LICENCIATURA EM ENGENHARIA INFORMÁTICA

TEORIA DA COMPUTAÇÃO

António Dourado Pereira Correia

Advertência. Este documento é um texto de trabalho para apoio ao estudo da cadeira de Teoria da Computação. Não inclui toda a matéria leccionada na disciplina e naturalmente não dispensa a consulta da bibliografia complementar. O autor agradece qualquer comentário ou sugestão que o tolerante e paciente leitor entenda por bem fazer.

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia
Universidade de Coimbra
Setembro 2009

Índice Geral

CAPÍTULO 1

INTRODUÇÃO E DEFINIÇÕES BÁSICAS

1.1. Introdução

As ciências da computação, numa interpretação genérica, têm a ver com todos os aspectos relacionados com os computadores e o seu funcionamento. Há no entanto três temas centrais que são delas estruturantes: autómatos, computabilidade, complexidade. As três visam dar resposta à questão primeira - quais são as capacidades e as limitações fundamentais dos computadores? O que se pode computar? Estas que têm ocupado intensamente os cientistas da computação desde os anos 30 do século passado, quando a noção e o significado de computação surgiu no panorama científico.

Nesta disciplina iremos abordar aqueles três temas. Começaremos pelos autómatos e veremos depois os elementos introdutórios fundamentais da computabilidade e da complexidade.

A teoria dos autómatos compõe o edifício formal que sustenta solidamente todo o desenvolvimento de processadores de texto, de compiladores e de hardware. Suporta também (através das gramáticas associadas aos autómatos) todo o desenvolvimento das linguagens de programação. Assim linguagens, gramáticas e autómatos são, com veremos, as três faces de um mesmo prisma. Vejamos formalmente em que consistem.

1.2.Linguagens formais

Tal como as linguagens humanas, as linguagens formais (assim chamadas por serem definidas por um conjunto de formalismos matemáticos abstractos) são “faladas” por autómatos, isto é, um autómato compreende, e sabe interpretar uma dada linguagem, de uma forma que veremos posteriormente. É aliás interessante constatar que os estudiosos das linguagens humanas (os linguistas) deram uma contribuição muito importante para o desenvolvimento da teoria das linguagens formais. Por exemplo Chomsky, de que falaremos no Capítulo 6, é um linguista.

Uma linguagem exprime-se através de símbolos de uma certa forma (tal como as linguagens humanas ao longo da história usaram símbolos diversos, desde os hieroglíficos até aos caracteres latinos, passando pelos chineses, pelos gregos e pelos árabes). Poderemos definir formalmente esta realidade.

1.2.1 Alfabeto, cadeias e operações sobre cadeias

Um **alfabeto** é composto por um conjunto não vazio de símbolos, usualmente representada pela letra grega **Σ (**sigma), de símbolo.

Σ={símbolos}, conjunto não vazio de símbolos (letras, algarismos,...)

Os símbolos de um alfabeto podem ser de qualquer natureza e assumir um aspecto arbitrário. Qualquer alfabeto é um conjunto, e esta é a única característica comum. A própria palavra alfabeto é composta pela concatenação do primeiro e segundo símbolos grego, o alfa e o beta.

Exemplos de alfabetos:

Σ = {*a,b*}, alfabeto composto pelas duas letras *a* e *b*.

Σ = {0,1}, alfabeto binário composto pelos valores binários 0 e 1.

Σ = {(,),[,] } , alfabeto dos parênteses

Σ = {0,1,2,3,4,5,6,7,8, 9 }, alfabeto dos algarismos árabes

Σ ={*a,b,c, ..., z*}, o alfabeto romano minúsculo

Σ ={conjunto de todos as caracteres ASCII}, o alfabeto ASCII.

Σ = {copos, facas, garfos, pratos, colheres, tachos}, o alfabeto dos instrumentos de cozinha

Σ = {maçã, pêra, ameixa, banana, … }, o alfabeto dos frutos.

Qualquer objecto pode pertencer a um alfabeto. Por isso um alfabeto pode ser um conjunto de qualquer espécie de coisas.

Por razões de simplicidade usam-se normalmente apenas letras, algarismos e outros caracteres comuns como #, $, &, etc.

Juntando-se vários caracteres de um alfabeto obtém-se uma **cadeia** (*string,* em inglês), *w*, formalmente definida como uma sequências finitas de símbolos do alfabeto.

Exemplo 1.2.1

*w= a, w=ab, w=abbaba, ...*cadeias no alfabeto $\Sigma = \{a,b\}$

Exemplo 1.2.2

w =1,*w*=2, *w*=23, *w*=5674*,* cadeias de algarismos árabes.

Por vezes uma cadeia também se chama **palavra**. De facto nas linguagens humanas as palavras são cadeias de caracteres seguindo uma certa regra de formação – a ortografia. Alguns autores de língua inglesa (por exemplo Linz) usam o termo *phrase* frase) como sinónimo de cadeia. De facto uma cadeia é gerada aplicando as regras (produções) de uma gramática, tal como uma frase de uma linguagem humana se constrói usando as regras da respectiva gramática. Nesse sentido as palavras são os elementos atómicos da linguagem humana que correspondem aos símbolos de um alfabeto de uma linguagem formal. Para evitar confusões usaremos preferencialmente o termo cadeia.

Fazendo a concatenação de duas cadeias *w* e *v* obtém-se uma terceira cadeia que por isso se chama a **concatenação** de *w* e *v.*

Pode-se obter fazendo a concatenação à direita (*v* à direita de *w*), por exemplo sendo

$$w= a_1a_2a_3 \ldots a_n \qquad v= b_1b_2b_3\ldots b_n$$

a concatenação à direita de *v* com *w* , *wv,* será

$$wv=a_1a_2a_3\ldots a_nb_1b_2b_3\ldots b_n$$

e a concatenação à esquerda, *vw*

$$vw=b_1b_2b_3\ldots b_na_1a_2a_3\ldots a_n$$

Os caracteres numa cadeia seguem uma certa ordem. Por vezes estamos interessados em alterar essa ordem, obtendo-se cadeias derivadas da original. Por exemplo a cadeia **reversa**, obtém-se invertendo (da direita para a esquerda) a ordenação dos caracteres. A reversa da cadeia *w* acima será, revertendo *w,*

$$w^R = a_n \ldots a_3 a_2 a_1$$

O número de caracteres de uma cadeia pode ser qualquer, sendo por isso umas longas e outras curtas. Chama-se **comprimento** de uma cadeia, anotando-se pelo módulo, |w|, o número de posições, ou de casas, ocupadas pelos caracteres na cadeia. Por exemplo 011011 tem comprimento 6. Por vezes também se define comprimento como o número de caracteres da cadeia, o que está certo se se incluírem as repetições nesse número. De outro modo a cadeia anterior tem 2 caracteres (0 e 1) que, repetidos, ocupam 6 posições.

Se uma cadeia não tem qualquer carácter (note-se que o singular de caracteres é carácter, e não carater), terá 0 caracteres, sendo por isso **vazia** (um conjunto vazio), e denota-se por λ (alguns autores denotam por ε) . O seu comprimento será nulo, $|\lambda|=0$, e pode escolher-se de qualquer alfabeto.

Se concatenarmos uma cadeia vazia com outra cadeia *w* qualquer, obtém-se

$$\lambda w = w\lambda = w, \quad \forall w,$$

para todo e qualquer *w,* tal como a união de um conjunto com o conjunto vazio dá o conjunto original, qualquer que ele seja.

Certas cadeias têm formas especiais. Por exemplo *abcdedcba* ou *abcdeedcba* podem ler-se igualmente da frente para trás ou de trás para a frente. Elas têm um ponto de simetria. Em linguagem corrente chamam-se capicuas e mais formalmente chamam-se **palíndromos** e são tais que $w = w^R$. Voltaremos a falar deles.

Subcadeia de uma cadeia *w* é qualquer cadeia composta por caracteres sucessivos de *w* . Se dividirmos qualquer cadeia *w* em duas partes *u* e *v*, tal que *w=uv,* diz-se que *u* é um **prefixo** e *v* um **sufixo** de *w.*

Se por exemplo

$$w=abbab$$

Poderemos definir os seguintes prefixos de w:

$$\{\lambda, a, ab, abb, abba, abbab\}$$

e os seguintes sufixos de *w*:

$$\{abbab,\ bbab,\ bab,\ ab,\ b,\ \lambda\}$$

Note-se que cada prefixo está associado a um sufixo. O prefixo *a* está associado ao sufixo *bbab* e o sufixo λ ao prefixo *abbab*. A ordem por que estão escritos acima reflecte essa associação.

Potência *n* de uma cadeia, w^n, é uma cadeia obtida pela concatenação *n* vezes da cadeia *w* consigo própria. Assim por exemplo

$$w^2 = ww$$

$$w^3 = www,$$

etc.

É fácil de ver que

$$w^1 = w$$

Quanto a w^0, ela será a concatenação de *w* consigo própria zero vezes. Se a concatenação uma vez dá a própria cadeia, a concatenação zero vezes dá nada, ou seja, a cadeia vazia e portanto

$$w^0 = \{\lambda\}$$

para todo e qualquer *w*.

O sinal de potência também se usa para exprimir a concatenação de um carácter consigo próprio. Por exemplo

$$a^3 = aaa,$$

$$1^2 0^4 = 110000,$$

$$a^n b^m = aa\ldots abb\ldots b.$$

Também se define **potência de um alfabeto**, Σ^n, como o conjunto das cadeias desse alfabeto de comprimento *n*. Assim se $\Sigma = \{0,1\}$,

$\Sigma^0 = \{\lambda\}$, cadeia vazia

Σ^1={0,1} , cadeias com um carácter

Σ^2={00,01,10,11}, cadeias com dois caracteres

Σ^3={000, 001, 0010, …, 111} cadeias com três caracteres

etc.

(note-se que enquanto Σ é um conjunto de caracteres (símbolos), Σ^1 é um conjunto de cadeias, mesmo que tenham apenas um carácter).

Se definirmos o conjunto de todas as potências possíveis de um alfabeto, esse conjunto terá todas as cadeias com zero, um, dois, …, qualquer número de caracteres do alfabeto. Para exprimir este facto usa-se o símbolo estrela, *****, para significar qualquer, e chama-se a Σ^* o **fecho estrela** do alfabeto (*star-closure*). Assim Σ^* é conjunto de todas as cadeias que se podem obter pela concatenação de zero ou mais símbolos de Σ. Contém sempre a cadeia nula λ. É simplesmente o conjunto de todas as cadeias possíveis no alfabeto Σ, e é uma forma simples e elegante de denotar esse conjunto infinito.

Se excluirmos a cadeia vazia do fecho estrela, obtemos o **fecho estrela positivo**, Σ^+

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$

É fácil de ver, pela definição, que quer Σ^* quer Σ^+ são sempre conjuntos infinitos, mesmo quando o alfabeto Σ é finito.

Chegamos agora finalmente à definição de linguagem, um dos conceitos chave da computação:

Definição 1.2.1. **Linguagem** : dado um alfabeto qualquer Σ, **linguagem *L*** é qualquer subconjunto de Σ^*.

Assim sendo, num alfabeto qualquer podem-se definir múltiplas linguagens. Elas terão propriedades diferentes que lhes dão características bem distintas, que estudaremos na ocasião oportuna. Uma linguagem num alfabeto não precisa de ter todos os caracteres desse alfabeto, pode ter apenas uma parte deles.

Define-se **frase** (ou **sentença**) como qualquer cadeia na linguagem formal *L*

A língua portuguesa é um conjunto de cadeias formadas pelas letras do nosso alfabeto latino. Na linguagem corrente cada cadeia é uma palavra e uma frase pode ter várias palavras.

Na linguagem Java, um programa qualquer correctamente escrito é um subconjunto de todas as cadeias possíveis que se podem formar com os caracteres da linguagem. O seu alfabeto é um subconjunto dos caracteres ASCII.

Seja por exemplo o alfabeto $\Sigma=\{a,b\}$.

Nele $\Sigma^*=\{\lambda,a,b,aa,ab,ba,bb,aaa,aab,aba,abb,baa,bab,bba,bbb, ...\}$, contém todas as combinações possíveis dos símbolos do alfabeto, incluindo a cadeia vazia.

(i) O conjunto $L_1=\{a, ab, abb\}$ é uma linguagem em Σ. É uma linguagem finita.

(ii) O conjunto $L_2=\{a^n b^n: n\geq 0\}$ é uma linguagem em Σ. É uma linguagem infinita.

As cadeias *ab, aabb, aaabb,* pertencem a L_2 porque se podem escrever na forma respectivamente a^1b^1, a^2b^2, a^3b^3. Como veremos a gramática da linguagem define as regras da formação das cadeias.

Já as cadeias *aba, abb, aaababbab,* não pertencem a L_2 (tente escrevê-las naquela forma de potências).

Por vezes define-se uma linguagem exprimindo verbalmente as suas propriedades específicas que a distinguem de outra qualquer. Por exemplo:

- o conjunto de cadeias em $\Sigma=\{0,1\}$ com um número de 0's igual ao número de 1's

$$L=\{\lambda, 01, 10, 0011, 1100, 0110, 0101, 1001, \ldots\}$$

- o conjunto dos números binários cujo correspondente decimal é primo

$$L=\{10,11,101,111,1011, 1101, 10001, \ldots\}$$

Também se pode constatar que ϕ, a linguagem vazia, não tem qualquer cadeia, é uma linguagem em qualquer alfabeto.

Também $\{\lambda\}$, a linguagem composta apenas pela carácter vazio, é uma linguagem em qualquer alfabeto.

Note-se a diferença entre ϕ e $\{\lambda\}$: a primeira não tem qualquer cadeia e a segunda tem uma cadeia (a cadeia vazia é de facto uma cadeia, mesmo que vazia).

1.2.2. Operações de conjuntos sobre linguagens e suas propriedades

Uma linguagem é, como vimos, um conjunto. Portanto aplicam-se-lhe todas as operações sobre conjuntos. Se tivermos duas linguagens L_1 e L_2, poderemos fazer com elas as seguintes operações:

União*:* todas as cadeias que pertencem ou a uma ou a outra,

$$L_1 \cup L_2 = \{w: w \in L_1 \text{ ou } w \in L_2\},$$

Intersecção*:* todas as cadeias que pertencem simultaneamente a uma e a outra,

$$L_1 \cap L_2 = \{w: w \in L_1 \text{ e } w \in L_2\},$$

Diferença de duas linguagens*:* todas as cadeias que pertencem a uma e não pertencem à outra,

$$L_1 - L_2 = \{w: w \in L_1 \text{ e } w \notin L_2\},$$

$$L_2 - L_1 = \{w: w \notin L_1 \text{ e } w \in L_2\}.$$

Complemento de uma linguagem: todas as cadeias possíveis no alfabeto respectivo (isto é, Σ^*) menos as cadeias da linguagem complementada,

$$Compl(L) = \overline{L} = \Sigma^* - L.$$

Reversão de uma linguagem: todas as cadeias que resultam da reversão das cadeias originais de L constituem a linguagem reversa de L denotada por L^R

$$L^R = \{w^R : w \in L\}.$$

Concatenação de duas linguagens*:* todas as cadeias em que uma parte (prefixo) pertence à primeira linguagem e a parte restante (sufixo), à segunda

$$L_1 \cdot L_2 = \{uv: u \in L_1, v \in L_2\},.$$

Dada a concatenação de L_1 e L_2 será que $L_1 \subseteq L_1 \cdot L_2$, para quaisquer L_1 e $\cdot L_2$ em quaisquer alfabetos Σ_1 e Σ_2 ?

Tal será possível se e só se $\lambda \in L_2$. Vejamos as várias situações possíveis.

(i) No caso em que os dois alfabetos são disjuntos, se λ não pertence a L_2, então a concatenação de uma qualquer cadeia de L_1 com uma qualquer cadeia de L_2 resulta numa cadeia que não pertence a L_1 e por isso nenhuma cadeia de L_1 neste caso pertence à concatenação.

(ii) no caso particular em que as duas linguagens partilham o mesmo alfabeto, ao concatenarmos a menor cadeia de L_1 com a menor cadeia de L_2, obtém-se uma cadeia maior do que a primeira, e portanto não é possível obter em L_1L_2 a menor cadeia de L_1. Pelo menos essa não pertence a L_1L_2 e por isso o mesmo sucede a L_1 (esta só pertence à concatenação se todas as suas cadeias pertencerem). No caso extremo em que $L_1=L_2$, verifica-se a mesma situação.

Mutatis mutandis da anterior $L_2 \subseteq L_1 \cdot L_2$ se e só se $\lambda \in L_1$

Distributividade da concatenação em ordem à união

Considerem-se, a título de exemplo, as linguagens

$$L_1=\{a, ab, b\},$$

$$L_2=\{0, 01\}$$

$$L_3=\{11, 111, 1111\}$$

Então

$L_2 \cup L_3=\{0,01,11,111,1111\}$ e

$L_1 . (L_2 \cup L_3) =\{a, ab, b\}.\{0,01,11,111,1111\}=$

$=\{a0,a01, a11,a111,a1111,ab0,ab01,ab11,ab111,ab1111,b0,b01, b11,b111,b1111\}$

$=\{(a0,a01, ab0,ab01, b0, b01), (a11,a111,a1111, ab11,ab111,ab1111, b11,b111,b1111)\}$

$= \{a,ab,b\}.\{0,01\} \cup \{a,ab,b\}\{11, 111, 1111\} = .\{L_1 \cdot L_2 \cup L_1 L_3$

Este resultado pode-se generalizar: a concatenação é distributiva em relação à união.

$$L_1 . \ (L_2 \cup L_3) = \ L_1 \cdot L_2 \cup L_1 L_3.$$

De facto se uma cadeia pertence à união de duas linguagens, ela irá entrar na concatenação. Por outro lado, se uma cadeia não pertence à união, ela não entrará na concatenação. Serão concatenadas todas as da união e apenas elas.

Será a união distributiva em relação à concatenação?

Uma propriedade definida assim genericamente tem que ser válida para todos e quaisquer casos. Se formos capazes de encontrar um só caso em que isso se não verifique, a propriedade não é geral e portanto a resposta é não.

Considerem-se novamente as três linguagens acima.

Faça-se

$$L_2 \cdot L_3 = \{0, 01\}.\{11, 111, 1111\} = \{011,0111,01111,0111,01111,011111\}$$

$$L_1 \cup (L_2 \cdot L_3) = \{a, ab, b\} \cup \{011,0111,01111,0111,01111,011111\} =$$

$$= \{a, ab, b, 011,0111,01111,0111,01111,011111\}$$

Por outro lado,

$$(L_1 \cup L_2) = L_1 = \{a, ab, b\} \cup \{0, 01\} = \{a, ab, b, 0, 01\}$$

$$(L_1 \cup L_3) = \{a, ab, b\} \cup \{11, 111, 1111\} = \{a, ab, b, 11, 111, 1111\}$$

Concatenando estas duas,

$$(L_1 \cup L_2).(L_1 \cup L_3) = \{a, ab, b, 0, 01\}.\{a, ab, b, 11, 111, 1111\}$$

$$= \{aa, aab, ab, a11, a111, a1111, ...\}$$

Ora as cadeias *aa, aab*, não pertencem a $L_1 \cup (L_2 \cdot L_3)$.

Logo

$$L_1 \cup (L_2 \cdot L_3) \neq (L_1 \cup L_2)\ (L_1 \cup L_3)$$

e concluímos que em geral a união não é distributiva sobre a concatenação.

A concatenação de uma linguagem com o conjunto vazio (linguagem vazia) resulta no conjunto vazio (analogamente ao produto de um número real por zero).

$$L \cdot \varnothing = \varnothing$$

A concatenação de uma linguagem L qualquer com a cadeia vazia resulta na linguagem L (analogamente ao produto de um número real pela unidade).

$$L \cdot \lambda = L$$

Na concatenação o conjunto vazio desempenha o papel de zero e a cadeia vazia o papel da unidade.

Potência de uma linguagem

Define-se a **potência *n* de uma linguagem** a partir da concatenação da linguagem com ela própria (auto-concatenação) n vezes.

Se concatenarmos uma linguagem zero vezes com ela própria , vem L^0.

Em que resulta ?

Para um qualquer número real, elevando-o a zero obtém-se a unidade. Então aqui será natural considerar-se que a potência zero de uma linguagem é a unidade da concatenação, ou seja, a cadeia vazia. Podemos também considerar que a potência nula de uma linguagem consiste em escolher zero cadeias dessa linguagem. Logo,

$$L^0 = \{\lambda\}$$

$$L^1 = L$$

$$L^2 = LL$$

...

Chama-se **fecho-estrela** (*star-closure, Kleene star*) de uma linguagem L^* ao conjunto de todas as cadeias que se podem obter por concatenação da linguagem com ela própria, um número arbitrário de vezes (incluindo zero vezes), ou seja,

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup .....$$

Se tivermos, por exemplo, a linguagem L={0,1}, o seu fecho estrela é o conjunto de todas as cadeias de 0's e 1's, incluindo a cadeia vazia. De facto

L^0={λ} - todas as cadeias com zero caracteres

L^1 ={0,1} - todas as cadeias de 0´s e 1´s com um carácter

L^2 ={0, 1}.{0, 1}={00,01,10,11} - todas as cadeias de 0's e1's com dois caracteres

L^3={0, 1}.{0, 1}.{0,1}={00,01,10,11}.{0,1}={000,001,010,011,110,111}

- todas as cadeias de 0's e 1's com três caracteres

L^4={0, 1}.{0, 1}.{0,1}.{0,1} - todas as cadeias de 0's e 1's com quatro caracteres,

E assim sucessivamente. Unindo agora todos os subconjuntos teremos o conjunto de todas as cadeias de 0's e 1's com zero, um, dois, três, ……, qualquer número de caracteres, ou seja, todas as cadeias possíveis de 0's e 1's. Assim o fecho estrela desta linguagem com duas cadeias é uma linguagem infinita. Note-se que L^* contém sempre λ, independentemente de L o conter ou não.

Se considerarmos a linguagem L={0, 11}, o que dará o seu fecho estrela ?

L^0={λ} - pelas razões anteriores

L^1 ={0,11}

L^2 ={0, 11}.{0, 11}={00,011,110,1111}

L^3={0, 11}.{0, 11}.{0,11}={00,011,110,1111}.{0,11}=

={000,0011,0110,01111,1100,11011,11110,111111}

E assim sucessivamente. Obtêm-se todas as cadeias de 0's e 1's em que os 1's aparecem sempre aos pares. Por exemplo a cadeia 00110110, mas não 0010110 nem 0011010. Como se poderá obter 00110110 a partir de λ ? De L^6 é possível: 0011 pertence a L^3, e 0110 pertence também a L^3.

Ainda neste exemplo o fecho estrela da linguagem finita com apenas duas cadeias resulta numa linguagem infinita.

Há algumas linguagens que têm a concatenação aparentemente estranhas.

Se considerarmos L a linguagem que contém todas as cadeias de 0´s (incluindo zero 0), então $L^*=L$. Tal como se for a linguagem conjunto de todas as cadeias de 1's . L é infinita, tal como o seu fecho estrela.

E o fecho estrela da linguagem vazia, ϕ^* ?

$$\phi^0=\{\lambda\}$$

$$\phi^1=\phi$$

$$\phi^2=\phi$$

Unindo tudo vem

$$\phi^*=\{\lambda\}\cup\phi\cup\phi=\{\lambda\}$$

Quanto à linguagem com o carácter vazio, $L=\{\lambda\}$, o seu fecho estrela é $L^*=\{\lambda\}$.

A linguagem ϕ e a linguagem $\{\lambda\}$ são as duas únicas linguagens que têm um fecho estrela finito.

Fecho-positivo (*positive closure*) de uma linguagem, L^+ , é o conjunto de todas as cadeias que se podem obter por concatenação da linguagem com ela própria uma ou mais vezes. Se L não tem λ, λ^+ também não,

$$L^+ : L^1\cup L^2\cup L^3\cup..... = L^* - \{\lambda\}$$

Exemplo: Seja

$$L=\{a^n b^n,\ n\geq 1\}$$

Logo:

$L=\{ab,\ aabb,\ aaabbb,\ aaaabbbb,\ ....\}$

$L^2 = LL=\{ab,\ aabb,\ aaabbb,\ aaaabbbb,\ ...$

$abab,\ abaabb,\ abaaabbb,\ abaaaabbbb,\ ...$

$aabbab,\ aabbaabb,\ aabbaaabbb,\ aabbaaaabbbb,\ ...$

..... }

Procurando uma forma compacta para L^2, pode-se escrever

$L^2 = \{a^n b^n a^m b^m,\ n\geq 1,\ m\geq 1\}$, n e m são independentes.

A reversa de L é fácil de calcular: basta revertermos a expressão que a define:

$$L^R=\{b^n a^n,\ n\geq 1\}$$

Se tivermos necessidade de escrever o complemento de L de uma forma sintética, como fazê-lo? Trata-se de todas as cadeias que não obedeçam à regra de formação de L. Por exemplo *aab, baa, ababab*, *bababa*, *aaaa, aaaa, bbaab, bbbbaa, ...*. Será fácil encontrar uma expressão compacta, em notação de conjuntos, que inclua todas as cadeias do complemento de L ? Fica o desafio ao leitor.

E o fecho estrela L^* ?

Nem sempre a notação de conjuntos é a mais adequada para representar linguagens. Por isso foram desenvolvidas representações alternativas: as expressões regulares e as gramáticas. Estudá-las-emos em capítulos posteriores. Mas vejamos desde já algumas noções básicas de gramáticas.

1.3. Gramáticas

A gramática de uma linguagem formal tem o mesmo objectivo da de uma linguagem humana: ela fixa, de modo que todos possamos entender, as regras de formação das palavras e das frases. Todas as linguagens escritas têm uma gramática. É a gramática que dá sentido à linguagem: sem ela as pessoas não se entenderiam

Uma gramática é também uma ferramenta para estudar matematicamente linguagens, muito poderosa, que ultrapassa as limitações da notação de conjuntos anterior.

Vejamos um exemplo da língua portuguesa

Exemplo 1.3.1

Há várias gramáticas da língua portuguesa, normalmente identificadas pelo nome dos seus autores. Se usarmos a de Mateus e outros (Gramática da Língua Portuguesa, Mateus, M,H.M, A. M.Brito, I. Duarte, I. H Faria, Caminho Série Linguística, 1989), poderemos representar simbolicamente as regras de formação de uma frase (aqui feita para fins ilustrativos, e portanto muito simplificadas em relação à riqueza da nossa língua):

Regra: uma frase é composta por um sintagma nominal e um sintagma verbal.

Pode não ter sintagma nominal ou o sintagma verbal; mas tem que ter pelo menos um deles.

Simbolicamente escreve-se a **regra de produção**

$$\langle \text{frase} \rangle \rightarrow \langle \text{sintagma nominal} \rangle \langle \text{sintagma verbal} \rangle \qquad \text{(produção 1)}$$

O sintagma nominal é composto por um determinante e um nome (pelo menos um deles) ou pode ser vazio:

$$\langle \text{sintagma nominal} \rangle \rightarrow \langle \text{determinante} \rangle \langle \text{nome} \rangle \mid \langle \text{vazio} \rangle \qquad \text{(produções 2 e 3)}$$

O determinante pode ter um artigo ou um deíctico, ou nada:

$$\langle \text{determinante} \rangle \rightarrow \langle \text{artigo} \rangle \mid \langle \text{deícticos} \rangle \mid \langle \text{vazio} \rangle \qquad \text{(produções 4,5,6)}$$

O sintagma verbal é composto por um verbo e um sintagma nominal (pelo menos um deles):

⟨sintagma verbal⟩ → ⟨ verbo⟩ ⟨sintagma nominal⟩ (produção 7)

Para podermos formar (produzir) frases temos que ter palavras concretas e não apenas categorias de palavras. Elas, as palavras concretas, são os **elementos terminais** da gramática. Por exemplo:

⟨verbo⟩ → *estuda* | *ama* | *compra* | *dorme*|*chove* (produções 8 a 12)

⟨artigo⟩ → *o* | *a* |*um* | *uma*|<vazio> (produções 13 a 17)

⟨deícticos⟩ → *este* | *esse* | *aquele* | *meu* | *teu* | *seu* |<vazio> (produções 18 a 24)

⟨nome⟩ → *Luís* | *Antónia* | *Isabel* | *livro* | *gelado* | (produções 25 a 29)

Com as regras de produção e os elementos terminais dados, poderemos agora formar frases.

Por exemplo para produzir a frase

a Antónia compra aquele gelado

usam-se as regras de produção na ordem adequada para o fim em vista:

⟨frase⟩ ⇒ ⟨sintagma nominal⟩ ⟨sintagma verbal⟩ produção 1

⇒ ⟨determinante⟩ ⟨nome⟩ ⟨sintagma verbal⟩ produção 2

⇒ ⟨determinante⟩ ⟨nome⟩ ⟨ verbo⟩ ⟨sintagma nominal⟩ produção 7

⇒ ⟨determinante⟩ ⟨nome⟩ ⟨ verbo⟩ ⟨determinante⟩ ⟨nome⟩ produção 2

⇒ ⟨determinante⟩ ⟨nome⟩ ⟨ verbo⟩ ⟨deíctico⟩ ⟨nome⟩ produção 5

⇒ *a* ⟨nome⟩ ⟨ verbo⟩ ⟨deíctico⟩ ⟨nome⟩ produção 13

⇒ *a Antónia* ⟨ verbo⟩ ⟨deíctico⟩ ⟨nome⟩ produção 26

⇒ *a Antónia compra* ⟨deíctico⟩ ⟨nome⟩ produção 10

⇒ *a Antónia compra aquele* ⟨nome⟩ produção 20

⇒ *a Antónia compra aquele gelado* produção 29

Verifique o leitor se as seguintes frases obedecem à gramática dada:

(i) O João foi ao cinema

(ii) Chove

(iii) A Isabel ama o Luís

Este exemplo mostra como uma frase (ou oração), conceito geral e complexo, pode ser definida à custa de elementos simples (decomposição da complexidade). Começa-se no conceito mais complexo (frase), e reduz-se sucessivamente até se obterem os elementos irredutíveis com que se constrói a língua.

A generalização deste princípio leva-nos às gramáticas formais isto é, gramáticas das linguagens formais, as linguagens dos computadores.

As linguagens formais são construções matemáticas (conjuntos) que obedecem a certas regras. Para que não haja equívocos, as gramáticas formais devem ter uma estrutura clara, coerente, completa. Por isso usam-se notação e conceitos matemáticos precisos para a sua definição e manipulação.

Definição 1.3.1. Gramática.

Uma gramática *G* é definida por um quarteto

$$G=(V, T, S, P)$$

em que

V : **variáveis,** conjunto não vazio finito de objectos,

T: **símbolos terminais,** conjunto não vazio finito de objectos,

$S \in V$: variável de **inicialização,** um símbolo especial, também chamado axioma,

P : um conjunto finito de **produções**

As variáveis não podem ser símbolos terminais e vice-versa, isto é V e T são conjuntos disjuntos. No nosso estudo são em geral caracteres.

As regras de produção constituem o cerne da gramática, pois é através delas que uma cadeia de caracteres se transforma noutra cadeia de caracteres. Por isso elas definem uma linguagem associada à gramática.

Gramáticas diferentes podem produzir a mesma linguagem. Nesse caso são gramáticas equivalentes. Mais à frente estudaremos técnicas e algoritmos para transformar uma gramática numa outra sua equivalente.

Embora uma linguagem possa ser produzida por várias gramáticas, uma gramática produz uma e uma só linguagem. Teremos a relação ilustrada na figura 1.3.1.

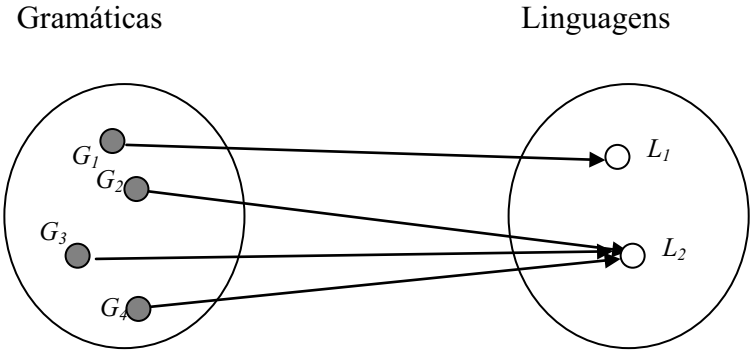

Figura 1.3.1 . Várias gramáticas podem produzir a mesma linguagem. Mas uma gramática não pode produzir duas linguagens.

As regras de produção de uma qualquer gramática têm sempre a forma padrão

$$x \rightarrow y$$

que se lê " (a cadeia) x produz (a cadeia) y" .

A cadeia produtora x tem que ter qualquer coisa, não pode ser a cadeia vazia (se assim não fosse produzir-se-ia a partir do nada, o que não teria significado). Este facto anota-se por

$$x \in (V \cup T)^+$$

querendo-se dizer que a cadeia x pode ter variáveis da gramática e/ou símbolos terminais, **sem** a cadeia vazia (por isso se escreve $^+$ e não *).

A cadeia produzida *y,* chamada o **corpo** da produção, pode conter variáveis da gramática, e/ou símbolos terminais, podendo ser a cadeia vazia. Neste caso quer dizer que a produção consiste simplesmente na anulação de *x*. Anota-se assim por

$$y \in (V \cup T)^*$$

sendo agora legítimo escrever (*) para admitir a possibilidade da cadeia vazia. De facto $(V \cup T)^0$ resulta na cadeia vazia.

A produção substitui, numa forma sentencial, a cadeia *x* pela cadeia *y*. Isto é,

dada uma cadeia $w = uxv$

e a regra de produção $x \rightarrow y$,

_____

por substituição de x por y em w, obtém-se a cadeia $z = uyv$

Diz-se que *w* produz z , ou que *z* é produzida por *w* e anota-se com seta larga

$$w \Rightarrow z$$

w e *z* podem variáveis e/ou símbolos terminsais, dependendo do caso, como veremos.

As regras de produção podem aplicar-se por qualquer ordem e tantas vezes quantas as necessárias. Este facto permite que com um número finito de produções se possa obter uma linguagem infinita. Duas produções e um símbolo inicial são suficientes para produzir uma linguagem infinita. Por exemplo as duas produções seguintes

$$S \rightarrow aS$$

$$S \rightarrow \lambda$$

geram qualquer cadeia no alfabeto $\Sigma=\{a\}$, incluindo a cadeia vazia. Experimente o leitor gerar a cadeia *aaa*.

Será possível obter, com uma só produção, uma linguagem infinita? Só em certas gramáticas especiais que estudaremos no Capítulo 12.

Entre uma forma sentencial inicial w_1 e uma final w_n poderemos passar por um grande número de formas sentencias intermédias, $w_1, w_2, \ldots$, ou seja,

$$w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$$

Diz-se que w_1 produz w_n ou, em escrita mais compacta (mais prática), usando (*) para significar uma sequência de produções simples,

$$w_1 \overset{*}{\Rightarrow} w_n$$

(*) exprime o facto de que para derivar w_n a partir de w_1 são necessários um número não especificado de passos (incluindo eventualmente zero passos, como em $w_1 \overset{*}{\Rightarrow} w_1$).

As regras de produção podem ser aplicadas numa ordem qualquer. Por cada ordem, produz-se uma cadeia terminal. Ordens de derivação diferentes podem dar cadeias terminais iguais ou diferentes, conforme a gramática concreta.

O conjunto de todas as cadeias terminais que se podem obter por uma gramática, compõem a **linguagem gerada por essa gramática**. Exprime-se pela expressão $L(G)$, linguagem L da gramática G. Em termos mais formais, dada a gramática definida pelo quarteto

$$G=(V, T, S, P)$$

a linguagem da gramática (ou gerada pela gramática) é composta por todas as cadeias terminais (sentenças, apenas com símbolos de T, ou eventualmente λ) que se podem gerar a partir de S com as produções de P, por qualquer ordem qualquer número de vezes, ou seja,

$$L(G) = \{w \in T^*: S \overset{*}{\Rightarrow} w\}$$

Se uma dada cadeia pertence à linguagem, $w \in L(G)$, então a sequência

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \ ... \ \Rightarrow w_n \Rightarrow w$$

é uma **derivação da cadeia** *w*. A cadeia *w* deve ter apenas símbolos terminais caracteres do alfabeto da linguagem, incluindo eventualmente λ.

Qualquer derivação da gramática começa sempre por *S*, que por isso é também chamado o **axioma** da gramática. Desde *S* até *w* passa-se por diversas expressões que têm variáveis da gramática ou uma mistura de variáveis e de símbolos terminais. A essas expressões chama-se **formas sentenciais** da derivação: são as cadeias $S, w_1, w_2, ..., w_n$.

Exemplo 1.3.2.

Seja a gramática definida pelo quarteto

$$G = (\{S\}, \{a,b\}, S, P)$$

Comparando com a definição 1, identificam-se os elementos do quarteto:

- Conjunto *V* das variáveis*: S*
- Símbolos terminais: *a,b* , as cadeias terminais terão apenas estes símbolos
- Variável de inicialização: *S*
- As produções *P* dadas por

  P1: $S \rightarrow aSb$: partindo do símbolo inicial *S,* coloca-se-lhe um *a* antes e um *b* depois.

  P2: $S \rightarrow \lambda$: o símbolo *S* pode ser substituído pela cadeia vazia (isto é, por nada).

Uma derivação pode seguir o seguinte percurso:

$S \Rightarrow aSb$	aplicando a	produção P1
$\Rightarrow aaSbb$		produção P1
$\Rightarrow aaaSbbb$		produção P1

$\Rightarrow \ldots$

pode-se continuar um número qualquer de vezes, até ao infinito. Vão-se obtendo sucessivas formas sentenciais.

Em qualquer altura se pode aplicar a produção P2, substituindo *S* por λ, ou seja por nada. Nessa altura obtém-se uma frase da linguagem. Assim,

ao fim da 1ª produção obtém-se *ab*

ao fim da 2ª produção obtém-se *aabb*

etc.,

ao fim da *n*èsima produção obtémse $aaa....abbb...b = a^n b^n$

Se aplicarmos primeiro a produção P2 obtém-se a cadeia vazia e não se pode prosseguir.

Conclui-se assim que esta gramática produz a linguagem composta por cadeias em que a primeira metade só tem *a*'s e a segunda metade só tem *b*'s, ou seja, de uma forma matematicamente elegante, usando as noções que vimos anteriormente,

$$L(G) = \{a^n b^n,\ n \geq 0\}$$

Este resultado é tão evidente que se pode dizer que não carece de demonstração. Mas pode-se demonstrar formalmente, usando uma das técnicas de prova conhecidas. Como se trata de demonstrar uma fórmula para um número infinito de casos, o mais natural será usar a prova por indução, associada à noção de recursividade.

O caso λ , correspondente a $n = 0$ tem que ser tratado à parte. Demonstra-se que ele se obtém aplicando a produção P_2 a partir de *S*.

Para os outros casos façamos a demonstração por recursividade da produção P1 e indução.

Queremos provar que qualquer que seja $n \geq 1$, todas as formas sentenciais são da forma

$$w_n = a^n S b^n$$

- **base** da indução, *n=1,* é verdade ?

sim, resultante de uma vez P1 a partir de S.

-**hipótese indutiva**: admitamos que é verdade para *n=k* e para todos os anteriores, ou seja, é verdade que

$$w_k = a^k S b^k$$

-passo indutivo: em consequência é verdade para *k+1* ou seja

$$w_{k+1} = a^{k+1} S b^{k+1}$$

Prova do passo indutivo (nesta prova há-de entrar a hipótese indutiva):

Partindo de w_k e por aplicação de P_1 obtém-se w_{k+1}

$$w_{k+1} \rightarrow a w_k b$$

mas substituindo a hipótese indutiva nesta produção,

$$w_{k+1} \rightarrow a(a^k S b^k) b$$

Manipulando a expressão

$$w_{k+1} \rightarrow (a a^k) S (b^k b)$$

ou ainda por definição de concatenação (potência) de caracteres,

$$w_{k+1} \rightarrow a^{k+1} S b^{k+1}$$

Falta finalmente provar que aplicando P_2 a qualquer forma sentencial na forma $w_n = a^n S b^n$, $n \geq 1$, se obtém uma sentença na forma $a^n b^n$.

$$a^n S b^n \rightarrow a^n b^n$$

o que é verdade.

E a demonstração está feita, *quod erat demonstrandum* (o que era preciso demonstrar) *q.e.d.*

Exemplo 1.3.3:

Encontrar uma gramática que gere a linguagem

$$L=\{a^{n+1}b^n,\ n\geq 0\}$$

Esta linguagem é parecida com a do exemplo anterior, com a diferença de que é necessário gerar um *a* adicional à esquerda de *b*. Pode-se simplesmente gerá-lo em primeiro lugar, aplicar depois regras de produção como no caso anterior:

P1: $S \Rightarrow aS$

P2: $S \rightarrow aSb$

P3: $S \rightarrow \lambda$

Teríamos assim a gramática definida exactamente como a anterior:

$$G = (\{S\},\ \{a,b\},\ S,\ P)$$

Como as produções de podem aplicar por ordem arbitrária e um número arbitrário de veses, aplicando P1, P1, P2, P3, obtém-se

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaSb \Rightarrow aaab$$

i.e. a^3b, que não faz parte da linguagem.

Para que isto não aconteça, P1 só se pode aplicar uma vez. Introduz-se com esse objectivo uma variável adicional, *A*, e definem-se as produções

P1: $S \Rightarrow aA$

P2: $A \rightarrow aAb$

P3: $A \rightarrow \lambda$

em que S é a variável de inicialização. Note-se que partindo de S não se pode voltar a ele, e por isso P1 só se aplica uma vez, sendo sempre a primeira. Poderemos agora definir a gramática *G* com o quarteto

$$G=(\{A\}, \{a,b\}, S, P)$$

sendo P composto pelas três produções P1, P2 e P3.

Para demonstrar que uma dada linguagem é gerada por uma certa gramática, tem que se mostrar que:

(i) por um lado toda a cadeia $w \in L$ pode ser derivada a partir de S usando as produções P da gramática ou seja

(ii) por outro lado qualquer cadeia gerada pela gramática G pertence à linguagem L.

Nem sempre é fácil fazer as duas demonstrações, sobretudo a primeira. Ver por exemplo em Linz, exemplo 1.13. Recorre-se frequentemente à prova por indução.

Uma linguagem pode ser gerada por muitas gramáticas, que por isso se dizem equivalentes ,

G_1 e G_2 são gramáticas equivalentes se $L(G_1)=L(G_2)$

Na Fig. 1.1 as gramáticas G_2, G_3 e G_4 são equivalentes.

Os palíndromos são cadeias que têm interesse especial para o estudo de gramáticas e autómatos. Vejamos mais em detalhe como se podem definir e gerar.

Definição formal de palíndromos (PAL) sobre um alfabeto Σ

Seja PAL a linguagem constituída por todos os palíndromos (capicuas) sobre um dado alfabeto. Pode-se definir pelas três regras seguintes

1. O carácter vazio é simétrico, logo pertence aos palíndromos.

$$\lambda \in \text{PAL}$$

2. Qualquer carácter isolado é simétrico, logo é um palíndromo

$$\text{Para todo o } a \in \Sigma,\ a \in \text{PAL},$$

3. Se tivermos uma cadeia simétrica e lhe adicionarmos o mesmo carácter no princípio e no fim, resulta uma cadeia simétrica,.

Para toda a cadeia $w \in$ PAL , e todo o $a \in \Sigma$, $awa \in$ PAL

Uma cadeia só pertence a PAL se puder ser gerada por estas 3 regras. Por outro lado se uma cadeia é um palíndromo, então ela pode ser gerada por aquelas três regras, aplicando-as as vezes necessárias pela ordem conveniente.

Exemplo 1.3.4

Seja o alfabeto $\Sigma = \{ a, b \}$

Aplicando aquelas 3 regras,

1. $\lambda \in$ PAL

2. $a \in$ PAL, $b \in$ PAL

3. Para qualquer $w \in$ PAL, $awa \in$ PAL

$bwb \in$ PAL

Qualquer cadeia pertence a PAL se e só se for gerada pelas regras 1, 2 e 3.

Por exemplo:

$b \in$ PAL ,	regra 2
$aba \in$ PAL,	regra 3
$babab \in$ PAL	regra 3
$abababa \in$ PAL	regra 3

Como se poderá escrever uma gramática para esta linguagem?

Poderemos tentar imitar as três regras: primeiro gera-se um palíndromo com a regra 1 ou a 2,e depois geram-se sucessivos palíndromo, cada vez maiores, com a regra 3.

Deste modo uma gramática para esta linguagem pode ser obtida pelas derivações seguintes:

1 P1: $S \rightarrow \lambda$ (regra 1)

2 P2: $S \rightarrow a$ (regra 2)

P3: S $\rightarrow b$ (regra 2)

3 P4: S $\rightarrow aSa$ (regra 3)

P5: S $\rightarrow bSb$ (regra 3)

Por exemplo para gerar *abba* teremos a sequência de produções

$$S \rightarrow aSa \rightarrow abSba \rightarrow ab\lambda ba \rightarrow abba$$

Pode-se escrever as 5 produções gramática na forma compacta, usando "|" para o "ou" lógico

$$S \rightarrow \lambda \mid a \mid b \mid$$

$$S \rightarrow aSa \mid bSb$$

1.4.Autómatos

Os autómatos finitos são modelos abstractos de muitos dispositivos de hardware e de software. Aplicam-se nomeadamente em (Hopcroft& col)

1- software para o projecto e o teste de circuitos digitais.

2- Analisadores lexicais dos compiladores (que verificam se uma palavra, por exemplo um identificador, está bem escrita num programa).

3- Software para busca de texto em ficheiros, na web, etc.

4- Software de verificação de sistemas que têm um número finito de **estados** distintos, tais como protocolos de comunicação, protocolos de segurança, etc.

Muitos componentes e sistemas digitais podem estar num entre um número finito de estados, e por isso a noção de estado é central num autómato a eles associado. O estado permite "lembrar" o passado relevante do sistema (ele chegou lá após um certo número de transições, que são o seu passado). Se o número de estados de um sistema é finito, poderemos representá-lo com uma quantidade limitada de recursos.

Considere-se por exemplo um interruptor *on-off.* Se está *off* e se se pressiona, passa a *on*. Se está *on* e se se pressiona, passa a *off* . Ele terá por isso dois estados, o *on* (A) e o *off* (F) Desenhando, teremos os dois estados como dois vértices de um grafo, Fig.1.3.1.

Figura 1.4.1. Os dois estados do autómato que representa o interruptor *on-off.* F-fechado, A-aberto.

Representando por arestas de um grafo as acções de pressionar ("*Press"*) teremos a Fig. 1.4.2

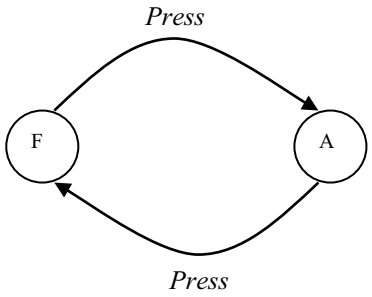

Figura 1.4.2. As transições entre os estados do autómato, provocadas pelo accionamento do interruptor.

Para se saber como se inicializou o interruptor, identifica-se o estado inicial por uma seta,

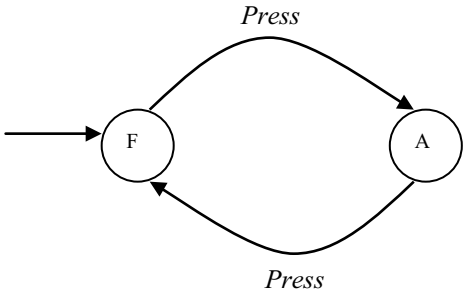

Figura 1.4.3. O autómato finito do interruptor.

e tem-se finalmente o desenho de um autómato finito (de dois estados) que representa o funcionamento do interruptor, na Fig. 1.4.3.

As etiquetas das arestas representam a acção exterior sobre o sistema (neste caso pressionar o interruptor) e as arestas a evolução do sistema em consequência dessa acção, que é sempre uma transição entre dois estados (ou, como veremos, e generalizando a noção de transição, de um estado para ele próprio).

Considere-se agora o autómato da figura 1.4.4.

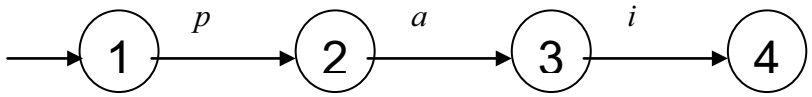

Figura 1.4.4. Um autómato finito que transita de estado pela entrada de letras.

Inicializa-se no estado 1, recebe do exterior (lê) a letra *p* e passa ao estão 2, depois com as letras *a* e *i* passa sucessivamente aos estados três e 4.

Poderíamos nomear os estados de um modo mais sugestivo, tal que a sua etiqueta nos sintetizasse o que se passou até aí. Teríamos por exemplo a Fig. 1.4.5.

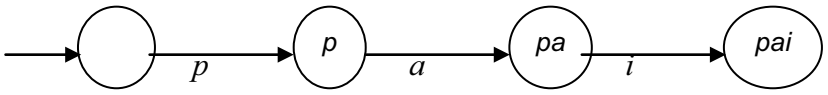

Figura 1.4.5. O autómato anterior da Fig 1.4.4 com nova etiquetagem dos estados.mais sugestiva.

Pode afirmar-se que o autómato de 4 estados é capaz de reconhecer a palavra *pai*: se chegou ao estado 4, ela pareceu-lhe, caso contrário nunca chega ao estado 4. Por esta razão diz-se que o autómato é um **aceitador** da palavra *pai*. Para assinalar este facto o estado 4 desenha-se com uma dupla circunferência, como na figura 1.4.6.

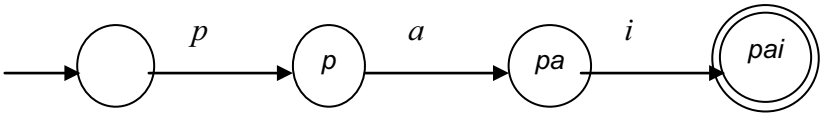

Figura 1.4.6. O autómato com estado final aceitador

E chama-se estado **final,** ou estado **aceitador**, conforme os autores. O primeiro chama-se estado **inicial**.

Vejamos um exemplo um pouco mais complexo.

Exemplo 1.4.3

Uma porta automática de entrada de um serviço público, rotativa, tem geralmente dois sensores, um frontal, e um na retaguarda, conforme esquematizado na Fig. 1.4.7.. O frontal detecta uma pessoa que se aproxima e o da retaguarda detecta a presença de uma pessoa na sua área de observação.

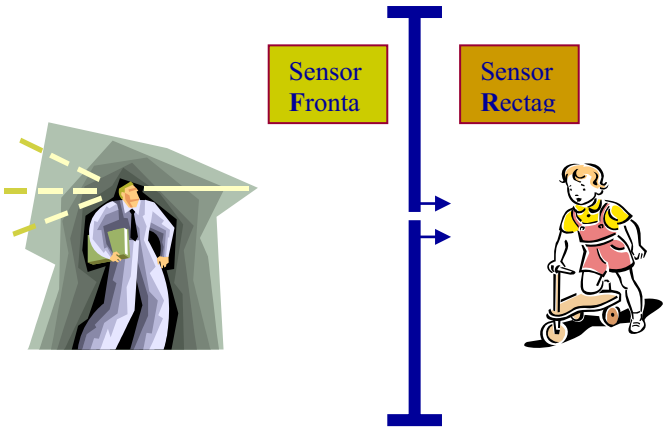

Figura 1.4.7. Uma porta de abertura automática, rotativa.

Quanto uma pessoa se aproxima para entrar a porta abre-se, girando sobre o seu eixo de fixação, se não estivar ninguém detrás da porta, caso contrário haveria um acidente. Há um dispositivo electrónico, o **controlador**, que recebe os sinais dos dois sensores e conforme o caso manda abrir, fechar, ou manter-se como está. O controlador terá dois estados, correspondentes a porta aberta, A, e porta fechada, F. Tal como no exemplo anterior, poderemos desenhar dois vértices de um gráfico para esses dois estados.

Figura 1.4.8. Os dois estado do autómato abridor da porta.

Os dois sensores enviam, cada um, informação de presença ou ausência de pessoas, que poderemos representar por 0 (ausência) e 1 (presença).Teremos quatro situações possíveis, considerando simultaneamente os dois sensores:

00 : ausência de frente e detrás

01: ausência de frente e presença detrás

10: presença de frente e ausência detrás

11: presença de frente e presença detrás.

Em cada uma destas quatro situações possíveis o controlador tem que tomar uma decisão: fecha, abre ou deixa ficar como está ?

Poderemos constatar que o alfabeto do autómato (o conjunto de informação externa que ele sabe ler) é composto por aquelas quatro situações, que poderemos associar a 4 caracteres de um alfabeto, conforme a tabela

Tabela 1.4.1. Codificação do alfabeto ao autómato. PF- presença frontal, PR – presença na rectaguarda.

PF	PR		Carácter
0	**0**	**00**	***a***
0	**1**	**01**	***b***
1	**0**	**10**	***c***
1	**1**	**11**	***d***

Considerando as restrições de segurança, teremos as transições possíveis representadas na tabela (de transições) seguinte:

Tabela 1.4.2. Transições entre os estados em função dos caracteres de entrada lidos.

Entrada → Estado ↓	**a 00**	**b 01**	**c 10**	**d 11**
F	**F**	**F**	**A**	**F**
A	**F**	**A**	**A**	**A**

Agora é fácil desenhar o grafo do autómato, com os dois nós correspondentes aos estados e as arestas correspondentes à 8 transições da tabela. Obtém-se a figura seguinte.

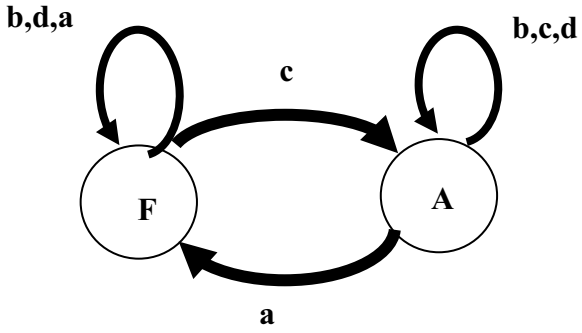

Figura 1.4.9. O autómato finito que representa o sistema de controlo de abertura da porta .

Como é que o controlador fecha e abre a porta? Enviando um sinal a um actuador mecânico. Poderemos assim considerar que o autómato tem uma **saída** que pode ter dois valores: 1 para abrir, 0 para fechar. Poderemos introduzir essa informação colocando sob a letra do estado o valor lógico correspondente da saída, como na figura seguinte

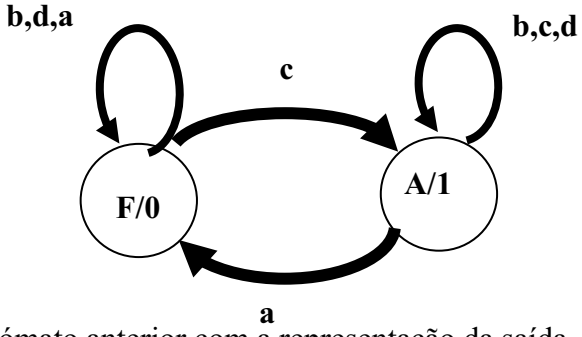

Figura 1.4.10. O autómato anterior com a representação da saída. Veremos no Cap. 2 que este autómato se chama por isso de Máquina de Moore.

Depois deste dois exemplos muito simples, estamos em condições de definir um autómato mais detalhadamente.

Um autómato genérico tem os seguintes componentes:

- um **ficheiro de entrada**, composto por uma cadeia num alfabeto

- um mecanismo para **leitura** do ficheiro de entrada de modo que

    lê a cadeia de entrada da esquerda para a direita, um carácter de cada vez,

    detecta o fim da cadeia

- uma **saída,** onde a unidade de controlo pode escrever

- um dispositivo de **memória temporária**

  composto por um número ilimitado de células, cada uma capaz de armazenar um símbolo de um alfabeto (que pode ser diferente do alfabeto de entrada). O autómato pode ler e alterar o conteúdo dos registos de memória.

- uma **unidade de controlo** que

  pode estar num de entre um certo número finito de **estados internos,**

  pode mudar de estado segundo alguma regra.

Juntando tudo teremos a Fig. 1.4.11.

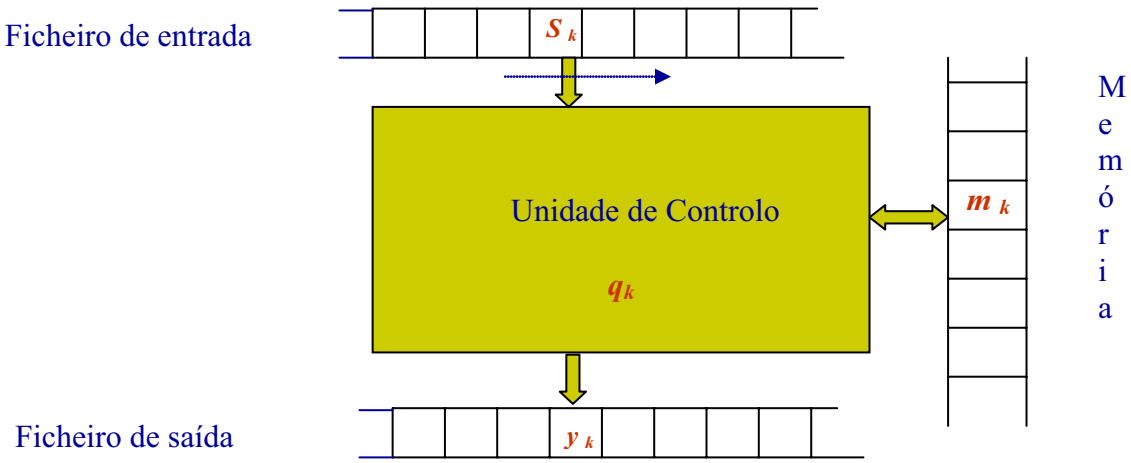

Figura 1.4.11. Representação genérica de um autómato finito.

O que lhe dá a característica de finito é tão só o número de estados da unidade de controlo: ele é finito.

Esta figura é muito geral. Um autómato concreto pode não ter memória (como os exemplos que vimos), ou pode não ter saída (como o aceitador de *pai*). Mas todos têm a unidade de controlo e a entrada.

O autómato funciona em tempo discreto, sequencialmente. Num dado instante k está num certo estado interno, ε_k, e o mecanismo de entrada está a ler um símbolo s_k no ficheiro de

entrada. No próximo instante de tempo, o estado interno do autómato vai ser determinado pelo estado actual, pelo símbolo lido à entrada e pelo conteúdo da memória, m_k. Essa dependência é definida pela **função de transição de estado *f*,** que depende por isso do estado actual, da entrada actual e do conteúdo actual da memória, ou seja,

$$q_{k+1} = f(q_k, s_k, m_k)$$

Entre dois instantes do tempo sucessivos pode produzir-se uma saída ou pode alterar-se o conteúdo da memória.

Chama-se **configuração do autómato** ao **conjunto** composto por

- o estado interno da unidade de controlo,

- o ficheiro de entrada e

- o conteúdo da memória.

Chama-se um **passo** à transição do autómato de uma configuração para a configuração seguinte.

Todos ao autómatos que estudaremos se enquadram nesta descrição genérica. Os diversos tipos que estudaremos distinguem-se por

i) a forma de produzir a saída , sendo

aceitadores se reconhecem ou não a cadeia de entrada; não têm saída informam pelo estado final em que terminam a leitura da entrada

transdutores se produzem cadeias de caracteres na saída com alguma utilidade

ii) a natureza da memória temporária (o factor mais importante)sendo

autómatos finitos se não sem têm memória temporário ou

autómatos de pilha se têm uma memória em pilha LIFO (em inglês *pushdown automata*, de empurrar (a pilha) para baixo),

máquinas de Turing se têm a memória em fita, de dimensão infinita, que pode ser lida e alterada em qualquer ordem

iii) a natureza da função de transição, dizendo-se

determinísticos se dada uma configuração, existe um e um só comportamento futuro possível ou

não determinísticos se dada uma configuração, o próximo passo pode ter várias alternativas.

1.5. Os três paradigmas da computação

Quando falamos em computação, o que queremos dizer exactamente ?

Em primeiro lugar computação poder ser o **cálculo do valor numérico de uma função**, como por exemplo da função *f*(*n*), que calcula o *n*-èsino número primo,

$$f(n) = n - ésimo - número - primo$$

trata-se de uma função **unária,** ou da função *g,* produto de dois números,

$$g(n,m) = n \times m$$

função **binária**, de dois argumentos *m* e *n* ou ainda da função *h*, soma de *n* números,

$$h = a_1 + a_2 + ... + a_n$$

função ***n*-ária** (de *n* argumentos).

Este tipo de computação é mais antigo do que os computadores digitais, é aliás tão antigo quanto a civilização humana, dado que desde muito cedo o homem precisou de fazer cálculos matemáticos (para calcular o calendário, por exemplo).

Na era do computador digital há outros tipos de computação. Por exemplo, poderemos ter uma cadeia de 0's e 1's e calcular a sua paridade, ou o número de zeros. Ou então queremos saber se uma cadeia é um palíndromo, ou se a palavra "*public*" está bem escrita num programa de computador em Java.

Este tipo de computação parece nada ter ver, à partida, com a computação do valor de funções. Trata-se do reconhecimento, ou classificação, de cadeias de símbolos. Como as cadeias de símbolos formam linguagens, este tipo de computação chama-se por isso **reconhecimento de linguagens,** o segundo paradigma da computação.

O problema de reconhecimento de linguagens domina a teoria da computação. Falaremos dele repetidamente até ao final da disciplina e ele ocupa a maior parte dos livros de teoria da computação.

Porque é assim tão importante?

Em primeiro lugar porque é muito representativo: pode-se reduzir qualquer problema de decisão a um problema de reconhecimento de linguagens.

Em segundo lugar porque o problema do cálculo do valor de uma função pode também ser reduzido a um problema de reconhecimento de uma linguagem.

Comecemos pela verificação da primeira razão.

Considere-se o problema de decisão seguinte

- o número natural *n* é primo ?

Trata-se de um problema de decisão: decidir se é ou não, mas decidir de modo fundamentado.

Construa-se no alfabeto $\Sigma=\{1\}$ a linguagem composta por cadeias de 1's que têm um comprimento igual a um número primo:

$$L=\{1^n, n \text{ primo}\},$$

em que como sabemos $1^n = 111\ldots1$, n vezes

Isto é

$$L=\{11, 111, 11111, 1111111, \ldots\}$$

Agora temos o problema de reconhecimento de linguagens

- a cadeia 1^n pertence à linguagem *L* ?

equivalente ao problema de decisão acima enunciado.

Além do cálculo do valor de uma função ou do reconhecimento de uma linguagem temos outro tipo de computação: **o da transdução de cadeias**, ou seja, a transformação de uma cadeia de símbolos por exemplo revertendo-a, deslocando-a para a esquerda ou para a direita, contando o número de *a*'s, etc. Temos aqui o terceiro paradigma da computação.

De entre os três paradigmas da computação, qual o mais importante?

De facto nenhum é mais importante do que o outro. Verifica-se até o princípio da inter-redutibilidade (Fig.1.5.1): qualquer instância de um dos três paradigmas de pode reduzir a uma instância de qualquer um dos outros dois.

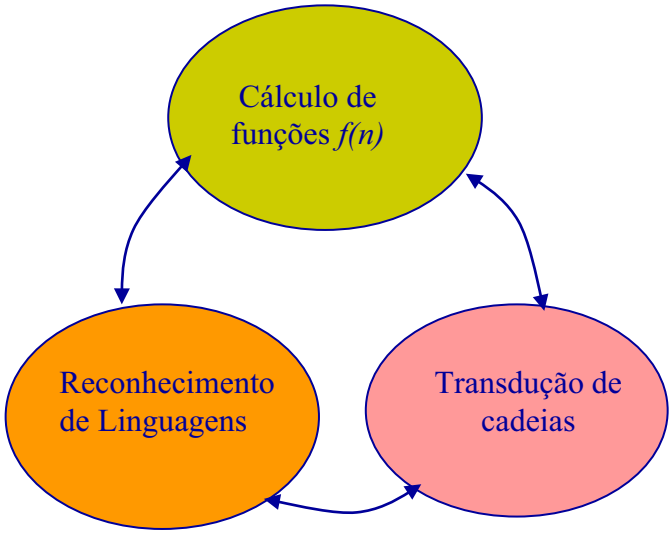

Figura 1.5.1. Inter-redutibilidade entre os três paradigmas da computação.

Vejamos como.

i) Qualquer caso de computação de uma função pode ser reduzido a uma instância do reconhecimento de uma linguagem.

Exemplo 1.5.1. Seja a função

$$f(n) = m\ , \quad m \text{ é o } n\text{-ésimo número primo}$$

Tabela 1.5.1. Função *n*-ésimo número primo

n	1	2	3	4	5	6	…
$f(n)$	2	3	5	7	11	13	…

Suponhamos que queremos calcular

$$f(5) = ??$$

Para reduzir este cálculo a um problema de reconhecimento de linguagens faça-se a construção engenhosa seguinte (inspirado em Taylor):

1º Usando a representação binária de números naturais, constroem-se as cadeias que resultam da concatenação de 101, isto é 5 em binário, com os números naturais (1, 2, 3, 4, 5, 6, …), usando por exemplo # como separador:

101#1
101#10
101#11
101#100
101#101
101#110
101#111
…

2º Define-se agora a linguagem L associada aos números primos

$$L = \{\text{Binário}(n)\#\text{Binário}(m)\}$$

(não esquecer que m é o n-ésio número primo).

A computação do valor de $f(5)$ é equivalente à determinação de qual a única cadeia daquele conjunto que pertence a L . De facto há-de lá estar, naquele conjunto de cadeias,

101#1011

Assim, procurando-a, ficamos a saber que

$$f(5)=11$$

ii) Redução de uma classificação de cadeias a uma computação numérica de uma função

Tomemos o caso do reconhecimento de palíndromos.

Em primeiro lugar faz-se uma codificação dos símbolos do alfabeto, de modo que cada símbolo fique associado a um e um só número natural em binário.

Por exemplo, em $\Sigma=\{a,b\}$, se $a = 01$ e $b = 10$, então o palíndromo *aba* resulta no número natural $011001=25_{10}$.

Em segundo lugar constrói-se a linguagem dos palíndromos, $L(Pal)$,.

$$L(Pal)= \{a,b,aa,bb,aaa,\ aba,bab,\ bbb, \ldots$$

e os seus códigos,

$$\text{Códigos}= \{01,10,0101,1010,010101,\ 011001,100110,101010, \ldots\}$$

A função característica de L será

$$\chi(n) = \begin{cases} 1, \text{se } n \text{ é o código de um palíndromo} \\ 0, \text{ se } n \text{ não o é} \end{cases}$$

Agora para se saber se 25 é o código de um palíndromo, basta calcular o valor da função característica $\chi(25)$ que é 1, e assim se conclui que *aba* é um palíndromo.

Claro que para cada caso é necessário desenvolver o engenho para encontrar a forma de implementar o princípio da inter-redutibilidade. O que importa saber é que é sempre possível encontrar uma solução.

Mas este princípio é muito importante: ele permite que se use o problema da identificação de linguagens em teoria da computação como representativo dos três paradigmas. Por isso é o que usaremos ao longo da cadeira.

Bibliografia

An Introduction to Formal Languages and Automata, Peter Linz, 3rd Ed., Jones and Bartelett Computer Science, 2001

Models of Computation and Formal Languages, R. Gregory Taylor, Oxford University Press, 1998.

Introduction to Automata Theory, Languages and Computation, 2nd Ed., John Hopcroft, Rajeev Motwani, Jeffrey Ullman, Addison Wesley, 2001.

Elements for the Theory of Computation, Harry Lewis and Christos Papadimitriou, 2nd Ed., Prentice Hall, 1998.

Introduction to the Theory of Computation, Michael Sipser, PWS Publishing Co, 1997.

Anexo 1. Tabela de números primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081 2083 2087 2089 2099 2111 2113 2129 2131 2137 2141 2143 2153 2161 2179 2203 2207 2213 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297 2309 2311 2333 2339 2341 2347 2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417 2423 2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 2557 2579 2591 2593 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797 2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903 2909 …. … …