Modelos de computación

Bibliografía:

- Apuntes de clase.
- Michael Sipser. *Introduction to the Theory of Computation, 3rd. ed.* Cengage Learning (2013).
- Wikipedia, la Enciclopedia Libre. *Lenguaje recursivamente enumerable*. Recuperado de https://es.wikipedia.org/wiki/Lenguaje_recursivamente_enumerable a fecha de 13 de septiembre de 2022.
- Wikipedia, the Free Encyclopedia. *Turing Machine*. Recuperado de https://en.wikipedia.org/wiki/Turing_machine a fecha de 11 de septiembre de 2022.

Capítulo 1

Lenguajes regulares

Un **alfabeto** Σ es un conjunto finito de **símbolos**. Una **cadena** en Σ es un elemento de $\Sigma^* \coloneqq \bigcup_{n\in\mathbb{N}} \Sigma^n$, que solemos escribir como $u_1 \cdots u_n \coloneqq (u_1, \ldots, u_n)$. Dadas dos cadenas $u, v \in \Sigma^*$, llamamos **concatenación** de u y v a $u \cdot v \coloneqq u_1 \cdots u_{|u|} v_1 \cdots v_{|v|}$. Σ^* es un monoide con la concatenación de cadenas, y llamamos ϵ a su elemento neutro. Dada $u = u_1 \cdots u_n \in \Sigma^*$, llamamos $u^{\mathrm{R}} \coloneqq u_n u_{n-1} \cdots u_1$.

Un **lenguaje** sobre Σ es un $L \in \mathcal{U}_\Sigma \coloneqq \mathcal{P}(\Sigma^*)$. La **concatenación** de dos lenguajes $L_1, L_2 \subseteq \Sigma^*$ es $L_1 \cdot L_2 \coloneqq \{uv\}_{u\in L_1, v\in L_2}$. $\mathcal{P}(\Sigma^*)$ es un monoide con la concatenación de lenguajes. La **clausura** o ***Kleene star*** de un lenguaje L es $L^* \coloneqq \bigcup_{n\in\mathbb{N}} L^n$.

1.1. Autómatas finitos

Un **autómata finito determinista** (**DFA**, *Deterministic Finite Automaton*) es una tupla $(Q, \Sigma, \delta, q_0, F)$ formada por un conjunto finito de **estados** Q, un alfabeto Σ, una **función de transición** $\delta : Q \times \Sigma \to Q$, un **estado inicial** $q_0 \in Q$ y un conjunto de **estados finales** o **de aceptación** $F \subseteq Q$.

Un **autómata finito no determinista** (**NFA**, *Nondeterministic Finite Automaton*) es una tupla $(Q, \Sigma, \delta, q_0, F)$ formada por un conjunto finito de **estados** Q, un alfabeto Σ, una **función de transición** $\delta : Q \times (\Sigma_\epsilon \coloneqq \Sigma^1 \cup \{\epsilon\}) \to \mathcal{P}(Q)$, un **estado inicial** $q_0 \in Q$ y un conjunto de **estados finales** o **de aceptación** $F \subseteq Q$.

Un NFA $(Q, \Sigma, \delta, q_0, F)$ **acepta** una cadena $w \in \Sigma^*$ si existen $m \in \mathbb{N}$, $(y_1, \ldots, y_m) \in (\Sigma_\epsilon)^m$ y $(r_0, \ldots, r_m) \in Q^{m+1}$ tales que $r_0 = q_0$, $r_m \in F$ y, para $i \in \{0, \ldots, m-1\}$, $r_{i+1} \in \delta(r_i, y_{i+1})$. Un DFA $(Q, \Sigma, \delta, q_0, F)$ **acepta** una cadena $w \in \Sigma^*$ si el NFA $(Q, \Sigma, \delta', q_0, F)$ lo acepta, donde δ' viene dado por $\delta'(q, a) \coloneqq \{\delta(q, a)\}$ y $\delta'(q, \emptyset) \coloneqq \emptyset$ para $q \in Q$ y $a \in \Sigma$. El **lenguaje aceptado** por un DFA o NFA $\mathcal{A}$, $L(\mathcal{A})$, es el conjunto de cadenas aceptadas por $\mathcal{A}$.

Llamamos $\mathcal{L}_{\mathrm{DFA}}$ a la clase de lenguajes aceptados por algún DFA, y $\mathcal{L}_{\mathrm{NFA}}$ a la de lenguajes aceptados por algún NFA. Como **teorema**, $\mathcal{L}_{\mathrm{DFA}} = \mathcal{L}_{\mathrm{NFA}}$.

Para dibujar un NFA $(Q, \Sigma, \delta, q_0, F)$, dibujamos un círculo para cada $q \in Q$ con su etiqueta dentro, o un doble círculo si $q \in F$, una flecha que señala a q_0 y, para cada $q \in Q$, $a \in \Sigma_\epsilon$ y $q' \in \delta(q, a)$, una flecha del círculo q al q' etiquetada con a. Para dibujar un DFA, dibujamos su NFA equivalente. Además, si en el DFA existe un $e \in Q \setminus F$ (estado de error) tal que para todo $a \in \Sigma$ es $\delta(e, a) = e$, podemos no dibujar e ni las flechas que van a parar a e.

También podemos representar un NFA con una tabla con un estado por fila, empezando por el estado inicial y marcando de alguna forma los estados finales, y con una columna por letra o ϵ, cuyas celdas contienen los valores de la función de transición.

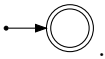
1.2. Expresiones regulares

Las **operaciones regulares** son la unión, la concatenación y la clausura de lenguajes. Una **expresión regular** sobre el alfabeto Σ es una de la forma $\emptyset$, ϵ, a para un $a \in \Sigma$, $r_1 \mid r_2$, $r_1 r_2$ o r_1^*, donde r_1 y r_2 son expresiones regulares y las operaciones, de mayor a menor prioridad, son r_1^*, $r_1 r_2$ y $r_1 \mid r_2$, y se usan paréntesis para desambiguar la prioridad. Un **lenguaje regular** es uno representado por una expresión regular, donde $\emptyset$ y ϵ se representan a sí mismos, a representa $\{a\}$, $r_1 \mid r_2$ representa la unión de los lenguajes correspondientes, $r_1 r_2$ la concatenación y r_1^* la clausura.

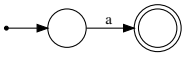
Llamamos $\mathcal{L}_{\text{ER}}$ o $\mathcal{REG}$ a la clase de lenguajes regulares, y se tiene $\mathcal{L}_{\text{ER}} = \mathcal{L}_{\text{NFA}}$.

$\subseteq$] $\emptyset$ es el lenguaje de

ϵ es el lenguaje de



a es el lenguaje de



Dados dos lenguajes regulares L_1 y L_2 con NFAs respectivos

$$(Q_1, \Sigma, \delta_1, q_1, F_1) \quad \text{y} \quad (Q_2, \Sigma, \delta_2, q_2, F_2),$$

$L_1 \cup L_2$ es aceptado por $(Q_1 \sqcup Q_2 \sqcup \{q_0\}, \Sigma, \delta, q_0, F_1 \cup F_2)$, donde

δ	Σ	ϵ
q_0	$\emptyset$	$\{q_1, q_2\}$
Q_1	$\delta_1(q,a)$	
Q_2	$\delta_2(q,a)$	

$L_1 L_2$ es aceptado por $(Q_1 \sqcup Q_2, \Sigma, \delta, q_1, F_2)$, con δ dada por

δ	Σ	ϵ
$Q_1 \setminus F_1$	$\delta_1(q,a)$	
F_1		$\delta_1(q,a) \cup \{q_2\}$
Q_2	$\delta_2(q,a)$	

L_1^* es aceptado por $(Q_1 \sqcup \{q_0\}, \Sigma, \delta, q_0, \{q_0\})$, con δ dada por

δ	Σ	ϵ
q_0	$\emptyset$	$\{q_1\}$
$Q_1 \setminus F_1$	$\delta_1(q,a)$	
F_1		$\delta_1(q,a) \cup \{q_0\}$

1.3. Propiedades de $\mathcal{REG}$

$\mathcal{REG}$ está cerrado bajo unión, concatenación, clausura, complemento, intersección y diferencia. **Demostración:** Los 3 primeros son por definición. Si el DFA $(Q, \Sigma, \delta, q_0, F)$ acepta el lenguaje L, el DFA $(Q, \Sigma, \delta, q_0, Q \setminus F)$ acepta $\overline{L} = \Sigma^* \setminus L$, pues para cada cadena w solo hay una secuencia de estados $\{r_0, \ldots, r_m\} \subseteq Q$ con $r_0 = q_0$ y cada $r_{i+1} = \delta(r_i, w_{i+1})$ y w se acepta si y sólo si r_m es final. Finalmente, $L \cap M = \overline{\overline{L} \cup \overline{M}}$ y $L \setminus M = L \cap \overline{M}$.

Lema del bombeo, **LBLR** o ***pumping lemma*:** para $L \in \mathcal{REG}$, existe $p \in \mathbb{N}$, la ***pumping length***, tal que toda $w \in L$ con $|w| \geq p$ puede ser dividida como $w = xyz$ con $|y| > 0$ y $|xy| \leq p$ de modo que $xy^k z \in L$ para todo $k \in \mathbb{N}$. **Demostración:** Sean $(Q, \Sigma, \delta, q_0, F)$ un DFA que acepta L, $p \coloneqq |Q|$, $w \in L$ con $n \coloneqq |w| \geq p$ y $\{q_0, \ldots, q_n\} \subseteq Q$ con cada $q_{i+1} = \delta(q_i, w_{i+1})$. Como $\{q_0, \ldots, q_p\} \subseteq Q$, existen $i, j \in \{0, \ldots, p\}$ con $i < j$ y $q_i = q_j$. Sean entonces $x \coloneqq w_1 \cdots w_i$, $y \coloneqq w_{i+1} \cdots w_j$ y $z \coloneqq w_{j+1} \cdots w_n$, entonces $w = xyz$, $|y| > 0$, $|xy| = j \leq p$ y, para $k \in \mathbb{N}$, el DFA acepta $xy^k z$ con la secuencia de estados $q_0 \cdots q_i (q_{i+1} \cdots q_j)^k q_{i+1} \cdots q_n$.

El lenguaje $\{0^n 1^n\}_{n \in \mathbb{N}}$ no es regular. **Demostración:** Supongamos que lo sea y sean L el lenguaje, $p \in \mathbb{N}$ una *pumping length* de L, $w \coloneqq 0^p 1^p$ y $w =: xyz$ una división dada por el lema del bombeo. Si $y = 0^k$, $xy^{2k}z \notin L$ porque tiene más 0s que 1s; análogamente, si $y = 1^k$, $xy^{2k}z \notin L$, y si $y = 0^k 1^l$ con $k, l > 0$, $xyyz \notin L$ porque tiene un 1 seguido de un 0#.

Los autómatas finitos son demasiado sencillos para teorizar sobre lo computable o no computable debido a su falta de memoria.

Capítulo 2

Lenguajes libres de contexto

2.1. Autómatas de pila

Un **autómata de pila con aceptación por estado final** (**PDA**, ***Push Down Automaton***) es una tupla $(Q, \Sigma, \Gamma, \delta, A_0, q_0, F)$ formada por un conjunto finito de **estados** Q, un alfabeto Σ, un alfabeto de **símbolos de la pila** Γ, una **función de transición** $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$, un **símbolo inicial de la pila** A_0,[1] un **estado inicial** $q_0 \in Q$ y un conjunto de **estados finales** o **de aceptación** $F \subseteq Q$. Un **autómata de pila con aceptación a pila vacía** (**PDA$^\mathbf{v}$**) es una tupla $(Q, \Sigma, \Gamma, \delta, A_0, q_0)$ similar a un PDA pero sin F.

Para representar un PDA, dibujamos un círculo por cada estado, o un doble círculo si el estado es final, y por cada par de estados (q, r), si existe (a, x, y) con $(r, y) \in \delta(q, a, x)$, dibujamos una flecha de q a r etiquetada con, por cada (a, x, y) con $(r, y) \in \delta(q, a, x)$, una línea «$a, x \to y$».

Dado un autómata de pila, una posición es una tupla $(q, \gamma, w) \in Q \times \Gamma^* \times \Sigma^*$. Una **acción** (**no determinista**) es un par de posiciones de la forma $((q, \gamma b, aw), (r, \gamma c, w))$, donde $q, r \in Q$, $\gamma \in \Gamma^*$, $b, c \in \Gamma_\epsilon$, $w \in \Sigma^*$, $a \in \Sigma_\epsilon$ y $(r, c) \in \delta(q, a, b)$. Se representa como $q \to^{a,b\to c} r$ o en **formato funcional** como «$(q, a, b) = (r, c)$».

Una secuencia de posiciones $(s_0, \ldots, s_k)$ en la que cada (s_i, s_{i+1}) es una acción se representa como una secuencia de acciones encadenadas, en la que el estado final de una acción y el inicial de la siguiente se representan como el mismo ($q_1 \to^{a_1,b_1\to c_1} q_2 \to^{a_2,b_2\to c_2} \ldots$). Un PDA acepta una cadena w si existe una secuencia de posiciones que empieza por (q_0, A_0, w) y termina por (q, ϵ, ϵ) con $q \in Q$ en el caso de un PDA con aceptación a pila vacía o por (q, γ, ϵ) con $q \in F$ y $\gamma \in \Gamma^*$ en el caso de un PDA con aceptación por estado final.

Podemos suponer que una transición añade o elimina más de un elemento de la pila ($\delta : Q \times \Sigma_\epsilon \times \Gamma^* \to \mathcal{P}(Q \times \Gamma^*)$), lo que equivale a tener varios estados intermedios en la transición para primero quitar elementos y luego añadir.

Un **autómata de pila determinista** (**PDAD**) es uno en el que, para cada $q \in Q$, $a \in \Sigma$ y $x \in \Gamma$, uno de entre $\delta(q, a, x)$, $\delta(q, a, \epsilon)$, $\delta(q, \epsilon, x)$ y $\delta(q, \epsilon, \epsilon)$ es unipuntual y el resto son vacíos. En una representación de un PDAD, si existen $q \in Q$, $a \in \Sigma$ y $x \in \Gamma$ tales que no hay ninguna

[1] Este es completamente innecesario y solo sirve para complicar las demostraciones. Aparece en las diapositivas pero no en el libro.

flecha saliente de q con una línea de la forma «$a, x \to y$», «$a, \epsilon \to y$», «$\epsilon, x \to y$» o «$\epsilon, \epsilon \to y$» para algún $y \in \Gamma_\epsilon$, se entiende que existe un estado adicional q_E no final no representado con transiciones $\delta(q_\mathrm{E}, b, \epsilon) = \{(q_\mathrm{E}, \epsilon)\}$ para cada $b \in \Sigma$ y una transición $\delta(q, a, x) = \{(q_\mathrm{E}, \epsilon)\}$.

Llamamos $L(\mathcal{A})$ al lenguaje aceptado por un PDA $\mathcal{A}$, $\mathcal{L}_{\mathrm{PDA}}$ a la clase de lenguajes aceptados por algún PDA, $\mathcal{L}_{\mathrm{PDA}^\mathrm{v}}$ a la de lenguajes aceptados por algún PDA$^\mathrm{v}$ y $\mathcal{L}_{\mathrm{PDAD}}$ a la de lenguajes aceptados por algún PDAD.

2.2. Gramáticas libres de contexto

Una **gramática libre de contexto** (**GLC**) es una tupla $(V, \Sigma, \mathcal{R}, S)$ formada por un **alfabeto de variables** V, un alfabeto de **símbolos terminales** Σ disjunto de V, un conjunto finito $\mathcal{R}$ de **reglas de producción**, pares $(S, w) \in V \times (V \dot\cup \Sigma)^*$ representados como $S \to w$, y una **variable inicial** $S \in V$. Se puede representar con una línea por cada variable $T \in V$, empezando por la variable S, de la forma $T \to w_1 \mid \cdots \mid w_n$, donde $\{w_1, \ldots, w_n\} = \{w \mid (T, w) \in \mathcal{R}\}$.

Dadas $v, w, x \in (V \cup \Sigma)^*$ y $R \in V$, escribimos $vRw \Rightarrow vxw$ si $(R \to x) \in \mathcal{R}$. Una **derivación** es una secuencia $\{v_1, \ldots, v_n\} \subseteq (V \cup \Sigma)^*$ tal que para cada i, $v_i \Rightarrow v_{i+1}$, y escribimos $v \Rightarrow w$ si existe una derivación que empieza por v y termina por w. El **lenguaje generado** por una GLC $G = (V, \Sigma, \mathcal{R}, S)$ es $L(G) \coloneqq \{w \in \Sigma^* \mid S \Rightarrow^* w\}$. Un lenguaje es **libre de contexto** si es generado por una GLC, y llamamos $\mathcal{CF}$ o $\mathcal{L}_{\mathrm{GCF}}$ a la clase de los lenguajes libres de contexto. Todo lenguaje generado por un PDA es libre de contexto.

Dada una GLC $G \coloneqq (V, \Sigma, \mathcal{R}, S)$, un **árbol de derivación** de $w \in L(G)$ en es un árbol ordenado construido usando como raíz S y añadiendo, para cada paso $uRv \Rightarrow uxv$ en la derivación $S \Rightarrow^* w$, aristas de R a cada símbolo de x, distinguiendo cada símbolo en la cadena de otros símbolos iguales. Una **derivación por la izquierda** es una en la que, en cada paso $uRv \Rightarrow uxv$, $u \in \Sigma^*$. Toda $w \in L(G)$ admite una derivación por la izquierda, que se obtiene recorriendo un árbol de derivación de w en preorden y expandiendo cada variable que se encuentre.

Una GLC está en **forma normal de Chomsky** si todas sus reglas son de la forma $S \to \lambda$, $A \to BC$ o $A \to a$, donde S es la variable inicial, A, B y C son variables arbitrarias con $B, C \neq S$ y a es un terminal arbitrario. Toda GLC se puede convertir a forma normal de Chomsky con el Algoritmo 1.

2.3. Propiedades de $\mathcal{CF}$

Como **teorema**, $\mathcal{REG} \subsetneq \mathcal{L}_{\mathrm{PDAD}} \subsetneq \mathcal{L}_{\mathrm{PDA}} = \mathcal{L}_{\mathrm{PDA}^\mathrm{v}} = \mathcal{CF}$.

$1 \subseteq 2$] El DFA $(Q, \Sigma, \delta, q_0, F)$ equivale al PDAD $(Q, \Sigma, \{\$\}, \{(q, a, \epsilon, \delta(q, a), \epsilon)\}_{q \in Q}^{a \in \Sigma}, \$, q_0, F)$.

$2 \nsubseteq 1$] $L = \{0^n c 1^n\}_{n \in \mathbb{N}}$ es reconocido por el PDAD de la figura 2.1, pero no es regular. Si lo fuera, tendría una *pumping length* $p \in \mathbb{N}$ y $w \coloneqq 0^p c 1^p \in L$ tendría una descomposición $w =: xyz$ en las condiciones del lema del bombeo, y como $|xy| \leq p$, debe ser $y = 0^k$ para un cierto $k > 0$, pero entonces xy^2z tiene más 0s que 1s y por tanto $w \notin L\#$.

Entrada: GLC $G = (V, \Sigma, \mathcal{R}, S)$.
Salida: GLC $G' = (V, \Sigma, \mathcal{R}, S_0)$ en forma normal de Chomsky con $L(G') = L(G)$.
añadir una nueva variable S_0 a V;
añadir $S_0 \to S$ a $\mathcal{R}$;
para $A \to \epsilon \in \mathcal{R}$ **hacer**
    eliminar $A \to \epsilon$ de $\mathcal{R}$;
    para $B \to w \in \mathcal{R}$ *con* A *en* w **hacer**
        añadir $B \to w'$ a $\mathcal{R}$ para cada w' resultante de eliminar 1 o más ocurrencias de A en w, con lo que si hay n ocurrencias de A se añaden $2^n - 1$ reglas, con la excepción de que no añadimos $B \to \epsilon$;
mientras *exista* $A \to B \in \mathcal{R}$ *con* $B \in V$ **hacer**
    eliminar $A \to B$ de $\mathcal{R}$;
    si $A \neq B$ **entonces**
        para $B \to u \in \mathcal{R}$ **hacer** añadir $A \to u$ a R;
para $A \to u_1 \cdots u_k \in \mathcal{R}$ *con* $k \geq 3$ **hacer**
    eliminar $A \to u_1 \cdots u_k$ de $\mathcal{R}$;
    añadir nuevas variables $A_1, \ldots, A_{k-2}$ a V;
    para $i \leftarrow 1$ *a* $k - 3$ **hacer** añadir $A_i \to u_{i+1}A_{i+1}$ a $\mathcal{R}$;
    añadir $A \to u_1A_1$ y $A_{k-2} \to u_{k-1}u_k$ a R;
para $X \to ab$ *con* $a \in \Sigma$ *o* $b \in \Sigma$ **hacer**
    si $a \in \Sigma$ **entonces**
        añadir una nueva variable A a V y $A \to a$ a $\mathcal{R}$;
        cambiar $X \to ab$ por $X \to Ab$;
    si $b \in \Sigma$ **entonces** hacer lo propio;

Algoritmo 1: Conversión de una GLC a forma normal de Chomsky.

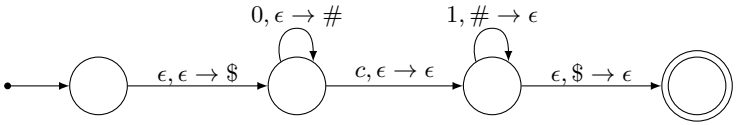


Figura 2.1: PDAD de $\{0^n c 1^n\}_{n \in \mathbb{N}}$.

Entrada: PDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Para simplificar la pila empieza vacía y no hay símbolo inicial, pero un PDA «normal» se puede convertir a uno de este tipo trivialmente.
Salida: GLC $G = (V, \Sigma, \mathcal{R}, S)$ con $L(G) = L(\mathcal{A})$.
añadir un nuevo q_a a Q;
para $q \in F$ **hacer** añadir $q \to^{\epsilon,\epsilon\to\epsilon} q_\mathrm{a}$ a δ;
$F \leftarrow \{q_\mathrm{a}\}$;
para $x \in \Gamma$ **hacer** añadir $q_\mathrm{a} \to^{\epsilon,x\to\epsilon} q_\mathrm{a}$ a δ;
para $q \to^{a,x\to y} r$ *en* δ *con* $x, y \neq \epsilon$ **hacer**
  añadir un nuevo s a Q;
  sustituir $q \to^{a,x\to y} r$ en δ por $q \to^{a,x\to\epsilon} s, s \to^{\epsilon,\epsilon\to y}$;
añadir un nuevo \$ a Γ;
para $q \to^{a,\epsilon\to\epsilon} r$ *en* δ **hacer**
  añadir un nuevo s a Q;
  sustituir $q \to^{a,\epsilon\to\epsilon} r$ en δ por $q \to^{a,\epsilon\to\$} s, s \to^{\epsilon,\$\to\epsilon} r$;
$V \leftarrow \{A_{pq}\}_{p,q\in Q}$;
$S \leftarrow A_{q_0 q_\mathrm{a}}$;
$\mathcal{R} \leftarrow \{A_{pq} \to A_{pr}A_{rq}\}_{p,q,r\in Q} \cup \{A_{pp} \to \epsilon\}_{p\in Q} \cup \{A_{pq} \to aA_{rs}b\}_{p,q,r,s\in Q;u\in\Gamma;a,b\in\Sigma\cup\{\epsilon\}}^{p\to^{q,\epsilon\to u}r,s\to^{b,u\to\epsilon}q\in\delta}$;

Algoritmo 2: Conversión de un PDA a una GLC que acepta el mismo lenguaje.

$2 \subseteq 3]$ Por definición.

$3 \subseteq 4]$ Un PDA $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, A_0, q_0, F)$ acepta el mismo lenguaje que el PDA$^\mathrm{v}$ dado por $\mathcal{A}' \coloneqq (Q \sqcup \{q_\mathrm{s}\} \sqcup \{q_\mathrm{e}\}, \Sigma, \Gamma \sqcup \{\$\}, \delta', \$, q_\mathrm{s})$ con

$$\delta' \coloneqq \delta \cup \{(q_\mathrm{s}, \epsilon, \epsilon, q_0, A_0)\} \cup \{(q, \epsilon, a, q_\mathrm{e}, \epsilon)\}_{q\in F\cup\{q_\mathrm{e}\}}^{a\in\Gamma}.$$

$4 \subseteq 3]$ Un PDA$^\mathrm{v}$ $\mathcal{A}' \coloneqq (Q, \Sigma, \Gamma, \delta, A_0, q_0)$ acepta el mismo lenguaje que el PDA dado por $\mathcal{A} \coloneqq (Q \sqcup \{q_\mathrm{s}\} \sqcup \{q_\mathrm{e}\}, \Sigma, \Gamma \sqcup \{\$\}, \delta', \$, q_\mathrm{s}, \{q_\mathrm{e}\})$ con

$$\delta' \coloneqq \delta \cup \{(q_\mathrm{s}, \epsilon, \epsilon, q_0, A_0)\} \cup \{(q, \epsilon, \$, q_\mathrm{e}, \epsilon)\}_{q\in Q}.$$

$3 \subseteq 5]$ Hay que probar la corrección del Algoritmo 2. Este primero modifica $\mathcal{A}$ pero sin cambiar el lenguaje que acepta: primero para que tenga un único estado final q_a, luego para que siempre que se acepte una cadena, se pueda aceptar con la pila vacía, y finalmente para que todas las transiciones añadan o eliminen un elemento de la pila pero no ambos. Entonces queremos ver que, para $p, q \in Q$ y $w \in \Sigma^*$, A_{pq} genera w si y sólo si existe una secuencia de acciones en $\mathcal{A}$ de (p, ϵ, w) a (q, ϵ, ϵ), pues entonces $A_{p_0 p_\mathrm{a}}$ genera $L(\mathcal{A})$.

$5 \subseteq 3]$ Dada una GLC $G = (V, \Sigma, \mathcal{R}, S)$, el PDA $\mathcal{A} \coloneqq (\{s, l, e\}, \Sigma, V \sqcup \Sigma \sqcup \{A_0\}, \delta, A_0, e)$ con

$$\delta \coloneqq \{(s, \epsilon, A_0, l, A_0 S), (\ell, \epsilon, A_0, e, \epsilon)\} \cup \{(\ell, a, a, \ell, \epsilon)\}_{a\in\Sigma} \cup \{(\ell, \epsilon, x, l, w^\mathrm{R})\}_{(x,w)\in\mathcal{R}}$$

cumple $L(G) = L(\mathcal{A})$.

Lema del bombeo o ***pumping lemma*:** Si $L \in \mathcal{CF}$, existe $p \in \mathbb{N}$ tal que toda $w \in L$ con $|w| \geq p$ admite una descomposición $w = uvxyz$ con $|vy| > 0$ y $|vxy| \leq p$ tal que para $n \in \mathbb{N}$, $uv^n xy^n z \in L$. **Demostración:** Sean $G = (V, \Sigma, \mathcal{R}, S)$ una GLC para L y $b \coloneqq \max_{(A \to v) \in \mathcal{R}} |v|$, en cualquier árbol de derivación de G ningún nodo tiene más de b hijos, con lo que si un árbol tiene altura h, hay no más de b^h nodos a distancia h de la raíz y por tanto la cadena tiene longitud máxima b^h, y por contrarrecíproco, una cadena con longitud $b^h + 1$ o más tiene altura al menos $h + 1$. Sean entonces $p \coloneqq b^{|V|+1}$ y $w \in L$ con $|w| \geq p$, tomamos un árbol de derivación τ de w con número de nodos mínimo, cuya altura será al menos $|V| + 1$ ya que $|w| \geq b^{|V|+1} \geq b^{|V|} + 1$. Un camino de longitud máxima de la raíz a una hoja tendrá al menos $|V| + 2$ nodos, de los que a lo sumo 1 será terminal y el resto variables. Así, de entre las $|V| + 1$ variables inferiores (más lejanas a la raíz), habrá una variable R repetida, y llamamos x a la subcadena generada por la R inferior, vxy a la generada por la superior y $uvxyz$ a w ($S \Rightarrow^* uRz \Rightarrow^* uvRyz \Rightarrow^* uvxyz$). Sustituyendo el subárbol de la inferior por el de la superior obtenemos $uv^2xy^2z \in L$, y repitiendo obtenemos $uv^n xy^n z \in L$ para $n > 1$, mientras que sustituyendo el de la superior por el de la inferior obtenemos $uxz \in L$. Para ver que $|vy| > 0$, si fuera $|vy| = 0$ sería $vxy = x$ y sustituyendo el subárbol de la R superior por el de la inferior obtendríamos un árbol de derivación de w con menos nodos que τ#. Finalmente, como el R superior está entre las $|V| + 1$ variables inferiores, la altura de su subárbol es como mucho $|V| + 1$ y $|vxy| \leq b^{|V|+1} = p$.

El lenguaje $\{a^n b^n c^n\}_{n \in \mathbb{N}}$ no es libre de contexto. **Demostración:** Si lo fuera, tendría una longitud de bombeo p por el lema anterior. Sean $w \coloneqq a^p b^p c^p$ y $uvxyz \coloneqq w$ una descomposición en las condiciones de dicho lema. Si v o y contuviera más de un tipo de símbolo, $uv^2xy^2z \in L$ contendría símbolos en orden incorrecto#, por lo que v e y contienen cada una un sólo tipo de símbolo y, como al menos una de las 2 no es vacía, hay un tipo de símbolo no contenido en ninguna, luego $uv^2xy^2z \in L$ no contiene el mismo número de cada tipo#.

$\mathcal{CF}$ es cerrado para la unión, concatenación y clausura. **Demostración:** Dadas gramáticas $(V_1, \Sigma_1, \mathcal{R}_1, S_1)$ y $(V_2, \Sigma_2, \mathcal{R}_2, S_2)$ que generan respectivamente L_1 y L_2, la gramática $G \coloneqq (V_1 \sqcup V_2 \sqcup \{S\}, \Sigma_1 \cup \Sigma_2, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1, S \to S_2\}, S)$ genera $L_1 \cup L_2$. En efecto, si $w \in L(G)$, $S \Rightarrow S_1 \Rightarrow^* w$ o $S \Rightarrow S_2 \Rightarrow^* w$, y si $w \in L_1 \cup L_2$, por ejemplo $w \in L_1$, entonces $S \Rightarrow S_1 \Rightarrow^* w$. $G \coloneqq (V_1 \sqcup V_2 \sqcup \{S\}, \Sigma_1 \cup \Sigma_2, \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{S \to S_1 S_2\}, S)$ genera $L_1 L_2$. En efecto, si $w \in L(G)$, de la raíz del árbol de derivación parte un nodo S_1 y uno S_2, mostrando que $S_1 \Rightarrow^* u$ y $S_2 \Rightarrow^* v$ para ciertos u y v con $uv = w$, y si $w \in L_1 L_2$, sean $uv = w$ con $u \in L_1$ y $v \in L_2$, $S \Rightarrow S_1 S_2 \Rightarrow^* uS_2 \Rightarrow^* uv = w$. Finalmente, $G \coloneqq (V_1 \sqcup \{S\}, \Sigma_1, \mathcal{R}_1 \cup \{S \to S_1 S, S \to \epsilon\}, S)$ genera L_1^*. Sea $w \in L(G)$, por inducción en la altura h de un árbol de derivación de w, si $h = 0$ la derivación es $S \Rightarrow \epsilon$, luego $w = \epsilon \in L_1^*$, y para $h > 0$, el primer paso es $S \Rightarrow^* S_1 S$, pero el subárbol de este último nodo S tiene altura $h - 1$ y por tanto genera un $v \in L_1^*$, y como el nodo S_1 genera un $u \in L_1$, $w = uv \in L_1 L_1^* = L_1^*$. Recíprocamente, para $n \in \mathbb{N}$, podemos hacer $S \Rightarrow S_1 S \Rightarrow S_1^2 S \Rightarrow \cdots \Rightarrow S_1^n S \Rightarrow S_1^n$ y por tanto $L(G) \supseteq L_1^n$.

$\mathcal{CF}$ no es cerrado para la intersección, el complemento y la diferencia. **Demostración:** $L_1 \coloneqq \{a^n b^n c^m\}_{n,m \in \mathbb{N}}$ y $L_2 \coloneqq \{a^m b^n c^n\}_{n,m \in \mathbb{N}}$ son libres de contexto, pero $L_1 \cap L_2 = \{a^n b^n c^n\}_{n \in \mathbb{N}} \notin \mathcal{CF}$. Si fuera cerrado para la diferencia, lo sería para el complemento ya que $\overline{L} = \Sigma^* \setminus L$, y entonces lo sería para la intersección ya que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

Los autómatas de pila no son un buen modelo de computación, pues un ordenador puede reconocer si una cadena está en $\{a^n b^n c^n\}_{n \in \mathbb{N}}$ o no y un autómata de pila no puede.

Capítulo 3

Máquinas de Turing

En 1928, David Hilbert planteó el **problema de decisión** o ***Entscheidungsproblem***, consistente en encontrar un proceso mecánico para determinar, en una cantidad finita de pasos, si una proposición lógica de primer orden es un teorema o no. Hilbert asumía que tal proceso existía, pero en 1936, Alonzo Church en Princeton y Alan Turing en Cambridge probaron, de forma independiente, que no. Para ello tuvieron que definir formalmente este tipo de procesos, llamados algoritmos, Church a partir de su **cálculo** λ, un lenguaje formal para escribir funciones computables, y Turing a partir de sus ***computing machines*** o **máquinas de Turing**, una extensión de los autómatas de pila que permite moverse a través de la memoria.

3.1. Máquinas de Turing

Un **modelo de computación** MOD es una clase de autómatas en la que cada uno tiene asociado un alfabeto Σ y dos conjuntos disjuntos $L_{\mathrm{A}}, L_{\mathrm{R}} \in \Sigma^*$ de cadenas que **acepta** y que **rechaza**, respectivamente.

Una **máquina de Turing** (MT) es una tupla $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$ formada por un conjunto finito de **estados** Q, un alfabeto de la **entrada** Σ que no contiene al **símbolo blanco** B, un alfabeto de la **cinta** $\Gamma \supseteq \Sigma \cup \{\mathrm{B}\}$, una **función de transición** $\delta : D \subseteq (Q \times \Gamma) \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$, un **estado inicial** q_0 y un **estado final** o **de aceptación** q_{F}. La representación gráfica es similar a la de una PDA, representando una transición $\delta(q, x) = (r, y, d)$ con la línea «$x : y, d$» sobre una flecha de q a r.

Una **configuración** de $\mathcal{M}$ es una tupla (c, q, p) formada por el **contenido de la cinta** $c \in \Gamma^{\mathbb{N}}$, el **estado actual** $q \in Q$ y la **posición de la cabeza de lectura/escritura** $p \in \mathbb{N}$, y tal que casi todos los $c_i = \mathrm{B}$. Podemos representar una configuración (c, q, p) como «$c_0 \cdots c_{p-1} q c_p \cdots c_n$», donde $n \geq \max(\{n \mid c_n \neq \mathrm{B}\} \cup \{p-1\})$.

La **configuración inicial** para una $w \in \Sigma^*$ es $q_0 w$. Sean $a, b, c \in \Gamma$, $u, v \in \Gamma^*$ y $q, r \in Q$, la configuración $uaqbv$ lleva a $uqacv$ si $\delta(q, b) = (q, c, \mathrm{L})$, y la configuración $uqbv$ lleva a $ucqv$ si $\delta(q, c) = (q, c, \mathrm{R})$. Una **configuración final**, **aceptante** o **de aceptación** es una con estado q_{F}. La secuencia de configuraciones de w es la que empieza por su configuración inicial y, desde cada configuración no final, la siguiente es la configuración a la que esta lleva. Esta termina en la primera configuración final, en cuyo caso la MT **acepta** w, o la primera que no lleva a ninguna, en cuyo caso la MT **rechaza** w, y puede no terminar.

Una versión equivalente termina solo cuando la configuración no lleva a ninguna otra, y entonces acepta la cadena si dicha configuración tiene un estado en un conjunto de estados finales $F \subseteq Q$ y la rechaza en otro caso. Otra tiene, en vez de q_{F}, dos estados q_{accept} y q_{reject}, de modo que $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ es total y la MT termina aceptando si llega a q_{accept} o rechazando si termina en q_{reject}. Otra variación equivalente, compatible con las otras, consiste en tener una cinta que se extiende infinitamente a ambos lados, no solo a uno, con lo que si $\delta(q, b) = (q, c, \mathrm{L})$, la configuración qbv lleva a $q\mathrm{B}cv$.

Es común describir una MT de forma algorítmica, con instrucciones de más alto nivel, pudiendo usar una instrucción de alto nivel si se puede implementar en una MT. Son instrucciones de alto nivel:

1. Una secuencia de instrucciones.

   Para terminar una instrucción, se establece el estado al de inicio de la siguiente.

2. Moverse sin escribir.

   Se escribe lo mismo que se leyó.

3. Escribir un símbolo y quedarse en la misma posición.

   Escribir y moverse a la derecha, luego moverse incondicionalmente a la izquierda.

4. Ejecutar una u otra instrucción, o ninguna, según el símbolo leído.

   Para estos símbolos, se mueve al inicio de la instrucción correspondiente, que termina en el inicio de la siguiente. Para los que no se hace ninguna, se pasa directamente al inicio de la siguiente.

5. Ejecutar una u otra instrucción según el símbolo leído y los n anteriores para n fijo.

   Se mueve n veces a la izquierda. Si se lee x_1, se mueve a la derecha y se pasa a un cierto estado $\overline{x_1}$, desde el que si se lee x_2, se mueve a la derecha y se pasa a $\overline{x_1 x_2}$, etc., hasta llegar al estado $\overline{x_1 \cdots x_n}$.

6. Ejecutar una instrucción mientras el símbolo leído cumpla una condición.

   Para estos símbolos, pasar a la instrucción, que al terminar vuelve a la comprobación. Para el resto, pasar a la siguiente.

7. Detectar el límite izquierdo de la cinta.

   Se añade un nuevo estado inicial q_{i} y el símbolo \$ al alfabeto de la cinta. Desde q_{i} se escribe \$ y, en bucle, se mueve a la derecha y se escribe el símbolo de la posición anterior, hasta que este sea B. Entonces se va moviendo a la izquierda hasta encontrar \$, y luego una posición a la derecha y se pasa a q_0. Esto mueve la cadena una posición a la derecha y escribe \$ al principio, con lo que detectar el límite izquierdo de la cinta es detectar \$.

8. Volver al principio de la cinta.

   Una vez hecho lo anterior, ir a la izquierda hasta que se lea \$.

9. Recorrer una cadena detectando una de cada $n \in \mathbb{N}^*$ apariciones de un cierto símbolo.

   Se usan n estados en ciclo. Mientras no se lea B, se pasa al estado siguiente si se lee el símbolo o al mismo en otro caso, y se mueve a la derecha. En el último estado, antes de hacer esto, se detecta si está el símbolo y se actúa en consecuencia.

10. Ejecutar una instrucción u otra según el número de apariciones de cierto símbolo módulo un $n \in \mathbb{N}^*$ fijo.

    Se hace un ciclo como el anterior y, cuando se lee B, se ejecuta una u otra instrucción según el estado.

11. Contar el número de apariciones de un cierto símbolo hasta un límite n.

    Se usan los estados $c_0, \ldots, c_n$ en ciclo. En c_i, si se lee B, se realiza la acción para i apariciones; si se lee el símbolo, se pasa a c_{i+1} y se mueve a la derecha o, si $i = n$, se ejecuta la acción para más de n apariciones; en otro caso se mueve a la derecha.

3.2. Tesis de Church-Turing

Un modelo de computación MOD **decide** un lenguaje L si existe un $\mathcal{M} \in \text{MOD}$ con $L_\text{A} = L$ y $L_\text{R} = \overline{L}$, y **enumera** L si existe un $\mathcal{M} \in \text{MOD}$ con $L_\text{R} = \overline{L}$. Claramente MOD enumera todas las cadenas que decide.

Dados dos modelos de computación MOD_1 y MOD_2, MOD_1 es **menos expresivo** que MOD_2, $\text{MOD}_1 \preceq \text{MOD}_2$, si MOD_2 enumera todo lenguaje enumerado por MOD_1 y decide todo lenguaje decidido por MOD_1, en cuyo caso es **equivalente** a MOD_2, $\text{MOD}_1 \equiv \text{MOD}_2$, si $\text{MOD}_2 \preceq \text{MOD}_1$, y **estrictamente menos expresivo** que MOD_2, $\text{MOD}_1 \prec \text{MOD}_2$, en otro caso.

Son equivalentes a MT:

1. Las **máquinas de Turing con *stay*** (SMT), como las MT pero con función de transición de la forma $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}, \text{S}\}$, donde S corresponde a no moverse en la cinta, es decir, si $q, r \in Q$, $a, b \in \Gamma$, $u \in \Gamma^*$ y $v \in \Gamma^\mathbb{N}$, $uqav$ lleva a $urbv$ si $\delta(q, a) = (r, b, \text{S})$.

   MT $\subseteq$ SMT, y toda SMT se puede convertir en una MT cambiando cada transición con S por una que primero se mueve a la derecha y luego a la izquierda.

2. Las **máquinas de Turing con *return*** (RMT), como las MT pero con función de transición de la forma $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{E}, \text{R}\}$, donde E corresponde a volver al principio de la cinta, es decir, si $q, r \in Q$, $a, b \in \Gamma$, $u \in \Gamma^*$ y $v \in \Gamma^\mathbb{N}$, $uqav$ lleva a $rubv$ si $\delta(q, a) = (r, u, \text{E})$.

3. Las **máquinas de Turing multicinta**, k-MT para cierto $k \in \mathbb{N}^*$, como las MT pero con función de transición $\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{\text{L}, \text{R}\})^k$ y configuraciones en $Q \times (\Gamma^\mathbb{N} \times \mathbb{N})^k$, de modo que hay k cintas cada una con su propio lector independiente y en cada transición se puede escribir y mover en cada una de las cintas. La configuración inicial para una entrada $w \in \Sigma^*$ es $(q_0, (w\text{B}\cdots, 0), (\text{B}\cdots, 0)^{k-1})$. Una configuración $(q, (u_1, n_1), \ldots, (u_k, n_k))$ lleva a otra $(r, (v_1, m_1), \ldots, (v_k, m_k))$ si, siendo $\delta(q, u_{1n_1}, \ldots, u_{kn_k}) = (r, (c_1, d_1), \ldots, (c_k, d_k))$, para $i \in \{1, \ldots, k\}$, v_i es como u_i pero cambiando el término n_i-ésimo por c y, bien $d_i = \text{L}$ y $m_i = n_i - 1$, bien $d_i = \text{R}$ y $m_i = n_i + 1$.

   Claramente una MT puede ser simulada por una k-MT que en todas las cintas salvo la primera simplemente se mueve a la derecha. Una k-MT $(Q, \Sigma, \Gamma, \delta, q_0, q_\text{F})$ puede ser simulada por una MT con alfabeto de la cinta $\Gamma \sqcup \Gamma \sqcup \{\#\}$, donde un símbolo x del segundo Γ lo representamos por $\dot{x}$. La idea es que cada cinta con contenido $u\text{B}\cdots$ se

representa por $\#u$ pero sustituyendo el símbolo x apuntado en esa cinta por $\dot{x}$. Al inicio, la MT sustituye la entrada w por $\#\dot{w}_1 w_2 \cdots w_{|w|}(\#\dot{\mathrm{B}})^{k-1}$, vuelve al principio y pasa a q_0. En un estado $q \in Q$, hace lo siguiente:

   a) Ir al principio e ir moviéndose a la derecha guardando en el estado las k primeras apariciones de símbolos del segundo Γ, $\dot{a}_1, \ldots, \dot{a}_k$, volviendo al principio al llegar a la k-ésima.

   b) Si $\delta(q, a_1, \ldots, a_k) = (r, (s_1, d_1), \ldots, (s_k, d_k))$, para i de 1 a k: Mientras no se lea $\#$, ir a la derecha. Mientras no se lea $\dot{a}_i$, ir a la derecha. Cambiar $\dot{a}_i$ por s_i. Si $d_i = \mathrm{L}$, ir a la izquierda, y si $d_i = \mathrm{R}$, ir a la derecha. Leer x y cambiarlo por $\dot{x}$, o si $x = \#$, rechazar si $d_i = \mathrm{L}$ o insertar antes $\dot{\mathrm{B}}$ desplazando el resto a la derecha si $d_i = \mathrm{R}$.

   c) Pasar al estado r.

4. Las **máquinas de Turing no deterministas** (MTND), como las máquinas de Turing pero con función de transición $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$. Para $a, b, c \in \Gamma$, $u, v \in \Gamma^*$ y $q, r \in Q$, la configuración $uaqbv$ lleva a $uqacv$ si $(q, c, \mathrm{L}) \in \delta(q, b)$, y la configuración $uqbv$ lleva a $ucqv$ si $(q, c, r) \in \delta(q, c)$. La MTND acepta una cadena w si, de todas las secuencias de configuraciones que empiezan con la configuración inicial para dicha cadena y cada configuración de la secuencia lleva a la siguiente, existe una finita que acaba en q_F, y la rechaza si toda secuencia de este tipo es finita y no acaba en q_F.

   Toda MT es equivalente a una MTND con función de transición δ' dada por $\delta'(q, a) = \{n\}$ si $(q, a) \in D$ y $\delta(q, a) = n$ y $\delta'(q, a) = \emptyset$ si $(q, a) \notin D$. Para el recíproco, consideramos una MTND $(Q, \Sigma, \Gamma, \delta, q_0, q_\mathrm{F})$ y la convertimos en una 3-MT, que guardará la entrada en la cinta 1, simulará la MTND en la cinta 2 recorriendo en anchura el árbol de posibilidades y guardará en la cinta 3 el camino actual en dicho árbol. Sea $b \coloneqq \max_{q \in Q}^{a \in \Gamma} |\delta(q, a)|$, el alfabeto de la cinta será $\Gamma \sqcup \{p_1, \ldots, p_b\} \sqcup \{\$, \#\}$, y hará lo siguiente:

   a) Escribir $\$p_1$ en la cinta 3 y guardar $c \leftarrow$ FALSE en el estado.

   b) Copiar la cinta 1 en la 2, escribiendo $\#$ tras el final de la cadena, y volver al principio en ambas, usando \$ como marca de inicio. Guardar $q \leftarrow q_0$ en el estado del 3-MT.

   c) Si $q = q_\mathrm{F}$, aceptar. Leer a de la cinta 2 y p_i de la 3. Si $a = \#$, $a \coloneqq \mathrm{B}$, escribir B, ir a la derecha, escribir $\#$ e ir a la izquierda. Si $p_i = \mathrm{B}$, hacer $c \leftarrow$ TRUE (indicando que existe una rama por la que se puede seguir si se da un paso más), moverse a la izquierda en la cinta 3 e ir al paso $4d$. Si $i > |\delta(q, a)|$, escribir B, moverse a la izquierda en la cinta 3 e ir al paso $4d$. Si $i \leq |\delta(q, a)|$, sea (r, b, d) el i-ésimo elemento de $\delta(q, a)$, escribir b en la cinta 2, moverla en la dirección d, guardar $q \coloneqq r$ y, si se lee \$, ir al paso $4d$, y si no mover la cinta 3 a la derecha y repetir este paso.

   d) Si en la cinta 3 se lee p_b, escribir B, moverse a la izquierda y repetir este paso. Si se lee \$, si $c =$ TRUE, hacer $c \leftarrow$ FALSE, ir a la izquierda y escribir p_1 hasta que se lea B, sobrescribiendo la primera B, y si $c =$ FALSE, rechazar. Si se lee p_i con $i < b$, cambiar por p_{i+1}. Volver al principio de la cinta (sin contar \$) e ir al paso $4b$.

En su artículo sobre el cálculo λ, Church probó que este es equivalente a las **funciones recursivas**, propuestas por Kurt Gödel en 1931, y Turing, cuyo artículo salió después, probó que las MT son equivalentes al cálculo λ. Esto llevó a la **tesis de Church-Turing**, que afirma que esta definición de algoritmo se corresponde con la noción intuitiva, o que la máquina de

Turing es el modelo de computación más expresivo posible y todos los modelos suficientemente expresivos son equivalentes a este.

Posteriormente aparecieron otras definiciones formales de algoritmo, como las **funciones parciales**, las **máquinas con infinitos registros** o el **lenguaje S** (simple), todas equivalentes a MT. Un lenguaje de programación es **Turing completo** si es equivalente a MT.

3.3. Lenguajes decidibles y enumerables

Un lenguaje es **decidible** o **recursivo** si lo decide MT e **indecidible** en otro caso, y es **Turing-reconocible**, **recursivamente enumerable** o **enumerado** si lo enumera MT. Llamamos $\mathcal{DEC}$ a la clase de lenguajes decidibles y $\mathcal{RE}$ a la de lenguajes recursivamente numerables.

Algunos lenguajes decidibles:

1. $\{0^{2^n}\}_{n\in\mathbb{N}}$.

   Si se lee B, rechazar (longitud 0). En bucle:

   a) Si solo hay un 0, se acepta.

   b) Si 0 aparece un número de veces impar, rechazar.

   c) Marcar con # uno de cada 2 ceros, reduciendo el número de ceros a la mitad.

2. $\{a^n b^n c^n\}_{n\in\mathbb{N}}$.

   Se va leyendo la cadena de izquierda a derecha y, si hay una a después de una b o c o una b después de una c, se rechaza. Se vuelve al principio. En bucle, primero se comprueba si hay alguna a, b o c, aceptando si no hay ninguna, se vuelve al principio de la cinta, se marca con # la primera a, luego la primera b y luego la primera c, rechazando si alguna de las 3 no existe, y se vuelve al principio.

Dados $L_1, L_2 \in \mathcal{DEC}$:

1. $L_1 \cup L_2 \in \mathcal{DEC}$.

   Sean $(Q_1, \Sigma, \Gamma_1, \delta_1, q_{01}, q_{F1})$ y $(Q_2, \Sigma, \Gamma_2, \delta_2, q_{02}, q_{F2})$ MTs que deciden L_1 y L_2, respectivamente, $(Q_1 \sqcup Q_2 \sqcup \{q_0\} \sqcup \{q_F\}, \Sigma, \Gamma_1 \cup \Gamma_2, \delta', q_0, q_F)$ con las transiciones de δ_1 y δ_2 más una de q_0 a q_{01} y q_{02} de forma no determinista y una de q_{F1} y otra de q_{F2} a q_F decide $L_1 \cup L_2$.

2. $L_1 L_2 \in \mathcal{DEC}$.

   Hacemos una 2-MT que, para cada posición en la cadena de la cinta 1 y la primera B en esta, marcando la posición actual de algún modo, copia la subcadena desde el principio a la posición anterior a la actual en la cinta 2, ejecuta una MT que reconoce L_1 en dicha cinta y, si acepta, hace lo propio con una MT que reconoce L_2 y la subcadena que empieza en la posición actual y termina al final, usando # como en la demostración de equivalencia de las MTND.

3. $L_1^* \in \mathcal{DEC}$.

4. $L_1 \cap L_2 \in \mathcal{DEC}$.

5. $\overline{L_1} \in \mathcal{DEC}$.

6. $L_1 \setminus L_2 \in \mathcal{DEC}$.

Como **teorema**, $\mathcal{CF} \subsetneq \mathcal{DEC} \subsetneq \mathcal{RE}$.

$1 \subseteq 2]$ Un DFA lo puede simular una 2-MT usando la cinta 1 para leer de la cadena y la cinta 2 para la pila.

$2 \nsubseteq 1]$ $\{a^n b^n c^n\}_{n\in\mathbb{N}} \in \mathcal{DEC} \setminus \mathcal{CF}$.

$2 \subseteq 3]$ Trivial.

Sean $L_1, L_2 \in \mathcal{RE}$:

1. $L_1 \cup L_2 \in \mathcal{RE}$.

   Tomamos una MTND que al inicio, de forma no determinista, ejecute una MT que enumere L_1 o una que enumere L_2.

2. $L_1 \cap L_2 \in \mathcal{RE}$.

   Sean $\mathcal{M}_1$ una MT que enumera L_1 y $\mathcal{M}_2$ una que enumera L_2, se ejecutan $\mathcal{M}_1$ y $\mathcal{M}_2$ a la vez sobre dos copias de la entrada (alternándolas), aceptando cuando ambas hayan aceptado y rechazando si una rechaza.

3. $L_1 L_2 \in \mathcal{RE}$.

   Similar al caso decidible pero haciendo todas las simulaciones en paralelo: primero 1 paso de cada una sobre una copia de la subcadena correspondiente, luego 2 pasos, 3, etc., aceptando si, para alguna división de la cadena de entrada, las dos máquinas aceptan.

4. $L_1^* \in \mathcal{RE}$.

   $L_1^* = \{\epsilon\} \cup (L_1 \setminus \{\epsilon\})^*$, por lo que si la entrada es ϵ aceptamos y, en otro caso, hacemos como en el caso anterior pero iterando sobre todas las particiones de la entrada con un número arbitrario de subcadenas y cada subcadena no vacía. Para ello en la cinta 2 se itera sobre $\#\{N, \#\}^{n-1}$, donde $\#$ indica el principio de una subcadena a considerar, y aceptando cuando todas las subcadenas de la misma partición han aceptado.

3.4. Algoritmos

Las máquinas de Turing no solo permiten reconocer cadenas, sino ejecutar algoritmos en general. La entrada siempre es una cadena, por lo que para que otro tipo de objeto actúe de entrada hay que representarla como una cadena y la máquina de Turing debe decodificar esta representación. Dado un objeto O, llamamos $\langle O \rangle$ a la cadena que codifica O en cierta representación, y dados objetos $O_1, \ldots, O_n$, llamamos $\langle O_1, \ldots, O_n \rangle$ a la cadena que codifica $(O_1, \ldots, O_n)$. La MT comprobará la entrada y rechazará si no tiene un formato válido. La salida podría escribirla como cadena al principio de la cinta.

Capítulo 4

Decidibilidad

Algunos lenguajes decidibles:

1. $\text{Acc}^{\text{DFA}} := \{\langle \mathcal{A}, w\rangle \mid \text{el DFA } \mathcal{A} \text{ acepta la cadena } w\}$.
   Hay que simular $\mathcal{A}$ con entrada w.
2. $\text{Acc}^{\text{NFA}} := \{\langle \mathcal{A}, w\rangle \mid \text{el NFA } \mathcal{A} \text{ acepta la cadena } w\}$.
3. $\text{Acc}^{\text{PDA}} := \{\langle \mathcal{A}, w\rangle \mid \text{el PDA } \mathcal{A} \text{ acepta la cadena } w\}$.
4. $\text{Empty}^{\text{DFA}} := \{\langle \mathcal{A}\rangle \mid \text{el DFA } \mathcal{A} \text{ no acepta ninguna cadena}\}$.
   Hay que comprobar si existe algún camino del estado inicial a algún estado final.
5. $\text{Empty}^{\text{NFA}} := \{\langle \mathcal{A}\rangle \mid \text{el NFA } \mathcal{A} \text{ no acepta ninguna cadena}\}$.
   Análogo.
6. $\text{Empty}^{\text{PDA}} := \{\langle \mathcal{A}\rangle \mid \text{el PDA } \mathcal{A} \text{ no acepta ninguna cadena}\}$.

Si $L \in \mathcal{DEC}$, $L^* \in \mathcal{DEC}$.

4.1. Numeración de Gödel

La clase de las máquinas de Turing es numerable, y en particular lo es el conjunto de lenguajes recursivamente enumerables. **Demostración:** hay infinitas máquinas de Turing, todas se pueden expresar como cadenas sobre un mismo alfabeto (describiendo los estados y símbolos como cadenas de ciertos símbolos con un terminador) y el conjunto de estas cadenas es numerable.

La **numeración de Gödel** es una asociación inyectiva de las cadenas de un cierto alfabeto a los números naturales, para lo cual se asigna un natural positivo m_s a cada símbolo s del alfabeto y, si $p_1, p_2, \ldots$ es la secuencia de todos los primos, una cadena $w_1 \cdots w_n$ se representa como $p_1^{m_{w_1}} \cdots p_n^{m_{w_n}}$. Esta permite codificar proposiciones lógicas o máquinas de Turing como naturales. La codificación y decodificación son **computables**, es decir, existe una MT que siempre termina y que puede codificar y decodificar números de Gödel.

Todo conjunto A cumple $|A| < |\mathcal{P}(A)|$. **Demostración:** Claramente $|A| \leq |\mathcal{P}(A)|$, pero si hubiera una biyección $f : A \to \mathcal{P}(A)$, sean $B \coloneqq \{x \in A \mid x \notin f(x)\}$ e $Y \coloneqq f^{-1}(B)$, si $y \in B$, $y \notin f(y) = B\#$, y si $y \notin B$, $y \in f(y) = B\#$.

Existen lenguajes no recursivamente enumerables, pues el conjunto de lenguajes sobre un alfabeto Σ no vacío, $\mathcal{P}(\Sigma^*)$, es no numerable y el conjunto de lenguajes recursivamente enumerables sobre Σ es a lo más numerable.

$\mathcal{REG}$, $\mathcal{CF}$, $\mathcal{DEC}$ y $\mathcal{RE}$ no son cerrados para subconjuntos, pues si $L \coloneqq 0^* \in \mathcal{REG} \subseteq \mathcal{CF} \subseteq \mathcal{DEC} \subseteq \mathcal{RE}$, L es numerable y por tanto $\mathcal{P}(L)$ no lo es, pero como $\mathcal{RE}$ es numerable existe $L' \in \mathcal{P}(L) \setminus \mathcal{RE}$.

4.2. La máquina de Turing universal

Existe una **máquina de Turing universal** $\mathcal{U}$ que permite simular cualquier máquina de Turing, recibiendo una máquina de Turing $\mathcal{M}$ y una cadena w y aceptando si $\mathcal{M}$ acepta w, rechazando si $\mathcal{M}$ rechaza w y no terminando en otro caso.

$$K \coloneqq \{\langle \mathcal{M}, w\rangle \mid \text{la MT } \mathcal{M} \text{ acepta con entrada } w\} \in \mathcal{RE} \setminus \mathcal{DEC}.$$

Demostración: $K \in \mathcal{RE}$ por ser el lenguaje enumerado por $\mathcal{U}$. Si fuera $K \in \mathcal{DEC}$, sea $\mathcal{H}$ una MT que decide K, existe $\mathcal{D}$ que decide $\{\langle \mathcal{M}\rangle \mid \mathcal{H} \text{ rechaza } \langle \mathcal{M}, \langle \mathcal{M}\rangle\rangle\}$, pero entonces $\mathcal{D}$ acepta $\mathcal{D}$ si y sólo si $\mathcal{H}$ rechaza $\langle \mathcal{D}, \langle \mathcal{D}\rangle\rangle$, si y sólo si $\mathcal{D}$ no acepta $\langle \mathcal{D}\rangle\#$.

Para un lenguaje $L \subseteq \Sigma^*$, $L, \overline{L} \in \mathcal{RE} \implies L, \overline{L} \in \mathcal{DEC}$, pues se puede crear una máquina que, dada su entrada, simule progresivamente pasos de máquinas que enumeren L y $\overline{L}$ hasta que una termine y aceptar o rechazar según cuál termine y con qué resultado.

Así, $\mathcal{RE}$ no es cerrado para el complemento, pues $K \in \mathcal{RE}$ y si fuera $\overline{K} \in \mathcal{RE}$ sería $K \in \mathcal{DEC}\#$. Tampoco lo es para la diferencia, pues de serlo lo sería para el complemento ya que $\Sigma^* \in \mathcal{RE}$.

Un lenguaje es **co-recursivamente enumerable** si su complementario es recursivamente enumerable, y llamamos CO-$\mathcal{RE}$ a la clase de lenguajes co-recursivamente enumerables. En particular $\mathcal{DEC} = \mathcal{RE} \cap \text{CO-}\mathcal{RE}$. Un lenguaje es **altamente indecidible** si no es $\mathcal{RE}$ ni CO-$\mathcal{RE}$.

Capítulo 5

Reducibilidad

Una **reducción** de un lenguaje $L_1 \subseteq \Sigma_1^*$ a un lenguaje $L_2 \subseteq \Sigma_2^*$ es una $f : \Sigma_1^* \to \Sigma_2^*$ tal que $\forall w \in \Sigma_1^*, (w \in L_1 \iff f(w) \in L_2)$. Una MT $\mathcal{M}$ **computa** una función $f : \Sigma_1^* \to \Sigma_2^*$ si siempre termina y, para $w \in \Sigma_1^*$, $\mathcal{M}$ termina conteniendo en su cinta únicamente $f(w)$, y entonces f es **computable**. Una ***mapping* reducción** de $L_1 \subseteq \Sigma_1^*$ a $L_2 \subseteq \Sigma_2^*$ es una reducción computable de L_1 a L_2. Si existe decimos que L_1 se puede **reducir** a L_2, $L_1 \leq_{\mathrm{m}} L_2$, y esta relación es claramente transitiva.

Equivalentemente, un **oráculo** para un lenguaje L es una caja negra que decide L, y un lenguaje A **se reduce** a un lenguaje B si existe una MT con un oráculo de B que decide A.

Teoremas de reducibilidad:

1. Si $A \leq_{\mathrm{m}} B$ y $A \notin \mathcal{DEC}$ entonces $B \notin \mathcal{DEC}$.

   Si $B \in \mathcal{DEC}$, sean $\mathcal{H}$ una MT que decide B y $\mathcal{F}$ una que calcula una *mapping* reducción de A a B, una MT que ejecuta $\mathcal{F}$ y luego $\mathcal{H}$ decide A, luego $A \in \mathcal{DEC}$.

2. Si $A \leq_{\mathrm{m}} B$ y $A \notin \mathcal{RE}$ entonces $B \notin \mathcal{RE}$.

   Análogo.

Ejemplos:

1. **Problema de la parada.**

   $$\mathrm{HALT}^{\mathrm{MT}} \coloneqq \{\langle \mathcal{M}, w\rangle \mid \mathcal{M} \text{ es una MT que para con entrada } w\} \notin \mathcal{DEC}.$$

   Sea $\mathcal{M}'$ una MT que ejecuta $\mathcal{M}$, acepta si $\mathcal{M}$ acepta y entra en bucle infinito en otro caso, $\langle \mathcal{M}, w\rangle \mapsto \langle \mathcal{M}', w\rangle$ es una *mapping* reducción de K a $\mathrm{HALT}^{\mathrm{MT}}$, y $K \notin \mathcal{DEC}$.

2. $\mathrm{EMPTY}^{\mathrm{MT}} \coloneqq \{\langle \mathcal{M}\rangle \mid \mathcal{M} \text{ es una MT que no acepta ninguna cadena}\} \notin \mathcal{DEC}$.

   Dadas una MT $\mathcal{M}$ y una cadena w, definimos $\mathcal{M}_w$ como una máquina que comprueba si la entrada es w, rechaza si no lo es y ejecuta $\mathcal{M}$, de modo que

   $$L(\mathcal{M}_w) = \begin{cases} \{w\}, & w \in L(\mathcal{M}); \\ \emptyset, & \text{en otro caso,} \end{cases}$$

y $\langle \mathcal{M}, w\rangle \mapsto \langle \mathcal{M}_w\rangle$ es una *mapping* reducción de K a $\overline{\text{EMPTY}^{\text{MT}}}$, con lo que $\overline{\text{EMPTY}^{\text{MT}}} \notin \mathcal{DEC}$ y por tanto $\text{EMPTY}^{\text{MT}} \notin \mathcal{DEC}$.

3. Pass $\coloneqq \{\langle \mathcal{M}, w, q\rangle \mid \mathcal{M}$ es una MT que, con entrada w, pasa por el estado $q\} \notin \mathcal{DEC}$.

   $\langle \mathcal{M}, w\rangle \mapsto \langle \mathcal{M}, w, q_f\rangle$, donde q_f es el estado final de $\mathcal{M}$, es una *mapping* reducción de K a Pass, pues si $\mathcal{M}$ acepta w, pasa por q_f cuando tiene entrada w, y si no la acepta es porque no pasa por q_f.

$L \in \mathcal{RE}$ es **completo** para $\mathcal{RE}$ si todo $L' \in \mathcal{RE}$ se puede reducir a L. Entonces:

1. K es completo en $\mathcal{RE}$.

   Dado $L \in \mathcal{RE}$, sea $\mathcal{M}$ una máquina que enumera L, $\langle w\rangle \mapsto \langle \mathcal{M}, w\rangle$ es una *mapping* reducción de L a K.

2. Si L es completo en $\mathcal{RE}$ y $L' \in \mathcal{RE}$ cumple $L \leq_{\text{m}} L'$, L' es completo en $\mathcal{RE}$.

Una **propiedad** de los lenguajes recursivamente enumerables es una proposición cuya única variable libre se refiere a un lenguaje en $\mathcal{RE}$, y se puede identificar con la subclase de los lenguajes en $\mathcal{RE}$ que cumplen la propiedad. Es **trivial** si es $\emptyset$ o $\mathcal{RE}$. Así, $\mathcal{REG}$, $\mathcal{CF}$, $\mathcal{DEC}$ y $\{L$ es finito$\}$ son propiedades no triviales.

Teorema de Rice: Para una propiedad P de $\mathcal{RE}$ no trivial,

$$\mathcal{L}_P \coloneqq \{\langle \mathcal{M}\rangle \mid \mathcal{M} \text{ es una MT con } L(M) \in P\} \notin \mathcal{DEC}.$$

Demostración: Supongamos $\mathcal{L}_P \in \mathcal{DEC}$, con lo que existe $\mathcal{M}_P$ que decide $\mathcal{L}_P$. Como P es no trivial, existen $L_1 \in \mathcal{L}_P$ y $L_2 \in \mathcal{RE} \setminus \mathcal{L}_P$, y podemos tomar que uno de los dos sea $\emptyset$. Si $L_2 = \emptyset$, sea $\mathcal{M}_{L_1}$ una MT que enumera L_1, para cada MT $\mathcal{M}$ y cada cadena w, definimos $\mathcal{M}_w$ como una MT que, ante una entrada x, ejecuta $\mathcal{M}$ con entrada w, entra en bucle infinito si $\mathcal{M}$ rechaza, y en otro caso ejecuta $\mathcal{M}_{L_1}$ con entrada x y devuelve el resultado de esta última máquina. Entonces

$$L(\mathcal{M}_w) = \begin{cases} L_1, & w \in L(\mathcal{M}); \\ \emptyset, & \text{en otro caso,} \end{cases}$$

luego $\langle \mathcal{M}, w\rangle \mapsto \langle \mathcal{M}_w\rangle$ es una *mapping* reducción de K a $\mathcal{L}_P$ y basta aplicar el primer teorema de reducibilidad. Si $L_1 = \emptyset$ por este argumento $\mathcal{RE} \setminus \mathcal{L}_P \notin \mathcal{DEC}$ y por tanto $\mathcal{L}_P \notin \mathcal{DEC}$.

Un lenguaje es **indecidible** si no es $\mathcal{RE}$, pero existen **grados de indecidibilidad** según los oráculos que hacen falta para poder enumerarlo.

Capítulo 6

Complejidad en tiempo

Una MT $\mathcal{M}$ que termina siempre tiene **tiempo de ejecución** o **complejidad en tiempo** $t : \mathbb{N} \to \mathbb{N}$ dada por $t(n) \coloneqq \text{máx}_{w \in \Sigma^n}\{\text{nº de pasos que } \mathcal{M} \text{ ejecuta con entrada } w\}$, donde cada paso es una regla de transición.

Dadas $f, g : \mathbb{N} \to \mathbb{R}^{\geq 0}$, $f(n)$ es O **grande** de $g(n)$, del **orden de magnitud** de $g(n)$ o $g(n)$ es un **límite superior asintótico** para $f(n)$, $f(n) = O(g(n))$, si $\exists c, n_0 \in \mathbb{N} : \forall n \geq n_0, (n) \leq cg(n)$. Casi nunca necesitamos conocer la complejidad en tiempo de un algoritmo con precisión, sino que basta conocer su orden de magnitud. Hay una jerarquía de órdenes $O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n \log n) \subseteq O(n^2) \subseteq O(n^3) \subseteq \cdots \subseteq O(2^n) \subseteq O(n^n)$, y siempre que sea posible, al especificar el orden de magnitud de una función, elegiremos el más pequeño.

Para $t : \mathbb{N} \to \mathbb{R}^{\geq 0}$, llamamos **clase de complejidad** $\text{TIME}(t(n))$ a la clase de los lenguajes decidibles por una MT con complejidad en tiempo $O(t(n))$. Esta magnitud se refiere al lenguaje o problema, no a un algoritmo concreto. En general, la clase de complejidad de un problema depende del modelo de computación usado, aun para modelos equivalentes. Este orden no difiere mucho entre modelos deterministas, pero sí entre un modelo determinista y uno no determinista.

Como **teorema**, si una k-MT tiene tiempo de ejecución $O(t(n))$ con $t(n) \geq n$, existe una MT equivalente con tiempo de ejecución $O(t(n)^2)$. **Demostración:** En la demostración de que k-MT $\equiv$ MT, para simular una k-MT con una MT, recorríamos la cinta para darle el formato correcto $(O(n))$ y, en cada paso de la k-MT, recorríamos la cinta 2 veces, primero para saber qué transición aplicar y luego para aplicarla, actualizando el contenido de las celdas y las posiciones de los cursores. Cada cinta tiene tamaño como mucho $O(\text{máx}\{t(n), n\}) = O(t(n))$, pues en cada transición se añade como mucho un caracter en cada cinta, con lo que los recorridos son $O(t(n))$ y, como se hacen $O(t(n))$ de estos, el tiempo total es $O(t(n)^2)$.

Una MTND es un **decisor** si, para toda cadena de entrada, todas sus secuencias de ejecución son finitas. Una MTND $\mathcal{N}$ que es un decisor tiene **tiempo de ejecución** o **complejidad en tiempo** $f : \mathbb{N} \to \mathbb{N}$ si $f(n)$ es el máximo número de pasos que ejecuta $\mathcal{N}$ en cualquiera de sus ramas para cualquier entrada de longitud n.

Como **teorema**, si una MTND tiene tiempo de ejecución $O(f(n))$ con $f(n) \geq n$, existe una MT equivalente con tiempo de ejecución $2^{O(f(n))}$. **Demostración:** En la demostración de que MTND $\equiv$ 3-MT, se calculan cada uno de los nodos haciendo búsqueda en anchura, pero como sabemos que todas las ramas terminan, podemos hacer búsqueda en profundidad

y solo detenernos al llegar a una hoja. Si hay hasta m pasos, hay como mucho c^m hojas y calcular cada una requiere una copia $O(n)$ y simular mc^m transiciones, más una cantidad de transiciones asintóticamente menor para llevar cuenta del *backtracking* y para evitar la interferencia entre una simulación y la siguiente, por lo que el total de transiciones es como mucho $O(n + mc^m) = O(n + f(n)c^{f(n)}) = O(f(n)c^{f(n)}) = O(2^{\log_2 f(n)+f(n)\log_2 c}) = 2^{O(\log_2 f(n)+f(n)\log_2 c)} = 2^{O(f(n))}$, y por el teorema anterior, al simular esto en un MT el orden máximo es $(2^{O(f(n))})^2 = 2^{O(2f(n))} = 2^{O(f(n))}$.

Nótese que $2^{O(f(n))}$ no es lo mismo que $O(2^{f(n)})$, pues por ejemplo $3^n \neq O(2^n)$ porque para cualesquiera $c, n_0 \in \mathbb{N}$ existe $n > n_0$ tal que $3^n > 2^n c$, sin más que tomar $n > \log_{\frac{3}{2}} c$, pero $3^n = 2^{\log_2 3^n} = 2^{n\log_2 3} = 2^{O(n)}$.

Capítulo 7

$\mathcal{P}$ y $\mathcal{NP}$

Un problema es **tratable** si se puede resolver de forma suficientemente eficiente, y es **intratable** si se puede resolver pero no lo suficientemente rápido como para que la solución sea práctica.

7.1. La clase $\mathcal{P}$

Un lenguaje es **decidible en tiempo polinómico** si está en

$$\mathcal{P} \coloneqq \bigcup_k \mathrm{TIME}(n^k).$$

Los problemas en esta clase se consideran tratables, y el resto intratables. Así, son intratables los problemas con orden de complejidad exponencial, que suelen corresponder a búsqueda exhaustiva o por fuerza bruta del espacio de búsqueda, y que solo se pueden resolver de forma general en casos sencillos. Realmente, si un problema en $\mathrm{TIME}(n^k)$ es tratable depende de k y de la aplicación particular, pero la distinción entre tiempo polinómico y exponencial es útil en la práctica.

Una **función polinómica** o **computable en tiempo polinómico** es una que una MT puede computar con complejidad en tiempo polinómica.

$\mathcal{P}$ no es afectada significativamente por las particularidades del modelo de computación, y de hecho es igual para una MT, una k-MT y un ordenador típico (extendido para que sea Turing-completo). Sin embargo, sí es afectada por la representación del problema, pues aunque casi todas las representaciones son equivalentes a este respecto, no todas lo son.

Una representación es **razonable** si existe una función polinómica con inversa polinómica para codificar objetos hacia una representación interna.[1] Las formas que hemos usado para representar autómatas, grafos, etc. son razonables, pero no lo es la representación unaria de números, pues es exponencialmente más larga que una representación en base 2 o más que sí es razonable.

[1] Esto es así pese a que no hemos definido «representación interna» y en principio esta depende de la implementación del algoritmo.

Entrada: Gramática $G = (V, \Sigma, \mathcal{R}, S)$ en forma normal de Chomsky y cadena $w_1 \cdots w_n \in \Sigma$.
Salida: Acepta si $w \in L(G)$, rechaza en otro caso.

```
si w = ε entonces
    si S → ε ∈ R entonces aceptar;
    sinó rechazar;
T ← {((i, j), ∅)}_{1≤i≤j≤n} ;        // T(i, j) contiene las variables que generan w_i ··· w_j.
para i ← 1 a n hacer
    para A ∈ V con A → w_i ∈ R hacer añadir A a T(i, i);
para l ← 2 a n hacer
    para i = 1 a n − l + 1 hacer
        j ← i + l − 1;
        para k ← i a j − 1 hacer
            para A → BC ∈ R hacer
                si B ∈ T(i, k) y C ∈ T(k + 1, j) entonces añadir A a T(i, j);
si S ∈ T(1, n) entonces aceptar;
sinó rechazar;
```

Algoritmo 3: Algoritmo CYK de programación dinámica para establecer si una cadena está en un cierto lenguaje libre de contexto en tiempo polinómico.

Están en $\mathcal{P}$:

1. La suma, resta saturada, producto, cociente entero y resto de números naturales.

2. RELPRIM $\coloneqq \{\langle x, y\rangle \mid x, y \in \mathbb{N}$ son primos relativos$\}$.

   Una forma de comprobarlo es usar el algoritmo de Euclides para ver que $\gcd\{x, y\} = 1$, repitiendo $(x, y) \leftarrow (y, x \bmod y)$ hasta que $y = 0$. La operación más cara en un paso es el módulo, y basta ver que el número de pasos es polinómico respecto al tamaño de la entrada, para lo que vemos que, excluyendo el primer paso, cada 2 pasos x se divide al menos por la mitad. En efecto, tras el primer paso $x > y$, de modo que si $\frac{x}{2} \geq y$, $x \bmod y < y \leq \frac{x}{2}$, y si $\frac{x}{2} < y$, $x \bmod y = x - y < \frac{x}{2}$, pero $x \bmod y$ será el valor de y en un paso y de x en dos.

3. PATH $\coloneqq \{\langle G, s, t\rangle \mid G$ es un grafo dirigido con un camino de s a $t\}$.

   Se añade el nodo s a una lista F. Mientras haya un nodo n en F, se elimina de n, se añade a C, se recorren los arcos añadiendo a F los que parten de n y van a un nodo que no está C. Se acepta si y sólo si al final t está en C.

4. 4-CLIQUE $\coloneqq \{\langle G\rangle \mid G$ es un grafo no dirigido con una 4-clique$\}$. Una **k-clique** es un subgrafo completo con k nodos.

   Si G tiene menos de 4 nodos, rechaza. En otro caso se enumeran las $\binom{n}{4} = \frac{n!}{4!(n-4)!} = O(n^4)$ combinaciones de 4 nodos y, para cada una, se comprueba si todos los pares de nodos están en el grafo.

5. EULCYCLE $\coloneqq \{\langle G\rangle \mid G$ es un grafo dirigido con un ciclo euleriano$\}$. Un **camino euleriano** es uno que pasa por todas las aristas exactamente una vez.

   Un teorema de Euler dice que un grafo dirigido G tiene un ciclo euleriano si y sólo si existe un camino entre cada par de vértices y todos los vértices tienen grado entrante igual a su grado saliente. Ver que hay un camino entre cada par de vértices es polinómico por serlo PATH y el número de pares de vértices, y ver que todos los vértices tienen grado entrante y saliente iguales también.

6. 2-COLOR $\coloneqq \{\langle G\rangle \mid G$ es un grafo no dirigido bipartito$\}$.

7. Todo $L \in \mathcal{CF}$.

   Algoritmo 3.

$\mathcal{P}$ es cerrada para unión, intersección, complemento, concatenación y clausura. **Demostración:** La unión, intersección y clausura son triviales. Respecto a la concatenación L_1L_2, para cada partición de la entrada en 2 partes, posiblemente vacías, se decide L_1 en la primera y L_2 en la segunda. Para ver que, si $L \in \mathcal{P}$, $L^* \in \mathcal{P}$, hacemos lo siguiente: Para una entrada $w_1 \cdots w_n$, si $n = 0$, acepta. En otro caso se usa una programación dinámica con una tabla $T(i, j)$ en la que, para $1 \leq i < j \leq n$, $T(i, j)$ indica si $w_i \cdots w_j \in L^*$. Para l de 1 a n, para i de 1 a $n - l + 1$, sea $j = i + l - 1$, si $w_i \cdots w_j \in L$, se marca $T(i, j)$, y de lo contrario, para k de i a $j - 1$, si están marcadas $T(i, k)$ y $T(k + 1, j)$, se marca $T(i, j)$. Finalmente se acepta o rechaza según $T(1, n)$ esté marcada o no. En total se hacen $O(n^3)$ consultas a T y $O(n^2)$ ejecuciones de la máquina que decide L.

7.2. La clase $\mathcal{NP}$

Dada una clase $\mathcal{C}$ de problemas que una MT puede solucionar con un cierto orden de complejidad en tiempo, llamamos $\mathcal{NC}$ a la clase de problemas que una MTND puede solucionar en dicho orden de complejidad.

Así, un lenguaje es NTIME($f(n)$) si existe una MTND que lo decide con complejidad en tiempo $O(f(n))$, y

$$\mathcal{NP} = \bigcup_k \text{NTIME}(n^k).$$

Un **verificador** para un lenguaje L es un algoritmo V tal que $L = \{w \mid \exists c : V \text{ acepta } \langle w, c\rangle\}$. Un lenguaje L es $\mathcal{NP}$ si y sólo si tiene un verificador que se ejecuta en tiempo $O(|w|^k)$ para algún k.

$\Longrightarrow$] Sea $\mathcal{M}$ una MTND que decide L en tiempo polinómico, definimos un verificador que, ante una entrada $\langle w, c\rangle$, simula $\mathcal{M}$ tratando c como una secuencia con la opción que toma $\mathcal{M}$ en cada paso, y que acepta o rechaza según lo haga $\mathcal{M}$ con estas elecciones.

$\Longleftarrow$] Sea V el verificador en tiempo $O(|w|^k)$, por definición existen $m, c \in \mathbb{N}$ tales que, para toda cadena w, V se ejecuta en tiempo máximo $t(w) \coloneqq \text{máx}\{m, c|w|^k\}$. Definimos una MTND que, ante una entrada w, elige de forma no determinista una cadena c de longitud máxima $t(w)$, ejecuta V sobre $\langle w, c\rangle$ y acepta o rechaza según lo haga V.

$\mathcal{NP}$ es cerrada para la unión, intersección, concatenación y clausura. Se desconoce si es cerrada para el complemento o no. $\mathcal{P}$ y $\mathcal{NP}$ no son cerradas para subconjuntos.

$\mathcal{P} \subseteq \mathcal{NP}$, pero es un problema abierto si $\mathcal{P} = \mathcal{NP}$ o no. La mayoría de investigadores consideran que $\mathcal{P} \neq \mathcal{NP}$ porque se han invertido muchos recursos en encontrar algoritmos en tiempo polinómico para ciertos problemas y no se han encontrado, y aunque se ha intentado probar la existencia de algoritmos en $\mathcal{NP}$ que no están en $\mathcal{P}$, no se han invertido tantos recursos porque el incentivo económico de probar algo que casi se da por hecho es relativamente bajo.

Un problema tiene **tiempo pseudo-polinómico** si su entrada es una cantidad fija de números y existe un algoritmo que lo resuelve en tiempo polinómico respecto a estos números pero no respecto a la longitud de la entrada.

PRIME, el conjunto de números primos, es $\mathcal{NP}$ y pseudo-polinómico, pues un algoritmo para decidir si $n \in$ PRIME es comprobar si es divisible por algún número entre 2 y $n-1$, haciendo $n-2$ divisiones. En 2002, los autores indios Agarwal, Kayal y Saxena encontraron un algoritmo que decide PRIME en tiempo polinómico, el ***AKS primality test***, probando que PRIME $\in \mathcal{P}$. Este algoritmo no es constructivo; no encuentra factores primos de un número compuesto.

Capítulo 8

Problemas $\mathcal{NP}$-completos

Una **reducción de tiempo polinómico** de un lenguaje L a otro L' es una función polinómica $f : \Sigma^* \to \Sigma^*$ tal que, para $w \in \Sigma^*$, $w \in L \iff f(w) \in L'$. Si existe decimos que L **se puede reducir polinómicamente** o **se polireduce** a L', $L \leq_{\mathrm{p}} L'$. Claramente si $L \leq_{\mathrm{p}} L'$ y $L' \in \mathcal{P}$ entonces $L \in \mathcal{P}$, y si $L \leq_{\mathrm{p}} L'$ y $L' \in \mathcal{NP}$ entonces $L \in \mathcal{NP}$.

Un lenguaje L es **$\mathcal{NP}$-duro** si cualquier lenguaje $\mathcal{NP}$ se polireduce a L, y es **$\mathcal{NP}$-completo** si es $\mathcal{NP}$-duro y $\mathcal{NP}$. $\mathcal{P} = \mathcal{NP}$ si y sólo si algún lenguaje $\mathcal{NP}$-completo está en $\mathcal{P}$.

$\Longrightarrow$] Por la existencia de lenguajes $\mathcal{NP}$-completos, que vamos a ver.

$\Longleftarrow$] En tal caso todo lenguaje en $\mathcal{NP}$ se polireduce a él y por tanto también está en $\mathcal{P}$.

Tercer teorema de reducibilidad: Si $L \leq_{\mathrm{p}} L'$ y L es $\mathcal{NP}$-completo entonces L' es $\mathcal{NP}$-completo.

8.1. El problema SAT

Una **proposición** o **fórmula booleana** es una **variable** o **proposición atómica** (dada por una secuencia de letras), una **conjunción** $(a \wedge b)$, una **disyunción** $(a \vee b)$ o una **negación** $(\neg a)$, donde a y b son proposiciones. Podemos reducir la cantidad de paréntesis entendiendo que la negación tiene precedencia sobre la disyunción y la conjunción y que estas son asociativas por la izquierda.

Una **asignación de valores** es una función que a cada variable de una cierta proposición le asigna un **valor de verdad**, «verdadero» o «falso». Dada una asignación de valores, el valor de verdad de $(a \wedge b)$ es verdadero si lo es el de a y el de b y falso en otro caso; el de $(a \vee b)$ es falso si lo es el de a y el de b y verdadero en otro caso, y el de $(\neg a)$ es verdadero si y sólo si el de a es falso, usando siempre la misma asignación.

Una proposición es **satisfacible** si existe una asignación de valores que le asigna verdadero. Definimos

$$\mathrm{SAT} \coloneqq \mathrm{SAT}_0 \coloneqq \mathrm{SAT}_{\mathrm{LP}} \coloneqq \{\langle \Phi \rangle \mid \Phi \text{ es una fórmula booleana satisfacible}\}.$$

Sean $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{f}})$ una MNTD en tiempo polinómico y $m, c, k \in \mathbb{N}^*$ tales que, para una entrada de tamaño n, N se ejecuta en no más de $t(n) \coloneqq \max\{m, cn^k\}$ pasos, un **tablón**

de cálculo para N es una matriz en $\mathcal{M}_{t(n)+1}(Q \sqcup \Gamma)$ en que cada fila es una configuración de N que se sigue de la anterior, siendo la primera una configuración inicial. Las configuraciones se representan como las primeras $t(n)$ posiciones de la cinta con el estado de la máquina justo a la izquierda de la posición donde está el cursor.

Para $i \in \{1, \dots, t(n)\}$ y $j \in \{1, \dots, t(n)+1\}$, la **ventana** (i, j) de un tablón (a_{ij}) es la submatriz 2×3 $(a_{pq})_{i \le p \le i+1}^{j-1 \le q \le j+1}$, donde $a_{p0} \coloneqq a_{p,t(n)+2} \coloneqq \# \notin Q \sqcup \Gamma$. Una **ventana legal** es una matriz en $\mathcal{M}_{2\times 3}(Q \sqcup \Gamma \sqcup \{\#\})$ que puede ser ventana de un tablón de cálculo de N sin restringirlo a que la primera fila sea una configuración inicial.

Teorema de Cook-Levin: SAT es $\mathcal{NP}$-completo. **Demostración:** SAT $\in \mathcal{NP}$, pues su número de variables es menor que la longitud y por tanto decidir una asignación de forma no determinista tiene tiempo polinómico, y el número de evaluaciones a realizar para determinar el valor de verdad es una por cada símbolo u operador y cada una se hace tiempo polinómico. Sea ahora $A \in \mathcal{NP}$, y queremos ver que $A \le_{\mathrm{p}} \mathrm{SAT}$. Sea $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{f}})$ una MTND que decide A en tiempo polinómico, para $w \in \Sigma^*$, si N tiene un tablón con entrada w que contiene q_{f} entonces $w = w_1 \cdots w_n \in A$, y el recíproco se cumple siempre que después de q_{f} se pueda seguir hasta completar las $t(n)$ derivaciones, lo que podemos modelar con una transición adicional $(q_{\mathrm{f}}, a, \mathrm{R}) \in \delta(q_{\mathrm{f}}, a)$ para cada $a \in \Gamma$. Entonces basta encontrar una fórmula booleana computable en tiempo polinómico respecto a n cuya satisfacibilidad equivalga a la existencia de un tablón de N con entrada w que contenga a q_{f}. Si $C \coloneqq Q \sqcup \Gamma$, la fórmula tendrá $t(n)^2|C|$ variables $\{x_{ijs}\}_{1 \le i,j \le t(n)}^{s \in C}$, donde x_{ijs} indica que el tablón contiene s en la posición (i, j). Queremos que la primera fila contenga la entrada,

$$\Phi_{\mathrm{start}} \coloneqq x_{11q_0} \wedge x_{12w_1} \wedge x_{13w_2} \wedge \cdots \wedge x_{1,n+1,w_n} \wedge x_{1,n+2,\mathrm{B}} \wedge \cdots \wedge x_{1,t(n)+1,\mathrm{B}}.$$

Tiene que aparecer q_{f},

$$\Phi_{\mathrm{accept}} \coloneqq \bigvee_{1 \le i,j \le t(n)+1} x_{ijq_{\mathrm{f}}}.$$

Cada celda debe contener uno y solo un valor,

$$\Phi_{\mathrm{cell}} = \bigwedge_{1 \le i,j \le t(n)+1} \left(\bigvee_{s \in C} x_{ijs} \wedge \bigwedge_{\substack{s,t \in C \\ s \ne t}} (\neg x_{ijs} \vee \neg x_{ijt}) \right).$$

Finalmente queremos ver que, si una fila es una configuración válida, la siguiente también lo es y se sigue de ella. Si esto es así, todas las ventanas entre las dos filas serán legales. Ahora bien, las ventanas legales son las siguientes, donde $a, b, c, d \in \Gamma$, $e \in \Gamma \sqcup \{\#\}$ y $\mathbf{q}, \mathbf{r} \in Q$:

$$\begin{bmatrix} a & b & c \\ a & b & c \end{bmatrix}, \quad \begin{bmatrix} \# & b & c \\ \# & b & c \end{bmatrix}, \quad \begin{bmatrix} a & b & \# \\ a & b & \# \end{bmatrix};$$

$$\begin{bmatrix} a & c & e \\ \mathbf{r} & c & e \end{bmatrix}, \quad \begin{bmatrix} \mathbf{q} & a & e \\ b & \mathbf{r} & e \end{bmatrix}, \quad \begin{bmatrix} e & \mathbf{q} & a \\ e & b & \mathbf{r} \end{bmatrix}, \quad \begin{bmatrix} e & c & \mathbf{q} \\ e & c & b \end{bmatrix}, \qquad (\mathbf{r}, b, \mathrm{R}) \in \delta(\mathbf{q}, a);$$

$$\begin{bmatrix} \mathbf{q} & a & e \\ c & b & e \end{bmatrix}, \quad \begin{bmatrix} c & \mathbf{q} & a \\ \mathbf{r} & c & b \end{bmatrix}, \quad \begin{bmatrix} e & c & \mathbf{q} \\ e & \mathbf{r} & c \end{bmatrix}, \quad \begin{bmatrix} e & c & d \\ e & c & \mathbf{r} \end{bmatrix}, \qquad (\mathbf{r}, b, \mathrm{L}) \in \delta(\mathbf{q}, a).$$

Con esto es claro que, si todas las ventanas entre dos filas son legales, los «#» a los lados se conservan y, si una fila tiene una única celda de estado, la siguiente tendrá una única que

estará a la izquierda o a la derecha, la transición estará en δ y los símbolos que no participan de la transición quedarán igual. Como hay un número finito de ventanas legales y estas solo dependen de N, la fórmula que buscamos es

$$\Phi_{\text{move}} \coloneqq \bigwedge_{\substack{1\le i\le t(n)\\ 1\le j\le t(n)+1}} \left(\bigvee_{\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}\text{ventana legal}} \begin{pmatrix} x_{i,j-1,a} \wedge x_{ijb} \wedge x_{i,j+1,c} \wedge \\ \wedge x_{i+1,j-1,d} \wedge x_{i+1,j,e} \wedge x_{i+1,j+1,f} \end{pmatrix} \right),$$

eliminando de la conjunción interior las variables $x_{i0\#}$ o $x_{i,t(n)+2,\#}$ (que no están definidas y siempre son ciertas) y de la disyunción las ventanas para las que hay x_{i0s} o $x_{i,t(n)+2,s}$ con $s \neq \#$ o $x_{ij\#}$ con $j \neq 0, t(n)+2$ (que no están definidas y siempre son falsas). Así, finalmente la fórmula es

$$\Phi \coloneqq \Phi_{\text{start}} \wedge \Phi_{\text{accept}} \wedge \Phi_{\text{cell}} \wedge \Phi_{\text{move}}.$$

Si $t(n) = O(n^k)$, Φ_{start} tiene $O(n^k)$ apariciones de variables (una por columna), Φ_{accept} tiene $O(n^{2k})$ (una por celda), Φ_{cell} tiene $O(n^{2k})$ (C^2 por celda) y Φ_{move} tiene $O(n^{2k})$ (hasta 6 por ventana legal y ventana, el número de ventanas legales es fijo y hay menos ventanas que celdas), por lo que Φ tiene tamaño $O(n^{2k})$ y, para N fijo, generarla a partir de w es sistemático, con lo que Φ se genera en tiempo polinómico. La demostración vista en clase[1] asume que la cota de tiempo siempre es de la forma $n^k - 3$, pese a que para valores pequeños de n esto es imposible; añade una columna llena de # a cada lado del tablón y dos filas al final inútiles para que este tenga tamaño $n^k \times n^k$, añadiendo dos proposiciones atómicas a Φ_{start} para establecerlas a # y que se propaguen al resto y extendiendo el resto de proposiciones a estas nuevas columnas y filas y al hecho de que ahora $C = Q \sqcup \Gamma \sqcup \{\#\}$, y llama proposiciones sólo a las proposiciones atómicas.

Este teorema lo descubrieron Stephen Cook y Leonid Levin en los 70, siendo la primera vez que se descubre que un lenguaje es $\mathcal{NP}$-completo.

Un **literal** es una variable o su negación. Una **cláusula** es una disyunción de uno o más literales. Una fórmula booleana está en **forma normal conjuntiva** si es una conjunción de una o más cláusulas, y está en **forma normal 3-conjuntiva** si además estas cláusulas tienen un máximo de 3 literales.

Llamando SAT_{CNF} al conjunto de fórmulas satisfacibles en forma normal conjuntiva y $\text{SAT}_{\text{3-CNF}}$ al de fórmulas satisfacibles en forma normal 3-conjuntiva, $\text{SAT}_{\text{3-CNF}} \subsetneq \text{SAT}_{\text{CNF}} \subsetneq \text{SAT}$, y $\text{SAT}_{\text{3-CNF}}$ y SAT_{CNF} son $\mathcal{NP}$-completos. Sin embargo, $\text{SAT}_{\text{2-CNF}}$, el conjunto de fórmulas satisfacibles en forma normal conjuntiva con a lo sumo 2 literales en cada cláusula, está en $\mathcal{P}$.

SAT_1, el conjunto de fórmulas satisfacibles en lógica de primer orden, es indecidible, y esto es más o menos lo que probaron Alonzo Church y Alan Turing en sus *papers* sobre el *Entscheidungsproblem* en 1936.

[1] La cual hay que saberse pese a que es incorrecta porque hacen preguntas sobre ella.

8.2. Otros lenguajes $\mathcal{NP}$-completos

Son $\mathcal{NP}$-completos:

1. CLIQUE $\coloneqq \{\langle G, k\rangle \mid G$ es grafo no dirigido con k-clique$\}$.

   Primero vemos que está en $\mathcal{NP}$. El verificador recibe la lista de k nodos de la clique, de tamaño claramente proporcional al grafo. Si tiene algún nodo que no está en el grafo, o más o menos de k nodos, rechaza. Para cada par de nodos en la clique, si el par no es un arco del grafo, rechaza. En otro caso acepta.

   Veamos que $\text{SAT}_{\text{3-cnf}} \leq_{\text{p}}$ CLIQUE. Dada una fórmula Φ, si algún literal de Φ tiene menos de 3 literales, lo completamos repitiendo uno de ellos. Sea la fórmula resultante $(l_{11} \vee l_{12} \vee l_{13}) \wedge \cdots \wedge (l_{k1} \vee l_{k2} \vee l_{k3})$, definimos el grafo $G_\Phi = (V, E)$ como $V \coloneqq \{(i,j)\}_{1 \leq i \leq k}^{1 \leq j \leq 3}$ y $E \coloneqq \{\{(a,b),(c,d)\}\}_{1 \leq a < c \leq k, 1 \leq b,d \leq 3, l_{ab} \neq \neg l_{cd}, \neg l_{ab} \neq l_{cd}}$, y Φ es satisfacible si y sólo si G_Φ tiene una k-clique.

   $\Longrightarrow$] Dada una asignación de valores que hace a Φ verdadera, para $i \in \{1, \ldots, k\}$, existe al menos un n_i tal que l_{in_i} es verdadero, y como obviamente ningún l_{in_i} es negación de otro, $\{(i, n_i)\}_{i=1}^{k}$ forma una k-clique.

   $\Longleftarrow$] Como ningún (i,j) conecta con un (i,k), la k-clique está formada por elementos correspondientes a literales de cláusulas distintas, sin que uno sea la negación de otro, por lo que es posible construir una asignación de verdad que haga verdaderos a los literales de la k-clique y por tanto a Φ.

   La función de conversión de $\langle \Phi \rangle$ a $\langle G_\Phi, k \rangle$ es claramente polinómica.

2. HAMPATH $\coloneqq \{\langle G, s, t\rangle \mid G$ es un grafo dirigido con camino hamiltoniano de s a $t\}$. Un **camino hamiltoniano** es uno que pasa por todos los vértices exactamente una vez.

   Primero vemos que es $\mathcal{NP}$. El verificador recibe el grafo y el camino hamiltoniano, cuyo tamaño es claramente polinómico respecto al del grafo. Marca los vértices de dicho camino, rechazando si encuentra un vértice que ya ha sido marcado. Si queda algún vértice sin marcar, rechaza. Para cada arco del camino, si el arco no está en el grafo, rechaza. Finalmente, acepta si el camino empieza por s y termina por t y rechaza en otro caso.

   Veamos que $\text{SAT}_{\text{3-cnf}} \leq_{\text{p}}$ HAMPATH. Sean Φ una fórmula 3-CNF con proposiciones $\{p_1, \ldots, p_m\}$ distintas y cláusulas $k_1 \wedge \cdots \wedge k_n$, que podemos considerar de 3 elementos cada una. Creamos un grafo dirigido $G_\Phi \coloneqq (V, E)$, donde

$$
\begin{aligned}
V \coloneqq & \{a_i\}_{i=0}^{m} \cup \{b_{ij}\}_{i \in \{1,\ldots,m\}}^{j \in \{1,\ldots,3n+3\}} \cup \{c_j\}_{j=1}^{n}, \\
E \coloneqq & \{(a_{i-1}, b_{i1}), (a_{i-1}, b_{i,3n+3}), (b_{i1}, a_i), (b_{i,3n+3}, a_i)\}_{i=1}^{m} \cup \\
& \cup \{(b_{ij}, b_{i,j+1}), (b_{i,j+1}, b_{ij})\}_{i \in \{1,\ldots,m\}}^{j \in \{1,\ldots,3n+2\}} \cup \\
& \cup \{(b_{i,3j}, c_j), (c_j, b_{i,3j+1})\}_{i \in \{1,\ldots m\}, j \in \{1,\ldots,n\}}^{p_i \text{ aparece en } k_j} \cup \\
& \cup \{(b_{i,3j+1}, c_j), (c_j, b_{i,3j})\}_{i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\}}^{\neg p_i \text{ aparece en } k_j}.
\end{aligned}
$$

   Φ es satisfacible si y sólo si G_Φ tiene un camino hamiltoniano de a_0 a a_m.

$\Longrightarrow$] Tomamos una asignación de valores que haga verdadera a Φ y, para cada k_j, elegimos un literal en k_j que valga verdadero. Para construir el camino, primero, para i de 1 a m, si p_i es verdadero, recorremos $a_{i-1}, b_{i1}, b_{i2}, \ldots, b_{i,3n+3}, a_i$, y si es falso, recorremos $a_{i-1}, b_{i,3n+3}, b_{i,3n+2}, \ldots, b_{i1}, a_i$. Entonces, para cada k_j, si el literal elegido de k_j es de la forma p_i, cambiamos el eje $(b_{i,3j}, b_{i,3j+1})$ del camino por $(b_{i,3j}, c_j), (c_j, b_{i,3j+1})$, y si es de la forma $\neg p_i$, cambiamos $(b_{i,3j+1}, b_{i,3j})$ por $(b_{i,3j+1}, c_j), (c_j, b_{i,3j})$.

$\Longleftarrow$] Para $i \in \{1, \ldots, m\}$, el camino debe ir de a_{i-1} bien a b_{i1} o a $b_{i,3n+3}$. En el primer caso le asignamos verdadero a p_i, y en el segundo le asignamos falso. Para $j \in \{1, \ldots, n\}$, si el nodo anterior a c_j en el camino es de la forma $b_{i,3j}$, queremos ver que el siguiente es $b_{i,3j+1}$. No puede ser $b_{i,3j}$ porque ya ha aparecido, por lo que bien es $b_{i,3j+1}$ o un nodo $b_{p,3j}$ o $b_{p,3j+1}$ con $p \neq i$. Si fuera lo segundo, el nodo anterior a $b_{i,3j+1}$ no puede ser c_j porque este ya es anterior a $b_{p,3j}$ o $b_{p,3j+1}$, por lo que debe ser $b_{i,3j+2}$, pero entonces de los sucesores a $b_{i,3j+1}$ ($b_{i,3j}$, $b_{i,3j+2}$ y posiblemente c_j) ninguno puede ser el siguiente a este.# Por tanto el siguiente a c_j en el camino es $b_{i,3j+1}$. La otra alternativa es que el nodo anterior a c_j sea de la forma $b_{i,3j+1}$, con lo que el siguiente es $b_{i,3j}$ por un argumento análogo. Entonces, para un mismo i, los b_{ij} se recorren con j siempre ascendente o siempre descendente. En el primer caso el anterior a b_{i1} es a_{i-1} y p_i es verdadero, y como k_j contiene a p_i ya que $(b_{i,3j}, c_j) \in E$, la asignación hace a k_j verdadera. En el segundo, el anterior a $b_{i,3n+3}$ es a_{i-1} y p_i es falso, y como k_j contiene a $\neg p_i$ ya que $(b_{i,3j+1}, c_j) \in E$, la asignación hace a k_j verdadera.

La función de conversión de $\langle \Phi \rangle$ a $\langle G_\Phi, a_0, a_m \rangle$ es claramente polinómica.

3. HAMCYCLE $\coloneqq \{\langle G \rangle \mid G$ es un grafo dirigido con un ciclo hamiltoniano$\}$.

   Claramente es $\mathcal{NP}$, pues basta tomar un nodo s de G arbitrario y decidir si $\langle G, s, s \rangle \in$ HAMPATH. Veamos ahora que HAMPATH $\leq_\mathrm{p}$ HAMCYCLE. Sean $G = (V, E)$ un grafo dirigido, $a, b \in V$ y $G' \coloneqq (V', E')$ un grafo dirigido dado por $V' \coloneqq V \sqcup \{a', b'\}$ y $E' \coloneqq E \sqcup \{(b, b'), (b', a'), (a', a)\}$. Entonces G tiene un camino hamiltoniano de a a b si y sólo si G' tiene un ciclo hamiltoniano.

   $\Longrightarrow$] Concatenamos al camino hamiltoniano de a a b el camino $bb'a'a$.

   $\Longleftarrow$] El ciclo hamiltoniano debe contener el subcamino $bb'a'a$ por ser esta la única forma que pase por b' y a'. Entonces, tomando el ciclo empezando por a, este debe acabar por dicho subcamino, y eliminando esta parte nos queda un camino de a a b en G que pasa por todos los nodos en V.

   La función de conversión de $\langle G, a, b \rangle$ a $\langle G' \rangle$ es claramente polinómica.

4. UHAMCYCLE $\coloneqq \{\langle G \rangle \mid G$ es un grafo no dirigido con un ciclo hamiltoniano$\}$.

   Claramente es $\mathcal{NP}$, pues equivale a decidir HAMCYCLE sobre un grafo dirigido cuyos nodos son los de G y cuyos arcos son (a, b) y (b, a) para cada eje $\{a, b\}$ de G. Para ver que es $\mathcal{NP}$-completo, vemos que HAMCYCLE $\leq_\mathrm{p}$ UHAMCYCLE. Dado un grafo dirigido $G = (V, E)$, definimos un grafo no dirigido $G' \coloneqq (V', E')$ dado por

$$V' \coloneqq \{(v, 0), (v, 1), (v, 2)\}_{v \in V},$$
$$E' \coloneqq \{\{(v, 0), (v, 1)\}, \{(v, 1), (v, 2)\}\}_{v \in V} \cup \{\{(u, 2), (v, 0)\}\}_{(u,v) \in E},$$

y G tiene un ciclo hamiltoniano si y sólo si lo tiene G'.

$\Longrightarrow$] Sea $u_1 \cdots u_n u_1$ un ciclo hamiltoniano en G,

$$(u_1, 0)(u_1, 1)(u_1, 2) \cdots (u_n, 0)(u_n, 1)(u_n, 2)(u_1, 0)$$

es uno en G'.

$\Longleftarrow$] Si V' es vacío, V también y hemos terminado. En otro caso, dado un ciclo hamiltoniano de G' y un nodo de este de la forma $(u_1, 1)$, podemos suponer que el anterior es $(u_1, 0)$ y el siguiente es $(u_1, 2)$, pues en otro caso ocurre al revés y damos la vuelta al ciclo. Vemos por inducción que, para $k \in \{1, \ldots, n\}$, el ciclo contiene un subcamino de la forma $(u_1, 0)(u_1, 1)(u_1, 2) \cdots (u_k, 0)(u_k, 1)(u_k, 2)$, pues entonces en particular esto ocurre para $k = n$ y $u_1 \cdots u_n u_1$ es un ciclo hamiltoniano en G. Para $k = 1$ ya lo hemos visto. Para $k > 1$, probado esto para $k - 1$, por construcción el elemento siguiente a $(u_{k-1}, 2)$ es de la forma $(u_k, 0)$. El elemento anterior a $(u_k, 1)$ debe ser $(u_k, 0)$ ya que de lo contrario sería $(u_k, 2)$ y el siguiente sería $(u_k, 0)$, pero $(u_k, 0)$ ya es el siguiente de $(u_{k-1}, 2)\#$. Por tanto a $(u_k, 0)$ le siguen $(u_k, 1)$ y luego $(u_k, 2)$.

Claramente la función de conversión de $\langle G \rangle$ a $\langle G' \rangle$ es polinómica.

5. COLOR $\coloneqq \{\langle G, k\rangle \mid G$ es un grafo no dirigido k-coloreable$\}$. Un grafo no dirigido $G = (V, E)$ es **k-coloreable** si existe una función $f : V \to \{1, \ldots, k\}$ tal que $f(u) \neq f(v)$ para $\{u, v\} \in E$, en cuyo caso f es un **coloreado** de G. En particular un grafo es bipartito si es 2-coloreable.

6. El **TSP** (***Traveling Salesman Problem***) consiste en encontrar el ciclo hamiltoniano de peso mínimo en un grafo no dirigido completo con pesos. La versión de decisión es TSP-DEC, formado por los pares $\langle G, k\rangle$ donde G es un grafo no dirigido completo con pesos en $\mathbb{N}$ con un ciclo hamiltoniano de coste no mayor a $k \in \mathbb{N}$, y es $\mathcal{NP}$-completo.

   TSP-DEC $\in \mathcal{NP}$, pues basta tomar un nodo cualquiera del grafo y marcarlo, tomar de forma no determinista otro nodo que no esté marcado y marcarlo, y así hasta marcar todos los nodos, formando así un ciclo, sumar el coste de todos los ejes del ciclo, rechazar si la suma es mayor a k y aceptar en otro caso. Para ver que es $\mathcal{NP}$-completo vemos que UHAMCYCLE $\leq_p$ TSP-DEC. Dado un grafo no dirigido $G = (V, E)$, definimos un grafo no dirigido con pesos $G' = (V, E', \omega)$ donde E' es tal que (V, E') es completo y

$$\omega(e) \coloneqq \begin{cases} 1, & e \in E; \\ 2, & e \notin E. \end{cases}$$

   Claramente la función de conversión de $\langle G \rangle$ a $\langle G', |V| \rangle$ es polinómica, y queda ver que G tiene un ciclo hamiltoniano si y sólo si G' tiene uno con peso no mayor que $|V|$.

   $\Longrightarrow$] Tomamos el mismo ciclo, y como sus $|V|$ aristas están en G, su peso es $|V|$.

   $\Longleftarrow$] Un ciclo hamiltoniano en G' tiene $|V|$ aristas y su peso es pues $|V|$ más el número de aristas en el ciclo fuera de E, por lo que si el ciclo tiene peso no mayor que $|V|$, no puede tener aristas fuera de E y por tanto está en G.

7. SUBSET-SUM $\coloneqq \{\langle S, t\rangle \mid S$ es una lista de naturales con una subsecuencia que suma $t\}$.

   Está en $\mathcal{NP}$, y un verificador toma una asignación de qué elementos de la lista se toman y cuáles no, suma los que se toman y compara. Para ver que es $\mathcal{NP}$-completo vemos que $\text{SAT}_{\text{3-CNF}} \leq_{\text{p}}$ SUBSET-SUM. Dada una fórmula Φ con variables $p_1, \ldots, p_m$ distintas y cláusulas $k_1 \wedge \cdots \wedge k_n$, que podemos considerar con 3 literales cada una, tomamos la lista $S \coloneqq (y_0, z_0, \ldots, y_{m-1}, z_{m-1}, g_0, h_0, \ldots, g_{n-1}, h_{n-1})$, donde y_i es la suma de 10^{n+i} más 10^j por cada k_j con el literal p_i, z_i es la suma de 10^{n+i} más 10^j por cada k_j con el literal $\neg p_i$ y $g_j \coloneqq h_j \coloneqq 10^j$, y $t \coloneqq \sum_{i=0}^{m-1} 10^{n+i} + 3\sum_{j=0}^{n-1} 10^j$. Nótese que para cada posición decimal, una suma de elementos de S no incluye más de 5 sumandos con ese dígito no nulo, por lo que nunca hay llevadas. Entonces Φ es satisfacible si y sólo si alguna sublista de S suma t.

   $\Longrightarrow$] Dada una asignación de valores de los p_i, tomamos los y_i con p_i verdadero y los z_i con p_i falso. Entonces las posiciones decimales de n a $n+m-1$ valen 1, y las posiciones $j \in \{0, \ldots, n-1\}$ valen de 1 a 3 ya que la cláusula k_j contiene entre 1 y 3 literales verdaderos que tendrán el dígito correspondiente en y_i o z_i a 1. Añadiendo elementos g_j o h_j se llega a t.

   $\Longleftarrow$] Para cada i, exactamente uno de entre y_i o z_i se toma, lo que nos da una asignación de valores con p_i a verdadero si se toma y_i o a falso si se toma z_i, y queda ver que esta hace a Φ verdadera. Para cada cláusula k_j, el dígito j es 3 y se han tomado entre 0 y 2 de g_j y h_j, por lo que al menos uno de los y_i y z_i que se han tomado tiene un 1 en la posición j. Si lo tiene un y_i, k_j contiene el literal p_i al que se ha asignado verdadero, y si lo tiene un z_i, k_j contiene el literal $\neg p_i$ habiendo asignado falso a p_i.

   La función de conversión de $\langle\Phi\rangle$ a $\langle S, t\rangle$ es polinómica. En efecto, la longitud de t es como mucho proporcional a la de Φ, como lo es la de cada elemento de S y la longitud de S, y las operaciones usadas se calculan en tiempo polinómico salvo la exponenciación, pero calcular las potencias de 10 corresponde a multiplicar por 10 $n+m-1$ veces, pudiendo hacer cada multiplicación en tiempo proporcional a la longitud de los factores, la cual es como mucho proporcional a $n+m-1$.

8. VERTEX-COVER $\coloneqq \{\langle G, k\rangle \mid G$ es un grafo no dirigido con una k-cobertura$\}$. Una **k-cobertura** de un grafo no dirigido es un conjunto de k nodos tales que toda arista contiene uno de esos nodos.

   Para ver que es $\mathcal{NP}$: si G tiene menos de k nodos, rechaza; se eligen k nodos de forma no determinista y, si alguna arista no contiene un nodo de los seleccionados, rechaza; en otro caso acepta. Ahora vemos que $\text{SAT}_{\text{3-CNF}} \leq_{\text{p}}$ VERTEX-COVER. Dada una fórmula 3-CNF Φ con variables $p_1, \ldots, p_m$ distintas y cláusulas $(l_{11} \vee l_{12} \vee l_{13}) \wedge \cdots \wedge (l_{n1} \vee l_{n2} \vee l_{n3})$, definimos un grafo no dirigido $G_\Phi \coloneqq (V, E)$ con

$$\begin{aligned} V \coloneqq & \{y_i, z_i\}_{i=1}^m \cup \{a_{j1}, a_{j2}, a_{j3}\}_{j=1}^n, \\ E \coloneqq & \{\{y_i, z_i\}\}_{i=1}^m \cup \{\{a_{j1}, a_{j2}\}, \{a_{j2}, a_{j3}\}, \{a_{j3}, a_{j1}\}\}_{j=1}^n \cup \\ & \cup \{\{a_{jk}, y_i\}\}_{l_{jk}=p_i} \cup \{\{a_{jk}, z_i\}\}_{l_{jk}=\neg p_i}. \end{aligned}$$

   Φ es satisfacible si y sólo si G tiene una $(m+2n)$-cobertura.

$\Longrightarrow$] Dada una asignación de valores que hace a Φ verdadera, la cobertura está formada por los y_i con p_i verdadero, los z_i con p_i falso y, para $j \in \{1, \ldots, n\}$, los a_{jk} salvo uno con el correspondiente l_{jk} verdadero. Las aristas $\{y_i, z_i\}$ y las $\{a_{jp}, a_{jq}\}$ quedan claramente cubiertas, y las de la forma $\{a_{jk}, y_i\}$ o $\{a_{jk}, z_i\}$ quedan cubiertas por a_{jk} si este está en la cobertura o por el otro nodo si no lo está, pues en tal caso l_{jk} es verdadero y, si $l_{jk} = p_i$, el otro nodo es y_i que está seleccionado, y si $l_{jk} = \neg p_i$ el otro nodo es z_i que está seleccionado.

$\Longleftarrow$] Para $i \in \{1, \ldots, m\}$ la cobertura debe tener a y_i o a z_i para cubrir $\{y_i, z_i\}$, y para $j \in \{1, \ldots, n\}$ debe tener dos de los a_{jk} para cubrir todos los $\{a_{jp}, a_{jq}\}$. Ya llevamos $m + 2n$ nodos, por lo que no se cubren más. Tomamos la asignación de valores con p_i a verdadero si la cobertura contiene a y_i o a falso si contiene a z_i, y entonces Φ es verdadera porque, para cada $j \in \{1, \ldots, n\}$, hay un a_{jk} fuera de la cobertura, de modo que si $\{a_{jk}, y_i\} \in E$ este está cubierto por y_i y $p_i = l_{jk}$ es verdadero, y si $\{a_{jk}, z_i\} \in E$ este está cubierto por z_i y $\neg p_i = l_{jk}$ es verdadero, y en cualquier caso la j-ésima cláusula contiene un literal verdadero.

La función de conversión de $\langle \Phi \rangle$ a $\langle G_\Phi, m + 2n \rangle$ es claramente polinómica.