Treball Fi de Grau - Curs 2023 / 2024

El teorema de Cook-Levin

Autora: **Alba Soler Moreno**

Tutor: Enric Cosme Llópez

Índex

Introducció

Què determina que alguns problemes siguen computacionalment difícils mentre que d'altres siguen fàcils? Com podem trobar en el llibre de Michael Sipser, [Sip13], aquesta és la pregunta central de la teoria de la complexitat per a la que, tot i que s'ha estat investigant durant més de 40 anys, no coneixem resposta. L'objectiu de la teoria de la complexitat és classificar els problemes com a fàcils o difícils, mentre que la teoria de la computabilitat classifica els problemes entre aquells que són solucionables i els que no ho són.

Durant la primera meitat del segle XX, alguns matemàtics com K. Gödel, A. Turing i A. Church, van descobrir que certs problemes no poden ser resolts per ordinadors. Tal com podem trobar al llibre [HMU07], la teoria d'autòmats és l'estudi de models matemàtics de computació. En la década dels 30, anterior a l'existència dels ordinadors, Alan Turing va estudiar una màquina abstracta que, pel que fa a allò que podia calcular, tenia totes les capacitats dels ordinadors actuals. Turing pretenia definir els límits de les capacitats d'una màquina de computació, distingint el que era possible fer amb ella i el que no. Durant les següents dos décades, nombrosos investigadors van estudiar tipus de màquines més simples, que en l'actualitat anomenem "autòmats finits". En els anys 60, S. Cook va extendre l'estudi de Turing i va ser capaç de separar aquells problemes que es poden resoldre eficientment per ordinador dels problemes que en principi es poden resoldre, però que necessiten tant de temps que els ordinadors són inútils excepte per a instàncies molt menudes del problema.

L'objectiu d'aquest treball és arribar a entendre l'enunciat i la demostració del Teorema de Cook-Levin, un resultat fonamental en el camp de la teoria de la complexitat computacional. El teorema de Cook-Levin demostra que el problema de satisfacibilitat booleana, és a dir, el problema de determinar si una fórmula proposicional admet una valoració en les variables que la fa vertadera, és un problema NP-complet. Per a fer-ho farem un estudi previ d'alguns conceptes i resultats bàsics de la computació i la teoria de la complexitat. Començarem al Capítol 1 presentant una serie de definicions bàsiques necessàries per a la comprensió de conceptes posteriors. Definirem, entre altres, els conceptes de paraula sobre un alfabet, concatenació de paraules i monoide lliure sobre un conjunt, i enunciarem i demostrarem la propietat universal del monoide lliure.

Seguirem al segon capítol explorant els models de computació clàssics. Introduirem els autòmats finits com a models matemàtics per a reconèixer llenguatges i presentarem les màquines de Turing. Veurem quan un llenguatge és decidible o reconeixible per una màquina de Turing i demostrarem l'existència de llenguatges que no són reconeixibles. Per finalitzar aquest capítol, introduirem el conepte de màquina de Turing no determinista, que ens permetrà explorar la computació des d'una perspectiva no determinista.

Continuarem al tercer capítol tractant conceptes clau de la complexitat computacional. Començarem aquest capítol presentant la reducció funcional, una eina fonamental en teoria computacional que ens permet relacionar la complexitat de diferents llenguatges mitjançant funcions computables. La reducció funcional ens permetrà transferir propietats de decidibilitat

d'un problema a un altre. Seguirem definint la dominància asimptòtica (notació $\mathcal{O}$) per mesurar la complexitat temporal de les màquines de Turing. Continuarem estudiant les classes de complexitat $\mathbf{P}$ i $\mathbf{NP}$, amb exemples de llenguatges que hi pertanyen. Finalitzarem el capítol abordant la $\mathbf{NP-completesa}$, explicant el concepte de reducció polinomial entre llenguatges. Explorarem alguns resultats que relacionaran les classes $\mathbf{P}$ i $\mathbf{NP}$ a través de la reducció polinomial i la $\mathbf{NP-completesa}$, i veurem un resultat que estableix una connexió entre la classe $\mathbf{NP}$ i les màquines de Turing no deterministes.

Al quart i últim capítol presentarem, finalment, el teorema de Cook-Levin, que ens diu que el problema de la satisfacibilitat booleana és $\mathbf{NP-complet}$. Començarem el capítol fent una breu introducció a la lògica proposicional, en la que presentarem els conceptes bàsics de Δ-àlgebres, que inclouen les operacions de negació, conjunció i disjunció, i definirem les valoracions, que són aplicacions que assignen valors de veritat a les variables proposicionals. Continuarem el capítol definint el llenguatge SAT i tot seguit enunciarem i demostrarem el teorema de Cook-Levin. Finalitzarem el capítol explorant altres llenguatges i veient com, aplicant la reducció polinomial i el teorema de Cook-Levin, podem demostrar que aquests llenguatges són també $\mathbf{NP-complets}$. Aquestes demostracions ens permetran entendre millor la interrelació entre diferents problemes $\mathbf{NP-complets}$ i la importància de les reduccions polinomials en la teoria de la complexitat computacional.

El que fem en aquest treball és introduir una serie de conceptes i resultat bàsics de la teoria de computació amb l'objectiu de poder enunciar i demostrar el teorema que li dóna nom al treball, el teorema de Cook-Levin. Al llarg d'aquest estudi, veurem una serie de resultats que demostrarem presentant sols la idea que s'hauria de seguir per a poder provar-los, sense entrar moltes vegades en la demostració més formal i tècnica.

Per a l'escriptura d'aquest treball hem seguit el curs [Bar21]. En aquest curs s'han fet servir referències bibliogràfiques del següents llibres: [GJ79], [Tur36], [Sip13] i [Ros09]

CAPíTOL 1

Preliminars

En aquest capítol presentarem algunes definicions i propietats que ens serviran més endavant per a poder desenvolupar la teoria dels autòmats.

En primer lloc presentarem la definició de monoide i continuarem introduint el concepte de paraula i concatenació de paraules. Definirem el concepte de monoide sobre un conjunt i posteriorment enunciarem i demostrarem la propietat universal del monoide lliure. Finalitzarem el capítol amb un exemple on apliquem aquesta propietat. Aquestes definicions i propietats establiran les bases necessàries per a la comprensió i l'anàlisi dels autòmats en capítols posteriors. En aquest capítol extraurem algunes definicions dels treballs [Tud23] i [Gal23].

1. Definicions bàsiques

Començarem veient la definició de monoide i homomorfisme de monoides. Seguirem veient les definicions de paraula sobre un alfabet, longitud d'una paraula i concatenació de paraules. Acabarem la secció introduint la definició de monoide lliure sobre A.

DEFINICIÓ 1.1. (Monoide) *Direm que* $\mathbf{M} = (M, \cdot, 1)$ *és un monoide si* M *és un conjunt,* $\cdot$ *és una operació binaria associativa en* M *i* $1 \in M$ *és element neutre per a l'operació* $\cdot$.

EXEMPLE 1.1. *Notem que donat un grup* $(G, \cdot, 1)$*, com l'operació* $\cdot$ *és associativa i 1 és el seu element neutre, aleshores qualsevol grup és en particular un monoide.*

DEFINICIÓ 1.2. (Homomorfisme de monoides) *Siguen* $\mathbf{M}$ *i* $\mathbf{N}$ *dos monoides i siga una aplicació* $f : \mathbf{M} \longrightarrow \mathbf{N}$. *Direm que* f *és* homomorfisme de monoides *si preserva l'operació de cada monoide, és a dir, si per a tot* $m_1, m_2 \in \mathbf{M}$*, es compleix que*

$$f(m_1 m_2) = f(m_1) f(m_2)$$

i, a més, $f(1) = 1$.

Veurem ara com, a partir d'un conjunt A que anomenarem alfabet, podem crear $\mathbf{A}^\star$, que anomenarem monoide lliure sobre A i que és una estructura de monoide els elements dels quals són paraules. Començarem definint el concepte de paraula.

DEFINICIÓ 1.3. (Paraula sobre un conjunt) *Siga* A *un conjunt, una paraula de longitud* $n \in \mathbb{N}$ *sobre* A *és una aplicació de la forma*

$$\begin{array}{rccc} w\colon & n & \longrightarrow & A \\ & i & \longmapsto & w_i \end{array}$$

Així podem representar la paraula w *com*

$$w = (w_i)_{i \in n} = w_0 \dots w_{n-1}.$$

DEFINICIÓ 1.4. (Paraules de longitud determinada). *Siga A un conjunt i $n \in \mathbb{N}$, definirem el conjunt de paraules de longitud n sobre el conjunt A com* $\mathrm{Hom}(n, A)$. *Denotarem per λ, i l'anomenarem paraula buida, a l'única aplicació en* $\mathrm{Hom}(0, A)$.

Donat un alfabet A, definirem $A^\star$ com el conjunt de totes les paraules sobre A.

DEFINICIÓ 1.5. (Conjunt de totes les paraules). *Donat un conjunt A, el conjunt de totes les paraules sobre A és la unió de tots els conjunts de paraules de longitud n, amb $n \in \mathbb{N}$*

$$A^\star = \bigcup_{n \in \mathbb{N}} \mathrm{Hom}(n, A).$$

Ara veurem que el conjunt de les paraules sobre un conjunt A té estructura de monoide, amb la concatenació de paraules com a operació i la paraula buida λ com a neutre.

DEFINICIÓ 1.6. (Concatenació de paraules) *siga A un conjunt, denotem amb $\curlywedge$ la concatenació de paraules, definida com segueix:*

$$\begin{array}{rccl} w \curlywedge v\colon & n+m & \longrightarrow & A \\ & i & \longmapsto & \begin{cases} w_i & i \in n \\ v_{i-n} & i \in [n, n+m-1] \end{cases} \end{array}$$

λ és el neutre per a la concatenació.

DEFINICIÓ 1.7. (Monoide lliure sobre A). *Siga A un conjunt, $\mathbf{A}^* = (A^\star, \cdot, 1)$ és un monoide que anomenarem* monoide liure sobre A.

2. Propietat universal del monoide lliure

En aquesta secció explorarem la propietat universal del monoide lliure. Començarem amb la definició de la inclusió de generadors, que permet identificar cada lletra d'un alfabet amb una paraula de longitud 1. A continuació, demostrarem que qualsevol aplicació d'un alfabet a un monoide pot extendre's de manera única a un homomorfisme de monoides. Aquesta demostració inclou una construcció recursiva i una prova de la seua unicitat. Finalment, veurem com aquesta propietat permet calcular la longitud d'una paraula dins d'un monoide lliure, oferint un exemple concret d'aplicació pràctica.

DEFINICIÓ 1.8. (Inclusió de generadors). *Donat un conjunt A, existeix una aplicació de A en $A^\star$ anomenada* inclusió de generadors.

$$\begin{array}{rccll} \mathrm{in}_A\colon & A & \longrightarrow & A^\star & \\ & a & \longmapsto & (a)\colon & 1 \longrightarrow A \\ & & & & 0 \longrightarrow a \end{array}$$

Aquesta aplicació serveix per a identificar la lletra $a \in A$ amb la paraula (a) de longitud 1 en $A^\star$.

Una vegada introduïts els conceptes bàsics necessaris, podem enunciar i demostrar la propietat universal del monoide lliure.

PROPOSICIÓ 1.1. (Propietat universal del monoide lliure). *Si tenim A un conjunt, $\mathbf{M} = (M, \cdot, 1)$ un monoide i $f\colon A \longrightarrow M$ una aplicació, llavors existeix un únic $f^\sharp\colon \mathbf{A}^\star \longrightarrow \mathbf{M}$ homomorfisme de monoides tal que*

$$f^\sharp \circ \mathrm{in}_A = f.$$

DEMOSTRACIÓ. Per provar l'existència d'aquest homomorfisme el que farem serà presentar una expressió per a aquest i demostrar que efectivament és homomorfisme. Construirem $f^\sharp$ de manera recursiva sobre la longitud de la paraula.

Siga $w \in A$, sabem que existeix $n \in \mathbb{N}$ tal que $w \in \mathrm{Hom}(n, A)$. Distingim casos segons el valor de n:

- Si $n = 0$ aleshores tindrem $w = \lambda$, la paraula buida. Definim llavors $f^\sharp(\lambda) = 1$ de manera que estem enviant el neutre d'$\mathbf{A}^\star$ al neutre de $\mathbf{M}$.

- si $n = 1$ tindrem que existeix $a \in A$ tal que $w = (a)$. Definimm llavors $f^\sharp(a) = f(a)$. Notem que $\mathrm{in}_A(a) = (a) \in A^\star$, paraula de longitud 1. Tenim per tant la igualtat $(f^\sharp \circ \mathrm{in}_A)(a) = f(a)$.

- Si $n > 1$, podem descomposar la paraula com a concatenació de dues paraules, una primera paraula de longitud $n - 1$ amb una de longitud 1. És a dir, $w = v \curlywedge (a)$ on $v \in \mathrm{Hom}(n-1, A)$ i $a \in A$. Açò ens permet definir $f^\sharp(w) = f^\sharp(v) \cdot f(a)$.

Desenvolupant aquest raonament de forma recursiva obtenim la següent aplicació:

$$\begin{array}{rccc} f^\sharp\colon & \mathbf{A}^\star & \longrightarrow & \mathbf{M} \\ & (w_i)_{i\in n} & \longmapsto & \displaystyle\prod_{i\in n} f(w_i) \end{array}$$

Veiem que aquesta aplicació està ben definida ja que, per a tot $i \in n$, $f(w_i) \in M$ i el producte de $\mathbf{M}$ és una operació binària associativa en M.

Ara veurem que, efectivament, aquesta aplicació és un homomorfisme comprovant que la concatenació de paraules s'envia al producte de les imatges de les paraules. Per a veure açò sols cal tenir en compte que la paraula $w = (w_i)_{i\in n}$ és concatenació de n paraules de longitud 1 i que $w_i = \mathrm{in}_A(w_i)$, per a tot $i \in n$.

Així siguen les paraules $w = (w_i)_{i\in n} = w_0 \curlywedge \cdots \curlywedge w_{n-1}$ i $v = (v_j)_{j\in m}$ tenim que :

$$\begin{aligned} f^\sharp(v \curlywedge w) &= f^\sharp((v \curlywedge w_0 \curlywedge \ldots \curlywedge w_{n-2}) \curlywedge w_{n-1}) \\ &= f^\sharp(v \curlywedge w_0 \curlywedge \ldots \curlywedge w_{n-2}) \cdot f(w_{n-1}) \\ &= f^\sharp(v \curlywedge w_0 \curlywedge \ldots \curlywedge w_{n-3}) \cdot f(w_{n-2}) \cdot f(w_{n-1}) \\ &\ldots \\ &= f^\sharp(v) \cdot f(w_0) \cdot \ldots \cdot f(w_{n-1}) \\ &= f^\sharp(v) \cdot f^\sharp(w). \end{aligned}$$

Per últim necessitem demostrar que aquest homomorfisme és únic. Per a fer-ho suposem que existeix $h\colon \mathbf{A}^\star \longrightarrow \mathbf{M}$ homomorfisme de monoides tal que $h \circ \mathrm{in}_A = f$.

Considerem $w = (w_i)_{i\in n} \in A^\star$, aleshores

$$\begin{aligned}
h(w) &= h(w_0 \dots w_{n-1}) && (w = (w_i)_{i\in n}) \\
&= h(w_0 \curlywedge \cdots \curlywedge w_{n-1}) && \text{(Per Definició)} \\
&= h(w_0) \cdot \ldots \cdot h(w_{n-1}) && (h \text{ és homomorfisme de monoides}) \\
&= h(\mathrm{in}_A(w_0)) \cdot \ldots \cdot h(\mathrm{in}_A(w_{n-1})) && (\text{ per definició de l'aplicació } \mathrm{in}_A) \\
&= f(w_0) \cdot \ldots \cdot f(w_{n-1}) && (h \circ \mathrm{in}_A = f) \\
&= f^\sharp(w). && (\text{ per la definició de } f^\sharp)
\end{aligned}$$

Com coincideixen les imatges de h i $f^\sharp$, hem provat que es tracta de la mateixa aplicació. L'homomorfisme és únic. □

Utilitzant la propietat universal del monoide lliure podem veure que la longitud d'una paraula és un exemple d'homomorfisme de monoides.

EXEMPLE 1.2. *Siga A un alfabet. Considerem el monoide $(\mathbb{N}, +, 0)$ i l'aplicació*

$$\begin{array}{rccc} |\cdot|\colon & A & \longrightarrow & \mathbb{N} \\ & a & \longmapsto & 1 \end{array}$$

Per la propietat universal existeix un únic homomorfisme de monoides $|\cdot|^\sharp\colon \mathbf{A}^\star \longrightarrow \mathbb{N}$ tal que $|\cdot|^\sharp \circ \mathrm{in}_A = |\cdot|$. Notem que

$$\begin{array}{rccc} |\cdot|^\sharp\colon & \mathbf{A}^\star & \longrightarrow & \mathbb{N} \\ & (w_i)_{i\in n} & \longmapsto & \displaystyle\sum_{i=0}^{n-1} |\cdot|(w_i) = n \end{array}$$

Aquesta aplicació s'anomena longitud i compleix:

(1) $|\lambda| = 0$.

(2) $|uw| = |u| + |w|$.

CAPíTOL 2

Computació i decidibilitat

En aquest capítol, explorarem les bases de la teoria de la computació, centrant-nos en les estructures computacionals i els problemes decidibles. Començarem introduint els autòmats finits, tant deterministes com no deterministes, com a models matemàtics per reconèixer llenguatges formals. A continuació definirem les màquines de Turing, un model de computació més potent, i veurem com aquestes poden acceptar o rebutjar paraules. Això ens permetrà introduir els conceptes de llenguatges decidibles i reconeixibles per màquines de Turing. Demostrarem també l'existència de llenguatges que no són semi-decidibles, destacant així les limitacions de la computació. Finalment introduirem el concepte de màquina de Turing no determinista, que ens permetrà explorar la computació des d'una perspectiva no determinista, obrint noves possibilitats en el desenvolupament d'algorismes i en la comprensió de la computabilitat.

1. Autòmats finits

En aquesta secció començarem introduint el concepte de llenguatge i presentarem les operacions definibles sobre aquests. Això ens introduirà en el món dels autòmats. Després abordarem els autòmats finits, tant deterministes com no deterministes, que són models matemàtics per a reconèixer llenguatges formals. També tractarem les expressions regulars i la seua relació amb els autòmats finits. Finalment discutirem els llenguatges reconeixibles per autòmats, establint l'equivalència entre llenguatges reconeixibles per autòmats deterministes, no deterministes i expressions regulars. Aquesta connexió ens ajudarà a comprendre millor la teoria de la computació i la relació entre diferents models de computació i l'expressivitat dels llenguatges formals.

DEFINICIÓ 2.1. (Llenguatge). *Donat un alfabet A, anomenem* llenguatge *a qualsevol subconjunt $L \subseteq A^\star$. Les operacions que poden ser definides en els llenguatges són:*

- *Operacions booleanes: l'unió, intersecció i la complementació.*
- *Quocients: siga $L \subseteq A^\star$ i $w \in A$, aleshores:*

$$w^{-1}L = \{v \in A^\star \mid wv \in L\}$$
$$Lw^{-1} = \{v \in A^\star \mid vw \in L\}$$

  En general, siguen $L, K \subseteq A^\star$, aleshores:

$$K^{-1}L = \{v \in A^\star \mid \exists k \in K(kv \in L)\}$$
$$LK^{-1} = \{v \in A^\star \mid \exists k \in K(vk \in L)\}$$

- *Producte: siguen L_1 i L_2 dos llenguatges, aleshores:*

$$L_1L_2 = \{u_1u_2 \in A^\star \mid u_1 \in L_1, u_2 \in L_2\}$$

- *Estrella i suma: si $L \subseteq A^\star$, $L^\star$ és el submonoide i L^+ el semigrup d'$\mathbf{A}^\star$ generat per L.*

$$L^\star = \{w_0 \curlywedge \cdots \curlywedge w_{n-1} \in A^\star \mid n \in \mathbb{N} \text{ i } w_0, \ldots, w_{n-1} \in L\}$$
$$L^+ = \{w_0 \curlywedge \cdots \curlywedge w_{n-1} \in A^\star \mid n \in \mathbb{N}\backslash\{0\} \text{ i } w_0, \ldots, w_{n-1} \in L\}$$

Notem que $L^\star = L^+ + 1$, on $1 = \{\lambda\}$.

També podem definir les potències de L com:

$$\begin{cases} L^0 = \{\lambda\} \\ L^{n+1} = L(L^n) \end{cases}$$

De manera que

$$L^\star = \bigcup_{n\in\mathbb{N}} L^n \quad i \quad L^+ = \bigcup_{n\in\mathbb{N}^\star} L^n$$

Veiem ara la definició d'autòmat finit i seguirem amb un exemple.

DEFINICIÓ 2.2. (Autòmat finit). *Un* autòmat finit *és una 5-tupla donada per $\mathcal{A} = (\mathcal{Q}, A, E, I, F)$ on $\mathcal{Q}$ és un conjunt finit anomenat conjunt d'estats, A és un alfabet, E és un subconjunt de $\mathcal{Q} \times \mathcal{A} \times \mathcal{Q}$, I és un subconjunt de $\mathcal{Q}$ anomenat conjunt d'estats inicials i finalment F és un subconjunt de $\mathcal{Q}$ anomenat conjunt d'estats finals.*

Direm que $\mathcal{A}$ és finit si $\mathcal{Q}$ és un conjunt finit. Podem representar l'autòmat $\mathcal{A}$ com un graf etiquetat d'acord amb les transicions en E.

Veiem ara un exemple d'autòmat finit.

EXEMPLE 2.1. *Considerem ara l'autòmat $\mathcal{A} = (\mathcal{Q}, A, E, I, F)$ on $\mathcal{Q} = \{q_0, q_1, q_2\}$, $I = \{q_0\}$, $F = \{q_2\}$, l'alfabet $A = \{a, b\}$ i les transicions*

$$E = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1), (q_1, a, q_2), (q_2, a, q_1), (q_2, b, q_1)\}.$$

Representem aquest autòmat mitjançant un graf representat a la següent figura:

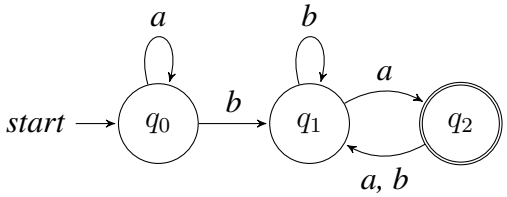

FIGURA 2.1. Exemple d'autòmat finit.

Definim ara els conceptes de transicions consecutives, camins i camí exitós.

DEFINICIÓ 2.3. *Siga $\mathcal{A} = (\mathcal{Q}, A, E, I, F)$ un autòmat i siguen $p, p', q, q' \in \mathcal{Q}$ i $a, a' \in A$. Direm que dues* transicions (p, a, q) i (p', a', q') són consecutives *si $q = p'$. Un* camí *en un autòmat $\mathcal{A}$ és una seqüència de transicions consecutives*

$$c = (q_0, a_0, q_1), (q_1, a_1, q_2), (q_2, a_2, q_3), \ldots, (q_{n-1}, a_{n-1}, q_n)$$

també denotat per

$$c : q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \longrightarrow q_{n-1} \xrightarrow{a_{n-1}} q_n.$$

Un camí en $\mathcal{A}$ s'anomena inicial *si q_0 és un estat inicial i* final *si q_n és un estat final. Un camí és* exitós *(o acceptat) si és inicial i final.*

EXEMPLE 2.2. (Camí exitós) *Si ens fixem en la Figura 2.1 podem observar que el camí $c = (q_0, b, q_1), (q_1, a, q_2)$ és un camí exitós.*

Dins dels autòmats finits diferenciarem entre autòmats deterministes i no deterministes. Els autòmats deterministes tenen la propietat que, per a cada estat i símbol de l'alfabet, hi ha exactament una transició definida, el que significa que l'estat següent està determinat de manera única per l'estat actual i el símbol d'entrada. D'altra banda, els autòmats no deterministes permeten múltiples transicions per a un mateix estat i símbol d'entrada, la qual cosa ofereix una major flexibilitat en la seua descripció.

DEFINICIÓ 2.4. (Autòmat determinista). *Un autòmat $\mathcal{A} = (\mathcal{Q}, A, E, I, F)$ és* determinista *si I conté exactament un estat inicial i si, per a cada estat $q \in \mathcal{Q}$ i per a cada lletra $a \in A$, sols existeix un estat q' de manera que $q \xrightarrow{a} q'$ és una transició en E.*

Escriurem $\mathcal{A} = (\mathcal{Q}, A, \delta, q_0, F)$ on δ ve definit com

$$\begin{array}{rccl} \delta\colon & A & \longrightarrow & \mathcal{Q}^{\mathcal{Q}} \\ & a & \longmapsto & \delta(a) : \begin{array}{ccc} \mathcal{Q} & \longrightarrow & \mathcal{Q} \\ q & \longmapsto & \delta(q,a) \end{array} \end{array}$$

Observem que $(\mathcal{Q}^{\mathcal{Q}}, \cdot, \mathrm{id}_{\mathcal{Q}})$ és un monoide. Per la propietat universal existeix un únic $\delta^{\#} : \mathbf{A}^{\star} \longrightarrow \mathcal{Q}^{\mathcal{Q}}$ homomorfisme de monoides tal que $\delta^{\#} \circ \mathrm{in}_A = \delta$. Aquest homomorfisme descriu la transició dins de l'autòmat que determina cada paraula en $A^{\star}$ per a cada estat.

NOTA 2.1. *Donat un autòmat determinista $\mathcal{A} = (\mathcal{Q}, A, \delta, q_0, F)$, el llenguatge generat per aquest és:*

$$\mathcal{L}(\mathcal{A}) = \{w \in A^{\star} \mid \delta^{\#}(w)(q_0) \cap F \neq \emptyset\} = \{w \in A^{\star} \mid \delta^{\#}(w)(q_0) \ \in F\}.$$

És a dir, el llenguatge que reconeix $\mathcal{A}$ són totes les paraules que etiqueten un camí existós en $\mathcal{A}$.

DEFINICIÓ 2.5. (Autòmat no determinista). *Un* automat no determinista *és un autòmat $\mathcal{A} = (\mathcal{Q}, A, \delta, q_0, F)$ en el què, per a un estat $q \in \mathcal{Q}$ i per a una lletra $a \in A$, pot existir més d'un estat q', q'' en $\mathcal{Q}$ de manera que $q \xrightarrow{a} q'$ i $q \xrightarrow{a} q''$ són transicions en δ*

$$\begin{array}{rccl} \delta\colon & A & \longrightarrow & \mathcal{P}(\mathcal{Q})^{\mathcal{Q}} \\ & a & \longmapsto & \delta(a) : \begin{array}{ccc} \mathcal{Q} & \longrightarrow & \mathcal{P}(\mathcal{Q}) \\ q & \longmapsto & \delta(q,a) \end{array} \end{array}$$

Observem que $(\mathcal{P}(\mathcal{Q})^{\mathcal{Q}}, *, \{\ \})$ és un monoide. Per la propietat universal existeix un únic $\delta^{\#} : \mathbf{A}^{\star} \longrightarrow \mathcal{P}(\mathcal{Q})^{\mathcal{Q}}$ homomorfisme de monoides. Aquest homomorfisme descriu els conjunts dins de l'autòmat accesibles per a cada estat i cada paraula en $A^{\star}$.

NOTA 2.2. *Si tenim $f : \mathcal{Q} \longrightarrow \mathcal{P}(\mathcal{Q})$ i $g : \mathcal{Q} \longrightarrow \mathcal{P}(\mathcal{Q})$, aleshores l'operació $*$ anterior ve donada per*

$$\begin{array}{rccc} g * f : & \mathcal{Q} & \longrightarrow & \mathcal{P}(\mathcal{Q}) \\ & q & \longmapsto & \bigcup_{z \in f(q)} g(z) \end{array}$$

Per visualitzar aquests conceptes, presentarem exemples gràfics d'autòmats finits deterministes i no deterministes que es poden representar visualment mitjançant gràfics etiquetats. En aquests grafs els vèrtexs representen estats que estan connectats per transicions etiquetades amb símbols d'un alfabet. Aquestes transicions representen com l'autòmat es mou d'un estat a un altre en resposta a l'entrada de símbols de l'alfabet.

EXEMPLE 2.3. (Autòmat determinista). *Siga $\mathcal{A} = (\mathcal{Q}, A, \delta, I, F) = (\mathcal{Q}, A, \delta, q_0, F)$ un autòmat amb $\mathcal{Q} = \{q_0, q_1, q_2, q_3\}$, $A = \{a, b\}$, $I = \{q_0\}$ i $F = \{q_3\}$.*

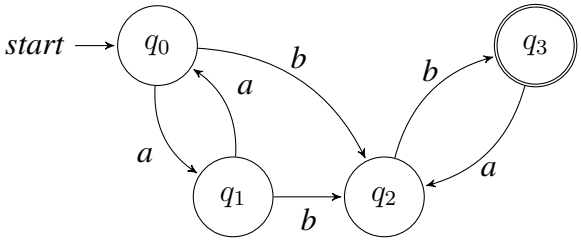

FIGURA 2.2. Exemple d'autòmat finit determinista.

És un autòmat finit determinista.

En aquest cas les paraules $aabb$ i abb estan en $\mathcal{L}(\mathcal{A})$, encara que hi han més.

EXEMPLE 2.4. *Un exemple d'autòmat no determinista seria agafar l'autòmat de la Figura 2.2 i afegir-li la transició (q_0, b, q_3). Així q_0 té més d'una transició que ix de q_0 etiquetada amb b. Notem que aquest autòmat reconeix b perquè hi ha almenys un camí existós etiquetat amb b. Açò no ocorria a l'autòmat anterior.*

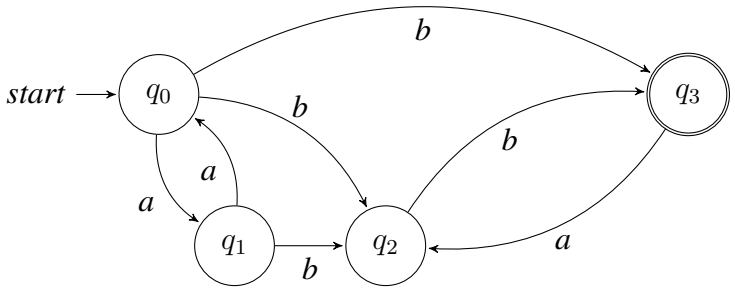

FIGURA 2.3. Exemple d'autòmat finit no determinista.

Introduirem ara el concepte d'expressions regulars sobre un alfabet, que són termes que representen llenguatges. Explorarem com les expressions regulars es relacionen amb els autòmats finits.

DEFINICIÓ 2.6. (Expressions regulars). *Siga A un alfabet, les* expressions regulars *sobre A són el menor conjunt de paraules en $A^\star$, denotat per $\mathrm{Reg}(A^*)$, que satisfà:*

(1) $\emptyset \in \mathrm{Reg}(A^*)$*;*

(2) $\lambda \in \mathrm{Reg}(A^*)$*;*

(3) $a \in \mathrm{Reg}(A^*)$*, per a tot* $a \in A$*;*

(4) *Si* $L, K \in \mathrm{Reg}(A^*)$*, aleshores* $LK \in \mathrm{Reg}(A^*)$ *i* $L + K \in \mathrm{Reg}(A^*)$*;*

(5) *Si* $L \in \mathrm{Reg}(A^*)$*, aleshores* $L^\star \in \mathrm{Reg}(A^*)$.

Tota expressió regular E *en* $\mathrm{Reg}(A^*)$ *dona lloc a un llenguatge en* A*, que denotarem per* $\mathcal{L}(E)$*, definit recursivament com segueix. En els casos bàsics:*

$$\mathcal{L}(\emptyset) = \emptyset \qquad \mathcal{L}(\lambda) = \{\lambda\} \qquad \mathcal{L}(a) = \{a\}$$

Suposem definits $\mathcal{L}(L)$ *i* $\mathcal{L}(K)$*, per a expressions regulars* $L, K \in \mathrm{Reg}(A^*)$*, aleshores:*

$\mathcal{L}(LK) = \mathcal{L}(L)\mathcal{L}(K)$.

$\mathcal{L}(L + K) = \mathcal{L}(L) \cup \mathcal{L}(K)$.

$\mathcal{L}(L^\star) = \bigcup_{n\in\mathbb{N}} \mathcal{L}(L)^n$.

Per a finalitzar la secció entrarem en la noció de llenguatges reconeixibles per autòmats, el que ens permetrà establir l'equivalència entre llenguatges reconeixibles per autòmats deterministes, no deterministes i expressions regulars. Aquesta connexió és fonamental en la teoria de la computació i ens permet comprendre millor la relació entre diferents models de computació i l'expressivitat dels llenguatges formals.

DEFINICIÓ 2.7. *Si* $L \subseteq A^\star$*, direm que*

(1) L *és reconeixible per un autòmat determinista si* $L = \mathcal{L}(\mathcal{A})$ *per a* $\mathcal{A}$ *un autòmat finit determinista sobre* A*.*

(2) L *és reconeixible per un autòmat no determinista si* $L = \mathcal{L}(\mathcal{A})$ *per a* $\mathcal{A}$ *un autòmat finit no determinista sobre* A*.*

(3) L *és regular si* $L = \mathcal{L}(E)$ *per a una expressió regular* $E \in \mathrm{Reg}(A^*)$.

Veiem ara un teorema que estableix una equivalència entre llenguatges reconeixibles per autòmats deterministes, no deterministes i expressions regulars.

TEOREMA 2.1. *Són equivalents:*

(1) L *és reconeixible per un autòmat determinista.*

(2) L *és reconeixible per un autòmat no determinista.*

(3) L *és regular.*

DEMOSTRACIÓ. Aquesta prova la podem trobar al Capítol 1 del llibre [Sip13]. En particular, podem trobar l'equivalencia entre autòmats finits deterministes i autòmats finits no deterministes al Capítol 1, Secció 1.2 (pàg. 55) d'aquest llibre. L'equivalència entre atòmats finits i expressions regulars està demostrada al Capítol 1, Secció 1.3. □

2. Màquines de Turing

A continuació definirem què és una màquina de Turing i veurem quan aquesta accepta o rebutja una paraula. Açò ens permetrà definir quan un llenguatge és decidible i quan reconeixible o semi-decidible per una màquina de Turing. A més, il·lustrarem aquests conceptes amb un exemple concret, una màquina de Turing que decideix el llenguatge dels palíndroms sobre un alfabet binari.

DEFINICIÓ 2.8. (Màquina de Turing). *Una màquina de Turing és una 7-tupla*

$$\mathcal{M} = (A, \Gamma, \mathcal{Q}, \delta, q_0, q_a, q_r)$$

on:

A *és un alfabet;*

Γ *és un alfabet de cinta que conté* $A \subseteq \Gamma$ *i un símbol especial* $\square \in \Gamma$ *anomenat final de cinta;*

$\mathcal{Q}$ *és un conjunt d'estats;*

$q_0, q_a, q_r \in \mathcal{Q}$ *i* $q_0 \neq q_a \neq q_r$ *són tres estats especials, d'inici, aceptació i rebuig, respectivament;*

$\delta : \mathcal{Q} \times \Gamma \longrightarrow \mathcal{Q} \times \Gamma \times \{L, R\}$ *és la funció de transició.*

DEFINICIÓ 2.9. (Configuració). *Donada una màquina de turing* $\mathcal{M}$*, una* configuració *de la màquina és un element de* $\Gamma^\star \times \mathcal{Q} \times \Gamma^\star$ *(normalment escriurem* $w_1 q w_2$ *en compte de* (w_1, q, w_2)*). Direm que hi ha una transició de la configuració* $w_1 q x w_2$ *a* $w_1 y q' w_2$ *si* $\delta(q, x) = (q', y, R)$*. Direm que hi ha una transició de la configuració* $w_1 z q x w_2$ *a* $w_1 q' z y w_2$ *si* $\delta(q, x) = (q', y, L)$*.*

DEFINICIÓ 2.10. *Una màquina de Turing* $\mathcal{M} = (A, \Gamma, \mathcal{Q}, \delta, q_0, q_a, q_r)$ accepta w *una paraula en* $A^\star$ *si des de la configuració* $q_0 w$ *podem aconseguir una configuració de la forma* $w_1 q_a w_2$ *sense passar per una configuració de rebuig* $(\_, q_r, \_)$ *. Direm que* rebutja $w \in A^\star$ *si des de la configuració* $q_0 w$ *podem aconseguir una configuració de la forma* $w_1 q_r w_2$*.*

Veiem ara la definició de llenguatges decidibles i semi-decidibles o reconeixibles per una màquina de Turing.

DEFINICIÓ 2.11. *Donat* $L \subseteq A^\star$ *direm que la màquina de Turing* $\mathcal{M}$*:*

Decideix L *si*

- *Accepta totes les paraules en* L*.*
- *Rebutja totes les paraules en* $A^\star - L$*.*

Reconeix *(o semi-decideix)* L *si*

- *Accepta totes les paraules en* L*.*
- *No accepta totes les paraules en* $A^\star - L$*.*

Cal emfatitzar que no acceptar no és el mateix que rebutjar. Diem que una màquina rebutja una paraula w si des de la configuració q_0w podem aconseguir una configuració de la forma $w_1q_rw_2$, mentre que diem que una màquina no accepta una paraula w si en cap moment arriba a una configuració de la forma $w_1q_aw_2$. No acceptar pot implicar o bé que la cadena siga rebutjada o bé que la màquina no acabe sobre aquesta.

Així acabem de veure que una màquina de Turing decideix un llenguatge L si accepta tota paraula en L i rebutja tota paraula que no està en L. D'altra banda direm que una màquina de Turing reconeix L si aquesta accepta tota paraula en L i rebutja o no acaba sobre tota paraula que no està en L.

NOTA 2.3. *Si $\mathcal{M}$ decideix L, aleshores $\mathcal{M}$ reconeix L.*

Veiem ara un exemple d'una màquina de Turing que decideix el llenguatge *Palíndroms*.

EXEMPLE 2.5. (Palíndroms). *Siga l'alfabet $A = \{0, 1\}$, siga $\Gamma = \{0, 1, \square\}$ i siga $\mathcal{M}$ la Màquina de Turing representada a la Figura 2.4. $\mathcal{M}$ decideix el llenguatge dels palíndroms o $L = \{ww^r \mid w \in A^\star\}$ on, si $w = (w_i)_{i\in n}$, aleshores $w^r = (w_{n-i-1})_{i\in n}$.*

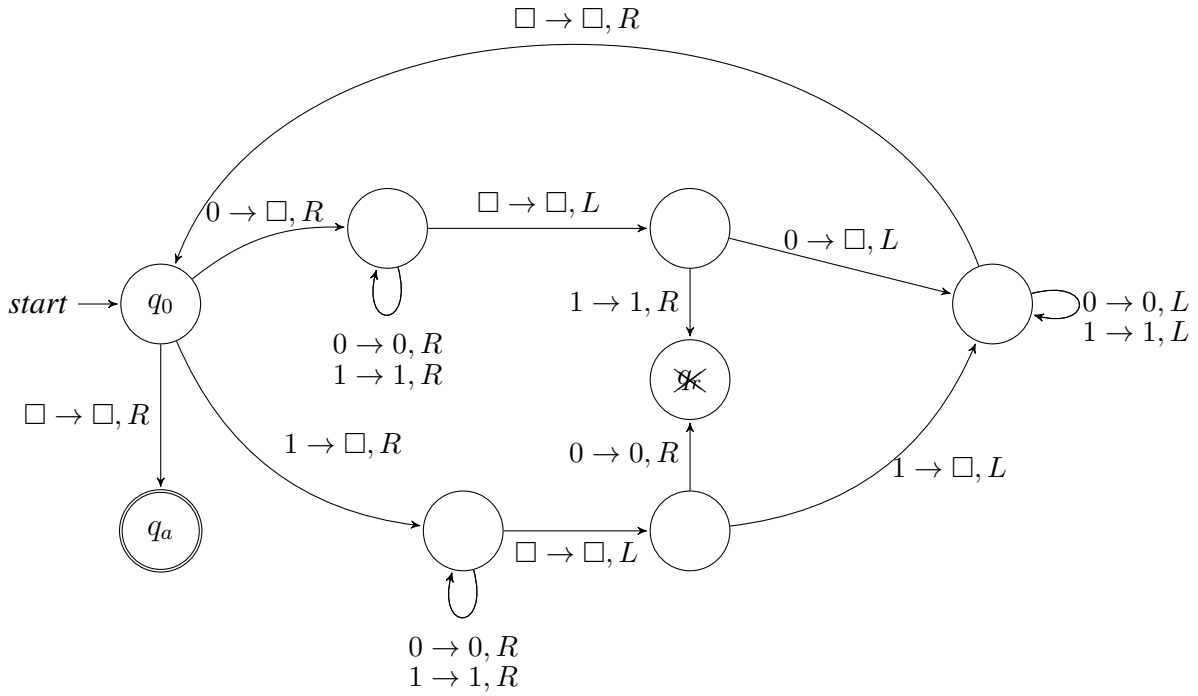

FIGURA 2.4. Màquina de Turing que decideix L.

Expliquem què fa aquesta màquina

(1) *Quan introduïm una cadena $w \in \Gamma$, la màquina comença comprovant quin és el primer caràcter. Si aquest és $\square$, accepta la cadena (paraula buida).*

(2) *Si el primer caràcter és 0, la màquina el canvia per $\square$ i desplaça el capçal cap a la dreta fins trobar el final de la cadena (marcat per $\square$). En aquest moment el capçal retrocedeix una posició cap a l'esquerra i comprova l'últim caràcter. Si és 1, la màquina*

rebutja la cadena (estat q_r). Si és 0, el canvia per $\square$ (per a indicar que ja ha comprovat aquest caràcter) i el capçal es desplaça cap a l'esquerra fins trobar un altre $\square$. A continuació, el capçal es mou una posició cap a la dreta i torna al pas (1).

(3) *Si el primer caràcter és 1, el procés és similar al descrit en el pas (2), però invertint el paper de 0 i 1 en les comprovacions.*

3. Problemes decidibles i semidecidibles

Començarem aquesta secció definint els conceptes de decidibilitat i semi-decidibilitat, i explorarem la seua relació amb les màquines de Turing. Presentarem exemples concrets de llenguatges tant decidibles com semi-decidibles i examinarem casos específics de llenguatges semi-decidibles que, tot i ser reconeixibles per una màquina de Turing, no són decidibles. Per concloure, demostrarem resultats importants que posen de manifest l'existència de llenguatges que no són semi-decidibles.

Recordem que un llenguatge és decidible si una màquina de Turing pot acceptar totes les paraules en el llenguatge i rebutjar les que no ho estan. En canvi, un llenguatge és semi-decidible si una màquina de Turing pot acceptar totes les paraules en el llenguatge, però pot no acabar d'executar-se per a les que no hi pertanyen. Introduïm les definicions formals d'aquests conceptes:

NOTA 2.4. (Llenguatges decidibles i semi-decidibles). *Siga A un alfabet, $L \subseteq A^\star$ un llenguatge i $\mathcal{M}$ una màquina de Turing. Siga $w \in L$, els diferents comportaments que pot tindre $\mathcal{M}$ sobre w són:*

$$\mathcal{M}(w) = \begin{cases} \textit{Accepta} \\ \textit{Rebutja} \\ \textit{No acaba} \end{cases}$$

Diguem que

- *$\mathcal{M}$ decideix L si $\mathcal{M}$ accepta tota paraula w en L i $\mathcal{M}$ rebutja tot $w \notin L$. Així direm que un llenguatge L és* decidible *si existeix $\mathcal{M}$, una màquina de Turing, que decideix L.*
- *$\mathcal{M}$ semi-decideix (reconeix) L si $\mathcal{M}$ accepta tota paraula w en L i $\mathcal{M}$ no accepta tot $w \notin L$. Per tant direm que un llenguatge L és* semi-decidible *si existeix $\mathcal{M}$ una màquina de Turing que semi-decideix L.*

NOTA 2.5. *Siga A un alfabet, les paraules d'A sempre es poden codificar en un llenguatge binari. Per tant, podem pensar que les màquines de Turing que treballem rebran sempre entrades en binari. Tot i que, en certes ocasions, el problema ens permetrà elegir l'alfabet que millor codifique el problema.*

Veiem ara un exemple de llenguatge decidible presentant una màquina de Turing que el decideix.

EXEMPLE 2.6. *Siga $A = \{a, b, c, d\}$. $L = \mathcal{L}((a + c)^\star)$ és un llenguatge decidible. Per a veure que açò és cert sols ens farà falta presentar una màquina de Turing que decideix el llenguatge. Agafem com a alfabet de cinta $\Gamma = \{a, b, c, d, \square\}$*

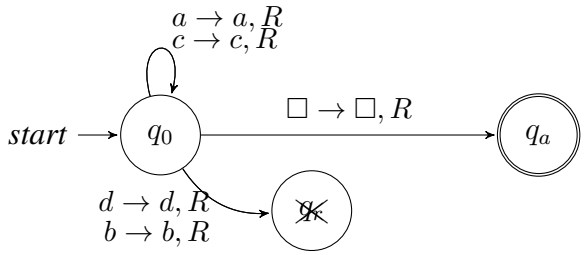

FIGURA 2.5. Màquina de Turing que decideix L.

Expliquem que fa aquesta màquina quan li introduïm una cadena $w \in \Gamma$:

(1) *La màquina comença en l'estat inicial q_0 i escaneja el primer símbol de la cinta.*

(2) *Si el símbol escanejat és $\square$, significa que ha arribat al final de l'entrada. En aquest punt, la màquina s'atura i accepta la cadena (arriba a un estat q_a). Si el símbol escanejat és b o d, la màquina es desplaça a un estat de rebuig q_r, indicant que la cadena no pertany al llenguatge L. Si el símbol escanejat és a o c, la màquina el deixa tal com està i mou el capçal cap a la dreta per a continuar llegint l'entrada repetint aquest procediment fins que s'arribe a un estat q_a o q_r.*

Ara anem a codificar les paraules d'A com a paraules en $2 = \{0, 1\}$. Definim $f : A \longrightarrow 2^\star$ tal que $f(a) = 00, f(b) = 01, f(c) = 10, f(d) = 11$. Per la propietat universal existeix un únic homomorfisme de monoides $f^\sharp : A^\star \longrightarrow 2^\star$ tal que $f^\sharp \circ \mathrm{in}_A = f$. Notem que reconèixer L és equivalent a reconèixer $f^\sharp[L] \subseteq 2^\star$.

Una vegada vista la definició formal, definirem alguns llenguatges que demostrarem que són decidibles, com són els llenguatges SUMA, MULT i AFND.

DEFINICIÓ 2.12. (SUMA) *Siga l'alfabet $A = \{a, b, c\}$*

$$\mathrm{SUMA} = \{a^n b^m c^k \in A^\star \mid n, m, k \in \mathbb{N}, n + m = k\}$$

TEOREMA 2.2. SUMA *és un llenguatge decidible, és a dir, existeix $\mathcal{M}$ una màquina de Turing tal que*

$$\mathcal{M}(w) = \begin{cases} \textit{accepta} & \textit{si } w \in \mathrm{SUMA} \\ \textit{rebutja} & \textit{si } w \notin \mathrm{SUMA} \end{cases}$$

DEMOSTRACIÓ. Per a demostrar aquest enunciat presentem una màquina de Turing $\mathcal{M}$ que decideix el llenguatge. Agafem com a alfabet de cinta $\Gamma = \{a, b, c, \square, x\}$. Podem comprovar que la següent màquina de Turing decideix el llenguatge SUMA.

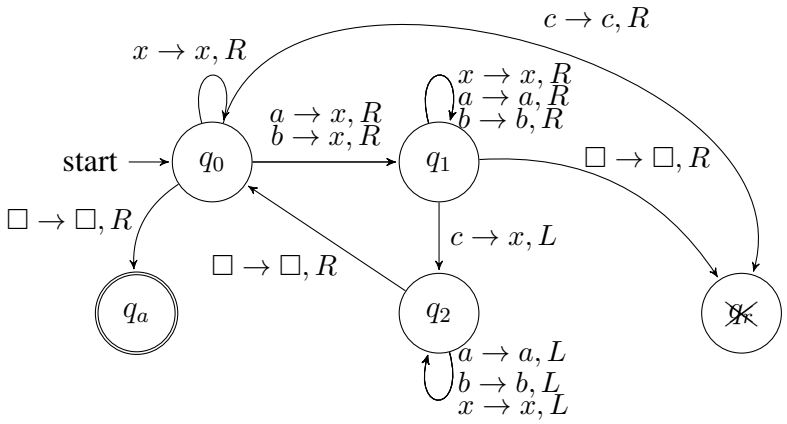

FIGURA 2.6. Màquina de Turing que decideix llenguatge SUMA.

Expliquem quin és el procediment que segueix aquesta màquina per a decidir si $w \in$ SUMA:

(1) La màquina comença en l'estat inicial q_0, amb el capçal situat sobre el primer símbol de la cadena w. Si aquest símbol és una a o una b, el substitueix per una x (per marcar que ha sigut processat), i desplaça el capçal una posició a la dreta, passant a l'estat q_1. Si el símbol és c, la màquina es desplaça a l'estat q_r (rebutja la cadena), ja que una c no pot aparèixer sense haver marcat abans una a o una b. Si el símbol és $\square$, la màquina arriba a l'estat d'acceptació q_a ja que açò indicaria que la cadena és buida i, per tant, compleix la condició $n + m = k$.

(2) En l'estat q_1, la màquina es mou cap a la dreta. Si el símbol actual és una x (és a dir, un símbol ja processat), una a o una b, la màquina no modifica el símbol i mou el capçal a la dreta mantenint l'estat q_1. Si el símbol és $\square$ mentre es troba en l'estat q_1, la màquina rebutja la cadena, ja que això indica que no hi ha prou c per a correspondre amb els a i b de la cadena. Quan el símbol és un c, la màquina el canvia per una x (per marcar-lo com processat) i passa a l'estat q_2.

(3) En l'estat q_2, el capçal es mou cap a l'esquerra fins a tornar al principi de la cadena. Quan troba el símbol blanc ($\square$), la màquina mou el capçal una posició a la dreta i canvia a l'estat q_0, preparant-se per començar un nou cicle de processament tornant al pas (1). $\square$

DEFINICIÓ 2.13. *(*MULT*) Siga l'alfabet* $A = \{a, b, c\}$

$$\text{MULT} = \{a^n b^m c^k \in A^\star \mid n, m, k \in \mathbb{N}, \quad nm = k\}.$$

TEOREMA 2.3. MULT *és un llenguatge decidible.*

DEMOSTRACIÓ. Construïm una màquina que decideix el llenguatge seguint aquest procediment:

(1) Marquem la primera lletra.

(2) Movem el capçal a la dreta fins a trobar la primera b i la marquem. A continuació movem el capçal a la dreta fins a trobar la primera c i la marquem.

(3) Movem el capçal a l'esquerra fins a trobar la primera b no marcada i la marquem. A continuació movem el capçal a la dreta fins a trobar la primera c no marcada i la marquem. Repetim aquest pas fins que no queden més b.

(4) Desmarquem les b. Tornem a la primera a no marcada, la marquem i repetim tot el procés des del pas (2).

Aquest procediment permet decidir el llenguatge MULT. □

NOTA 2.6. *Tot autòmat determinista $\mathcal{A} = (\mathcal{Q}, A, \delta, q_0, F)$ es pot codificar com una paraula en un llenguatge generat per l'alfabet $\Gamma = A \cup \mathcal{Q} \cup \{\text{“}\{\text{”}, \text{“}\}\text{”}, \text{“,”}, \text{“(”}, \text{“)”}\}$ que afegeix a l'alfabet A els estats en $\mathcal{Q}$ i els signes auxiliars d'obrir i tancar claudàtors, coma i obrir i tancar parèntesi. Denotem per $\langle\mathcal{A}\rangle$ a la codificació d'$\mathcal{A}$ com a paraula en l'alfabet Γ.*

Veiem un exemple de codificació d'un autòmat $\mathcal{A}$.

EXEMPLE 2.7. *L'autòmat representat en la Figura 2.7 es pot codificar també com segueix:*

$$\langle\mathcal{A}\rangle = (\{a,b\}, \{q_0,q_1,q_2\}, \{(q_0,a,q_1),(q_0,b,q_0),\\ (q_1,a,q_2),(q_2,a,q_2),(q_2,a,q_2),(q_2,b,q_0)\}, q_0, \{q_2\})$$

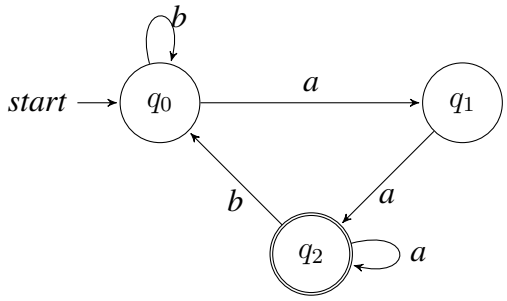

FIGURA 2.7. Autòmat determinista.

DEFINICIÓ 2.14. *Si $\mathcal{A} = (\mathcal{Q}, A, \delta, q_0, F)$ és un autòmat finit no determinista i Γ un alfabet capaç de codificar l'autòmat $\mathcal{A}$, definim el següent llenguatge:*

$$\mathrm{AFND} = \{\langle\mathcal{A}\rangle w \in \Gamma^\star \mid \mathcal{A} \textit{ no determinista i } w \in \mathcal{L}(\mathcal{A})\}.$$

És a dir, AFND *conté totes les paraules de la forma codificació d'un autòmat finit no determinista juntament amb una paraula que reconeix $\mathcal{A}$.*

TEOREMA 2.4. AFND *és decidible.*

DEMOSTRACIÓ. Construïm una màquina de Turing $\mathcal{M}$ que decideix el llenguatge AFND:

Siga $\alpha \in \Gamma^\star$, el primer que fa $\mathcal{M}$ és verificar si $\alpha = \langle\mathcal{A}\rangle w$.

- En cas negatiu, rebutgem.

- En cas afirmatiu, és a dir, si $\alpha = \langle \mathcal{A} \rangle w$, simulem el comportament d'$\mathcal{A}$ sobre w. Si la simulació arriba a un estat d'acceptació per a la paraula w, la màquina de Turing acceptarà la cadena. En cas contrari, la cadena serà rebutjada.

Notem així que $\mathcal{M}$ decideix AFND. □

Presentem ara un llenguatge semi-decidible però que demostrarem que no és decidible:

DEFINICIÓ 2.15. *($\mathcal{L}_{MT}$) Siga Γ un alfabet suficientment complex per a codificar expressions del tipus $\langle \mathcal{M}, w \rangle$ on $\mathcal{M}$ és una màquina de Turing i w una paraula en binari codificant una paraula en l'alfabet d'entrada per a $\mathcal{M}$. Definim el llenguatge*

$$\mathcal{L}_{MT} = \{\langle \mathcal{M}, w \rangle \in \Gamma^\star \mid \mathcal{M} \textit{ és màquina de Turing i } \mathcal{M} \textit{ accepta } w\}.$$

El primer que farem és demostrar que aquest és un llenguatge semi-decidible.

TEOREMA 2.5. *$\mathcal{L}_{MT}$ és semi-decidible.*

DEMOSTRACIÓ. Construïm una màquina de Turing $\mathcal{U}$ que semi-decideix el llenguatge $\mathcal{L}_{MT}$:

Siga $\alpha \in \Gamma^\star$, el primer que fa $\mathcal{U}$ és verificar si $\alpha = \langle \mathcal{M}, w \rangle$.

- En cas negatiu, rebutgem.
- En cas afirmatiu, és a dir, si $\alpha = \langle \mathcal{M}, w \rangle$, simulem el comportament de $\mathcal{M}$ sobre w
  - Si $\mathcal{M}$ accepta w, aleshores $\mathcal{U}$ accepta α.
  - Si $\mathcal{M}$ rebutja w, aleshores $\mathcal{U}$ rebutja α.
  - Si $\mathcal{M}$ no finalitza sobre w, aleshores $\mathcal{U}$ no finalitza sobre α.

Notem així que $\mathcal{U}$ accepta α si, i només si, $\mathcal{M}$ accepta w. La màquina $\mathcal{U}$ semi-decideix $\mathcal{L}_{MT}$. □

NOTA 2.7. *La màquina de Turing $\mathcal{U}$ que hem construit en el teorema anterior s'anomena* màquina universal de Turing.

Veiem ara que el llenguatge $\mathcal{L}_{MT}$ no és decidible.

TEOREMA 2.6. *El llenguatge $\mathcal{L}_{MT}$ no és decidible.*

DEMOSTRACIÓ. Farem la demostració per reducció a l'absurd. Suposem que ho és, és a dir, que existeix una màquina de Turing $\mathcal{N}$ tal que si $\alpha \in \mathcal{L}_{MT}$ aleshores $\mathcal{N}$ accepta α, i si $\alpha \notin \mathcal{L}_{MT}$ aleshores $\mathcal{N}$ rebutja α.

Construim $\mathcal{D}$ una màquina de Turing tal que, donat $\alpha \in \Gamma^\star$, comprove que $\alpha = \langle \mathcal{M} \rangle$ (α té la codificació de màquina de Turing).

- En cas negatiu rebutgem .
- En cas positiu:
  - $\mathcal{D}$ rebutja $\langle \mathcal{M} \rangle$ si, i només si, $\mathcal{N}$ accepta $\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle$. A més $\mathcal{N}$ accepta $\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle$ si, i només si, $\mathcal{M}$ accepta la seua codificació.

- $\mathcal{D}$ accepta $\langle \mathcal{M} \rangle$ si, i només si, $\mathcal{N}$ rebutja $\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle$. A més $\mathcal{N}$ rebutja $\langle \mathcal{M}, \langle \mathcal{M} \rangle \rangle$ si, i només si, $\mathcal{M}$ rebutja la seua codificació.

Queda així definida la màquina de Turing $\mathcal{D}$. Ens preguntem ara què fa $\mathcal{D}$ sobre la codificació $\langle \mathcal{D} \rangle$:

- $\mathcal{D}$ rebutja $\langle \mathcal{D} \rangle$ si, i només si, $\mathcal{D}$ accepta $\langle \mathcal{D} \rangle$.

- $\mathcal{D}$ accepta $\langle \mathcal{D} \rangle$ si, i només si, $\mathcal{D}$ rebutja $\langle \mathcal{D} \rangle$.

Arribem a una contradicció que ve de suposar que $\mathcal{L}_{MT}$ és decidible. Per tant podem concloure que $\mathcal{L}_{MT}$ no és decidible. □

Així amb els dos últims teoremes acabem de demostrar que $\mathcal{L}_{MT}$ és semi-decidible però no decidible.

En el que portem de secció hem definit què són els llenguatges decidibles i semi-decidibles, hem presentat alguns exemples de llenguatges decidibles i hem vist un exemple d'un llenguatge que és semi-decidible però no decidible. A continuació veurem que hi han llenguatges que no són ni tan sols semi-decidibles.

LEMA 2.1. *Si un llenguatge L és semi-decidible i $\Gamma^\star - L$ també, aleshores L és decidible.*

DEMOSTRACIÓ. Per a demostrar que el llenguatge L és decidible sols haurem de presentar una màquina de Turing que el decideix.

Com que L és semi-decidible, hi ha una màquina de Turing $\mathcal{M}_1$ tal que

Per a tota paraula w, w està en L si, i només si, $\mathcal{M}_1$ accepta w.

Com que $\Gamma^\star - L$ és semi-decidible, hi ha una màquina de Turing $\mathcal{M}_2$ tal que

Per a tota paraula w, w està en $\Gamma^\star - L$ si, i només si, $\mathcal{M}_2$ accepta w.

Podem construir ara una nova màquina de Turing $\mathcal{M}_3$ tal que, donada una paraula w, $\mathcal{M}_3$ simule de forma alterna per a cada pas l'acció d'$\mathcal{M}_1$ sobre w i l'acció d'$\mathcal{M}_2$ sobre w. És a dir, construim una màquina de Turing $\mathcal{M}_3$ que simula en paral·lel les màquines $\mathcal{M}_1$ i $\mathcal{M}_2$.

Construim la màquina $\mathcal{M}_3$ tal que si $\mathcal{M}_1$ accepta w, aleshores $\mathcal{M}_3$ accepta w i si $\mathcal{M}_2$ accepta w, aleshores $\mathcal{M}_3$ rebutja w. Com que $w \in L$ o $w \notin L$, tenim garantit que $\mathcal{M}_3$ acabarà sobre w. Així $\mathcal{M}_3$ decideix el llenguatge L. □

Amb l'ajuda d'aquest lema demostrarem el següent teorema.

TEOREMA 2.7. *Hi han llenguatges que no són semi-decidibles.*

DEMOSTRACIÓ. Per a demostrar-ho presentarem un llenguatge que no és semi-decidible i que per tant demostrarà l'existència de llenguatges que no són semi-decidibles.

En resultats anteriors hem vist que $\mathcal{L}_{MT}$ és semi-decidible i que $\mathcal{L}_{MT}$ no és decidible, aleshores pel lema anterior podem concloure que el llenguatge $\Gamma^\star - \mathcal{L}_{MT}$ no és semi-decidible, com volíem. □

4. Màquines de Turing no deterministes

En aquesta secció introduirem les màquines de Turing no deterministes, que són una extensió de les màquines de Turing clàssiques. En aquest context, els processos computacionals no estan restringits a una única seqüència d'operacions determinada, sinó que aquestes màquines permeten múltiples transicions des d'una mateixa configuració, cosa que ens permet modelar el concepte de no determinisme. Aquestes ens serán útils al final del següent capítol.

DEFINICIÓ 2.16. (Màquina de Turing no determinista). *Una* màquina de Turing no determinista *és una 7-tupla*

$$\mathcal{M} = (A, \Gamma, \mathcal{Q}, \delta, q_0, q_a, q_r)$$

on:

A *és un alfabet;*

Γ *és un alfabet de cinta que conté* $A \subseteq \Gamma$ *i un símbol especial* $\square \in \Gamma$ *anomenat final de cinta;*

$\mathcal{Q}$ *és un conjunt d'estats;*

$q_0, q_a, q_r \in \mathcal{Q}$ *i* $q_0 \neq q_a \neq q_r$ *són tres estats especials, d'inici, d'acceptació i de rebuig, respectivament;*

$\delta : \mathcal{Q} \times \Gamma \longrightarrow \mathcal{P}(\mathcal{Q} \times \Gamma \times \{L, R\})$ *és la funció de transició.*

Ací la diferència respecte a les Màquines de Turing és la funció de transició. En aquest cas podem veure que el conjunt d'arribada és el conjunt de parts. Açò significa que des d'una configuració de la màquina podem passar a varies configuracions distintes.

Direm que una Màquina de Turing no determinista accepta una cadena si existeix un camí que arriba a l'estat q_a*. De la mateixa forma direm que una Màquina de Turing no determinista rebutja una cadena si existeix un camí que arriba a l'estat* q_r*.*

Veiem ara un teorema que presenta una relació entre les màquines de Turing no deterministes i les màquines de Turing deterministes.

TEOREMA 2.8. *Tota màquina de Turing no determinista es pot simular per una màquina de Turing determinista. Dit d'altra forma, cada màquina de Turing no determinista té una màquina de Turing determinista equivalent.*

DEMOSTRACIÓ. La prova d'aquest teorema la podem trobar a la Secció 3.2 del llibre [Sip13]. □

CAPíTOL 3

Complexitat

En aquest capítol tractarem conceptes clau de la teoria de la computació i la complexitat. Primer, explorarem la reducció funcional, que permet relacionar la complexitat de diferents llenguatges mitjançant funcions computables. A continuació, introduirem la notació $\mathcal{O}$ per mesurar la complexitat temporal de les màquines de Turing, utilitzant exemples i teoremes per il·lustrar el concepte de dominància asimptòtica. Després estudiarem les classes de complexitat $\mathbf{P}$ i $\mathbf{NP}$, amb exemples de llenguatges i la seua relació. Finalment, abordarem la $\mathbf{NP-completesa}$, explicant les reduccions polinomials i la importància dels llenguatges $\mathbf{NP-complets}$, i com aquests connecten les classes $\mathbf{P}$ i $\mathbf{NP}$.

Aquest capítol proporciona les bases per comprendre i comparar la dificultat dels problemes computacionals.

1. Reducció funcional

En aquesta secció explorarem el concepte de reducció funcional, una eina fonamental en teoria computacional que ens permet relacionar la complexitat de diferents problemes. Una reducció funcional d'un problema A a un problema B és una funció computable que transforma instàncies del problema A en instàncies del problema B, de forma que la solució de la instància del problema B és també solució del problema A original. Això ens permet transferir propietats de decidibilitat i semi-decidibilitat d'un problema a un altre.

Començarem definint formalment les funcions computables i les reduccions funcionals, i després explorarem diversos resultats que mostren com aquestes reduccions ens poden ajudar a entendre millor la natura dels problemes computacionals. Finalment, il·lustrarem aquests conceptes amb un exemple concret de reducció funcional entre dos llenguatges. Aquesta exploració ens donarà una visió de com es poden utilitzar les reduccions per a demostrar propietats importants en diferents classes de problemes.

DEFINICIÓ 3.1. (Funció computable). *Una funció $f : A^\star \longrightarrow A^\star$ és* computable *si existeix una màquina de Turing $\mathcal{M}$ tal que, per a tot w en $A^\star$, si iniciem $\mathcal{M}$ amb w, aleshores $\mathcal{M}$ es deté i el contingut final de la cinta és $f(w)$.*

DEFINICIÓ 3.2. (Reducció funcional). *Siga A un alfabet i $L, K \subseteq A^\star$ llenguatges. Direm que L* redueix *a K si existeix f una funció computable $f : A^\star \longrightarrow A^\star$ tal que, per a tot w en $A^\star$, es té que*

$$w \in L \text{ si, i només si, } f(w) \in K.$$

Denotarem aquest fet per $L \leq_f K$.

El següent teorema estableix una relació important entre la reducció funcional i la decidibilitat dels llenguatges.

TEOREMA 3.1. *Si $L \leq_f K$ i K és decidible, aleshores L és decidible.*

DEMOSTRACIÓ. Com que $L \leq_f K$, aleshores existeix $f : A^\star \longrightarrow A^\star$ una funció computable tal que, per a tot w en $A^\star$, es té que w està en L si, i només si, $f(w)$ està en K.

Com que K és decidible, existeix $\mathcal{R}$, una màquina de Turing, tal que

$$\mathcal{R}(w) = \begin{cases} \text{accepta} & \text{si } w \in K \\ \text{rebutja} & \text{si } w \notin K \end{cases}$$

Definim $\mathcal{S}$ una màquina de Turing tal que $\mathcal{S}(w) = \mathcal{R}(f(w))$. Observem que

- $\mathcal{S}(w)$ accepta si, i només si, $f(w) \in K$. Sabem que $f(w) \in K$ si, i només si, $w \in L$. Aleshores $\mathcal{S}$ accepta w si, i només si, $w \in L$.
- $\mathcal{S}(w)$ rebutja si, i només si, $f(w) \notin K$. Sabem que $f(w) \notin K$ si, i només si, $w \notin L$ Aleshores $\mathcal{S}$ rebutja w si, i només si, $w \notin L$.

Així veiem que la màquina de Turing $\mathcal{S}$ decideix L. És a dir, L és decidible. □

COROL·LARI 3.1. *Si $L \leq_f K$ i L és indecidible, aleshores K és indecidible.*

El següent teorema estableix una relació important entre la reducció funcional i la semi-decidibilitat dels llenguatges.

TEOREMA 3.2. *Si $L \leq_f K$ i K és semi-decidible, aleshores L és semi-decidible.*

DEMOSTRACIÓ. Anàloga a la demostració del teorema anterior. □

COROL·LARI 3.2. *Si $L \leq_f K$ i L no és semi-decidible, aleshores K no és semi-decidible*

La reducció funcional ens pot servir per a demostrar propietats sobre un llenguatge. Ja hem vist als corol·laris anteriors que, si tenim dos llenguatges L i K tals que $L \leq_f K$ i L és indecidible, aleshores K és indecidible (anàleg per a L no reconeixible).

Veiem ara una proposició que relaciona la reducció funcional amb la complementarietat dels llenguatges.

PROPOSICIÓ 3.1. *Si $L \leq_f K$, aleshores $A^\star - L \leq_f A^\star - K$.*

DEMOSTRACIÓ. Si $L \leq_f K$, aleshores existeix una funció computable $f : A^\star \longrightarrow A^\star$ tal que, per a tot w en $A^\star$, es té que w està en L si, i només si, $f(w)$ està en K. Així, siga w en $A^\star - L$, tenim que w està en $A^\star - L$ si, i només si, w no està en L. Per la definició de reducció funcional tenim que w no està en L si, i només si, $f(w)$ no està en K. Finalment podem observar que $f(w)$ no està en K si, i només si, $f(w)$ està en $A^\star - K$.

Acabem de demostrar que $w \in A^\star - L$ si, i només si, $f(w) \in A^\star - K$, que és just la definició de $A^\star - L \leq_f A^\star - K$. □

Veiem ara un exemple en el que presentarem dos llenguatges i demostrarem que un redueix funcionalment a l'altre.

EXEMPLE 3.1. (Exemple de reducció) *Considerem els següents llenguatges*

$$\begin{aligned} ALL_{AFD} &= \{\langle \mathcal{D} \rangle \mid \mathcal{D} \text{ és un AFD tal que } \mathcal{L}(\mathcal{D}) = A^\star\}. \\ Eq_{AFD} &= \{\langle \mathcal{D}_1, \mathcal{D}_2 \rangle \mid \mathcal{D}_1, \mathcal{D}_2 \text{ són AFDs tal que } \mathcal{L}(\mathcal{D}_1) = \mathcal{L}(\mathcal{D}_2)\}. \end{aligned}$$

És a dir, ALL_{AFD} conté les codificacions d'autòmats finits deterministes que reconeixen totes les paraules en $A^\star$, mentre que Eq_{AFD} conté les codificacions de parelles d'autòmats finits deterministes que reconeixen els mateixos llenguatges.

Ara anem a reduir ALL_{AFD} a Eq_{AFD}.

Considerem el següent autòmat finit determinista (Figura 3.1) i notem que $\mathcal{L}(\mathcal{A}) = A^\star$.

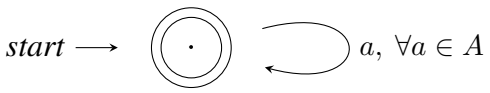

FIGURA 3.1. Autòmat $\mathcal{A}$.

Considerem f la següent aplicació computable:

$$f\colon \quad \begin{array}{ccc} \Gamma^\star & \longrightarrow & \Gamma^\star \\ w & \longmapsto & \begin{cases} \langle \mathcal{D}, \mathcal{A} \rangle & \text{si } w = \langle \mathcal{D} \rangle, \ \mathcal{D} \text{ autòmat finit determinista;} \\ \lambda & \text{altre cas.} \end{cases} \end{array}$$

Una vegada definida la funció f anem a demostrar les dues implicacions de l'afirmació $w \in ALL_{AFD}$ si, i només si, $f(w) \in Eq_{AFD}$.

Siga $w \in ALL_{AFD}$. Per la definició del llenguatge ALL_{AFD} tenim que $w = \langle \mathcal{D} \rangle$, amb $\mathcal{D}$ un autòmat finit determinista i $\mathcal{L}(\mathcal{D}) = A^\star$. En aquest cas $f(w) = \langle \mathcal{D}, \mathcal{A} \rangle$. Així tindrem que $\mathcal{A}$ és un autòmat finit determinista, $\mathcal{D}$ és un autòmat finit determinista i $\mathcal{L}(\mathcal{D}) = A^\star = \mathcal{L}(\mathcal{A})$. Per tant, per la definició del llenguatge, $f(w) \in Eq_{AFD}$, demostrant així que, si $w \in ALL_{AFD}$, aleshores $f(w) \in Eq_{AFD}$.

Siga ara $f(w) \in Eq_{AFD}$, aleshores $f(w) = \langle \mathcal{D}_1, \mathcal{D}_2 \rangle$ amb $\mathcal{D}_1, \mathcal{D}_2$ dos autòmats finits deterministes i $\mathcal{L}(\mathcal{D}_1) = \mathcal{L}(\mathcal{D}_2)$. L'única opció per a que açò passe és que $w = \langle \mathcal{D}_1 \rangle$ i $\mathcal{D}_2 = \mathcal{A}$. Així tindrem que $\mathcal{L}(\mathcal{D}_1) = \mathcal{L}(\mathcal{D}_2) = \mathcal{L}(\mathcal{A}) = A^\star$. Per tant $w \in ALL_{AFD}$, demostrant així que si $f(w) \in Eq_{AFD}$, aleshores $w \in ALL_{AFD}$.

Acabem de demostrar les dues implicacions, $w \in ALL_{AFD}$ si, i només si, $f(w) \in Eq_{AFD}$ i per tant que

$$ALL_{AFD} \leq_f Eq_{AFD}.$$

2. Dominància asimptòtica, notació $\mathcal{O}$

En aquesta secció treballarem amb llenguatges decidibles i ens enfocarem en la mesura de la complexitat en funció de la longitud de les cadenes d'entrada.

Començarem definint formalment la complexitat temporal d'una màquina de Turing que decideix un llenguatge donat. A continuació introduirem el concepte de dominància asimptòtica, que ens permetrà establir relacions d'ordre entre funcions en termes del seu creixement. Mitjançant exemples concrets, il·lustrarem com utilitzar la notació $\mathcal{O}$ per analitzar la complexitat de funcions. Presentarem també diversos resultats que demostren propietats importants de la dominància asimptòtica. Finalment explorarem algunes de les relacions de dominància asimptòtica més rellevants en l'àmbit de la complexitat computacional.

Aquest estudi proporcionarà una base sòlida per comprendre com es poden utilitzar les tècniques de complexitat per avaluar l'eficiència dels algorismes i prendre decisions sobre quines solucions són més pràctiques per a problemes computacionals específics.

DEFINICIÓ 3.3. *Siga L un llenguatge sobre un alfabet d'entrada Γ ($L \subseteq \Gamma^\star$) i $\mathcal{M}$ una màquina de Turing que decideix el llenguatge L, és a dir*

$$\textit{Siga } w \in \Gamma^\star, \quad \mathcal{M}(w) \begin{cases} \textit{accepta} & \textit{si } w \in L \\ \textit{rebutja} & \textit{si } w \notin L \end{cases}$$

Com $\mathcal{M}$ sempre para amb qualsevol entrada, té sentit definir l'aplicació $T_{\mathcal{M}}(w)$, que ens indica el nombre de passos que necessita fer $\mathcal{M}$ per a finalitzar amb l'entrada w.

$$\begin{array}{rccc} T_{\mathcal{M}}: & \Gamma^\star & \longrightarrow & \mathbb{N} \\ & w & \longmapsto & T_{\mathcal{M}}(w) \end{array}$$

Podem pensar en una aplicació que done informació sobre $T_{\mathcal{M}}(w)$ tenint en compte la longitud de w. Presentem diferents opcions:

- *L'aplicació que agafa el millor temps d'entre totes les paraules d'una mateixa longitud:*

$$\begin{array}{rccc} T^{mir}_{\mathcal{M}}: & \mathbb{N} & \longrightarrow & \mathbb{N} \\ & n & \longmapsto & \min\{T_{\mathcal{M}}(w) \in \mathbb{N} \mid |w| = n\} \end{array}$$

- *L'aplicació que agafa el pitjor temps, o temps més alt, d'entre totes les paraules d'una mateixa longitud:*

$$\begin{array}{rccc} T^{pir}_{\mathcal{M}}: & \mathbb{N} & \longrightarrow & \mathbb{N} \\ & n & \longmapsto & \max\{T_{\mathcal{M}}(w) \in \mathbb{N} \mid |w| = n\} \end{array}$$

- *L'aplicació que ens retorna el temps promedi que tarda $\mathcal{M}$ en finalitzar sobre les paraules de longitud n. Aquesta aplicació sols té sentit quan $|\Gamma|$ és finit.*

$$\begin{array}{rccc} T^{pro}_{\mathcal{M}}: & \mathbb{N} & \longrightarrow & \mathbb{R}^+ \\ & n & \longmapsto & \dfrac{\displaystyle\sum_{w \in \Gamma^\star, |w| = n} T_{\mathcal{M}}(w)}{|\Gamma|^n} \end{array}$$

Diferents màquines poden decidir el mateix llenguatge. El temps promedi de cadascuna pot servir per comparar-les. Si podem elegir, preferirem sempre màquines de Turing que tinguen menor temps promedi.

A continuació explicarem com comparar aplicacions de naturals en els reals positius mitjançant la dominància asimptòtica.

DEFINICIÓ 3.4. (Dominància asimptòtica). *Si tenim les aplicacions $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ i $g : \mathbb{N} \longrightarrow \mathbb{R}^+$, direm que g* domina asiptomàticament *a f si existeixen $n_0 \in \mathbb{N}$ i $c \in \mathbb{R}^+$ de forma que, per a tot natural $n \geq n_0$, tenim que $f(n) \leq c \cdot g(n)$. En aquest cas escriurem $f \in \mathcal{O}(g)$.*

Presentem ara alguns exemples molt simples del que acabem de definir.

EXEMPLE 3.2. *$n \in \mathcal{O}(n+5)$ ja que $n \leq c \cdot (n+5)$ per a tot $n \geq n_0$ amb $c = 1, n_0 = 1$. A la mateixa vegada $n + 5 \in \mathcal{O}(n)$ ja que, agafant $c = 2, n_0 = 5$, tenim que $n + 5 \leq 2n$ per a tot $n \geq 5$.*

EXEMPLE 3.3. $sin(n) \in \mathcal{O}(1)$.

Sabem que $sin(n) \leq 1$*, per a tot* $n \in \mathbb{N}$*, aleshores tindrem que* $sin(n)+1 \leq 2$*, per a tot* $n \in \mathbb{N}$*. Així podem agafar* $n_0 = 1, c = 2$ *i es compleix la definició.*

En la teoria de la complexitat, és important entendre com es comporten les funcions de complexitat quan es combinen. El següent lema ens proporciona una propietat útil sobre la suma de dues funcions en termes de dominància asimptòtica.

LEMA 3.1. *Si* $f_1 \in \mathcal{O}(g)$ *i* $f_2 \in \mathcal{O}(g)$*, aleshores* $f_1 + f_2 \in \mathcal{O}(g)$.

DEMOSTRACIÓ. Suposem que $f_1, f_2 \in \mathcal{O}(g)$, és a dir,

Com $f_1 \in \mathcal{O}(g)$, existeixen $c_1 > 0$ i $n_1 \in \mathbb{N}$ tals que $f_1(n) \leq c_1 \cdot g(n)$, per a tot $n \geq n_1$.
Com $f_2 \in \mathcal{O}(g)$, existeixen $c_2 > 0$ i $n_2 \in \mathbb{N}$ tals que $f_2(n) \leq c_2 \cdot g(n)$, per a tot $n \geq n_2$.

Agafem $c = c_1 + c_2$ i $n_0 = \max\{n_1, n_2\}$. Així si $n \geq n_0$, aleshores serà cert que $n \geq n_1$ i $n \geq n_2$. Per tant

$$f_1(n) + f_2(n) \leq c_1 g(n) + c_2 g(n) = (c_1 + c_2)g(n) = c \cdot g(n) \text{ per a tot } n \geq n_0$$

amb $n_0 \in \mathbb{N}$ i $c \in \mathbb{R}^+$. □

Veiem ara alguns exemples de dominància asimptòtica que es poden demostrar fàcilment emprant el lema anterior.

EXEMPLE 3.4. $3n^2 + n \in \mathcal{O}(n^2)$.

Pel lema anterior, seria suficient veure que $3n^2 \in \mathcal{O}(n^2)$ *i* $n \in \mathcal{O}(n^2)$

- $3n^2 \in \mathcal{O}(n^2)$ *ja que* $3n^2 \leq 3 \cdot n^2$ *per a tot* $n \geq 1$*. Hem agafat* $c = 3, n_0 = 1$.
- $n \in \mathcal{O}(n^2)$ *ja que* $n \leq n^2$ *per a tot* $n \geq 1$*. Hem agafat* $c = 1, n_0 = 1$.

Presentem ara una serie de lemes que demostren algunes de les relacions de dominància asimptòtica més rellevants, incloent-hi el producte per un escalar, la transitivitat, l'ordenació de logaritmes, la dominància de les funcions exponencials sobre les polinomials i altres resultats fonamentals.

LEMA 3.2. *Si* $f \in \mathcal{O}(g)$ *i* $\lambda \in \mathbb{R}^+$*, aleshores* $\lambda f \in \mathcal{O}(g)$.

DEMOSTRACIÓ. Com que $f \in \mathcal{O}(g)$, per definició sabem que existeix $c > 0$ i $n_0 \in \mathbb{N}$ tals que $f(n) \leq c \cdot g(n)$ per a tot $n \geq n_0$. Com que $\lambda \in \mathbb{R}^+$ tindrem que $\lambda c \in \mathbb{R}^+$ i

$$\lambda f(n) \leq \lambda c \cdot g(n), \text{ per a tot } n \geq n_0.$$

Com volíem demostrar. □

Abans de continuar, definirem una aplicació polinomial de grau d, que ens permetrà descriure funcions amb un comportament específic.

DEFINICIÓ 3.5. *Una aplicació* $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ *es diu* polinomial de grau d *si es compleix que* $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots a_d n^d$ *amb* $a_d \neq 0$ *i* $d = \delta(f)$ *el grau de* f.

Ara presentem un lema que estableix la relació entre una funció polinomial de grau d i la seua dominància asimptòtica.

LEMA 3.3. *Si f és polinomial de grau d, aleshores $f \in \mathcal{O}(n^d)$.*

DEMOSTRACIÓ. Conseqüència del Lema 3.1. □

Veiem ara un exemple d'un cas de dominància asimptòtica que no es dóna.

EXEMPLE 3.5. $n \notin \mathcal{O}(1)$.

És fàcil de demostrar per reducció a l'absurde. Suposem que $n \in \mathcal{O}(1)$, aleshores existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tal que $n \leq c$ per a tot $n \geq n_0$. Siga $n := \max\{c, n_0\} + 1$, tindrem que $c + 1 \leq n \leq c$ arribant així a una clara contradicció.

Ara explorarem com funciona la transitivitat en el context de la dominància asimptòtica entre funcions.

LEMA 3.4. (Transitivitat). *Si $f \in \mathcal{O}(g)$ i $g \in \mathcal{O}(h)$, aleshores $f \in \mathcal{O}(h)$.*

DEMOSTRACIÓ. Suposem que $f \in \mathcal{O}(g)$ i $g \in \mathcal{O}(h)$, aleshores

Com $f \in \mathcal{O}(g)$, existeixen $c_1 > 0$ i $n_1 \in \mathbb{N}$ tals que $f(n) \leq c_1 \cdot g(n)$, per a tot $n \geq n_1$.
Com $g \in \mathcal{O}(h)$, existeixen $c_2 > 0$ i $n_2 \in \mathbb{N}$ tals que $g(n) \leq c_2 \cdot h(n)$, per a tot $n \geq n_2$.

Agafem $n_0 := \max\{n_1, n_2\}$ i $c = c_1 \cdot c_2$, aleshores

$$f(n) \leq c_1 \cdot g(n) \leq c_1 \cdot c_2 \cdot h(n) = c \cdot h(n), \text{ per a tot } n \geq n_0.$$

Com volíem demostrar. □

Veiem ara com es comporta la dominància asimptòtica respecte al producte.

LEMA 3.5. (Producte). *Si $f_1 \in \mathcal{O}(g_1)$ i $f_2 \in \mathcal{O}(g_2)$, aleshores $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.*

DEMOSTRACIÓ. Si $f_1 \in \mathcal{O}(g_1)$ i $f_2 \in \mathcal{O}(g_2)$, aleshores

Com $f_1 \in \mathcal{O}(g_1)$, existeixen $c_1 > 0$ i $n_1 \in \mathbb{N}$ tals que $f_1(n) \leq c_1 \cdot g_1(n)$, per a tot $n \geq n_1$.
Com $f_2 \in \mathcal{O}(g_2)$, existeixen $c_2 > 0$ i $n_2 \in \mathbb{N}$ tals que $f_2(n) \leq c_2 \cdot g_2(n)$, per a tot $n \geq n_2$.

Agafem $n_0 := \max\{n_1, n_2\}$ i $c = c_1 \cdot c_2$, aleshores

$$f_1(n) \cdot f_2(n) \leq c_1 g_1(n) \cdot c_2 g_2(n) = c \cdot g_1(n) \cdot g_2(n), \text{ per a tot } n \geq n_0.$$

Com volíem demostrar. □

Veiem ara com es comporten els logaritmes en termes de dominància asimptòtica.

LEMA 3.6. (Ordre dels logaritmes) *Si $a, b > 1$, aleshores $log_a(n) \in \mathcal{O}(log_b(n))$.*

LEMA 3.7. (Exponencial domina a lineal) $n \in \mathcal{O}(2^n)$. *Amb açò veiem que $an+b \in \mathcal{O}(c^2)$ amb $c \geq 2$ per a tot $a, b \in \mathbb{R}$.*

LEMA 3.8. (Lineal domina a logarítmica) $log(n) \in \mathcal{O}(n)$.

TEOREMA 3.3. (Exponencial domina a polinomial) *Siga $d \in \mathbb{N}$, $n^d \in \mathcal{O}(2^n)$.*

DEMOSTRACIÓ. Primer que tot demostrarem que $\left(\frac{n}{p}\right)^k \leq (2^{n/p})^k$, per a tot $k \in \mathbb{N}$.

Sabem que $\frac{n}{p} \leq 2^{n/p}$, per tant

$$\left(\frac{n}{p}\right)^k = \left(\frac{n}{p}\right) \cdots \left(\frac{n}{p}\right) \leq (2^{n/p}) \cdots (2^{n/p}) = (2^{n/p})^k.$$

En particular si $k = p$, per a tot $n \geq p$ tindrem que $\frac{n^p}{p^p} \leq 2^p$.

Per tant $n^p \leq 2^n p^p$, per a tot $n \geq p$. Aleshores $n^p \in \mathcal{O}(2^n)$. □

Veiem ara un diagrama (Figura 3.2) en el que podem veure algunes de les dominàncies asimptòtiques que acabem de demostrar.

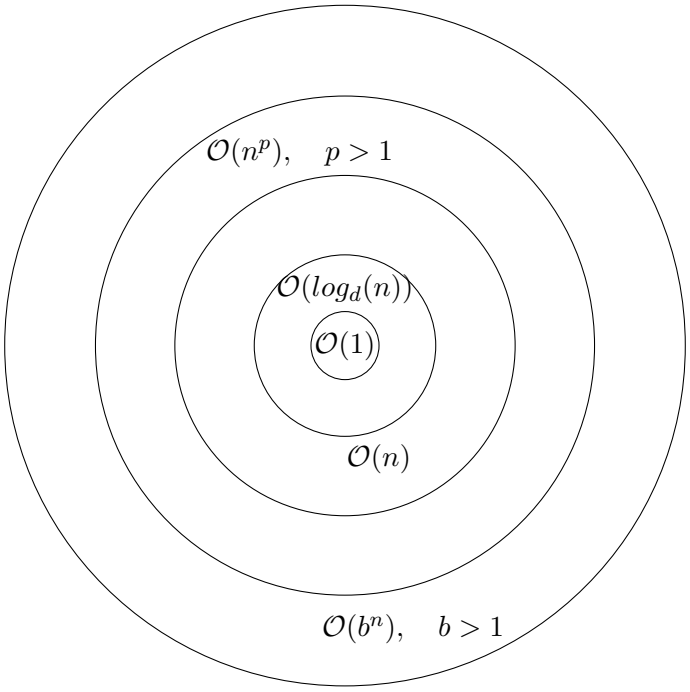

FIGURA 3.2. Diagrama de dominància asimptòtica.

3. Classes P i NP

En aquesta secció explorarem les classes de complexitat **P** i **NP**, que són conjunts de llenguatges amb certes propietats en termes del seu temps de computació. Aquestes classes són fonamentals en la teoria de la complexitat computacional, ja que permeten categoritzar problemes segons la facilitat o dificultat per a ser resolts o verificats per màquines de Turing. Introduirem exemples de llenguatges pertanyents a aquestes classes i discutirem la relació entre elles.

Considerarem a partir d'ara un alfabet A fixe que serà capaç de codificar tots els problemes que plantegem durant aquesta secció.

DEFINICIÓ 3.6. ($\mathrm{Time(T(n))}$). *Si* $T : \mathbb{N} \longrightarrow \mathbb{R}^+$, *definim*

$$\mathrm{Time(T(n))} = \left\{ L \subseteq A^\star \;\middle|\; \begin{array}{l} \exists \mathcal{M} \textit{ màquina de Turing que decideix } L \\ \textit{i té temps d'execució de l'ordre } \mathcal{O}(T(n)) \end{array} \right\}$$

Veiem ara la definició de la classe de llenguatges $\mathbf{P}$

DEFINICIÓ 3.7. (Llenguatges polinomials $\mathbf{P}$). *Definim el conjunt de llenguatges (sobre A) polinomials, denotat per* $\mathbf{P}$*, com*

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathit{Time}(n^k)$$

És a dir, $\mathbf{P}$ *conté tots aquells llenguatges que són decidibles en temps polinomial.*

Veiem ara un exemple de llenguatge que demostrarem que està en $\mathbf{P}$, és a dir, que és decidible en temps polinomial.

DEFINICIÓ 3.8. *(*PATH*). Definim el llenguatge*

$$\text{PATH} = \left\{ \langle \mathcal{G}, v, w \rangle \mid \begin{array}{c} \mathcal{G} \textit{ és un graf dirigit, } v, w \textit{ vèrtexs de } \mathcal{G} \\ \textit{i existeix un camí de } v \textit{ en } w \end{array} \right\}$$

És a dir, PATH *conté la codificació de la terna* $\langle \mathcal{G}, v, w \rangle$ *on* $\mathcal{G}$ *és un graf dirigit,* v *i* w *són dos vèrtexs en* $\mathcal{G}$ *i existeix un camí en* $\mathcal{G}$ *que va de* v *a* w*.*

TEOREMA 3.4. PATH $\in \mathbf{P}$.

DEMOSTRACIÓ. Utilitzem l'algorisme Breadth First Search (BFS).

```
# Donat G, v, w
visitats ={v}
cua=[v]
while not cua.buida()
    x:=cua.desencuar()
    if x=w {return true}
    for each arista x-y en el graf G
        if y no esta en visitats
            cua.encuar(y)
            visitats:=visitats + {y}
return false
```

Explicació de l'algorisme:

(1) Inicialització: S'inicialitza una llista de booleans, *visitats*, que guarda l'estat de cada node del graf. En un principi, tots els nodes estan marcats com a no visitats. A més, s'inicialitza una cua, que s'utilitzarà per realitzar el recorregut BFS.

(2) Inicialització del node inicial: S'estableix que el node inicial v està visitat i s'encua a la cua.

(3) Bucle principal: Mentre la cua no estiga buida, es procedeix amb la següent iteració.

(4) Desencuar un node: En cada iteració, s'extreu el primer element de la cua i es guarda com a x.

(5) Comprovació de destinació: Es comprova si $x = w$. Si ho és, significa que s'ha trobat un camí entre els nodes v i w, i per tant, es retorna vertader.

(6) Exploració dels veïns: Per a cada veí de x, si el veí no ha estat visitat anteriorment, es marca com a visitat i s'encua a la cua.

(7) Finalització: Si es recorre tot el graf i no es troba cap camí entre els nodes v i w, es retorna fals, indicant que no existeix un camí entre aquests dos nodes.

En resum, l'algorisme BFS funciona explorant el graf des del node inicial v de manera sistemàtica i ampliant-se gradualment als nodes veïns. Això es fa mitjançant l'ús d'una cua per gestionar els nodes que s'han de visitar. Si es troba el node destí w, es retorna vertader, si no, es retorna fals.

En total el cost d'aquest algorisme és $\mathcal{O}(n^3)$ (és a dir, ordre polinomial). Per tant tindrem que $\mathrm{PATH} \in \mathrm{Time}(\mathrm{n}^3) \in \mathbf{P}$. □

Veiem ara altre llenguatge que està també en $\mathbf{P}$.

DEFINICIÓ 3.9. *(*RELPRIME*). Definim el següent llenguatge:*

$$\mathrm{RELPRIME} = \{\langle n, m\rangle \mid \mathrm{mcd}(n, m) = 1\}$$

Així RELPRIME *conté la codificació de parells de nombres naturals que són coprimers.*

TEOREMA 3.5. $\mathrm{RELPRIME} \in \mathbf{P}$.

DEMOSTRACIÓ. Per a demostrar aquest teorema emprarem l'algorisme d'Euclides per a calcular el màxim comú divisor.

```
function gcd(n,m)
    if m=0
        return n
    else
        return gcd(m, n mod m) %n mod m es el residu de la divisio de n/m
```

Es pot veure fàcilment que si d divideix a n i a m, aleshores també divideix a n i a $n \bmod m$ i viceversa. Faltaria comprovar que aquest algorisme té complexitat polinomial.

En calcular el màxim comú divisor entre n i m o $\mathrm{mcd}(n, m)$, pot passar el següent:

- Si $n < m$, tindrem que $\mathrm{mcd}(m, n) = \mathrm{mcd}(m, n \bmod m)$.
- Si $n = m$, tindrem que $\mathrm{mcd}(m, n) = \mathrm{mcd}(m, 0) = m$.
- Si $n > m$, tindrem dos casos.
  - El cas en que $\frac{n}{2} \geq m$. En aquest cas tindrem $n \bmod m < m \leq \frac{n}{2}$.
  - El cas en que $\frac{n}{2} < m$. En aquest cas tindrem $n \bmod m < n - m < n - \frac{n}{2} = \frac{n}{2}$.

En resum, tenim que, quan calculem el $\mathrm{mcd}(n, m)$, aquest es transforma en $\mathrm{mcd}(m, n')$ sent $n' = n \bmod m < \frac{n}{2}$ i d'igual forma aquest últim es transformarà en $\mathrm{mcd}(n', m')$ amb $m' = m \bmod n' < \frac{m}{2}$. Així els nombres es van fent cada vegada més xicotets, reduïnt-se a la meitat o fins i tot més.

El temps d'execució d'aquest algorisme és $\mathcal{O}(log_2 n + log_2 m)$ i sabem, per resultats anteriors, que $\mathcal{O}(log_2 n + log_2 m) \subseteq \mathcal{O}(n)$. Per tant podem dir que, com a molt, el temps d'execució d'aquest algorisme serà polinomial. □

Introduïm ara la definició de camí Hamiltonià, que necessitarem per a definir un altre llenguatge.

DEFINICIÓ 3.10. (Camí Hamiltonià). *En un graf dirigit $\mathcal{G}$, un* camí hamiltonià *és un camí dirigit que passa per tots els vèrtes de $\mathcal{G}$ exactament una vegada.*

Veiem ara un exemple simple.

EXEMPLE 3.6. *Siga el següent graf dirigit, marquem en color roig el que podria ser un camí Hamiltonià.*

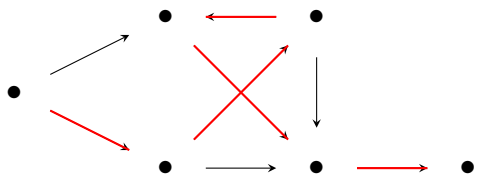

FIGURA 3.3. Exemple de camí Hamiltonià.

Veiem ara la definició del llenguatge HAMPATH.

DEFINICIÓ 3.11. (HAMPATH). *Definim el següent llenguatge:*

$$\text{HAMPATH} = \left\{ \langle \mathcal{G}, v, w \rangle \mid \begin{array}{l} \mathcal{G} \text{ és un graf dirigit i} \\ \text{existeix un camí Hamiltonià que va de } v \text{ a } w \end{array} \right\}$$

És a dir, HAMPATH *conté la codificació de la terna $\langle \mathcal{G}, v, w \rangle$, on $\mathcal{G}$ és un graf dirigit, v i w són vèrtexs en $\mathcal{G}$ i existeix un camí Hamiltonià en $\mathcal{G}$ que comença en v i acaba en w.*

NOTA 3.1. *D'aquest llenguatge el que podem dir és que es pot decidir en temps exponencial (aquesta afirmació està justificada amb més deteniment a la Secció 7.3 del llibre [Sip13]) però no sabem si es pot decidir en temps polinomial. És a dir, no podem dir si* HAMPATH *està o no en* $\mathbf{P}$. *En canvi el que sí que podem fer és comprovar en temps polinomial si un camí concret d'un graf és Hamiltonià.*

Veiem ara la definició de verificador, el que ens permetrà comprovar o verificar en temps polinomial un llenguatge, concepte que ja hem avançat a la Nota 3.1.

DEFINICIÓ 3.12. (Verificador). *Un verificador per a un llenguatge $L \subseteq A^\star$ és una màquina de Turing $\mathcal{V}$ tal que*

$$L = \{w \mid \textit{ existeix } c \in A^\star \textit{ tal que } \mathcal{V}(\langle w, c \rangle) = \textit{ accepta}\}$$

És a dir, direm que una màquina de Turing $\mathcal{V}$ és un verificador per al llenguatge L si per a tota paraula $w \in L$ existeix $c \in A^\star$ tal que $\mathcal{V}(\langle w, c \rangle)$ arriba a un estat d'acceptació. Per a un $w \in L$, la paraula c per la que $\mathcal{V}(\langle w, c \rangle)$ accepta, s'anomena el certificat *de w.*

Medim el temps d'execució d'un verificador en funció del tamany de w ignorant el certificat. Així un verificador té temps d'execució polinomial si el seu temps d'execució és polinomial.

Direm que un llenguatge $L \subseteq A^\star$ és polinomialment verificable *si té un verificador en temps polinomial.*

Una vegada definits aquests conceptes podem definir el següent conjunt de llenguatges.

DEFINICIÓ 3.13. *(*$\mathbf{NP}$*) Definim la classe* $\mathbf{NP}$

$$\mathbf{NP} = \{L \subseteq A^\star \mid L \textit{ polinomialment verificable}\}$$

És a dir, $\mathbf{NP}$ *conté tots aquells llenguatges que són verificables en temps polinomial. És a dir, aquells llenguatges que es poden acceptar si ens donen un certificat concret.*

Veiem ara un teorema que demostra que el llenguatge $\mathrm{HAMPATH}$ és polinomialment verificable.

TEOREMA 3.6. $\mathrm{HAMPATH} \in \mathbf{NP}$.

DEMOSTRACIÓ. Si existeix un camí Hamiltonià en un graf dirigit $\mathcal{G}$ que va de v a w, podem passar-lo llistat com a certificat. Construirem la màquina de Turing $\mathcal{V}$ com segueix:

Per a l'entrada $\langle\langle \mathcal{G}, v, w\rangle, \langle x_1, \ldots x_n\rangle\rangle$

- Comprovem primer que $x_1 = v$ i que $x_n = w$.
- Comprovem que no hi ha elements repetits en la llista.
- Comprovem que tots els elements de la llista estan en $\mathcal{G}$ i que són adjacents.
- Finalment comprovem que tots els elements de $\mathcal{G}$ estan en la llista.

Si totes aquestes condicions es compleixen, aleshores acceptem. Al contrari, rebutgem.

Per tant $\mathrm{HAMPATH}$ és un llenguatge polinomialment verificable. □

Relacionem ara $\mathbf{P}$ i $\mathbf{NP}$.

PROPOSICIÓ 3.2. *Si un llenguatge està en* $\mathbf{P}$*, aleshores també està en* $\mathbf{NP}$*. És a dir,* $\mathbf{P} \subseteq \mathbf{NP}$.

DEMOSTRACIÓ. Suposem $L \in \mathbf{P}$, és a dir, tenim una màquina de Turing $\mathcal{M}$ que decideix L en temps polinomial. Construim un verificador polinomial per a L com segueix: $\mathcal{V}(w, \lambda)$ ignora la paraula buida (el verificador) i simula $\mathcal{M}(w)$. Notem que $\mathcal{V}$ té el mateix temps d'execució que $\mathcal{M}$. Així el verificador és de temps polinomial i verifica el llenguatge L, per tant podem dir que L està en $\mathbf{NP}$. □

NOTA 3.2. *La classe* $\mathbf{P}$ *inclou llenguatges que poden ser decidits per una màquina de Turing en temps polinomial, mentre que la classe* $\mathbf{NP}$ *inclou aquells llenguatges que poden ser verificats en temps polinomial per una màquina de Turing. La relació* $\mathbf{P} \subseteq \mathbf{NP}$ *ens indica que qualsevol llenguatge que es pot decidir en temps polinomial també es pot verificar en temps polinomial.*

Per a finalitzar la secció veurem altre exemple de llenguatge en $\mathbf{NP}$.

DEFINICIÓ 3.14. *En un graf no dirigit, una* clique *és un subgraf tal que, per a tot parell de vèrtexs en la clique, hi ha una aresta entre ells.*

Veiem un exemple del que acabem de definir.

EXEMPLE 3.7. *En verd indiquem la clique.*

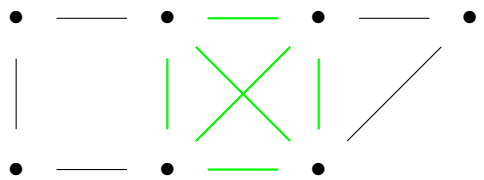

FIGURA 3.4. Exemple de clique.

Definim el llenguatge CLIQUE, que després demostrarem que està en la classe $\mathbf{NP}$.

DEFINICIÓ 3.15. *(*CLIQUE*). Definim el següent llenguatge*

$$\text{CLIQUE} = \left\{ \langle \mathcal{G}, k \rangle \mid \begin{array}{l} \mathcal{G} \textit{ és un graf no dirigit que admet} \\ \textit{una clique de tamany } k \in \mathbb{N} \end{array} \right\}$$

És a dir, CLIQUE *conté la codificació de tots aquells parells $\langle \mathcal{G}, k \rangle$ on $\mathcal{G}$ és un graf no dirigit i k és el tamany d'una clique en $\mathcal{G}$.*

TEOREMA 3.7. $\text{CLIQUE} \in \mathbf{NP}$.

DEMOSTRACIÓ. El certificat pot ser una llista contenint tots els vèrtexs de la clique. Construïm el verificador $\mathcal{V}$ com segueix:

Amb l'entrada $\langle \langle \mathcal{G}, k \rangle, \langle x_1, \dots x_n \rangle \rangle$ caldrà comprovar:

- Que la llista $x_1, \dots x_n$ és una llista de vèrtexs en $\mathcal{G}$.
- Dos a dos, els vèrtexs $x_1, \dots x_n$ són adjacents en $\mathcal{G}$.
- La llista té k elements.

Si totes aquestes condicions es compleixen, aleshores acceptem. Al contrari, rebutgem.

Per tant CLIQUE és un llenguatge polinomialment verificable. □

4. NP-completesa

En aquesta última secció introduirem el concepte de reducció polinomial entre llenguatges i explorarem com aquesta es relaciona amb la reducció funcional, ja definida al principi d'aquest capítol. Després d'haver definit les reduccions polinomials, presentarem la noció de llenguatge $\mathbf{NP-complet}$. Explorarem alguns resultats clau que ens ajudaran a entendre millor les relacions entre les classes $\mathbf{P}$ i $\mathbf{NP}$ a través de la reducció polinomial i la $\mathbf{NP-completesa}$. Provar que un llenguatge és $\mathbf{NP-complet}$ pot ser un procés complex, però una vegada identificat un llenguatge $\mathbf{NP-complet}$, podem utilitzar-lo com a punt de

referència per a demostrar la $\mathbf{NP-completesa}$ d'altres llenguatges. Finalment, presentarem un teorema que estableix la connexió entre els llenguatges en la classe $\mathbf{NP}$ i les màquines de Turing no deterministes, demostrant que un llenguatge és a $\mathbf{NP}$ si, i només si, existeix una màquina de Turing no determinista que decideix aquest llenguatge en temps polinomial.

NOTA 3.3. *Per a començar recordarem la definició de reducció funcional presentada en seccions anteriors: Siguen L i K dos llenguatges sobre un mateix alfabet, direm que L redueix funcionalment a K, i ho denotarem per $L \leq_f K$ si existeix $f : A^\star \longrightarrow A^\star$ una funció computable tal que, per a tot w en $A^\star$, w està en L si, i només si, $f(w)$ està en K.*

DEFINICIÓ 3.16. *(*Computable en temps polinomial*). Siga $f : A^\star \longrightarrow A^\star$, direm que f és* computable en temps polinomial *si existeix $\mathcal{M}$, una màquina de Turing que sempre acaba, té temps d'execució polinomial i a més es compleix que, per a tot w en $A^\star$, $\mathcal{M}(w)$ finalitza amb la paraula $f(w)$ escrita a la cinta.*

DEFINICIÓ 3.17. *(*Reducció polinomial*). Donats dos llenguatges $L, K \subseteq A^\star$, direm que L* redueix polinomialment *a K, i escriurem $L \leq_p K$, si existeix $f : A^\star \longrightarrow A^\star$ computable en temps polinomial, tal que, per a tot w en $A^\star$, w està en L si, i només si, $f(w)$ està en K. És a dir, si per a tot $w \in A^\star$,*

$$w \in L \text{ si, i només si, } f(w) \in K.$$

NOTA 3.4. *Notem que si $L \leq_p K$, aleshores $L \leq_f K$.*

NOTA 3.5. *Si $L \leq_p K$ i K és decidible, aleshores L és decidible.*

A continuació presentarem un teorema important relacionat amb la reducció polinomial i la classe de problemes que es poden resoldre en temps polinomial.

TEOREMA 3.8. *Si $L \leq_p K$ i $K \in \mathbf{P}$, aleshores $L \in \mathbf{P}$.*

DEMOSTRACIÓ. Suposem que $L \leq_p K$. Això significa que existeix una funció $f : A^\star \longrightarrow A^\star$ computable en temps polinomial tal que, per a tot $w \in A^\star$, $w \in L$ si, i només si, $f(w) \in K$.

Sabem que $K \in \mathbf{P}$, és a dir, existeix una màquina de Turing $\mathcal{M}$ que decideix K en temps polinomial. Formalment,

$$\mathcal{M}(w) = \begin{cases} \text{accepta} & \text{si } w \in K \\ \text{rebutja} & \text{si } w \notin K \end{cases}$$

Construirem una màquina de Turing $\mathcal{M}'$ que decidisca L en temps polinomial. La màquina $\mathcal{M}'$ funciona de la següent manera:

Per a una entrada $w \in A^\star$:

(1) Calcula $f(w)$ en temps polinomial $\mathcal{O}(n^k)$.

(2) Utilitza $\mathcal{M}$ per decidir si $f(w) \in K$ en temps polinomial $\mathcal{O}(n^r)$.

La màquina $\mathcal{M}'$ accepta w si, i només si, $\mathcal{M}(f(w))$ accepta. Això es compleix si, i només si, $f(w) \in K$, i per definició de f, això es compleix si, i només si, $w \in L$.

El temps total de computació de $\mathcal{M}'$ és la composició de dos polinomis, que també és un polinomi. Així, $\mathcal{M}'$ funciona en temps $\mathcal{O}(n^{kr})$, que és un temps polinomial. Per tant, hem construït una màquina de Turing $\mathcal{M}'$ que decideix L en temps polinomial, i així, $L \in \mathbf{P}$. $\square$

Veiem ara la definició de $\mathbf{NP-completesa}$ que és un cas encara més fort de llenguatges en $\mathbf{NP}$.

DEFINICIÓ 3.18. *($\mathbf{NP-complet}$). Direm que un llenguatge $K \subseteq A^\star$ és $\mathbf{NP-complet}$ si es compleix aquestes dues condicions:*

(1) $K \in \mathbf{NP}$.

(2) *per a tot $L \subseteq A^\star$, si $L \in \mathbf{NP}$, aleshores $L \leq_p K$ (L redueix polinomialment a K).*

Si K satisfà (2), diem també que K és $\mathbf{NP}$-difícil.

Demostrarem ara un teorema que afirma que, si podem trobar un llenguatge que siga $\mathbf{NP-complet}$ i també estiga en la classe $\mathbf{P}$, aleshores haurem demostrat que $\mathbf{P} = \mathbf{NP}$.

TEOREMA 3.9. *Si $K \subseteq A^\star$ és $\mathbf{NP-complet}$ i $K \in \mathbf{P}$, aleshores $\mathbf{P} = \mathbf{NP}$.*

DEMOSTRACIÓ. Veiem les dues inclusions per a demostrar la igualtat:

$\subseteq$) Ja l'hem vist a la secció anterior.

$\supseteq$) Siga $L \in \mathbf{NP}$, com que K és $\mathbf{NP-complet}$, tindrem per definició que $L \leq_p K$. Tenim per tant que $L \leq_p K$ i $K \in \mathbf{P}$, aleshores pel Teorema 3.8, tindrem que $L \in \mathbf{P}$. □

Ací podem observar que la importància dels llenguatges $\mathbf{NP-complets}$ està en el fet que, si trobarem un algorisme en temps polinomial per decidir qualsevol d'ells, això implicaria que tots els problemes en $\mathbf{NP}$ podrien ser decidits en temps polinomial, és a dir, $\mathbf{P} = \mathbf{NP}$.

Presentem una última proposició abans del teorema amb el que finalitzarem el capítol.

PROPOSICIÓ 3.3. *Siguen dos llenguatges $L, K \subseteq A^\star$ tals que*

1) *L és $\mathbf{NP-complet}$.*

2) *$L \leq_p K$.*

3) *K és $\mathbf{NP}$.*

Aleshores K és $\mathbf{NP-complet}$.

DEMOSTRACIÓ. Per hipòtesi tenim que K és $\mathbf{NP}$. Quedarà per demostrar la segona condició que fa que un llenguatge siga $\mathbf{NP-complet}$. Haurem de demostrar que, per a tot llenguatge $J \subseteq A^\star$ que està en la classe $\mathbf{NP}$, es compleix que $J \leq_p K$.

Suposem per tant que tenim un llenguatge $J \subseteq A^\star$ que està en la classe $\mathbf{NP}$. Com que L és $\mathbf{NP-complet}$ tindrem que $J \leq_p L$, i per hipòtesi tenim que $L \leq_p K$.

Com $J \leq_p L$, per definició de reducció polinomial tenim que existeix $g : A^\star \longrightarrow A^\star$ una funció computable en temps polinomial tal que, una paraula w està en J si, i només si, $g(w)$ està en L. D'igual forma, com que $L \leq_p K$, per definició de reducció polinomial, tenim que existeix $f : A^\star \longrightarrow A^\star$ computable en temps polinomial, tal que $g(w)$ està en L si, i només si, $f(g(w))$ està en K.

Si agafem $f \circ g : A^\star \longrightarrow A^\star$ (computable en temps polinomial per ser composició de dues funcions computables en temps polinomial), tindrem que una paraula w està en J si, i només

si, $(f \circ g)(w)$ està en K, demostrant així $J \leq_p K$. Per tant es compleix la segona condició i podem dir que K és un llenguatge $\mathbf{NP-complet}$. □

Finalment, anem a veure un teorema que ens permetrà demostrar, d'una forma alternativa a la que ja hem vist, que un llenguatge està en la classe $\mathbf{NP}$.

TEOREMA 3.10. *Un llenguatge $L \subseteq A^\star$ està en $\mathbf{NP}$ si, i sols si, existeix una màquina de Turing no determinista $\mathcal{N}$ que decideix L en temps polinomial.*

DEMOSTRACIÓ. Per a demostrar l'equivalència demostrarem les dues implicacions.

Suposem que $L \in \mathbf{NP}$, és a dir, suposem que existeix $\mathcal{V}$ un verificador polinomial per a L. Suposem que, per a una entrada w amb longitud n, $\mathcal{V}$ acaba en n^k passos. Construïm una màquina de Turing $\mathcal{N}$ que decideix L en temps polinomial com segueix. Siga $w \in L$, aleshores

- Elegim no determinísticament un certificat c de com a màxim tamany n^k.
- Simulem $\mathcal{V}(w, c)$.

Suposem que existeix una màquina de Turing no determinista $\mathcal{N}$ que decideix L en temps polinomial. Agafarem com a certificat c el camí que condueix desde l'inici fins a la configuració en la què està l'estat q_a. Construïm el verificador $\mathcal{V}$ com segueix:

- Simulem $\mathcal{M}(w)$. En cada transició elegim l'opció indicada pel certificat c.
- Si s'arriba a q_a, acceptar. Si no, rebutjar. □

COROL·LARI 3.3. *Si $L \in \mathbf{NP}$, aleshores L és un llenguatge decidible per una màquina de Turing determinista.*

DEMOSTRACIÓ. Conseqüència directa del Teorema 2.8 i del Teorema 3.10. □

CAPíTOL 4

Teorema de Cook-Levin

En aquest capítol explorarem un dels resultats fonamentals de la teoria de la complexitat computacional: el Teorema de Cook-Levin, que ens diu que un problema de la lògica proposicional, el de la satisfacibilitat booleana, és $\mathbf{NP-complet}$. Començarem amb una breu introducció a la lògica proposicional. A continuació enunciarem i demostrarem el Teorema de Cook-Levin i per a finalitzar el capítol veurem alguns resultats que són conseqüència directa d'aquest teorema.

1. Introducció a la lògica proposicional

En aquesta secció farem una breu introducció a la lògica proposicional, que és fonamental per a comprendre el tema central d'aquest capítol. Presentarem els conceptes bàsics de les Δ-àlgebres, que inclouen les operacions de negació, conjunció i disjunció. A més, definirem les valoracions, que són aplicacions que assignen valors de veritat a les variables proposicionals. També explorarem les propietats de les Δ-àlgebres lliures i com aquestes es poden utilitzar per a modelar la lògica proposicional. Per a redactar aquesta secció hem fet servir el llibre [Fer04].

NOTA 4.1. *Considerem el conjunt de símbols d'operació* $\Delta = \{\neg, \wedge, \vee\}$ *on :*

$\neg$ *és un símbol d'operació unària anomenat negació.*

$\wedge$ *és un símbol d'operació binària anomenat conjunció.*

$\vee$ *és un símbol d'operació binària anomenat disjunció.*

DEFINICIÓ 4.1. *(*Δ-àlgebra*). Una* Δ-àlgebra *és una tupla* $(A, \neg, \wedge, \vee)$ *on:*

A *és un conjunt.*

$\neg : A \longrightarrow A$ *és l'interpretació de la negació com a operació unària en A.*

$\wedge : A \times A \longrightarrow A$ *és l'interpretació de la conjunció com a operació binària en A.*

$\vee : A \times A \longrightarrow A$ *és l'interpretació de la disjunció com a operació binària en A.*

Escriurem $\mathbf{A}$ *per a referir-nos a la tupla* $(A, \neg, \wedge, \vee)$ *quan l'interpretació de les operacions estiga clara.*

DEFINICIÓ 4.2. *Siguen* $\mathbf{A} = (A, \neg, \wedge, \vee)$ *i* $\mathbf{B} = (B, \neg, \wedge, \vee)$ *dues* Δ*-àlgebres. Un* Δ-homomorfisme *d'*$\mathbf{A}$ *a* $\mathbf{B}$ *és una aplicació* $f : A \longrightarrow B$ *que satisfà:*

$$f(\neg a) = \neg f(a), \qquad f(a \wedge a') = f(a) \wedge f(a'), \qquad f(a \vee a') = f(a) \vee f(a').$$

Escriurem $f : \mathbf{A} \longrightarrow \mathbf{B}$ *per a denotar els* Δ*-homomorfismes.*

EXEMPLE 4.1. *Exemple d'una Δ-àlgebra.*

$\mathbf{2} = (2, \neg, \wedge, \vee)$ *on* $2 = \{0, 1\}$ *i les interpretacions de les operacions venen donades per*

	0	1
$\neg$	1	0

TAULA 4.1. Negació.

$\wedge$	0	1
0	0	0
1	0	1

TAULA 4.2. Conjunció.

$\vee$	0	1
0	0	1
1	1	1

TAULA 4.3. Disjunció.

Veiem ara la definició de Δ-àlgebra lliure.

DEFINICIÓ 4.3. *(Δ-àlgebra lliure). Siga $\mathcal{V}$ un conjunt els elements del qual anomenarem variables i que suposarem disjunt de Δ. Les paraules sobre el conjunt $\mathcal{V} \cup \Delta$, és a dir, $(\mathcal{V} \cup \Delta)^\star$, té estructura de Δ-àlgebra, amb la següent interpretació de les operacions.*

$$\begin{array}{rccc} \neg: & (\mathcal{V} \cup \Delta)^\star & \longrightarrow & (\mathcal{V} \cup \Delta)^\star \\ & P & \longmapsto & \neg \curlywedge P \end{array}$$

$$\begin{array}{rccc} \wedge: & (\mathcal{V} \cup \Delta)^\star \times (\mathcal{V} \cup \Delta)^\star & \longrightarrow & (\mathcal{V} \cup \Delta)^\star \\ & (P, Q) & \longmapsto & P \curlywedge \wedge \curlywedge Q \end{array}$$

$$\begin{array}{rccc} \vee: & (\mathcal{V} \cup \Delta)^\star \times (\mathcal{V} \cup \Delta)^\star & \longrightarrow & (\mathcal{V} \cup \Delta)^\star \\ & (P, Q) & \longmapsto & P \curlywedge \vee \curlywedge Q \end{array}$$

Anomenem Δ-àlgebra lliure generada per $\mathcal{V}$ a la Δ-subalgebra de $(\mathcal{V} \cup \Delta)^\star$ generada per $\mathcal{V}$. A aquesta la denotarem per $\mathbf{T}_\Delta(\mathcal{V})$ i per $\mathrm{T}_\Delta(\mathcal{V})$ al conjunt subjacent.

NOTA 4.2. *Notem que un element R de $\mathrm{T}_\Delta(\mathcal{V})$ sols pot ser d'una de les següents 4 formes (diferents dos a dos):*

$R = x$, per a una variable x de $\mathcal{V}$.

$R = \neg P$, per a un únic terme P de $\mathrm{T}_\Delta(\mathcal{V})$.

$R = P \wedge Q$, per a uns únics termes P, Q de $\mathrm{T}_\Delta(\mathcal{V})$.

$R = P \vee Q$, per a uns únics termes P, Q de $\mathrm{T}_\Delta(\mathcal{V})$.

Veiem ara un exemple del que acabem de definir.

EXEMPLE 4.2. *Siga $\mathcal{V} = \{x, y, z\}$. Aleshores $\mathrm{T}_\Delta(\mathcal{V})$ conté per exemple als termes $x, (x \wedge (\neg z)) \vee (((\neg y) \wedge x), x \wedge y, \neg y, x \vee y$.*

Una de les propietats més importants de les Δ-àlgebres lliures és la seua propietat universal, que permet extendre qualsevol aplicació definida sobre $\mathcal{V}$ a un homomorfisme de Δ-àlgebres de manera única. A continuació, presentem el teorema que estableix aquesta propietat d'existència i unicitat de l'homomorfisme.

TEOREMA 4.1. (Propietat universal de la Δ-àlgebra lliure). *Siga $\mathcal{V}$ un conjunt de variables i considerem la Δ-àlgebra lliure sobre $\mathcal{V}$, $\mathbf{T}_\Delta(\mathcal{V})$. Siga $\mathbf{A} = (A, \neg, \wedge, \vee)$ una Δ-àlgebra, i siga $f : \mathcal{V} \to A$ una aplicació. Aleshores existeix un únic Δ-homomorfisme $f^\sharp : \mathbf{T}_\Delta(\mathcal{V}) \longrightarrow \mathbf{A}$ satisfent que $f^\# \circ \mathrm{in}_\mathcal{V} = f$, on $\mathrm{in}_\mathcal{V} : \mathcal{V} \longrightarrow \mathrm{T}_\Delta(\mathcal{V})$ és la inserció canònica de generadors que envia una variable x de $\mathcal{V}$ a la paraula de longitud 1 que sols té la lletra x.*

DEMOSTRACIÓ. Definim $f^{\#}$ per recursió com segueix:

$f^{\#}(x) = f(x)$, on x es una variable de $\mathcal{V}$.

$f^{\#}(\neg P) = \neg f^{\#}(P)$, per a $P \in T_{\Delta}(\mathcal{V})$.

$f^{\#}(P \wedge Q) = f^{\#}(P) \wedge f^{\#}(Q)$, per a tot terme P i Q de $T_{\Delta}(\mathcal{V})$.

$f^{\#}(P \vee Q) = f^{\#}(P) \vee f^{\#}(Q)$, per a tot terme P i Q de $T_{\Delta}(\mathcal{V})$.

És fàcil comprovar que $f^{\#}$ és l'unic Δ-homomorfisme que satisfà $f^{\#} \circ \mathrm{in}_{\mathcal{V}} = f$. □

En l'estudi de les Δ-àlgebres, les valoracions són eines que permeten avaluar expressions dins d'aquestes estructures. Una valoració assigna a cada variable un valor en el conjunt $\{0, 1\}$. A continuació, presentem formalment la definició de valoració.

DEFINICIÓ 4.4. (Valoracions). *Siga $\mathcal{V}$ un conjunt de variables. Una valoració (la de l'exemple 4.1) és una aplicació del tipus $v : \mathcal{V} \longrightarrow 2$. Com $\mathbf{2} = (2, \neg, \wedge, \vee)$ té estructura de Δ-àlgebra, sabem que tota valoració indueix un Δ-homomorfisme $v^{\#} : T_{\Delta}(\mathcal{V}) \longrightarrow \mathbf{2}$.*

EXEMPLE 4.3. *(Continuació de l'exemple anterior) Siga $\mathcal{V} = \{x, y, z\}$. Considerem la següent valoració:*

$$\begin{aligned} v : \mathcal{V} &\longrightarrow 2 \\ x &\longmapsto 0 \\ y &\longmapsto 0 \\ z &\longmapsto 1 \end{aligned}$$

Aquesta valoració determina de forma única una valoració sobre les fórmules $x,(x \wedge (\neg z)) \vee (((\neg y) \wedge x), x \wedge y, \neg y, x \vee y$, com segueix

- $v^{\#}(x) = 0$.
- $v^{\#}((x \wedge (\neg z)) \vee ((\neg y) \wedge x)) = v^{\#}(x \wedge (\neg z)) \vee v^{\#}((\neg y) \wedge x)$
  $= (v^{\#}(x) \wedge \neg v^{\#}(z)) \vee (\neg v^{\#}(y) \wedge v^{\#}(x)) = (0 \wedge \neg 1) \vee (\neg 0 \wedge 0) = 0 \wedge 0 \vee 1 \wedge 0 = 0 \vee 0 = 0$.
- $v^{\#}(x \wedge y) = v^{\#}(x) \wedge v^{\#}(y) = 0 \wedge 0 = 0$.
- $v^{\#}(\neg y) = \neg v^{\#}(y) = \neg 0 = 1$.
- $v^{\#}(x \vee y) = v^{\#}(x) \vee v^{\#}(y) = 0 \vee 0 = 0$.

DEFINICIÓ 4.5. *Siga $\mathcal{V}$ un conjunt de variables i siga P una fórmula en $T_{\Delta}(\mathcal{V})$. Direm que*

(1) *P és una tautologia si, per a tota valoració $v : \mathcal{V} \to 2$, es té que $v^{\#}(P) = 1$.*

(2) *P és una contradicció si, per a tota valoració $v : \mathcal{V} \longrightarrow 2$, es té que $v^{\#}(P) = 0$.*

(3) *P és contingent si existeix una valoració $v : \mathcal{V} \to 2$ tal que $v^{\#}(P) = 1$ i existeix $w : \mathcal{V} \to 2$ tal que $w^{\#}(P) = 0$.*

(4) *P és satisfactible si existeix una valoració $v : \mathcal{V} \to 2$ tal que $v^{\#}(P) = 1$.*

EXEMPLE 4.4. *Veiem a continuació alguns exemples del que acabem de definir*

- *$x \wedge \neg x$ és una contradicció.*

- $x \vee \neg x$ *és una tautologia.*

- $(x \wedge y) \vee z$ *és satisfactible per a la valoració de l'Exemple 4.3. A més la següent valoració la fa falsa*

$$\begin{aligned} w : \mathcal{V} &\longrightarrow 2 \\ x &\longmapsto 1 \\ y &\longmapsto 0 \\ z &\longmapsto 0 \end{aligned}$$

Per tant $(x \wedge y) \vee z$ *és també contingent.*

2. Teorema de Cook-Levin

En aquesta secció, ens centrarem en el Teorema de Cook-Levin, que estableix que el problema de la satisfacibilitat booleana (SAT) és $\mathbf{NP-complet}$, i per tant, és un dels problemes més difícils dins de la classe $\mathbf{NP}$. Començarem definint el llenguatge SAT i veurem que aquest llenguatge és decidible. A continuació, enunciarem i demostrarem el Teorema de Cook-Levin.

A la secció anterior hem vist que una fórmula lògica és satisfactible si podem trobar una assignació de variables a valors boleans que fan vertadera la fórmula. Definim ara el llenguatge SAT.

DEFINICIÓ 4.6. (SAT). *El llenguatge* SAT *es defineix així:*

$$\mathrm{SAT} = \{\langle\varphi\rangle \mid \varphi \textit{ és un fórmula satisfactible } \}$$

És a dir, SAT *conté la codificació d'aquelles fórmules que són satisfactibles.*

PROPOSICIÓ 4.1. *El llenguatge* SAT *és decidible.*

DEMOSTRACIÓ. La decidibilitat del problema SAT es demostra veient que es pot construir un algorisme per determinar si una fórmula booleana determinada es pot fer vertadera assignant valors a les seues variables. Aquest algorisme explora sistemàticament totes les assignacions possibles de valors booleans a les variables mitjançant una cerca exhaustiva.

Veiem com actauria la màquina de Turing $\mathcal{M}$ sobre una fórmula d'entrada φ:

(1) Enumerem totes les possibles assignacions de valors vertader/fals a les variables de la fórmula φ.

(2) Avaluem la fórmula: Per a cada assignació de valors a les variables, s'avalua la fórmula booleana.

(3) Acceptació/Rebuig: Si alguna assignació fa que la fórmula siga veritat, acceptem (la fórmula és satisfactible); si cap assignació ho fa, rebutgem (la fórmula no és satisfactible).

Aquest procés és finit, ja que el nombre de variables en la fórmula φ és finit, i per tant, el nombre de possibles assignacions també ho és. La màquina de Turing que implementa aquest algorisme sempre acabarà en un temps finit, demostrant així que SAT és decidible. □

NOTA 4.3. *Notem que l'algorisme que hem donat en la demostració de la Proposició 4.1 és exponencial, ja que per a una fórmula* φ *amb* n *variables, cal fer* 2^n *comprovacions.*

Enunciem i demostrem ara el teorema de Cook-Levin, que ens diu que el problema de la satisfacibilitat booleana és $\mathbf{NP-complet}$.

TEOREMA 4.2. (Cook-Levin). SAT *és un llenguatge* $\mathbf{NP-complet}$.

DEMOSTRACIÓ. Per a veure que SAT és un llenguatge $\mathbf{NP-complet}$ hem de veure que compleix la definició. La demostració es divideix en dues parts: primer, veurem que SAT està en $\mathbf{NP}$, i segon, demostrarem que qualsevol llenguatge en $\mathbf{NP}$ pot reduir-se polinomialment a SAT, és a dir, si $L \subseteq A^\star$ i $L \in \mathbf{NP}$, aleshores $L \leq_p \text{SAT}$.

Començarem demostrant $\text{SAT} \in \mathbf{NP}$. Veiem que SAT és polinomialment verificable. Volem veure que que existeix un verificador $\mathcal{V}$ tal que $\langle\varphi\rangle \in \text{SAT}$ si, i només si, existeix c tal que $\mathcal{V}(\langle\varphi\rangle, c) =$ accepta. En aquest cas, el certificat serà la llista de variables en la fórmula que han de ser vertaderes per a que la fórmula siga vertadera. Construïm $\mathcal{V}(\langle\varphi\rangle, c)$ com segueix.

- El certificat c ha de ser una llista de variables $[x_1, x_2, \ldots, x_n]$ (si no, rebutjar).
- Evaluar la fórmula φ donant-li valor vertader a les variables que estan en la llista i valor fals a la resta de variables.
- Si la fórmula té valor vertader, acceptar. Si no, rebutjar.

Així demostrem que existeix un verificador $\mathcal{V}$ tal que $\langle\varphi\rangle \in \text{SAT}$ si, i només si, existeix un certificat c tal que $\mathcal{V}(\langle\varphi\rangle, c) =$ accepta i, per tant, que SAT és polinomialment verificable.

Anem a demostrar ara que SAT és $\mathbf{NP}$-difícil, és a dir, si tenim L un llenguatge en $\mathbf{NP}$, aleshores $L \leq_p \text{SAT}$.

Suposem $L \in \mathbf{NP}$, és a dir, pel Teorema 3.10 existeix $\mathcal{N}$ una màquina de Turing no determinista que decideix L en temps polinomial. Suposarem que, donada una cadena $w \in A^\star$, amb $|w| = n$, la màquina $\mathcal{N}$ acaba la seua execució en un temps màxim de $|w|^k = n^k$ passos. Notem que una cadena $w \in A^\star$ és acceptada per la màquina $\mathcal{N}$ si existeix una seqüència de configuracions $\alpha_0, \alpha_1, \alpha_2, \ldots \alpha_m$ amb $m \leq n^k$ tal que α_0 és la configuració inicial ($\alpha_0 = q_0 w$), α_m està en un estat final i, per a tot $i \in m$, hi ha una transició de α_i a α_{i+1}.

Anem a voler transformar una cadena $w \in A^\star$ en una fórmula φ que siga satisfactible si, i sols si, existeix una seqüència de configuracions de la màquina $\mathcal{N}$ que accepta la cadena w. Dit d'altra forma, volem trobar una funció f computable en temps polinomial tal que w està en L (o el que és el mateix, $\mathcal{N}$ accepta w) si, i només si, $f(w) = \varphi$ està en el llenguatge SAT ($L \leq_p \text{SAT}$). Busquem aquesta f.

Una taula d'execució és una matriu de tamany $n^k \times n^k$ en la que en cada fila tenim una configuració de la màquina

#	q_0	w_1	w_2	w_3	w_4	w_5	w_6	...		#
#	z	q_3	w_2	w_3	w_4			...		#
#	z	k	q_7	w_3	w_4			...		#
#								...		#
#										#
#										#
#										#
⋮										#
⋮								...		#
#								...		#

TAULA 4.4. Taula d'execució.

En aquesta taula tindrem variables de la forma χ_{ijs} que representen el fet que en la fila i, columna j, es troba en la taula el símbol s:

- $1 \leq i \leq n^k$ indica un número de fila.
- $1 \leq j \leq n^k$ indica un número de columna.
- s és un símbol en $C = \Gamma \cup \mathcal{Q} \cup \{\#\}$.

Construïrem ara una fórmula φ que serà diferent segons la cadena w que li introduïm a la màquina $\mathcal{N}$ i que dependrà de les configuracions que segueix la màquina quan li introduïm aquesta paraula. Vejam-ho:

$$\varphi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{accept} \wedge \varphi_{move}.$$

Expliquem ara cadascuna de les components d'aquesta fórmula:

- La component φ_{cell} garanteix que la taula estiga construïda correctament.

$$\varphi_{cell} = \bigwedge_{i=1\ldots n^k} \bigwedge_{j=1\ldots n^k} (\bigvee_{s\in C} \chi_{ijs}) \wedge (\bigwedge_{s\in C} \bigwedge_{s\neq t\in C} (\neg\chi_{ijs}) \vee (\neg\chi_{ijt})).$$

  Aquesta fórmula està dividida en dues parts. La primera part, $\bigwedge_{i=1\ldots n^k} \bigwedge_{j=1\ldots n^k} (\bigvee_{s\in C} \chi_{ijs})$ assegura que en cada casella de la taula hi ha almenys un símbol $s \in C$, és a dir, que la taula estiga emplenada correctament. La segona part, $(\bigwedge_{s\in C} \bigwedge_{s\neq t\in C} (\neg\chi_{ijs}) \vee (\neg\chi_{ijt}))$, garanteix que en cada casella de la taula hi ha exactament un símbol $s \in C$.

  Així, φ_{cell} ens diu que la posició (i, j) de la taula conté algun símbol $s \in C$ i a més ens diu que aquest és únic, és a dir, hi ha un i només un símbol en cada casella.

- La fórmula φ_{start} correspon a la primera fila de la taula, que és justament la configuració inicial de la màquina $\mathcal{N}$ quan li introduïm la cadena w.

$$\varphi_{start} = \chi_{11\#} \wedge \chi_{12q_0} \wedge \chi_{12w_1} \wedge \chi_{14w_2} \wedge \cdots \wedge \chi_{1(n+2)w_n} \wedge \cdots \wedge \chi_{1n^k\#}.$$

- φ_{accept} és una fórmula que ens diu que, en alguna casella de la taula, hi ha un estat d'acceptació.

$$\varphi_{accept} = \bigvee_{i=1\ldots n^k} \bigvee_{j=1\ldots n^k} \chi_{ijq_a}.$$

- La fórmula φ_{move} indica que, en la construcció de la taula d'execució, cada fila representa una configuració de la màquina de Turing, i cada transició de fila a fila correspon al moviment de la màquina en el seu procés d'execució, utilitzant una configuració de la màquina. Això implica que la transició d'una fila a la següent sempre es realitza utilitzant una configuració de la màquina.

$$\varphi_{move} = \bigwedge_{i=1\ldots n^k} \bigwedge_{j=1\ldots n^k} \left(\begin{array}{c} \text{La finestra de dos files i tres columnes} \\ \text{centrada en } (i,j) \text{ correspon a una} \\ \text{transició possible de la màquina } \mathcal{N} \end{array} \right).$$

Així, donada una paraula $w \in A^\star$ hem construit una fórmula $\varphi = f(w)$ i, a més, es compleix que $w \in L$ si, i només si, $f(w) \in \mathrm{SAT}$. La funció f que transforma la paraula en una fórmula lògica és computable en temps polinomial (ja que la fórmula φ que hem construit abans és conjunció de fórmules de tamany polinomial).

Tindrem per tant que $L \leq_p \mathrm{SAT}$. □

La demostració anterior ens fa veure com de complicades poden arribar a ser les fórmules lògiques, que fins i tot, tenen capacitat de codificar les configuracions i passos d'una màquina de Turing.

COROL·LARI 4.1. *Si demostràrem que* $\mathrm{SAT} \in \mathbf{P}$*, aleshores* $\mathbf{P} = \mathbf{NP}$.

Això significa que si poguérem resoldre SAT en temps polinomial, podríem resoldre qualsevol problema en $\mathbf{NP}$ en temps polinomial, implicant açò que $\mathbf{P} = \mathbf{NP}$.

3. Altres resultats

En aquesta secció explorarem altres llenguatges i veurem com podem aplicar el Teorema de Cook-Levin per a demostrar que aquests llenguatges són també $\mathbf{NP-complets}$. Començarem definint alguns conceptes importants com els literals, les clàusules, i les fórmules en forma normal conjuntiva (FNC) i en $3\mathrm{FNC}$. Després, introduirem els llenguatges $3\mathrm{SAT}$ i CLIQUE, i demostrarem que $3\mathrm{SAT}$ es pot reduir polinomialment a CLIQUE. A partir d'aquesta reducció provarem que $3\mathrm{SAT}$ és $\mathbf{NP-complet}$ i, finalment, que CLIQUE també és $\mathbf{NP-complet}$. Aquestes demostracions ens permetran entendre millor la interrelació entre diferents problemes $\mathbf{NP-complets}$ i la importància de les reduccions polinomials en la teoria de la complexitat computacional.

DEFINICIÓ 4.7. *Presentem una serie de definicions prèvies*

- *Un* literal *és una variable booleana afirmada o negada.*

  Ex: $x, \neg x$.

- *Una clàusula és una disjunció de literals.*

  Ex: $x \vee \neg y \vee z$.

- *Una fórmula està en* forma normal conjuntiva (FNC) *si és una conjunció de clàusules.*

  Ex: $(x \vee \neg y \vee z) \wedge (z \vee \neg x) \wedge (y \vee \neg z)$.

- *Una fórmula està en* $3\mathrm{FNC}$ *si està en* FNC *i a més totes les clàusules tenen exactament tres literals.*

Ex: $(x \vee y \vee y) \wedge (z \vee \neg x \vee y) \wedge (y \vee \neg z \vee z)$ *està en* 3FNC.

DEFINICIÓ 4.8. *(*3SAT*). Definim el llenguatge* 3SAT *com:*

$$3\mathrm{SAT} = \{\langle\varphi\rangle \mid \varphi \textit{ és un fórmula en } 3\mathrm{FNC} \textit{ satisfactible }\}$$

És a dir, 3SAT *conté la codificació de totes aquelles fórmules en* 3FNC *que són satisfactibles.*

NOTA 4.4. *Podem veure que* $3\mathrm{SAT} \subseteq \mathrm{SAT}$.

NOTA 4.5. *Recordem la definició del llenguatge* CLIQUE *introduïda en la Definició 3.15*

$$\mathrm{CLIQUE} = \left\{ \langle \mathcal{G}, k\rangle \mid \begin{array}{c} \mathcal{G} \textit{ és un graf no dirigit que admet} \\ \textit{una clique de tamany } k \in \mathbb{N} \end{array} \right\}$$

Veiem ara un teorema que relaciona els llenguatges 3SAT i CLIQUE.

TEOREMA 4.3. $3\mathrm{SAT} \leq_p \mathrm{CLIQUE}$.

DEMOSTRACIÓ. Per definició volem trobar una funció $f : A^\star \longrightarrow A^\star$ que siga computable en temps polinomial tal que $\langle\varphi\rangle$ està en 3SAT si, i només si, $f(\langle\varphi\rangle) = \langle\mathcal{G}, k\rangle$ està en CLIQUE.

Construïm l'aplicació f com següeix:

- Si φ no està en 3FNC, definim $f(\varphi) = \langle :, 2\rangle$, on ":"és un graf de dos vèrtexs sense cap connexió. Tindrem que $\langle\varphi\rangle$ no està en 3SAT i $f(\langle\varphi\rangle)$ no està en CLIQUE.
- Si tenim una fórmula φ que està en 3FNC, la fórmula és la conjunció de k clàusules de tres literals cadascuna.

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$$

  Anem a construir un graf de $3k$ vèrtexs i ens interessarà la pregunta de si té o no una $k-$clique. El graf que construirem tindrà un vèrtex per cada literal ($3k$ vèrtexs en total). Tots els vèrtexs del graf estaran conectats entre ells per una arista, excepte que es done algun d'aquests dos casos:

  (1) Si els vèrtexs pertanyen a la mateixa terna, és a dir, venen de la mateixa clàusula (a_j, b_j i c_j no estaran conectats entre si).

  (2) Si els dos vèrtexs pertanyen a literals oposats (afirmació i negació d'una mateixa variable).

Aquest és un exemple de graf que vindria de la fórmula $\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$:

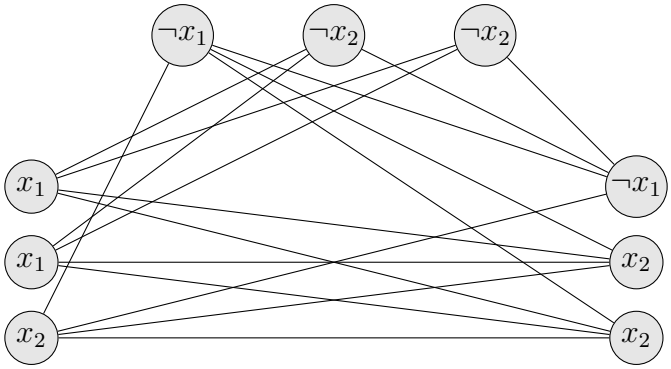

FIGURA 4.1. Exemple de graf per a $k = 3$.

Amb aquesta construcció vejam ara que $\langle\varphi\rangle$ està en 3SAT si, i només si, $f(\langle\varphi\rangle) = \langle\mathcal{G}, k\rangle$ està en CLIQUE.

Suposem que $\langle\varphi\rangle$ està en 3SAT, és a dir, φ és una fórmula en 3FNC satisfactible. Considerarem una assignació de variables a valors booleans que fa vertadera la fórmula. Notem que si

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k),$$

en cada clàusula ha d'haver almenys un literal que tinga valor vertader. Quan fem $f(\langle\varphi\rangle)$, aquests k vèrtexs aniràn conectats entre si ja que pertanyen a clàusules distintes i no corresponen a literals oposats. Per tant notem que el nostre graf tindrà una k-clique. Acabem de veure que si $\langle\varphi\rangle$ està en 3SAT, aleshores $f(\langle\varphi\rangle) = \langle\mathcal{G}, k\rangle$ està en CLIQUE.

Suposem ara que el graf té una k-clique. Els vèrtexs de la k-clique corresponen a ternes distintes. Podem elegir una assignació de variables que li otorgue valor vertader als literals de la clique (notem que no pot haver una contradicció, és a dir, no pot ser que x i $\neg x$ estiguen els dos en la clique). A les variables que no apareguen en la clique se'ls pot assignar qualsevol valor. Aquesta assignació de variables fa vertadera a φ. Acabem de demostrar que si $f(\langle\varphi\rangle) = \langle\mathcal{G}, k\rangle$ està en CLIQUE, aleshores $\langle\varphi\rangle$ està en 3SAT.

Per tant hem reduït en temps polinomial 3SAT a CLIQUE. □

NOTA 4.6. *Després de demostrar el teorema anterior, podem concloure que si tinguérem un algorisme per a resoldre* CLIQUE *en temps polinomial, podriem resoldre* 3SAT *en temps polinomial.*

Presentem ara un corol·lari del teorema de Cook-Levin que ens dona informació sobre el llenguatge 3SAT.

COROL·LARI 4.2. 3SAT *és* $\mathbf{NP - complet}$.

DEMOSTRACIÓ. Ja sabem que $3\text{SAT} \subseteq \text{SAT}$. Veiem ara la reducció de SAT a 3SAT:

Tota fórmula lògica és equivalent a la seua forma normal disjuntiva $(x_1 \vee x_2 \vee \cdots \vee x_n)$ i tota disjunció és equivalent a una fórmula en 3FNC,

$$(x_1 \vee x_1 \vee a_1) \wedge (\neg a_1 \vee x_2 \vee a_2) \wedge (\neg a_2 \vee x_3 \vee a_3) \wedge \cdots \wedge (\neg a_{n-1} \vee x_n \vee x_n).$$

Així $\text{SAT} = 3\text{SAT}$ i pel teorema de Cook-Levin sabem que SAT és $\mathbf{NP - complet}$, per tant podem concloure que 3SAT és $\mathbf{NP - complet}$. □

COROL·LARI 4.3. CLIQUE *és* $\mathbf{NP-complet}$.

DEMOSTRACIÓ. Hem vist que $3\mathrm{SAT} \leq_p \mathrm{CLIQUE} \in \mathbf{NP}$ i acabem de veure en el corol·lari anterior que $3\mathrm{SAT}$ és $\mathbf{NP-complet}$, aleshores queda demostrat que CLIQUE és $\mathbf{NP-complet}$. □

Conclusions

En aquest treball hem realitzat un recorregut a través de conceptes bàsics i essencials de la teoria de la computació i la complexitat computacional amb l'objectiu final d'entendre l'enunciat i la demostració del teorema de Cook-Levin i la seua importància en aquest àmbit. Hem començat amb una introducció a les definicions bàsiques i propietats del monoide lliure, seguit d'un anàlisi dels models de computació clàssics, entre els que trobem els autòmats finits i les màquines de Turing. Hem estudiat els conceptes de decidibilitat i semi-decidibilitat i hem vist exemples de llenguatges decidibles, semi-decidibles i no reconeixibles per màquines de Turing. Hem entrat també en el món de la complexitat computacional i hem estudiat conceptes clau com la reducció funcional, la dominància asimptòtica i les classes de complexitat $\mathbf{P}$ i $\mathbf{NP}$. Tot açò fins a arribar a introduir la noció de $\mathbf{NP-completesa}$, una classe de problemes que són els més difícils dins de la classe $\mathbf{NP}$. Aquest concepte ha permés identificar i classificar problemes de gran complexitat i ha donat lloc a una millor comprensió de la relació entre les classes $\mathbf{P}$ i $\mathbf{NP}$.

El teorema de Cook-Levin estableix que un problema de la lògica proposicional, el problema de la satisfacibilitat booleana (SAT), és $\mathbf{NP-complet}$. Açò significa que SAT és un problema en $\mathbf{NP}$ i que qualsevol altre problema en $\mathbf{NP}$ es pot reduir a SAT en temps polinomial. Aquest resultat té implicacions importants per a la teoria de la computació, ja que ens diu que si poguérem trobar un algorisme per resoldre SAT en temps polinomial, podríem resoldre qualsevol problema en $\mathbf{NP}$ en temps polinomial, implicant açò que $\mathbf{P} = \mathbf{NP}$. Així, la qüestió de si $\mathbf{P} = \mathbf{NP}$ continua sent un dels problemes més importants i oberts en la teoria de la computació.

En conclusió, en aquest treball no sols hem conseguit entendre el teorema de Cook-Levin, sinó que hem fet un breu recorregut sobre alguns aspectes bàsics de la teoria de la computació i complexitat computacional i hem vist la rellevància d'aquest teorema en el món de la computació. A més, hem vist com podem aplicar les reduccions polinomials per a demostrar la $\mathbf{NP-completesa}$ d'altres problemes. Açò ens permet explorar la naturalesa de la dificultat computacional i continuar avançant en la investigació de solucions eficients per a problemes complexos.

Bibliografia

[Bar21] P. Barenbaum. Teoría de la computación, 2021.

[Fer04] F. J. Ferri. *Teoria d'autòmats i llenguatges formals*. Educació. Sèrie Materials, 75. PUV, 2004.

[Gal23] N. Gallego. Llenguatges formals i autòmats finits. Teball Fi de Grau, Universitat de València, 2023.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

[HMU07] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3rd edition, 2007.

[Ros09] A. L. Rosenberg. *The Pillars of Computation Theory: State, Encoding, Nondeterminism*. Springer Science & Business Media, 2009.

[Sip13] M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2013.

[Tud23] C. Tudela. Autòmats finits i llenguatges formals. Teball Fi de Grau, Universitat de València, 2023.

[Tur36] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.