\mathcal{G} \ = \ (\Omega_1 \cup \Omega_2 \cup \{Z\}, \Sigma, \mathcal{P}, Z)$$

onde $Z \notin \Omega_1 \cup \Omega_2$ e $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \{Z \longrightarrow S_1 S_2\}$.

Se $\omega = \omega_1\omega_2 \in \mathcal{L}_1\mathcal{L}_2$, com $\omega_1 \in \mathcal{L}_1$ e $\omega_2 \in \mathcal{L}_2$, então existem as derivações $S_1 \longrightarrow^* \omega_1$ e $S_2 \longrightarrow^* \omega_2$ em $\mathcal{G}_1$ e $\mathcal{G}_2$, respectivamente. Assim, em $\mathcal{G}$, temos a derivação $Z \longrightarrow S_1S_2 \longrightarrow^* \omega_1\omega_2$. Logo $\mathcal{L}_1\mathcal{L}_2 \subseteq \mathcal{L}(\mathcal{G})$.

Conversamente, suponhamos $\omega \in \mathcal{L}(\mathcal{G})$. A derivação $Z \longrightarrow^* \omega$ inicia-se por $Z \longrightarrow S_1S_2$ e, pelo teorema 3.3.1, teremos $\omega = \omega_1\omega_2$ e $S_1 \longrightarrow^* \omega_1$ e $S_2 \longrightarrow^* \omega_2$ derivações em $\mathcal{G}$. Como $\Omega_1 \cap \Omega_2 = \emptyset$, a primeira derivação pertence a $\mathcal{G}_1$ e a segunda a $\mathcal{G}_2$. Logo, $\omega_1 \in \mathcal{L}_1$ e $\omega_2 \in \mathcal{L}_2$ e, portanto, $\omega = \omega_1\omega_2 \in \mathcal{L}_1\mathcal{L}_2$. Donde, $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}_1\mathcal{L}_2$.

iii) Definamos, por fim,

$$\mathcal{G} \ = \ (\Omega_1 \cup \{Z\}, \Sigma, \mathcal{P}, Z)$$

onde $Z \notin \Omega_1$ e $\mathcal{P} = \mathcal{P}_1 \cup \{Z \longrightarrow S_1 Z, Z \longrightarrow \varepsilon\}$.

Seja $\omega = \omega_1\omega_2...\omega_n \in \mathcal{L}_1^+$, com $\omega_1, \omega_2, ..., \omega_n \in \mathcal{L}_1$. Então, existe em $\mathcal{G}_1$ uma derivação $S_1 \longrightarrow^* \omega_i$ para cada $i \in \{1, 2, ..., n\}$. Em $\mathcal{G}$ teremos a derivação

$$Z \longrightarrow S_1Z \longrightarrow^* \omega_1 Z \longrightarrow \omega_1 S_1 Z \longrightarrow^* \omega_1\omega_2 Z \longrightarrow^* ... \longrightarrow^* \omega_1\omega_2...\omega_n Z \longrightarrow \omega_1\omega_2...\omega_n$$

Logo, $\mathcal{L}_1^+ \subseteq \mathcal{L}(\mathcal{G})$. Igualmente $\lambda \in \mathcal{L}(\mathcal{G})$, atendendo à produção $Z \longrightarrow \varepsilon$, pelo que $\mathcal{L}_1^* \subseteq \mathcal{L}(\mathcal{G})$.

Conversamente, seja $\omega \in \mathcal{L}(\mathcal{G})$. A derivação $Z \longrightarrow^* \omega$ será $Z \longrightarrow \varepsilon$, se $\omega = \lambda$, ou, caso contrário, deverá iniciar-se por $Z \longrightarrow S_1Z$. Esta produção pode ser usada várias vezes até ao desaparecimento de Z da palavra parcial. O resultado $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}_1^*$ segue por indução sobre o número de utilizações da produção referida na derivação de ω.

❑

Teorema 3.3.9. *O conjunto das linguagens de contexto livre não é fechado para a intersecção .*

Prova: Seja $\mathcal{L} = \{a^n b^n \mid n \geq 1\}$, $\mathcal{L}' = \{b^n a^n \mid n \geq 1\}$ e $\mathcal{L}'' = a^+$ três linguagens de contexto livre[9]. Pelo teorema 3.3.8, as linguagens

$$\begin{array}{rcccl} \mathcal{L}_1 & = & \mathcal{L}\mathcal{L}'' & = & \{a^n b^n a^m \mid m, n \geq 1\} \\ \mathcal{L}_2 & = & \mathcal{L}''\mathcal{L}' & = & \{a^m b^n a^n \mid m, n \geq 1\} \end{array}$$

são igualmente de contexto livre. No entanto, a sua intersecção

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \{a^n b^n a^n \mid n \geq 1\}$$

não o é, conforme discutimos no exemplo 3.3.7.

❑

Apesar da intersecção de duas linguagens de contexto livre não ser geralmente de contexto livre, como acabamos de provar, a afirmação é válida quando uma delas é regular[10]. Este facto pode ser usado para averiguar se uma dada linguagem é ou não de contexto livre, conforme se ilustra no seguinte exemplo.

Exemplo 3.3.8. Suponhamos que $\mathcal{L} = \{\omega^2 \mid \omega \in \{a, b\}^+\}$ é uma linguagem de contexto livre. Sabemos que a linguagem $\mathcal{L}' = a^+ b^+ a^+ b^+$ é regular. Então, pelo teorema referido, $\mathcal{L} \cap \mathcal{L}' = \{a^m b^n a^m b^n \mid m, n \geq 1\}$ também deveria ser de contexto livre, o que, sendo falso, contradiz a hipótese.

---

[9] Recorde que as primeiras foram discutidas no exemplo 3.3.2 e a última é regular.

[10] O leitor é referido a [CL89] para a prova deste resultado.

Capítulo 4

Autómatos Recorrentes

Capítulo 5

Máquinas de Turing e Computabilidade

Capítulo 6

Dos Autómatos aos Sistemas de Transição

Bibliografia

[Bar80] H. Barendregt. *The Lambda-Calculus: Its Syntax and Semantics.* North-Holland, 1980.

[Boo67] T. L. Booth. *Sequential Machines and Automata Theory.* John Wiley & Sons, 1967.

[Cho56] N. Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory,* (2):113–124, 1956.

[Cho69] N. Chomsky. On Certain Formal Properties of Grammars. *Information and Control,* (2):137–167, 1969.

[CL89] Carrol and Long. *Theory of Finite Automata.* Series in Computer Science. Prentice-Hall International, 1989.

[Eil74] S. Eilenberg. *Automata, Languages and Machines.* Academic Press (2 vol.), 1974.

[Han94] C. Hankin. *Lambda Calculi: A Guide for Computer Scientists.* Graduate Texts in Computer Science. Clarendon Press, Oxford, 1994.

[Hen88] M. C. Hennessy. *Algebraic Theory of Processes.* Series in the Foundations of Computing. MIT Press, 1988.

[Hol82] W. M. L. Holcombe. *Algebraic Automata Theory.* Cambridge University Press, 1982.

[How91] Howie. *Automata and Languages.* Cambridge University Press, 1991.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata, Languages and Computation.* Addison-Wesley, 1979.

[Joh82] P. Johnstone. *Stone Spaces.* Cambridge University Press, 1982.

[McC65] W. S. McCulloch. *Embodiments of Mind.* Graduate Texts in Computer Science. MIT Press, Cambridge, Mass., 1965.

[Mil80] A. J. R. G. Milner. *A Calculus of Communicating Systems.* LNCS (92), 1980.

[ML86] S. Mac Lane. *Mathematics Form and Function.* Springer Verlag, 1986.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Pub. Co., 1994.

[Pop56] K. Popper. *The Open Universe.* Hutchinson Co. Ltd, London, 1956.

[Rut95] J. Rutten. A calculus of Transition Systems (Towards Universal Co-algebra). Technical Report CS-R9503, CWI, 1995.

[Tur50] A. M. Turing. Computing Machinery and Intelligence. *Mind*, 59(236), 1950. reprinted in *The Mind's I* (Hofstadter, D., Dennett, D. eds, Penguin Books, 1981).

[WN93] G. Winskel and M. Nielsen. Models for Concurrency. In *Handbook of Theoretical Computer Science*. Draft version as Lecture Notes of Tempus Summer School on Algebraic and Categorical Methods in Computer Science, Brno, 1993.