Teoria de la computació

Computabilitat i indecidibilitat

Maria Serna
Carme Àlvarez
Rafel Cases
Antoni Lozano

Índex

1 Màquines de Turing

Per tal d'identificar el conjunt de problemes que es poden resoldre mitjançant un procediment mecànic, ens cal formalitzar la noció intuïtiva d'algorisme. El pas fonamental per establir-ne un model vàlid es va donar al final de la dècada de 1930, quan es va demostrar que els diferents intents proposats per caracteritzar aquest concepte conduïen a models equivalents entre si. Presentarem en aquest capítol un model teòric de computació introduït per A. Turing en 1936 [40], la màquina de Turing, i veurem com utilitzar aquest tipus de màquina per reconèixer llenguatges i computar funcions.

Una característica del model de màquina de Turing, molt important, és la possibilitat de definir sense ambigüitats el concepte de pas de càlcul i, en conseqüència, el de temps d'execució per a un algorisme. Aquest concepte, en general, és difícil de definir sobre llenguatges de programació concrets, ja que certes instruccions triguen més o menys temps i normalment no es disposa d'un criteri uniforme de mesura del temps.

Finalment, tot model necessita ser robust, en el sentit que petites modificacions, afegides per facilitar el disseny de màquines, han de conservar la capacitat de càlcul. En aquesta direcció, finalitzem el capítol introduint algunes variacions sobre el model bàsic i esbossem els arguments que porten a la demostració de l'equivalència amb aquest model bàsic.

1.1 Definició de la màquina de Turing

Una màquina de Turing, abreujadament TM, és un mecanisme que funciona com un autòmat finit amb una cinta semiinfinita addicional (vegeu la figura 1.1). La cinta està dividida en cel·les i la màquina accedeix a la informació, emmagatzemada a la cinta, mitjançant un capçal lector/escriptor, que pot llegir o escriure, en un instant donat, sobre una única cel·la. El capçal es pot moure a dreta i a esquerra, però només una cel·la per moviment.

Una transició d'una màquina de Turing comença amb la unitat de control en un estat, i el capçal posicionat per accedir a una cel·la de la cinta. A cada pas, es realitza:

- un canvi d'estat,
- una escriptura a la cel·la accessible a través del capçal,
- un moviment del capçal.

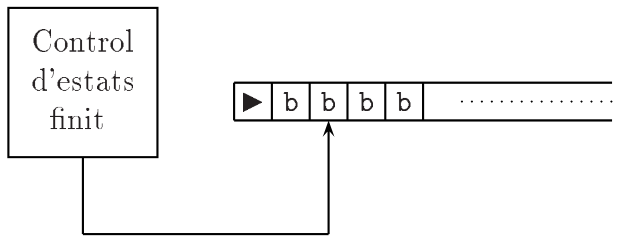


Fig. 1.1: *Esquema d'una màquina de Turing.*

Inicialment, la posició més a l'esquerra de la cinta conté un caràcter especial ▶, anomenat símbol d'*inici de cinta*. A la dreta d'aquesta posició es troba el mot d'entrada. La resta de cel·les de la cinta contenen un caràcter especial anomenat *blanc*, que representem pel símbol b. Utilitzem el terme *símbol actual* per referir-nos al símbol accessible a través del capçal en un instant donat.

Admetem tres tipus de moviments del capçal: una cel·la a la dreta, una cel·la a l'esquerra i no-moviment. Si el capçal està en la posició de més a l'esquerra de la cinta i es demana un moviment cap a l'esquerra, la màquina s'atura.

L'evolució d'una TM amb una entrada donada queda determinada per una *funció de transició*. Aquesta funció especifica les regles de canvi de les configuracions de la màquina. El seu argument és un parell, de la forma (estat, caràcter), que correspon a l'estat actual de l'autòmat i al caràcter actual. La imatge de la funció de transició és un triple (estat, caràcter, moviment), que correspon al nou estat, al nou símbol que s'escriurà a la cinta i al moviment que farà el capçal.

Inicialment, la màquina es troba en un *estat inicial*, habitualment representat per q_0, amb un mot $w \in \Sigma^*$ escrit a la part esquerra de la cinta, precedit pel símbol ▶. La posició del capçal és la segona cel·la, que conté el primer símbol de w, si $w \neq \lambda$. La màquina efectua transicions mentre tingui una transició definida a partir de l'estat i del símbol actuals. Si s'arriba a un punt en què la màquina es troba en un estat q amb un caràcter actual a, i no hi ha cap transició definida per a aquest parell (q, a), la màquina s'atura.

Per raons de simplicitat, sobretot en la formalització de certes propietats, es requereix que les màquines del nostre model tinguin un únic estat inicial i un únic *estat final* o *estat acceptador*. A més, demanem que *des de l'estat final no hi hagi transicions* definides. Això garanteix que quan la TM arriba a l'estat acceptador, s'atura. Així doncs, la funció de transició no pot ser mai una funció total.

A continuació definirem formalment els conceptes de màquina de Turing i de *pas* de càlcul com una relació entre configuracions d'una TM.

DEFINICIÓ Una *màquina de Turing* és una estructura de la forma $(Q, \Sigma, \Gamma, \delta, q_0, q_F)$, on

Q és un conjunt finit d'estats.

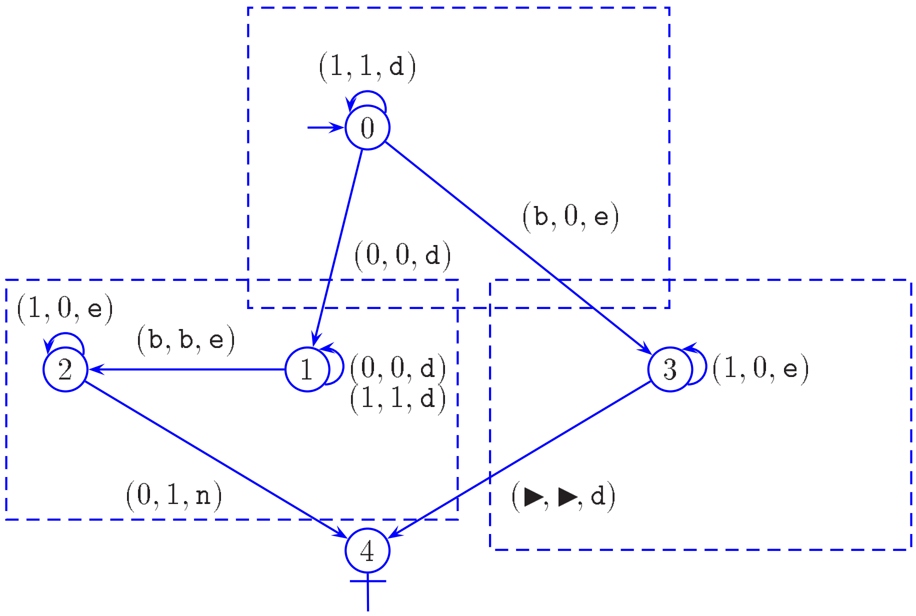


Fig. 1.2: TM *que computa la funció* ***següent***

Σ és un alfabet, l'alfabet d'entrada.

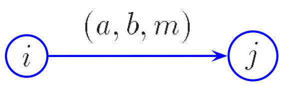
Γ és un alfabet, l'alfabet de cinta, tal que $\Sigma \cup \{\mathtt{b}, \blacktriangleright\} \subseteq \Gamma$, on $\mathtt{b}, \blacktriangleright \notin \Sigma$.

δ és la funció de transició, $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathtt{e}, \mathtt{d}, \mathtt{n}\}$

q_0 és l'estat inicial ($q_0 \in Q$).

q_{F} és l'estat final o acceptador ($q_{\mathrm{F}} \in Q$).

Representem gràficament una màquina de Turing mitjançant un graf etiquetat dirigit. Tenim un vèrtex per a cada estat, etiquetat amb un número; l'etiqueta i representa l'estat q_i. Un arc entre dos vèrtexs i i j, etiquetat amb (a, b, m), representa que la funció de transició té com a imatge del parell (q_i, a) el triple (q_j, b, m).



Com és usual en teoria d'autòmats, utilitzem una fletxa per marcar l'estat inicial i una creu per marcar l'estat final.

estat	cinta	estat	cinta	estat	cinta
0	$\blacktriangleright\boxed{0}10010101\mathtt{bb}\cdots$	0	$\blacktriangleright\boxed{\mathtt{b}}\mathtt{b}\cdots$	0	$\blacktriangleright\boxed{1}1111111\mathtt{bb}\cdots$
1	$\blacktriangleright0\boxed{1}0010101\mathtt{bb}\cdots$	3	$\boxed{\blacktriangleright}0\mathtt{b}\cdots$	0	$\blacktriangleright1\boxed{1}111111\mathtt{bb}\cdots$
1	$\blacktriangleright01\boxed{0}010101\mathtt{bb}\cdots$	4	$\blacktriangleright\boxed{0}\mathtt{b}\cdots$	0	$\blacktriangleright11\boxed{1}11111\mathtt{bb}\cdots$
1	$\blacktriangleright010\boxed{0}10101\mathtt{bb}\cdots$			0	$\blacktriangleright111\boxed{1}1111\mathtt{bb}\cdots$
1	$\blacktriangleright0100\boxed{1}0101\mathtt{bb}\cdots$			0	$\blacktriangleright1111\boxed{1}111\mathtt{bb}\cdots$
1	$\blacktriangleright01001\boxed{0}101\mathtt{bb}\cdots$			0	$\blacktriangleright11111\boxed{1}11\mathtt{bb}\cdots$
1	$\blacktriangleright010010\boxed{1}01\mathtt{bb}\cdots$			0	$\blacktriangleright111111\boxed{1}1\mathtt{bb}\cdots$
1	$\blacktriangleright0100101\boxed{0}1\mathtt{bb}\cdots$			0	$\blacktriangleright1111111\boxed{1}\mathtt{bb}\cdots$
1	$\blacktriangleright01001010\boxed{1}\mathtt{bb}\cdots$			0	$\blacktriangleright11111111\boxed{\mathtt{b}}\mathtt{b}\cdots$
1	$\blacktriangleright010010101\boxed{\mathtt{b}}\mathtt{b}\cdots$			3	$\blacktriangleright1111111\boxed{1}0\mathtt{b}\cdots$
2	$\blacktriangleright01001010\boxed{1}\mathtt{bb}\cdots$			3	$\blacktriangleright111111\boxed{1}00\mathtt{b}\cdots$
2	$\blacktriangleright0100101\boxed{0}0\mathtt{bb}\cdots$			3	$\blacktriangleright11111\boxed{1}000\mathtt{b}\cdots$
4	$\blacktriangleright0100101\boxed{1}0\mathtt{bb}\cdots$			3	$\blacktriangleright1111\boxed{1}0000\mathtt{b}\cdots$
				3	$\blacktriangleright111\boxed{1}00000\mathtt{b}\cdots$
				3	$\blacktriangleright11\boxed{1}000000\mathtt{b}\cdots$
				3	$\blacktriangleright1\boxed{1}0000000\mathtt{b}\cdots$
				3	$\blacktriangleright\boxed{1}00000000\mathtt{b}\cdots$
				3	$\boxed{\blacktriangleright}000000000\mathtt{b}\cdots$
				4	$\blacktriangleright\boxed{0}00000000\mathtt{b}\cdots$

Taula 1.1: *El comportament de la* TM *de la figura 1.2*

Exemple 1.1 A la taula 1.1 es mostra el funcionament de la TM de la figura 1.2 amb tres entrades diferents. S'exposa l'evolució del contingut de la cinta a cada transició. El símbol requadrat indica la posició del capçal.

El comportament global d'aquesta TM és el següent:

1. Cerca un zero anant cap a la dreta.
2. Si no en troba cap:
   - 2.1. Escriu 0 a sobre del primer blanc.
   - 2.2. Torna al començament de la cinta i canvia tots els 1's que troba per 0's.
3. Si troba un zero:
   - 3.1. Avança el capçal cap a la dreta fins a trobar un blanc.
   - 3.2. Torna cap al començament de la cinta i canvia tots els 1's que troba per 0's, fins que troba un zero.
   - 3.3. Canvia el 0 per un 1.

Si l'entrada és un mot w, el que queda a la cinta quan la TM s'atura és el mot següent a w en l'ordre usual (lexicogràfic per longituds, vegeu l'apèndix **??**).

Donada una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, suposem que els conjunts Q i Γ són disjunts per poder utilitzar els estats com a símbols dintre d'una cadena. Una *configuració instantània*, o simplement configuració, és una descripció de la situació en què es troba M en un instant donat. Consta de la informació relativa a l'estat, la posició del capçal i el contingut de la cinta. Representem una configuració instantània mitjançant un mot del llenguatge $\Gamma^* Q \Gamma^*$, assumint que $Q \cap \Gamma = \varnothing$. Un mot $\alpha q \beta$, en què $q \in Q$, α i β pertanyen a Γ^* i β no acaba en b, representa la configuració en què M es troba en l'estat q, el contingut de la cinta és $\alpha\beta$, seguit de blancs a la dreta, i la posició del capçal és la del primer caràcter de β o d'un blanc si $\beta = \lambda$.

Podem definir relacions entre configuracions depenent del nombre de vegades que la TM ha d'aplicar la funció de transició per passar de l'una a l'altra.

DEFINICIÓ Donades una màquina de Turing M i dues configuracions instantànies, $\alpha q \beta$ i $\alpha' q' \beta'$, és a dir, dos mots de $\Gamma^* Q \Gamma^*$, diem que *$\alpha q \beta$ produeix $\alpha' q' \beta'$ en un pas* de M, i ho expressem en la forma

$$\alpha q \beta \underset{M}{\vdash} \alpha' q' \beta'$$

si en M passem de la configuració $\alpha q \beta$ a la configuració $\alpha' q' \beta'$ aplicant una vegada la funció de transició.

Diem que *$\alpha q \beta$ produeix $\alpha' q' \beta'$ en t passos* de M, i ho expressem en la forma

$$\alpha q \beta \overset{t}{\underset{M}{\vdash}} \alpha' q' \beta'$$

si a partir de $\alpha q \beta$, aplicant t vegades la funció de transició, M arriba a la configuració $\alpha' q' \beta'$. Finalment, diem que *$\alpha q \beta$ produeix $\alpha' q' \beta'$* en un *nombre finit de passos* de M, i ho expressem en la forma

$$\alpha q \beta \overset{*}{\underset{M}{\vdash}} \alpha' q' \beta'$$

si existeix un $t \geqslant 0$ per al qual $\alpha q \beta \overset{t}{\underset{M}{\vdash}} \alpha' q' \beta'$.

Diem que una configuració és *terminal* quan no pot anar seguida de cap configuració en la relació $\underset{M}{\vdash}$. Això pot tenir només una d'aquestas causes:

a) No hi ha cap transició definida en la funció de transició per a la combinació estat-símbol de la configuració.

b) L'estat apareix al principi de la configuració (indicant que el capçal és a la primera cel·la de la cinta) i la transició que correspon comporta un moviment a l'esquerra.

La *configuració inicial* és la configuració $\blacktriangleright q_0 w$, on w és el mot d'entrada, és a dir, la configuració en què l'estat és l'estat inicial, a la cinta tenim el mot d'entrada i la posició del capçal és la segona cel·la de la cinta.

El *càlcul* de M amb entrada $w \in \Sigma^*$ és una seqüència de configuracions, possiblement infinita, la primera de les quals és la inicial $\blacktriangleright q_0 w$; cadascuna de les restants és produïda per l'anterior en un pas, i l'última, si existeix, és terminal. Quan el càlcul de M amb entrada w és una seqüència infinita, diem que *M amb entrada w no s'atura* o que *divergeix*, i quan és finita diem que *M amb entrada w s'atura*.

Utilitzem la notació següent per referir-nos al comportament d'una màquina de Turing davant d'un mot d'entrada w escrit a la cinta:

- $M(w)\uparrow$ significa que la TM M amb entrada w no s'atura.
- $M(w)\downarrow$ significa que la TM M amb entrada w s'atura.
- En cas que $M(w)\downarrow$, $M(w)$ representa el mot de Σ^* corresponent a l'última configuració del càlcul de M amb entrada w, contingut a la cinta i comprés entre el primer i el segon caràcter que no són de Σ. Si no s'ha modificat, $\blacktriangleright$ és el primer caràcter que no pertany a Σ.

Observeu que l'única manera de aturar-se que té una TM és arribar a una configuració terminal. A més la nostra definició d'aturada pot no concidir amb l'absència de moviment del capçal, perquè hi poden haver bucles infinits amb moviment `n`.

1.2 Llenguatge reconegut i funció computada

Utilitzem l'estat acceptador per determinar quan una TM accepta una entrada w.

DEFINICIÓ Un mot $w \in \Sigma^*$ és *acceptat* per una màquina de Turing M si M amb entrada w s'atura i ho fa en l'estat acceptador.

A partir d'aquest concepte podem definir el de llenguatge reconegut per una TM.

DEFINICIÓ El *llenguatge reconegut o acceptat* per una TM M, representat per $\mathsf{L}(M)$, està format pel conjunt de mots acceptats per M. Donat un llenguatge L, diem que M *reconeix o accepta* L si $\mathsf{L}(M) = L$. Diem que un llenguatge L és *enumerable recursivament* si existeix una TM que reconeix L.

Quan una TM s'atura amb una entrada, podem extreure informació del que ha quedat escrit a la cinta i associar-la com a sortida. Això ens permet introduir els conceptes de funció computada i funció computable.

DEFINICIÓ La *funció computada* per una TM M, representada per f_M, es defineix així: $\mathsf{f}_M(w)$ és $M(w)$ si $M(w)\downarrow$, i està indefinida en cas contrari. Donada una funció $f : \Sigma^* \longrightarrow \Sigma^*$ diem que M *computa* f si $f = \mathsf{f}_M$. Finalment, diem que una funció és *computable* si hi ha una TM M per a la qual $f = \mathsf{f}_M$.

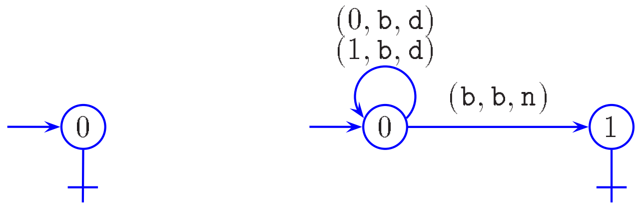


Fig. 1.3: TM *que reconeixen* Σ^*

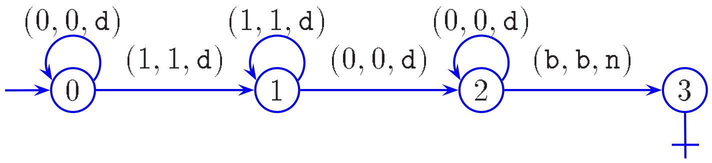


Fig. 1.4: TM *que reconeix* $0^*1^+0^+$

Per raons històriques en el desenvolupament de la teoria de la calculabilitat, és habitual trobar en textos clàssics, com el de H. Rogers [32], el terme *funció recursiva parcial*, referit al nostre concepte de funció computable.

Exemple 1.2 És fàcil demostrar que les màquines de Turing de la figura 1.3, amb alfabet d'entrada $\Sigma = \{0, 1\}$ i alfabet de cinta $\Gamma = \{0, 1, \blacktriangleright, \mathtt{b}\}$, reconeixen el llenguatge Σ^*. Mentre que la funció computada per la primera és la funció identitat, la segona computa la funció constant $f(w) = \lambda$ per a tot $w \in \Sigma^*$.

Exemple 1.3 La TM donada a la figura 1.2 computa la funció **següent**, que fa correspondre a cada mot $x \in \Sigma^*$ el següent segons l'ordre lexicogràfic per longituds. Com que sempre s'atura en l'estat acceptador, aquesta màquina reconeix Σ^*.

Exercici 1.1 Construeix una màquina de Turing M per a la qual $\mathsf{L}(M) = \varnothing$ i tal que per a tot $w \in \Sigma^*$, $\mathsf{f}_M(w) = w$.

Exercici 1.2 ¿Hi ha alguna relació entre el llenguatge reconegut per una TM M i el domini de la funció computada per M?

Exemple 1.4 Considerem la TM de la figura 1.4 que reconeix el llenguatge $0^*1^+0^+$. Els càlculs amb entrada 0001001 i 001100 són:

$\blacktriangleright q_0 0001001 \vdash$	$\blacktriangleright q_0 001100 \vdash$
$\vdash \blacktriangleright 0q_0 001001 \quad \vdash \blacktriangleright 00q_0 01001$	$\vdash \blacktriangleright 0q_0 01100 \quad \vdash \blacktriangleright 00q_0 1100$
$\vdash \blacktriangleright 000q_0 1001 \quad \vdash \blacktriangleright 0001q_1 001$	$\vdash \blacktriangleright 001q_1 100 \quad \vdash \blacktriangleright 0011q_1 00$
$\vdash \blacktriangleright 00010q_2 01 \quad \vdash \blacktriangleright 000100q_2 1$	$\vdash \blacktriangleright 00110q_2 0 \quad \vdash \blacktriangleright 001100q_2$
	$\vdash \blacktriangleright 001100q_3$

Exemple 1.5 Màquina de Turing que reconeix $\{0^n 1^{n^2} \mid n \in \mathbb{N}\}$ (vegeu la figura 1.5). La màquina utilitza tres símbols addicionals, `&`, `$` i `#`, per marcar els caràcters originals i poder-ne controlar el tractament. Segueix l'esquema:

- La màquina realitza una fase de càlcul per a cada 0 al bloc inicial de n 0's consecutius de l'entrada. (El cicle d'estats que comença a l'estat q_1.)
- Per controlar la fase en què es troba i per poder detectar el final del tractament, canvia un 0 per un `$`. Inicialment marca el primer 0, i la fase i comença quan està marcat el 0 a la posició i. Al final d'una fase torna a deixar el símbol marcat com al començament, i marca el zero següent, canviant-lo per un `$`.
- A cada fase, utilitzant la mateixa tècnica de marcatge, per a cada element al segment inicial de 0's es treu un 1 de la part final del mot. La fase acaba bé sempre que al començament quedin n 1's després del bloc inicial de 0's.
  - Per controlar els símbols tractats es fa, com abans, un recorregut marcant el símbol tractat; ara podem tenir un 0 o un `$`, la màquina utilitza `&` per marcar un 0 i `#` per marcar un `$`. Una vegada ha marcat el símbol esborra un 1 del final i marca l'element següent del bloc inicial de 1's. (El cicle d'estats que comença a l'estat q_2.) L'única diferència és que ara el marcatge és incremental, el tractament acaba quan tenim tot el segment inicial de 0's marcat.
  - Quan acaba el procés, ja ha esborrat els n 1's, i ha de tornar a deixar la part inicial marcada com al començament (estat q_9). Per això retrocedeix el capçal cap al començament de la cinta reescrivint els caràcters originals.
- Si el mot pertany al llenguatge, quan acabi la fase n ens hem de trobar amb un `b` a continuació del símbol `$`.

1.3 Màquines de Turing d'aturada segura. Temps de càlcul

Una família molt important de TM està formada per aquelles màquines que tenen garantida la finalització de qualsevol càlcul.

Definició Una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ és d'*aturada segura* si, per a tot mot $w \in \Sigma^*$, es compleix que $M(w){\downarrow}$.

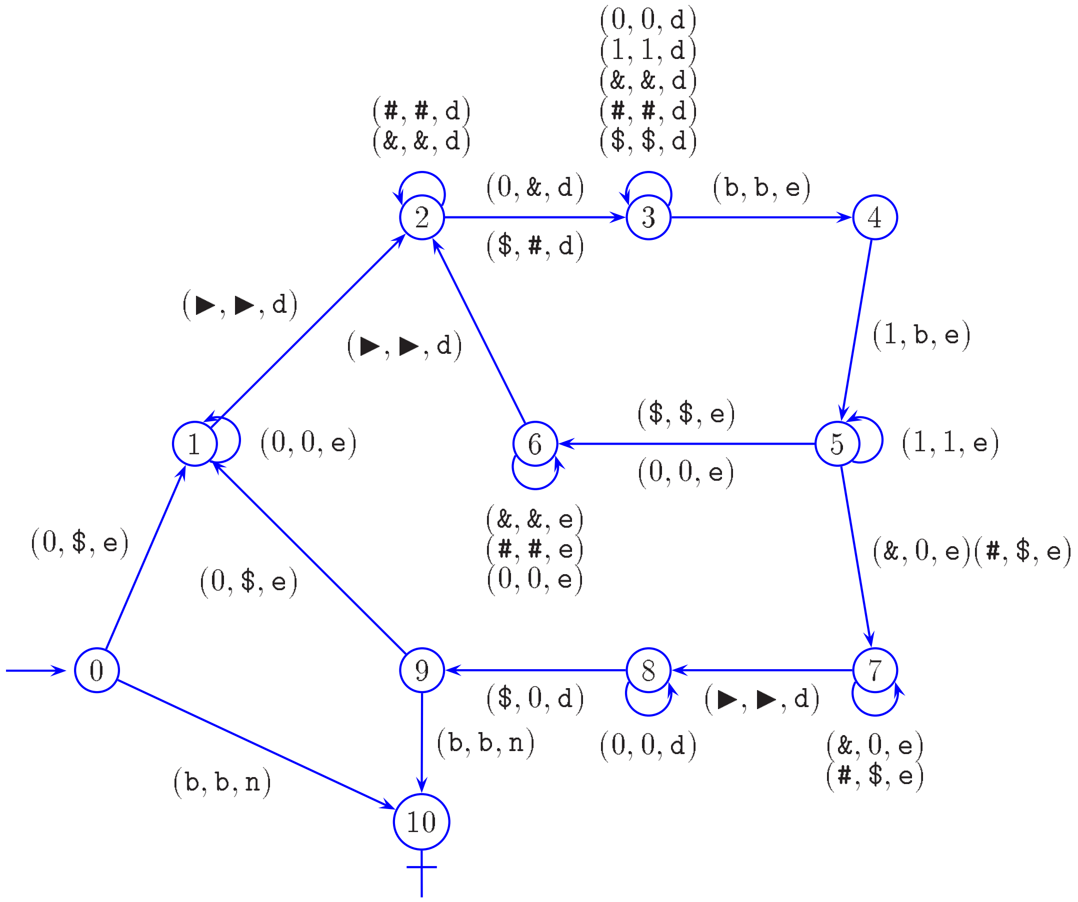


Fig. 1.5: TM *que reconeix* $\{0^n 1^{n^2} \mid n \in \mathbb{N}\}$

Quan tenim una TM d'aturada segura, utilitzem sovint el terme *decidir* en lloc de *reconèixer*. En aquest cas, la màquina o bé accepta (aturant-se en un estat acceptador) o bé rebutja (fent-ho en un estat no acceptador). Observeu que la funció computada per una TM d'aturada segura és sempre una funció total.

Utilitzem un terme particular per als llenguatges que es poden reconèixer per TM d'aturada segura.

DEFINICIÓ Un llenguatge L es *decidible* si hi ha una TM M d'aturada segura per a la qual $L = \mathsf{L}(M)$.

Clàssicament els llenguatges decidibles reben el nom de llenguatges *recursius*.

EXERCICI 1.3 Doneu dues TM que reconeguin el mateix llenguatge tals que una d'elles és d'aturada segura i l'altre no ho és.

EXERCICI 1.4 Doneu un exemple de llenguatge per al qual totes les TM que el reconeguin són d'aturada segura.

Remarquem el caràcter semàntic, però de cap manera sintàctic, de la definició de TM d'aturada segura. És a dir, la definició no es basa en cap propietat de l'estructura de la TM considerada, sinó en una propietat del comportament de la TM quan processa les entrades. Com veurem més endavant, és impossibe establir cap forma de codificació específica per al conjunt de TM d'aturada segura.

Donada una TM d'aturada segura, ens interessa analitzar el temps de càlcul de la màquina, sobre entrades d'una mateixa longitud. D'aquesta manera, podem descriure el temps de càlcul de la màquina en funció del nombre de caràcters processats. En aquest text ens centrem en l'*anàlisi del cas pitjor*.

DEFINICIÓ Donats una màquina de Turing M i un mot w, definim la funció $\mathsf{T}(M,w)$ de la manera següent: si $M(w)\downarrow$, $\mathsf{T}(M,w)$ és el nombre de passos de càlcul que realitza M amb l'entrada w; altrament T no està definida per a (M,w).

Quan M és una TM d'aturada segura, la funció $\mathsf{T}(M,w)$ està definida per a tot mot $w \in \Sigma^*$, llavors podem definir una funció de cost en el cas pitjor.

DEFINICIÓ Donada una màquina de Turing d'aturada segura M, definim la funció *temps de càlcul* de M com

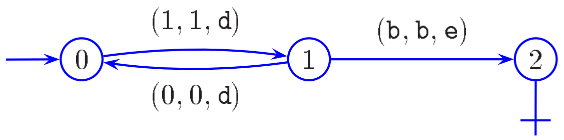
$$\mathsf{t}_M(n) = \mathsf{max}_{\,|w|=n} \mathsf{T}(M,w).$$

Aquesta funció dóna el temps màxim de càlcul requerit per M, sobre totes les entrades d'una mateixa talla n. Per això es diu que es tracta del cas pitjor, perquè de totes les entrades d'una mateixa talla pren el valor de la que requereix més temps de càlcul. En general, no ens interessa calcular el valor exacte de la funció de temps. Cercarem una fita superior que reflecteixi el cost temporal de la màquina. Això és el que fem amb les definicions següents.

DEFINICIÓ Sigui t una funció de $\mathbb{N}$ en $\mathbb{N}$. Diem que una màquina de Turing d'aturada segura M *decideix* un llenguatge L en *temps* t si $\mathsf{L}(M) = L$ i $\mathsf{t}_M = \mathrm{O}(t)$. Diem que un llenguatge L és *decidible en temps* t si hi ha una TM d'aturada segura que decideix L en temps t.

DEFINICIÓ Sigui t una funció de $\mathbb{N}$ en $\mathbb{N}$. Diem que una màquina de Turing d'aturada segura M *computa* una funció total $f : \Sigma^* \longrightarrow \Sigma^*$ en *temps* t, si $\mathsf{f}_M = f$ i $\mathsf{t}_M = \mathrm{O}(t)$. Diem que una funció total f és *computable en temps* t si hi ha una TM d'aturada segura que computa f en temps t.

EXEMPLE 1.6 Considerem la TM M següent



M reconeix el llenguatge $L = \{1(01)^i \mid i \in \mathbb{N}\}$ i és d'aturada segura. Amb qualsevol entrada w tal que $|w| = n$, la màquina s'atura com a molt en $n+1$ passos, ja que en el cas pitjor $w \in L$ i ha de recórrer tota l'entrada. Així, tenim $\mathsf{T}(M, w) \leqslant |w| + 1$, és a dir M decideix L en temps $\mathrm{O}(n)$.

Exercici 1.5 Analitzeu el temps de càlcul de les TM d'aturada segura construïdes en aquest capítol.

Moltes vegades ens trobem amb funcions que no són totals; per exemple, la funció pred, que assigna a cada mot de Σ^* el seu predecessor en ordre lexicogràfic per longituds. En aquest cas, la funció no és total ja que el mot λ no té cap predecessor. D'acord amb la nostra definició, no hi ha cap TM d'aturada segura que computi la funció pred tot i que és fàcil determinar en quins valors la funció no està definida. Sovint en aquests casos ens interessa saber si és possible trobar una TM d'aturada segura, que computi la funció en els casos en què estigui definida i que ens torni algun tipus de missatge quan la funció no ho estigui. En aquest exemple, és fàcil; en tenim prou afegint un símbol especial al nostre alfabet d'entrada, per exemple \$, i definint una funció que coincideix amb la funció pred quan el mot d'entrada no és λ i que val \$ quan l'entrada és λ.

Sovint, tenim funcions amb conjunt de partida que no és Σ^*. En aquest cas, el conjunt de dades d'entrada E és codificable, però el conjunt de codificacions dels elements de E és un subconjunt estricte de Σ^*. Quan el llenguatge de codificacions és decidible, és a dir, disposem d'una TM d'aturada segura que reconeix els mots de Σ^* que són codificacions vàlides d'elements de E, podem dissenyar una TM que computi la funció donada sota la precondició que l'entrada sigui una codificació vàlida. És a dir, només demanem que la TM computi la funció quan l'entrada és una codificació vàlida, i no ens preocupem del comportament de la TM sobre mots que no ho siguin. Habitualment diem, per abús de llenguatge, que aquesta TM computa la funció.

Exemple 1.7 Volem computar amb una TM la funció $f : \mathbb{N} \longrightarrow \mathbb{N}$ definida per $f(n) = n + 1$. Representem un nombre natural, tal com hem dit, per la seva expansió binària. El llenguatge corresponent a les codificacions vàlides és el definit per l'expressió regular (vegeu l'apèndix **??**) $1(0+1)^* + 0$, que és decidible en temps constant. En aquest context, diem que la TM de la figura 1.6 computa f. Aquesta TM s'atura (i retorna λ) amb entrada λ. En canvi, es penja quan l'entrada és diferent de λ però no representa cap nombre natural.

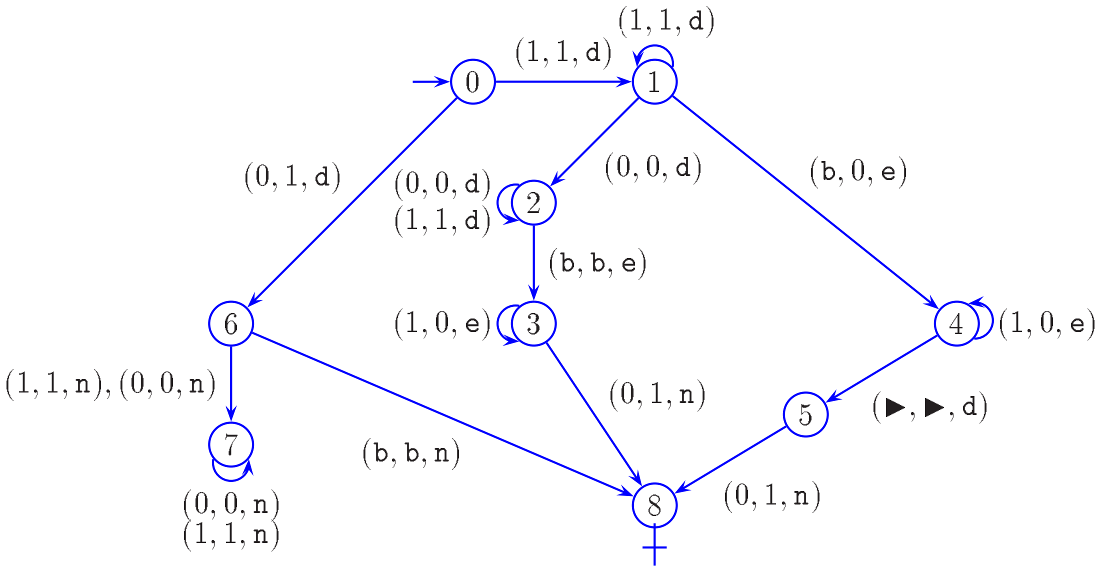


Fig. 1.6: *Una* TM *que calcula la funció $f(x) = x + 1$ definida sobre el conjunt dels nombres naturals*

Si a la TM de la figura 1.6 li treiem l'estat q_7, també tenim una TM que computa f sobre el conjunt dels naturals; en aquest cas la TM és d'aturada segura.

Quan el llenguatge de codificacions és decidible diem, també per abús de llenguatge, que la TM és d'aturada segura si s'atura sobre tot mot que representi un element de E.

1.4 Alguns problemes sobre màquines de Turing

Utilitzant la notació que hem introduït podem formalitzar amb claredat moltes propietats sobre TM. A continuació, enunciem alguns problemes sobre TM i formalitzem les propietats que els defineixen. L'obtenció d'algorismes que ens permetin la resolució d'aquests problemes serà discutida al llarg dels capítols següents.

Aturada (HALT)
Donats un mot w i una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, determinar si la màquina de Turing M s'atura amb entrada w.

La condició $M(w)\downarrow$ es pot expressar com

$$\boxed{\exists\, q \in Q\ \exists\, \alpha, \beta \in \Gamma^*\ \exists\, a \in \Gamma \quad \blacktriangleright q_0 w \underset{M}{\overset{*}{\vdash}} \alpha q a \beta \ \wedge\ (q, a) \notin \mathsf{Dom}\,\delta}$$

Pertinença (PERT)
Donats un mot w i una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, determinar si la màquina de Turing M accepta w.

La propietat $w \in \mathsf{L}(M)$ és pot expressar com

$$\boxed{\exists \alpha, \beta \in \Gamma^* \quad \blacktriangleright q_0 w \underset{M}{\overset{*}{\vdash}} \alpha q_F \beta}$$

Equivalència (TM-EQUIV)
Donades dues màquines de Turing M i N, determinar si reconeixen el mateix llenguatge.

La propietat $\mathsf{L}(M) = \mathsf{L}(N)$ és equivalent a

$$\boxed{\forall w \in \Sigma^* \quad w \in \mathsf{L}(M) \Longleftrightarrow w \in \mathsf{L}(N)}$$

que es pot expressar fàcilment utilitzant la formalització del problema PERT.

Minimització (TM-MINQ)
Donada una màquina de Turing M amb alfabet de cinta $\{0, 1, \blacktriangleright, \mathtt{b}\}$, obtenir una màquina de Turing M' amb alfabet de cinta $\{0, 1, \blacktriangleright, \mathtt{b}\}$ tal que $\mathsf{L}(M) = \mathsf{L}(M')$ i que M' tingui un nombre mínim d'estats.

Aquest és un problema funcional i la correspondència entrada-sortida es pot expressar per a TM, M, M', M'', que tinguin alfabet de cinta $\{0, 1, \mathtt{b}\}$, i conjunt d'estats Q, Q' i Q'' respectivament, com

$$\boxed{\mathsf{L}(M) = \mathsf{L}(M') \ \wedge \ \forall M'' \ \ (\mathsf{L}(M) = \mathsf{L}(M'') \Rightarrow \|Q''\| \geqslant \|Q'\|).}$$

1.5 Extensions del model bàsic de màquina de Turing

Hi ha moltes definicions alternatives de TM que inclouen petites variacions, com pot ser tenir més d'una cinta o incorporar indeterminisme al control d'estats. Tant el model bàsic com les variants raonables tenen el mateix potencial de càlcul. En aquesta secció, descrivim dues d'aquestes variants, la TM multicinta i la TM indeterminista; a més, esbossem la demostració de la seva equivalència des del punt de vista computacional.

1.5.1 La màquina de Turing multicinta

Una màquina de Turing multicinta és una TM que disposa d'un nombre finit de cintes semiinfinites (vegeu la figura 1.7). La màquina té un capçal amb moviment independent per a cada cinta.

A partir de l'estat i dels símbols actuals a cada cinta, una màquina de Turing multicinta, en una transició, realitza les accions següents:

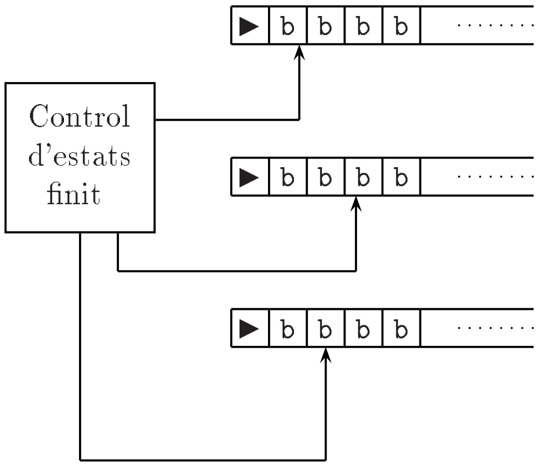


Fig. 1.7: *Esquema d'una màquina de Turing amb tres cintes*

- canvia l'estat;
- escriu a les cel·les apuntades pels capçals;
- mou els capçals de forma independent a cada cinta.

La funció de transició especifica el canvi d'estat de la unitat de control de la màquina i els canvis en les cintes. Un $(k+1)$-tuple (estat, caràcter, ... , caràcter) descriu l'argument de la funció de transició amb l'estat i els k caràcters actuals. El tuple que descriu el moviment ha d'especificar el nou estat, els nous caràcters a les cintes i els moviments dels capçals.

Continuem demanant que des de l'estat acceptador no hi hagi transicions definides. Quant a l'entrada, una de les cintes actua com a cinta d'entrada i conté el mot d'entrada. Inicialment la unitat de control de la màquina es troba en un estat q_0, amb un mot $x \in \Sigma^*$ escrit al començament de la cinta d'entrada i tots els capçals estan a sobre del primer caràcter després del símbol ▶. El funcionament de la màquina és similar al que s'ha descrit per al model bàsic amb una sola cinta, fent en un pas els canvis a cadascuna de les cintes.

La definició formal d'una màquina de Turing amb k cintes és:

DEFINICIÓ Una *màquina de Turing amb k cintes* és una estructura de la forma $(Q, \Sigma, \Gamma, \delta, q_0, q_F)$, on

Q és un conjunt finit d'estats.

Σ és un alfabet, l'alfabet d'entrada.

Γ és un alfabet, l'alfabet de cinta, tal que $\Sigma \cup \{\mathtt{b}, \blacktriangleright\} \subseteq \Gamma$, on $\mathtt{b}, \blacktriangleright \notin \Sigma$.

δ és la funció de transició, $\delta : Q \times \Gamma^k \longrightarrow Q \times (\Gamma \times \{\mathtt{e}, \mathtt{d}, \mathtt{n}\})^k$.

q_0 és l'estat inicial ($q_0 \in Q$).

q_{F} és l'estat acceptador ($q_{\mathrm{F}} \in Q$).

Les convencions gràfiques són similars, tenint en compte que ara hem d'especificar les transformacions de les k cintes,

$$i \xrightarrow{(a_1, b_1, m_1, \ldots, a_k, b_k, m_k)} j$$

De vegades, quan en una cinta es produeixen moviments, mentre que les altres cintes no es modifiquen, utilitzem un format comprimit en què el símbol $*$ s'utilitza com a jòquer, per indicar que el contingut de la cinta corresponent no es modifica. La figura 1.8 dóna un exemple de representació d'una TM amb quatre cintes.

La representació d'una configuració instantània s'estén també sense dificultat al cas de la TM multicinta. Donada una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$ amb k cintes, assumim, com és habitual, que $Q \cap \Gamma = \varnothing$ i considerem dos nous símbols $\#$ i $\$$, que no pertanyen a $Q \cup \Gamma$. Representem una configuració instantània mitjançant un mot del llenguatge $\Gamma^* Q \Gamma^* (\# \Gamma^* \$ \Gamma^*)^{k-1}$. El mot

$$\alpha_1 q \beta_1 \# \alpha_2 \$ \beta_2 \# \cdots \# \alpha_k \$ \beta_k$$

representa la configuració en la qual q és l'estat actual de M; per a cada i, $1 \leqslant i \leqslant k$, el contingut de la cinta i és $\alpha_i \beta_i$ seguit de $\mathtt{b}$ (la resta de la cinta està tota en blanc), i el i-èsim capçal és a sobre del primer caràcter de β_i o d'un blanc si $\beta_i = \lambda$.

A partir d'aquesta representació, podem generalitzar de forma natural la noció de pas de càlcul i la relació entre configuracions.

Definició Diem que una TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$ amb k cintes *accepta* un mot $w \in \Sigma^*$ si existeixen $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k \in \Gamma^*$ per als quals es compleix

$$\blacktriangleright q_0 w \# \blacktriangleright \$ \# \cdots \# \blacktriangleright \$ \underset{M}{\overset{*}{\vdash}} \alpha_1 q_{\mathrm{F}} \beta_1 \# \alpha_2 \$ \beta_2 \# \cdots \# \alpha_k \$ \beta_k.$$

El resultat d'un càlcul, quan la màquina s'ha aturat, s'obté amb el mateix conveni que en el model bàsic. Fixem una cinta com a cinta de sortida i n'extraiem el resultat. Podem fixar com a cinta de sortida una cinta diferent de la d'entrada .

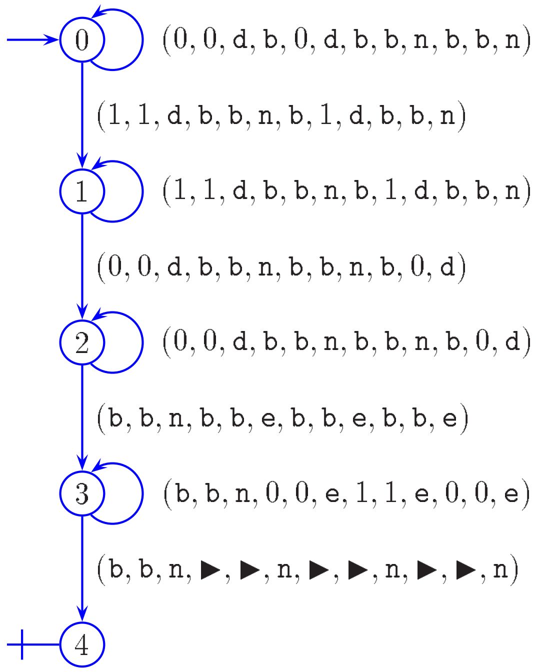


Fig. 1.8: *Una màquina de Turing que reconeix* $\{0^n1^n0^n \mid n > 0\}$.

A partir d'aquestes definicions, les nocions de llenguatge reconegut i funció computada s'estenen sense dificultat. Vegem alguns exemples d'ús del model.

EXEMPLE 1.8 Màquina de Turing que reconeix el llenguatge $\{0^n1^n0^n \mid n > 0\}$.

Utilitzarem una TM amb 4 cintes, c1, c2, c3, c4, on c1 és la cinta d'entrada.

1. Mentre el capçal de c1 llegeix 0, copiar de c1 a c2; avançar cap a la dreta els capçals de c1 i c2.

2. Mentre el capçal de c1 llegeix 1, copiar de c1 a c3; avançar cap a la dreta els capçals de c1 i c3.

3. Mentre el capçal de c1 llegeix 0, copiar de c1 a c4; avançar cap a la dreta els capçals de c1 i c4.

4. Si el capçal de c1 llegeix `b`:

    4.1. Fem retrocedir alhora els capçals de c2, c3 i c4, fins a trobar el símbol d'inici de cinta en alguna de les cintes.

    4.2. Si els tres capçals són a sobre del símbol d'inici de cinta, acceptar.

L'esquema d'una màquina que implementa aquest algorisme es dóna a la figura 1.8.

Exemple 1.9 Volem obtenir la representació en base 1 d'un nombre representat en base 2 i, a l'inrevés, obtenir la representació en base 2 d'un nombre representat en base 1.

Recordem que en la representació d'un nombre en base 2 s'utilitzen dues xifres diferents, el 0 i l'1. En la representació d'un nombre en base 1, s'utilitza una única xifra, que en el nostre cas és l'1. Així, el valor 10 en decimal té la representació 1010 en base 2, i la representació 1111111111 en base 1.

Vegem com passar de base 1 a base 2. Utilitzem una TM amb dues cintes: la cinta 1 s'utilitza com a cinta d'entrada i la cinta 2 com a cinta de sortida. L'esquema global és

1. Escriu 0 a c2.
2. Mentre el capçal de c1 no vegi el símbol `b`:
   2.1. Sumar 1 en c2 (podem adaptar la TM de l'exemple 1.7), garantint que el capçal quedi a l'inici de la cinta 2.
   2.2. Avançar el capçal de c1.

Per al canvi de base 2 a base 1, utilitzem també una TM amb dues cintes, que es pot implementar utilitzant l'algorisme següent:

1. Mentre el contingut de c1 no sigui 0:
   1.1. Restar 1 en c1, garantint que el capçal quedi a l'inici de la c1.
   1.2. Escriure 1 en c2 i avançar el capçal una posició cap a la dreta.

Exercici 1.6 Acabeu d'especificar les transicions de les dues màquines de canvi de base.

L'ús de màquines multicinta facilita el disseny de TM sense variar, però, la potència de càlcul del model bàsic. El teorema següent estableix aquesta equivalència.

Teorema 1.1 *El conjunt de llenguatges reconeguts (funcions computades) per màquines de Turing multicinta és el mateix que el dels llenguatges reconeguts (funcions computades) per màquines de Turing unicinta.*

Demostració *(Esbós)* Com que tota màquina unicinta també és multicinta, només cal veure com podem obtenir una TM unicinta que simuli el funcionament d'una TM multicinta.

Sigui M una TM que utilitza k cintes. Com sempre, assumim que Q i Γ no tenen símbols en comú. Construïm una TM M' unicinta que tingui com a alfabet de cinta el format pels símbols de Q, els de Γ i dos nous símbols `#` i `$`.

M' mantindrà a la seva cinta l'estat i el contingut de les k cintes de la forma següent:

$$q\#\alpha_1\$\beta_1\#\alpha_2\$\beta_2\#\cdots\#\alpha_k\$\beta_k\#.$$

Els símbols examinats pels capçals de M són els que figuren a la dreta dels símbols `$`, o bé blancs si a la dreta d'un `$` hi ha `#`.

Inicialment, la màquina comença desplaçant el contingut de la cinta d'entrada cap a la dreta, per poder escriure q_0`#$`, després de ▶ i abans del mot d'entrada. A continuació escriu $k-1$ vegades els símbols `$#`.

A partir d'aquest moment simula el moviment en les k cintes. Cada vegada que una de les cintes necessita utilitzar un espai en blanc addicional, M' l'insereix abans del símbol `#` corresponent, desplaçant una posició cap a la dreta el contingut de la cinta. En tot moment, els símbols `$` indiquen la posició dels capçals. Aquesta simulació es fa per les k cintes i després es canvia l'estat. □

1.5.2 La màquina de Turing indeterminista

Una màquina de Turing indeterminista és una TM que, a partir de l'estat i del símbol accessible a través del capçal, té associat un conjunt de ternes (estat, caràcter, moviment). En una transició, selecciona, de forma indeterminista, una d'entre les possibles ternes i modifica la seva configuració d'acord amb les mateixes regles que un TM del model bàsic. La definició formal d'una màquina de Turing indeterminista és:

DEFINICIÓ Una *màquina de Turing indeterminista* (abreujadament, NTM) és una estructura de la forma $(Q, \Sigma, \Gamma, \Delta, q_0, q_F)$, on

Q és un conjunt finit d'estats.

Σ és un alfabet, l'alfabet d'entrada.

Γ és un alfabet, l'alfabet de cinta, tal que $\Sigma \cup \{\mathtt{b}, \blacktriangleright\} \subseteq \Gamma$, on $\mathtt{b}, \blacktriangleright \notin \Sigma$.

Δ és el gràfic de transicions, $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{\mathtt{e}, \mathtt{d}, \mathtt{n}\})$.

q_0 és l'estat inicial ($q_0 \in Q$).

q_F és l'estat acceptador ($q_F \in Q$).

El *gràfic de transicions* especifica el conjunt de possibles canvis de configuració de la màquina en un estat. Continuem demanant que des de l'estat acceptador no hi hagi cap transició definida. Assumim també que la nostra màquina té una única cinta; les definicions es poden estendre fàcilment a un model de màquina de Turing indeterminista multicinta, encara que no ho farem.

Inicialment, la màquina es troba en l'estat inicial q_0, amb un mot $w \in \Sigma^*$ escrit a la cinta i el capçal a sobre del primer caràcter després del símbol ▶. El funcionament de la màquina és idèntic al descrit per al model determinista, amb una única diferència: la selecció arbitrària de transició. Els conceptes de configuració instantània i de relació de producció entre configuracions són idèntics als definits per a TM deterministes. La

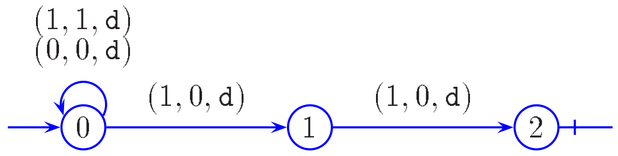


Fig. 1.9: *Un exemple d'*NTM

figura 1.9 dóna un exemple d'NTM. Com es pot veure, utilitzem les mateixes convencions per representar gràficament TM indeterministes.

El conjunt de càlculs possibles d'una màquina indeterminista amb una entrada donada es pot representar mitjançant un *arbre de computació*. Cada nus de l'arbre té associada un configuració de la màquina. L'arrel té associada la configuració inicial. Un nus té tants fills com configuracions diferents a què es pot arribar en un pas de l'NTM. Aquest arbre reflecteix totes les computacions possibles amb una entrada donada. Però hem de tenir clar que en una execució se selecciona de forma indeterminista un únic camí a partir de l'arrel. L'arbre de computació pot ser infinit, però tot i això pot tenir una branca que sí que porta a una configuració terminal. La figura 1.10 mostra l'arbre de computació de la TM de la figura 1.9 amb l'entrada 10110.

La noció d'acceptació pel model indeterminista es formula, a la vista del contingut de l'arbre de computació del mot d'entrada, de la manera següent:

DEFINICIÓ Diem que una NTM M indeterminista *accepta* un mot $w \in \Sigma^*$ si l'arbre de computació de M amb entrada w té alguna fulla que conté l'estat acceptador. Diem que una TM indeterminista M *accepta* w en temps t si l'arbre de computació de M amb entrada w té alguna fulla que conté l'estat acceptador a distància t de la configuració inicial.

Formalment, aquesta propietat s'expressa de manera idèntica al cas de les màquines deterministes, és a dir,

$$\mathsf{L}(M) = \{w \in \Sigma^* \mid \exists \alpha, \beta \in \Gamma^* \quad \blacktriangleright q_0 w \underset{M}{\vdash^*} \alpha q_{\mathrm{F}} \beta\}.$$

EXEMPLE 1.10 Volem una TM indeterminista que reconegui el llenguatge $L = \{ww \mid w \in \{0,1\}^*\}$. Considerem la màquina de Turing indeterminista de la figura 1.11 que correspon a l'esquema següent:

- Selecciona de manera indeterminista una posició al mot d'entrada.
- Utilitzant la posició seleccionada al pas precedent, comprova que el submot format per tots els caràcters des del començament fins a aquesta posició i el submot format pels caràcters des d'aquesta posició fins al final són el mateix mot.

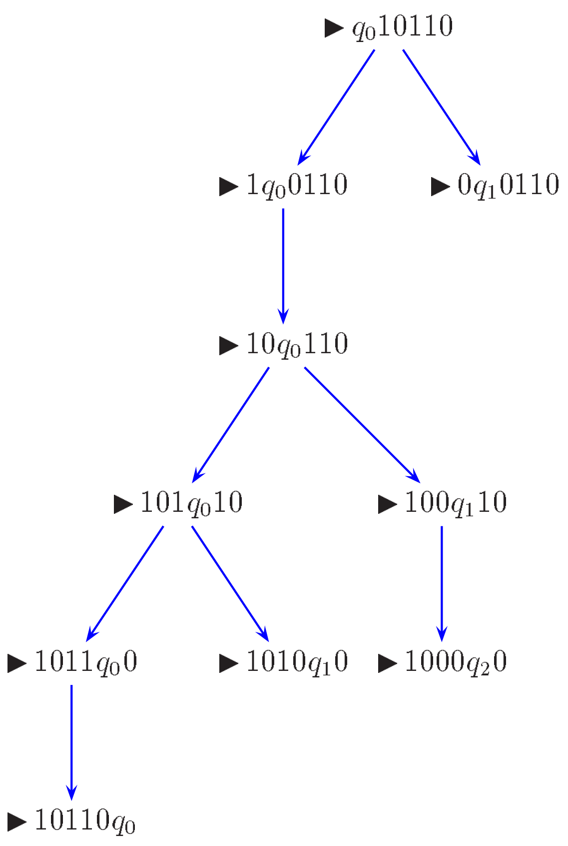


Fig. 1.10: *L'arbre de computació de la* TM *de la figura 1.9 amb entrada 10110*

L'ús de màquines indeterministes facilita el disseny de TM, però sense variar la potència de càlcul del model bàsic. El teorema següent estableix aquesta equivalència.

TEOREMA 1.2 *El conjunt de llenguatges reconeguts per màquines de Turing indeterministes és el mateix que el conjunt de llenguatges reconeguts per màquines de Turing deterministes.*

DEMOSTRACIÓ (*Esbós*) Com que tota màquina determinista unicinta és també una màquina indeterminista, només ens cal veure com es pot construir una màquina determinista que simuli el funcionament d'una màquina indeterminista. Sigui $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_F)$ una TM indeterminista. Com sempre, assumim que Q i Γ no tenen símbols en comú. Construïm una TM M' determinista que utilitza dues cintes. En la primera cinta emmagatzema configuracions de la màquina M i fa servir la segona per simular un recorregut en amplada de l'arbre de computació.

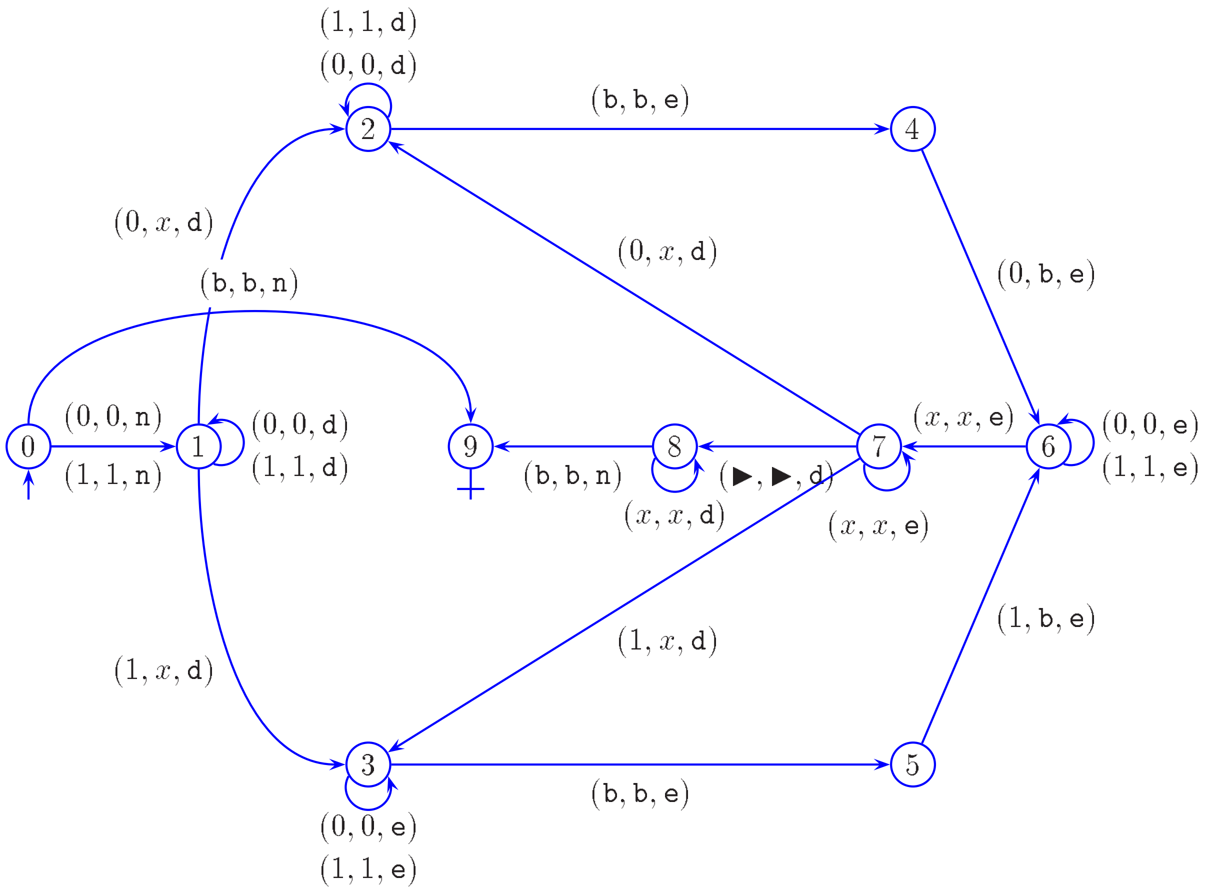


Fig. 1.11: *Una* NTM *que reconeix* $\{ww \mid w \in \Sigma^*\}$

Per fer un recorregut en amplada, podem fer servir una seqüència de valors per determinar el nus en què estem, de manera que sigui fàcil determinar quin és el nus següent. Si per a un parell (estat, símbol) la relació de transició té com a màxim k possibilitats diferents, podem utilitzar un recorregut en ordre lexicogràfic per longituds dels mots de l'alfabet $\{1, \ldots, k\}$. Partint de l'arrel, cada mot identifica un camí; per exemple, si tenim 1213 hem de començar a l'arrel, passar al primer fill, després passar al segon fill del nus actual, després al primer, etc. La longitud del mot que descriu el camí que cal seguir ens indica la profunditat a què està el nus. Hem de tenir en compte que alguns d'aquests mots no corresponen a nusos que existeixin a l'arbre, ja que en aquesta aproximació estem suposant que es tracta d'un arbre complet amb grau de sortida k.

Combinant la TM que calcula el mot següent en ordre lexicogràfic, amb una TM que utilitza el mot per seleccionar a cada pas la transició marcada pel mot, i utilitzant l'esquema de simular un pas, obtenim una TM determinista.

Si en algun moment s'arriba a una configuració acceptadora, la màquina M' s'atura i ho fa acceptant. Altrament, M' procedeix indefinidament assajant noves possibilitats. $\square$

1.6 Problemes

1.1 Suposem que tenim una màquina de Turing en què $q_0 = q_F$. Quin llenguatge reconeix? Quina funció computa?

1.2 Dissenyeu màquines de Turing que reconeguin els llenguatges següents:

1. $\{0^{2n} \mid n \in \mathbb{N}\}$
2. $\{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$
3. $\{0^n 1^n \mid n \in \mathbb{N}\}$
4. $\{0^n 1^m 0^{n+m} \mid n, m \in \mathbb{N}\}$
5. $\{www \mid w \in \{0,1\}^*\}$

1.3 Siguin M una màquina de Turing amb alfabet d'entrada Σ, i $x \in \Sigma^*$. Digueu si les afirmacions següents són certes o no i per què:

a) Si M amb entrada x no repeteix cap configuració, llavors $M(x)\downarrow$.

b) Si $M(x)\downarrow$, llavors M amb entrada x no repeteix cap configuració.

c) Si M amb entrada x no repeteix cap configuració, llavors $M(x)\uparrow$.

1.4 Dissenyeu màquines de Turing amb una cinta d'entrada i una de sortida (on quedi enregistrada la sortida en acabar el càlcul) per a les funcions següents:

1. $f : 0^* \to 0^*$, $f(0^n) = 0^{n+1}$.
2. $f : 0^* \times 0^* \to 0^*$, $f(0^i, 0^j) = 0^{i-j}$ si $j < i$, i λ altrament.
3. $f : \{0,1\}^* \to \{0,1\}^*$, $f(w) = 1$ si $w > 0$, i 0 altrament.

1.5 Una TM amb *cinta biinfinita* és similar a una TM unicinta excepte que la seva cinta és infinita a la dreta i a l'esquerra. Inicialment la cinta està plena de blancs, excepte la porció on s'escriu l'entrada. La computació és com la usual, excepte que ara no tenim cap símbol de començament de cinta. Doneu una definició formal de TM amb cinta biinfinita i demostreu l'equivalència amb el model bàsic de TM.

1.6 Una TM *enumeradora* és una TM amb una cinta de sortida, que es fa servir per escriure mots d'un alfabet separats per blancs, i cintes auxiliars. Una TM enumeradora comença amb totes les cintes en blanc. A mesura que fa càlculs, pot escriure mots a la cinta de sortida. El capçal de la cinta de sortida només es pot moure cap a la dreta. Si la màquina no s'atura, pot imprimir una llista infinita de mots.

El llenguatge enumerat per una TM enumeradora E està format per tots els mots que escriu a la cinta de sortida.

Doneu una definició formal de TM enumeradora. Demostreu la igualtat entre el conjunt de llenguatges enumerats per alguna TM enumeradora i el conjunt de llenguatges reconeguts per alguna TM.

2 Màquines de Turing i algorismes

En aquest capítol justificarem l'ús de la màquina de Turing com a model teòric del concepte d'algorisme. Primer, veurem que les estructures algorísmiques bàsiques es poden implementar amb TM. Després, veurem la manera d'afegir altres funcionalitats com la compilació o la interpretació, que es poden esperar de qualsevol llenguatge de programació. Totes aquestes raons ens porten a admetre la màquina de Turing com a model formal del concepte d'algorisme.

2.1 Els esquemes algorísmics bàsics

Donem implementacions amb TM dels esquemes algorísmics bàsics, tal com els podem trobar en qualsevol llibre bàsic d'iniciació a la programació, per exemple [10]. Assumint que una instrucció es correspon amb l'execució d'una TM, donem construccions que ens permetin introduir la seqüenciació, els condicionals i les iteracions, i finalitzem introduint valors constants. A més, donem indicacions, per a cada construcció, de com calcular el temps de càlcul d'una TM d'aturada segura, definida mitjançant l'esquema corresponent.

Composició seqüencial Donades dues TM, M i N, volem construir una TM que implementi la seva *composició seqüencial*, és a dir, l'esquema

$$M; N.$$

La nova TM ha de connectar la primera màquina amb la segona, preservant l'estat de la cinta. Observeu que, perquè N comenci, M ha hagut d'aturar-se. A més, si M s'atura acceptant, la TM que implementi la composició ha d'acceptar. A la figura 2.1 tenim una representació esquemàtica de la construcció.

Recordem que M només pot parar quan arribi a una configuració per a la qual no hi hagi cap transició definida. Aquests casos es poden detectar, mirant tots els parells (estat, caràcter), en la definició de la funció de transició de M.

Si $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$ i $N = (Q', \Sigma, \Gamma, \delta', q'_0, q'_{\mathrm{F}})$, suposem que els dos conjunts d'estats són disjunts; si no ho fossin, hauríem de redenominar els estats d'una de les dues màquines. Així mateix, considerarem que els alfabets de cinta i d'entrada són els mateixos a les dues màquines; si no, agafaríem la reunió dels respectius alfabets com a alfabet de la nova màquina.

Una màquina que correspon a la composició seqüencial (amb les condicions descrites abans) és

$$\mathbf{cs}(M, N) = (Q'', \Sigma, \Gamma, \delta'', q_0, q'_{\mathrm{F}}),$$

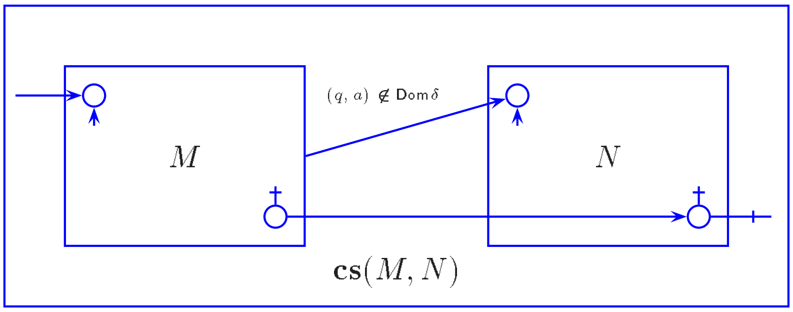


Fig. 2.1: *Esquema per a la composició seqüencial.*

on

1. $Q'' = Q \cup Q'$
2. δ'' conté les transicions de δ i de δ'. A més, per a tot $q \in Q$ i $a \in \Gamma$, si $q \neq q_{\mathrm{F}}$ i $(q, a) \notin \mathsf{Dom}\,\delta$, llavors afegim $\delta''(q, a) = (q'_0, a, \mathtt{n})$. Finalment, per a qualsevol $a \in \Gamma$ afegim $\delta''(q_{\mathrm{F}}, a) = (q'_{\mathrm{F}}, a, \mathtt{n})$

El criteri d'acceptació que hem fet servir per a la composició seqüencial de dues màquines de Turing M i N no és únic, i tampoc no és el que utilitzarem en tots els casos. De tota manera, les idees utilitzades en aquesta construcció són bàsiques per resoldre composicions amb altres restriccions.

Exercici 2.1 Dissenyeu una màquina de Turing que calculi la composició seqüencial de M i N, de manera que la màquina de Turing final, quan M para i accepta, no faci cas de l'acceptació i passi el control a N. Així, la nova TM només accepta un mot quan N ho fa.

Exercici 2.2 Donades dues TM, M i N, dissenyeu una màquina de Turing que calculi la funció $\mathsf{f}_M(\mathsf{f}_N(x))$. Observeu que, en aquest cas, cal deixar la cinta neta, només amb $\blacktriangleright N(x)$, i el capçal en la posició inicial, abans d'engegar el procés de M.

Quan M i N són TM d'aturada segura, la seva composició seqüencial també és una TM d'aturada segura. A més, el temps de càlcul de la composició és la suma dels temps que triguin les dues màquines.

Exemple 2.1 Considerem la TM descrita per

$$x := x + 1; x := 2 * x,$$

on x és un enter positiu. Hem vist a l'exemple 1.7 una TM M que suma una unitat a un enter positiu. El temps d'aquesta màquina és, com a molt, lineal en la longitud de l'entrada. Una TM N que multiplica per dos, aprofitant la representació en binari, només ha d'afegir un zero al final

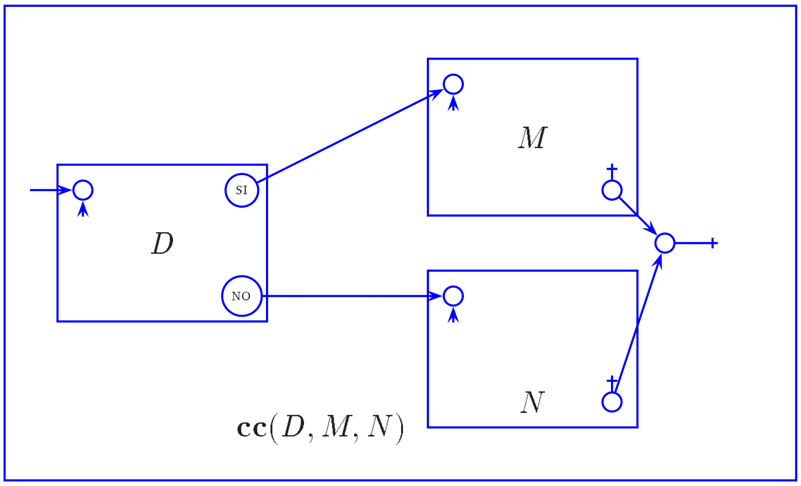


Fig. 2.2: *Esquema per a la composició condicional*

del mot que representa el valor actual, i també té cost lineal. Així, tenim dues màquines amb cost lineal i, per tant,

$$\mathsf{t}_{\mathsf{cs}(M,N)}(n) = \mathrm{O}(n).$$

A les següents definicions, anomenem TM *condicional* una TM D amb dos estats diferenciats, que denomienem q_{SI} i q_{NO}, per als quals no hi ha cap transició definida i que interpretarem com a respostes SI/NO a la pregunta associada a D.

Composició condicional Donades tres màquines de Turing M, N i D, on D és una màquina condicional, volem una TM que implementi la seva *composició condicional*, que correspon a l'esquema:

$$\textbf{si } D \textbf{ llavors } \ M \textbf{ si no } N \textbf{ fi si}.$$

La màquina D avalua la condició afirmativament quan s'atura en l'estat q_{SI} i negativament quan ho fa en l'estat q_{NO}. La figura 2.2 esquematitza aquesta construcció quan $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{F}})$, $N = (Q', \Sigma, \Gamma, \delta', q'_0, q'_{\text{F}})$ i $D = (Q'', \Sigma, \Gamma, \delta'', q''_0, q_{\text{SI}}, q_{\text{NO}})$.

Formalment, una màquina que correspon a l'esquema de composició condicional presentat és

$$\mathbf{cc}(D, M, N) = (Q''', \Sigma, \Gamma, \delta''', q''_0, q'''_{\text{F}}),$$

on

1. $Q''' = Q \cup Q' \cup Q'' \cup \{q'''_{\text{F}}\}$.

2. δ''' conté totes les transicions de δ, δ' i δ'' i, a més, per a tot $a \in \Gamma$, les transicions:

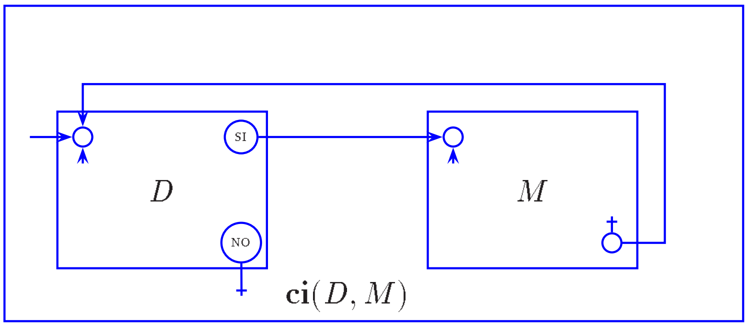


Fig. 2.3: *Esquema per a la composició iterativa.*

$$\delta'''(q_{\mathrm{SI}}, a) = (q_0, a, \mathtt{n}),$$
$$\delta'''(q_{\mathrm{NO}}, a) = (q_0', a, \mathtt{n}),$$
$$\delta'''(q_{\mathrm{F}}, a) = (q_{\mathrm{F}}''', a, \mathtt{n}),$$
$$\delta'''(q_{\mathrm{F}}', a) = (q_{\mathrm{F}}''', a, \mathtt{n}).$$

Quan D, M i N són TM d'aturada asegura, la seva composició condicional també és una TM d'aturada segura. El temps de càlcul de la composició és el temps que triga D a avaluar la condició, més el temps que triga la màquina que s'executi.

EXEMPLE 2.2 Considerem la TM descrita per

si $x = 0$ **llavors** $x := 1$ **si no** $x := x/2$ **fi si**.

Una TM D que comprova si $x = 0$ ho pot fer en temps constant. Una TM M que computa la funció constant amb valor 1, també ho pot fer en temps constant. Finalment una TM N que divideix per dos pot fer-ho en temps lineal. Així, tenim

$$\mathsf{t}_{\mathsf{cc}(D,M,N)}(n) = \mathrm{O}(1) + \mathsf{max}\ (\mathrm{O}(1) + \mathrm{O}(n)) = \mathrm{O}(n).$$

Composició iterativa Donades dues màquines de Turing M i D, on D és una màquina condicional, volem construir una TM que implementi la seva *composició iterativa*, que correspon a l'esquema bàsic

mentre D **fer** M **fi mentre**.

A la figura 2.3 donem un esquema de la construcció que fem servir quan tenim $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$ i $D = (Q', \Sigma, \Gamma, \delta', q_0', q_{\mathrm{SI}}, q_{\mathrm{NO}})$.

Una màquina que implementa la composició iterativa de M i N és

$$\mathbf{ci}(D, M) = (Q'', \Sigma, \Gamma, \delta'', q'_0, q_{\text{NO}}),$$

on

1. $Q'' = Q \cup Q'$.

2. δ'' conté totes les transiciones de δ i de δ' i, a més, per a tot $a \in \Gamma$, les transicions:

$$\delta''(q_{\text{SI}}, a) = (q_0, a, \mathtt{n}),$$
$$\delta''(q_{\text{F}}, a) = (q'_0, a, \mathtt{n}).$$

Observeu que quan D i M són TM d'aturada segura, la seva composició iterativa pot ser o no ser una TM d'aturada segura. Per garantir que ho sigui, hem de poder demostrar que, per a qualsevol entrada, el nombre d'iteracions és finit. Observem que les eines que cal fer servir per demostrar, en aquest context, l'obtenció d'una TM d'aturada segura són les mateixes que s'utilitzen per demostrar la 'finalització' d'un programa (referim el lector a [6]).

Quan sabem que la composició iterativa és una TM d'aturada segura, el seu temps de càlcul és la suma, per a cada iteració i, dels temps de càlcul de D i M en la iteració i-èsima. En general cerquem fites superiors del temps de càlcul de D i M en funció de la longitud màxima de l'entrada que reben a les diferents iteracions. Si tenim aquestes fites, sumant-les i multiplicant pel nombre d'iteracions obtenim una fita superior del temps de la composició iterativa.

EXEMPLE 2.3 Considerem la TM descrita per

$$\textbf{mentre } x > 1 \textbf{ fer } x := x - 1 \textbf{ fi mentre},$$

on x és un enter positiu. Una TM D que comprova $x > 1$ ho pot fer en temps constant. Una TM M pot restar una unitat en temps lineal. La condició d'entrada en el bucle i el fet que a cada iteració es decrementi el valor de x garanteixen que aquesta construcció correspon a una TM d'aturada segura. Sigui quin sigui el valor inicial X de x, la longitud màxima de l'entrada de M, a qualsevol iteració, és $|X|$, ja que a cada iteració decreix el valor emmagatzemat a x i la longitud inicial és $|X|$. Si $X \leqslant 1$, no es fa cap iteració i, si $X > 1$, el nombre d'iteracions és X. Llavors, quan $n = |X|$, podem fitar el nombre d'iteracions per $X \leqslant 2^n$ i el temps total de càlcul per iteració per $\mathrm{O}(n)$, i tenim

$$\mathsf{t}_{\mathsf{ci}(D,M)}(n) = \mathrm{O}(2^n)\,\mathrm{O}(n) = 2^{\mathrm{O}(n)}.$$

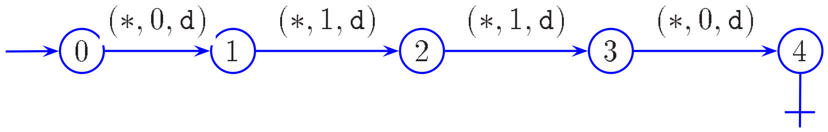


Fig. 2.4: *La màquina* C_{0110}

Introducció de valors constants A més dels esquemes algorísmics bàsics, tot programa té la capacitat d'incorporar valors constants. Per fer-ho amb el nostre model veiem com construir per a cada mot de Σ^* una TM que escriu aquest mot. Formalment, donat un mot $w \in \Sigma^*$, volem construir una TM, a la qual ens referim com C_w, que escriu w a partir de la posició on es trobi el capçal.

La construcció és molt senzilla: hem d'escriure, un a un, tots els símbols de w. Sigui $w = w_1 \cdots w_n$; fem servir un conjunt de $n+1$ estats $q_0, \ldots, q_n$ i, per a cada $a \in \Gamma$, definim la funció de transició com $\delta(q_i, a) = (q_{i+1}, w_{i+1}, \mathtt{d})$ per a $0 \leqslant i < n$. A la figura 2.4 es mostra un exemple de màquina que escriu la constant 0110.

Com en el cas de la composició seqüencial, hi ha moltes alternatives en la manera d'introduir els valors constants. La màquina proposada actua sempre en mode *sobreescriptura* i no en mode *inserció*. Totes les variants incorporen en algun moment l'escriptura dels símbols de la constant.

EXERCICI 2.3 Dissenyeu una TM Ins_w que insereix w a partir de la posició on es trobi el capçal. En aquest cas, cal desplaçar el contingut de la cinta $|w|$ posicions cap a la dreta.

Observeu que, per a qualsevol w, la TM C_w és d'aturada segura. A més, el temps que triga és constant.

Vegem ara com computar el temps de càlcul d'algunes TM descrites mitjançant construccions algorísmiques. En general, suposem que disposem d'una cinta per a cadascuna de les variables que apareguin en la descripció de l'algorisme.

EXEMPLE 2.4 Considerem la TM M, que té com a entrada un mot de Σ^*, descrita per l'esquema següent

entrada x
$y := \lambda; k := 0;$
mentre $y \neq x$ **fer**
    $y := \mathsf{següent}(y); k := k+1$
fi mentre;
escriure k.

on la funció $\mathsf{següent}$ torna el següent mot en l'ordre usual, lexicogràfic per longituds. M comença amb el primer mot i va recorrent Σ^* en ordre fins a trobar x. M és d'aturada segura i computa el número d'ordre del mot d'entrada. Sigui X el valor inicial de x. Com que y sempre conté un mot que és predecessor o igual a X, la longitud de l'entrada de la TM que computa el següent o que comprova la igualtat amb x està fitada per $n = |X|$. A més, en temps lineal es poden comparar dos mots, i també es pot calcular el següent en l'ordre usual. El nombre d'iteracions és exactament el nombre de mots predecessors de X. Aquest conjunt conté tots els mots de longitud menor que la longitud de X i, possiblement, alguns de longitud n. Així, tenim que el nombre d'iteracions està fitat per $\|\Sigma\|^{|X|+1} = 2^{\mathrm{O}(n)}$. Multiplicant les dues fites tenim $\mathbf{t}_M(n) = n2^{\mathrm{O}(n)} = 2^{\mathrm{O}(n)}$.

EXEMPLE 2.5 Considereu la TM M següent que té com a entrada un enter positiu.

> **entrada** x
> $y := 0; k := 0;$
> **mentre** $y \leqslant x$ **fer**
> $y := y + 5;$
> $k := k + 1$
> **fi mentre**;
> **escriure** k.

Com que el valor de y augmenta a cada iteració, la TM és d'aturada segura. Podem analitzar el temps de càlcul de M en funció del valor inicial X de x, i la longitud de l'entrada $n = |X|$. Les dues primeres assignacions es fan en temps constant. Com que y i k estan fitades superiorment, al llarg de tota l'execució, per X, tenim que les dues operacions dintre del bucle i la comparació es poden fer en temps $\mathrm{O}(n)$. El nombre d'iteracions és $\lfloor \frac{X}{5} + 1 \rfloor$ i, per tant, tenim $\mathbf{t}_M(n) = 2^{\mathrm{O}(n)}$.

EXEMPLE 2.6 Considereu la TM M següent que té com a entrada un enter positiu.

> **entrada** x
> $y := 1; k := 0;$
> **mentre** $y \leqslant x$ **fer**
> $y := y * 5;$
> $k := k + 1$
> **fi mentre**;
> **escriure** k.

Com que el valor de y augmenta a cada iteració, en un temps finit arribarà a tenir un valor que sobrepassi el valor inicial de x. Llavors, la TM és d'aturada segura.

Podem analitzar el temps de càlcul de M en funció del valor inicial X de x i de la longitud de l'entrada $n = |X|$. Com que y i k estan fitades superiorment, al llarg de tota l'execució, per X, les dues operacions dintre del bucle i la comparació es poden fer en temps $\mathrm{O}(n)$. Per determinar el nombre d'iteracions, relacionem el valor de y amb el nombre d'iteracions. Sigui y_i el valor de

y després de la iteració i. Tenim $y_0 = 1$ i, per a tot $i > 0$, $y_i = 5 * y_{i-1} = 5^i$. A partir d'aquesta expressió és fàcil trobar el nombre d'iteracions. Hem de trobar el primer i per al qual $5^i > X$, és a dir, $O(\mathsf{log}\ X)$. Així, tenim que el nombre d'iteracions és $O(\mathsf{log}\ X) = O(n)$. Com que la resta d'operacions dintre del bucle té cost $O(n)$, tenim $\mathsf{t}_M(n) = O(n^2)$.

EXEMPLE 2.7 Considereu la TM M següent que té com a entrada un enter positiu.

> **entrada** x
> $y := 2;$
> $k := 0;$
> **mentre** $y \leqslant x$ **fer**
> $y := y * y;$
> $k := k + 1$
> **fi mentre**;
> **escriure** k.

Utilitzant la mateixa tècnica que a l'exemple anterior, l'expressió del valor de y en funció del número d'iteració és ara $y_i = 2^{2^i}$ i el nombre d'iteracions és $O(\mathsf{log}\ (\mathsf{log}\ X))$, és a dir, $O(\mathsf{log}\ n)$. L'operació més costosa és el producte de dos enters, que es pot fer amb una TM en temps quadràtic, utilitzant per exemple l'algorisme tradicional que s'ensenya a l'escola per multiplicar dos nombres. Llavors, tenim $\mathsf{t}_M(n) = O(n^2 \mathsf{log}\ n)$.

A vegades tenim estructures de dades més complexes, com llistes o taules. En aquests casos, d'acord amb les condicions d'una codificació, tenim una TM que pot descodificar l'entrada i extreure les seves components. Aquestes components se solen emmagatzemar en diferents cintes. En general farem servir el nom de la dada com identificador de la funció de descodificació. Per exemple si el conjunt d'entrada es el conjunt dels grafs, disposem de la funció graf que ens torna el nombre de vèrtexs i la matriu d'adjacència.

EXEMPLE 2.8 La següent TM M, amb la codificació d'un graf G com a entrada, calcula el màxim grau dels vèrtexs de G.

> **entrada** x
> $(n, A) := \mathsf{graf}(x);$
> $max := 0; i := 1;$
> **mentre** $i \leqslant n$ **fer**
> $d := 0; j := 1;$
>   **mentre** $j \leqslant n$ **fer**
>   **si** $A[i, j]$ **llavors** $d := d + 1$ **fi si**;
> $j := j + 1$
>   **fi mentre**;
>   **si** $d > max$ **llavors** $max := d$ **fi si**;
> $i := i + 1$

fi mentre;
retorna (max).

Recordem que si utilitzem la codificació proposada al capítol **??** per a grafs, un graf amb n vèrtexs té talla n^2. Utilitzem la lletra m per referir-nos a la longitud de l'entrada, $m = |G| = n^2$. És clar que M és d'aturada segura, ja que els bucles s'executen n vegades cadascun. El bucle exterior s'executa n vegades, i a cada iteració s'executa n vegades el bucle interior. La resta d'operacions tenen cost lineal en la longitud dels seus operands. Llavors tenim que el temps total està fitat per $n^2 \mathsf{log}\ n$ i, expresant-lo en funció de la longitud de l'entrada, tenim $\mathsf{t}_M(m) = \mathrm{O}(m \log m)$.

Observeu que hem considerat que accedir a un element de la taula té cost constant. De fet en una implementació amb TM no tenim acces directe i, per tant, s'hauria de afegir un factor addicional de $\mathrm{O}(m)$. Nosaltres no afegirem aquest factor, amb el benentès que, amb una TM, com a molt hi afegim un factor polinòmic en la longitud de l'entrada.

2.2 Una representació per a màquines de Turing

Sovint ens interessa resoldre problemes en els quals l'entrada és un programa, com fan per exemple, un compilador o un intèrpret. Una propietat dels llenguatges de programació és que un intèrpret per a un llenguatge es pot escriure en el mateix llenguatge de programació. El nostre interès és comprovar que aquest tipus de mecanismes també es poden plantejar sobre màquines de Turing i resoldre'ls algorísmicament també amb TM.

Abans de plantejar-nos la resolució d'algun d'aquests problemes, en què l'entrada o la sortida o un component d'elles és una TM, necessitem fer un primer pas que consisteix a comprovar que el conjunt de màquines de Turing és representable mitjançant una estructura de dades finita i, en definitiva, amb una cadena de caràcters, és a dir, un mot d'un alfabet.

Dissenyar una estructura de dades per representar una màquina de Turing és fàcil. Assumim que els estats estan numerats de 0 a n, que l'estat inicial és q_0 i que l'estat acceptador és q_n. Amb aquest conveni, per representar una màquina de Turing en què hi hagi definides m transicions utilitzem una taula de tuples, de dimensió $m \times 5$, de manera que en cada posició tenim emmagatzemat un tuple que representa una entrada de la funció de transició. La representació de la màquina de la figura 1.2 com a taula de tuples es troba a la figura 2.5.

A partir de l'estructura de dades podem definir una funció de codificació de TM en mots de l'alfabet $\{0, 1\}^*$. Per fer-ho cal definir funcions de codificació per a cadascun dels *tipus* que apareixen a l'estructura de dades. Un estat està representat pel seu número expressat en binari. Per representar els alfabets, necessitem saber el nombre de símbols de cadascun d'ells i assignar-los codis binaris. Si Γ té g símbols i Σ té s símbols, $g \geqslant s + 2$ i podem utilitzar els primers g mots de longitud k, per a un valor k tal que $2^{k-1} < g \leqslant 2^k$. Els símbols de Σ correspondran als primers s codis, i `b` i $\blacktriangleright$ als dos últims. Tal com hem codificat els alfabets, ens cal guardar les seves talles per tal de poder reconstituir els components de la TM a partir de la seva codificació. Finalment, per codificar els moviments utilitzem tres codis binaris: 10 (`e`), 01 (`d`), 00 (`n`).

0	1	0	1	d
0	0	1	0	d
0	b	3	0	e
1	0	1	0	d
1	1	1	1	d
1	b	2	b	e
2	1	2	0	e
2	0	4	1	n
3	1	3	0	e
3	▶	4	▶	d

```
#10#100##0#01#0#01#01#
           #0#00#1#00#01#
          #0#10#11#00#10#
           #1#00#1#00#01#
           #1#01#1#01#01#
          #1#10#10#10#10#
         #10#01#10#00#10#
        #10#00#100#01#00#
         #11#01#11#00#10#
       #11#11#100#11#01##
```

Fig. 2.5: *Representacions de la* TM *de la figura 1.2*

Com que l'estructura de dades és una taula, podem definir una funció de codificació en la forma que ja hem vist; utilitzem un nou símbol #, diferent de 0 i de 1, i codifiquem la taula i les talles dels dos alfabets així:

$$\texttt{\#}s\texttt{\#}g\texttt{\#\#}\langle \text{ fila } 1 \rangle\texttt{\#\#}\cdots\texttt{\#\#}\langle \text{ fila } m \rangle\texttt{\#\#}.$$

Els cinc components de cada fila estan separats per #. A la figura 2.5 es dóna el mot que codifica la TM de la figura 1.2.

Aquesta codificació compleix els requeriments d'una bona codificació.

Exercici 2.4 Escriviu algorismes detallats que permetin fer la codificació i la descodificació d'una TM.

Exercici 2.5 Construïu una TM per determinar si un mot és o no una codificació vàlida d'alguna TM.

Amb aquesta representació (o amb una altra) podem considerar el conjunt de TM com un conjunt de dades, que pot constituir part de l'entrada i/o sortida de la definició d'un problema. A partir d'ara assumirem que tenim definida una representació per TM i la corresponent codificació. El que acabem de veure és un exemple de com fer-ho, però tot el tractament que fem és independent de quina sigui la codificació, sempre que tinguem els algorismes corresponents per codificar, descodificar i verificar. A més, donada $w \in \Sigma^*$, utilitzem la següent notació:

- M_w representa la màquina de Turing amb codificació w.
- φ_w representa la funció computada per M_w.
- L_w representa el llenguatge reconegut per M_w.

La notació anterior presenta un problema quan w no és una codificació d'una màquina de Turing. En aquest cas, podem fer servir el següent conveni: si w no és la codificació de cap màquina, M_w és la TM que té un únic estat i no té transicions definides. Observeu que podem fer ús de la TM de l'exercici 2.5 per comprobar si w és o no una codificació vàlida de TM.

EXERCICI 2.6 Construïu una TM que, amb entrada w, calculi la codificació de la màquina C_w, definida a la secció 2.1, que escriu la constant w.

Una vegada tenim una forma de codificar TM, podem associar un llenguatge a cadascun dels problemes decisionals sobre TM presentats al capítol 1.

- Problema de l'Aturada

$$\text{HALT} = \{\langle w, v\rangle \mid \mathsf{M}_w(v)\downarrow\}.$$

- Problema de la Pertinença

$$\text{PERT} = \{\langle w, v\rangle \mid v \in \mathsf{L}_w\}.$$

- Problema de l'Equivalència

$$\text{TM-EQUIV} = \{\langle w, v\rangle \mid \mathsf{L}(\mathsf{M}_w) = \mathsf{L}(\mathsf{M}_v)\}.$$

També considerarem el problema següent:

Autoaturada (K)
Donada una TM, determinar si s'atura quan té com a entrada la seva codificació.

El llenguatge associat és

$$\text{K} = \{w \mid \mathsf{M}_w(w)\downarrow\}.$$

També podem associar un gràfic als problemes funcionals

- Problema de la Minimització

$$\text{TM-MINQ} = \{(u, v) \mid \mathsf{L}(\mathsf{M}_u) = \mathsf{L}(\mathsf{M}_v) \wedge \forall w \in \Sigma^* \; \mathsf{L}(\mathsf{M}_u) = \mathsf{L}(\mathsf{M}_w) \Rightarrow \|\mathsf{Q}_w\| \geqslant \|\mathsf{Q}_v\|\},$$

on Q_x representa el conjunt d'estats de la TM M_x.

2.3 Intèrprets i simuladors

Una de les característiques fonamentals dels llenguatges de programació és l'automatització de l'execució d'un programa utilitzant verificadors sintàctics i intèrprets. En el nostre model de màquina, un *verificador sintactic* és una TM que, amb un mot $w \in \{0,1\}^*$ com a entrada, determina si w és una codificació vàlida d'alguna TM. L'exercici 2.5 ha consistit en la construcció d'un verificador sintàctic per a la codificació presentada a la secció precedent.

El segon mecanisme, el d'interpretació, és fonamental per a l'execució dels programes. Sense ell, canviar la programació d'un ordinador implicaria canviar algun dels seus components, algun circuit, com passava amb els primers computadors. En la terminologia del model de la màquina de Turing, la noció d'intèrpret correspon a la de *màquina universal*.

Vegem ara com podem dissenyar una TM que simula el comportament de qualsevol màquina de Turing amb una entrada donada. L'entrada per a aquesta màquina serà un parell format per una TM M, representada per un mot, i un mot de l'alfabet d'entrada de M. Assumim que la codificació de TM es fa amb mots de $\{0,1\}^*$ i que totes les TM tenen un mateix alfabet d'entrada, $\{0,1\}$. Per a la nova màquina, l'alfabet d'entrada és $\{0,1\}$. Fem la construcció en fases; primer veurem com es pot simular una pas de càlcul.

PROPOSICIÓ 2.1 *Existeixen TM d'aturada segura que, amb entrada $\langle w, v\rangle$, on $v = \alpha q \beta$ és una configuració instantània de la màquina M_w, decideixen, respectivament, les propietats*

a) v és una configuració acceptadora.

b) v és una configuració terminal.

DEMOSTRACIÓ Ometem la fase inicial en què la màquina comprova que el primer component de l'entrada és una codificació vàlida d'una màquina de Turing i que el segon component és una configuració vàlida per M_w, i només donem l'esquema per comprovar-ne les condicions.

Donem l'esquema d'una TM amb tres cintes. La cinta 1 actua com a cinta d'entrada i conté la codificacó de la TM i la configuració. La cinta 2 es fa servir per emmagatzemar la codificació de la màquina M_w en $\{0, 1, \texttt{\#}\}$, tal com s'ha descrit a la secció 2.2. La cinta 3 la farem servir com a cinta auxiliar. Per comprovar que la configuració sigui acceptadora:

1. Obtenir w i escriure-la a c2.
2. Obtenir l'estat acceptador de M_w i escriure-lo a c3.
3. Comparar el contingut de c3 amb l'estat a v. Si són iguals la condició és certa i si no serà falsa.

Per comprovar que la configuració sigui terminal:

1. Copiar w a c2.

2. Cercar l'estat a v, i copiar $q\#a$ en c3, essent a la primera letra de β o un b si $\beta = \lambda$.
3. Si el contingut de c3 ($q\#a$) apareix en c2 en una posició que correspon a la part esquerra d'una transició, llavors sabem que la configuració no és terminal. En cas contrari la configuració és terminal.

□

La proposició prèvia garanteix la decidibilitat de dues propietats que farem servir sovint en construccions algorísmiques.

- x és acceptadora per a M.
- x és terminal per a M.

on x representa una configuració per la TM M.

La construcció següent ens proporciona una forma sistemàtica de simular un pas d'una TM.

PROPOSICIÓ 2.2 *Existeix una màquina de Turing* PAS *d'aturada segura, que, amb entrada $\langle w, v\rangle$, on $v = \alpha q \beta$ és una configuració no terminal de la màquina M_w, calcula la configuració següent de v en M_w.*

DEMOSTRACIÓ La construcció segueix el mateix esquema que les TM de la proposició 2.1; tot i que hi afegirem una quarta cinta que ens servirà com a cinta de sortida.

Una vegada hem arribat al punt en què sabem que la configuració no és terminal, tenim seleccionada en c1 la transició corresponent, i només ens falta simular la transició per obtenir la nova configuració. Hem de tenir en compte que entre dues configuracions consecutives canvien, com a molt, tres símbols consecutius, que inclouen el símbol d'estat.

La nostra màquina ha de copiar la part inicial, canviar els símbols que calgui i copiar (si n'hi ha) la part final de la configuració. □

La màquina PAS dóna lloc a la funció

- $y :=$ configuració-següent(M, x).

que utilitzarem en les construccions algorísmiques.

La màquina universal, l'intèrpret, és una petita variant de la màquina del teorema precedent; només hi hem d'afegir una iteració.

TEOREMA 2.3 (Màquina universal) *Existeix una màquina de Turing* UNIV *que, amb entrada $\langle w, v\rangle$, simula el càlcul de M_w amb entrada v.*

DEMOSTRACIÓ Construïm una màquina universal amb les mateixes característiques que les TM precedents. La simulació es fa pas a pas, utilitzant la cinta 2 per mantenir la configuració actual. Per tal de finalitzar correctament cada iteració, hem de modificar la màquina PAS de manera que després d'un pas la nova configuració quedi a c2. A més, no hem de tornar a repetir la fase inicial que separa la codificació de la TM i la de la seva entrada i, en canvi, hem d'afegir el càlcul de la configuració inicial. L'esquema de la simulació és el següent:

entrada $\langle w, v\rangle$
$x :=$ configuració-inicial(M_w, v);
mentre x no és terminal per a M_w **fer**
    $x :=$ configuració-següent(M_w, x)
fi mentre;
escriure x.

$\square$

Observem que el nombre de passos que fa la TM PAS per simular un pas d'una altra TM és lineal en funció de la descripció de M_w i que, per tant, la simulació del càlcul d'una TM a partir d'una configuració només produeix un factor multiplicatiu constant del nombre de passos simulats.

Al llarg del text fem ús de la TM universal; per referir-nos-hi fem servir l'expressió algorísmica

- simular M amb entrada v

Quan acaba la simulació (si ho fa), tenim tota la informació sobre la configuració terminal i, en conseqüència, podem demanar per condicions com

- M accepta v

o fer assignacions com

- $y := M(v)$.

Una vegada tenim una TM que és capaç de simular un pas, podem dissenyar una altra TM que pot simular un cert nombre de passos. En aquest cas, l'entrada és un triple format per una TM, un mot d'entrada i el nombre t de passos que volem simular. La TM computa la configuració exactament després de t passos, o la configuració terminal a què s'arriba en, com a molt, t passos. Hi afegim el temps amb què s'ha arribat a aquesta configuració.

entrada $\langle w, v, t\rangle$
$x :=$ configuració-inicial(M_w, v);
$tt := 0$;
mentre $tt \leqslant t \ \wedge\ x$ no és terminal per a M_w **fer**
    $x :=$ configuració-següent(M_w, x);
    $tt := tt + 1$
fi mentre;
escriure $\langle x, tt\rangle$.

Aquesta TM és d'aturada segura; a més ens dóna informació sobre la configuració després de t passos o la configuració final a què s'ha arribat. En conseqüència podem afegir les construccions algorísmiques

- M accepta v en t o menys passos.
- M amb entrada v s'atura en t o menys passos.

Vegem ara alguns exemples d'utilització de les màquines introduïdes per obtenir d'altres TM.

EXEMPLE 2.9 Afegint un comptador de passos, podem utilitzar l'esquema de la TM universal per, donats w i v, computar la funció $\mathsf{T}(\mathsf{M}_w, v)$ definida a la secció 1.3.

> **entrada** $\langle w, v\rangle$
> $x :=$ configuració-inicial(M_w, v);
> $t := 0$;
> **mentre** x no és terminal per a M_w **fer**
> $x :=$ configuració-següent(M_w, x);
> $t := t + 1$
> **fi mentre**;
> **escriure** t.

EXEMPLE 2.10 Una construcció tradicional és la de la funció rellotge. Fem rellotge$(\langle w, v, t\rangle) = 0$ si M_w amb entrada v no s'ha aturat després de t passos i rellotge$(\langle w, v, t\rangle) = 1\$\mathsf{M}_w(v)$ si M_w amb entrada v s'ha aturat en t o menys passos, on $w, v \in \Sigma^*$ i $t \in \mathbb{N}$. L'esquema precedent es pot adaptar fàcilment, canviant la sortida.

> **funció** rellotge$(\langle w, v, t\rangle)$
> $x :=$ configuració-inicial(M_w, v);
> $tt := 0$;
>   **mentre** $tt \leqslant t \wedge x$ no és terminal per a M_w **fer**
> $x :=$ configuració-següent(M_w, x);
> $tt := tt + 1$
>   **fi mentre**;
>   **si** x és terminal per a M_w
>   **llavors retorna** $1\$\mathsf{M}_w(v)$
>   **altrament retorna** 0
>   **fi si**
> **fi funció**.

2.4 La tesi de Church-Turing

Al llarg dels capítols 1 i 2 hem justificat que la TM proporciona totes les funcionalitats que es poden esperar d'un algorisme. En concret, hem vist que:

- És un model robust, en el sentit que aparents millores no porten a cap modificació de potència de càlcul.
- Ens permet implementar tots els esquemes algorísmics bàsics que s'inclouen a la definició de qualsevol llenguatge de programació i, a més, es poden implementar els mecanismes bàsics de compilació i interpretació.
- Hem vist com incorporar valors constants i com realitzar operacions aritmètiques bàsiques, com ara sumar (o restar) una unitat a un natural, o multiplicar i dividir per dos. Els esquemes d'aquest capítol ens permeten implementar amb TM tota l'aritmètica, a partir d'aquestes operacions.

Aquestes consideracions justifiquen la utilització de la TM com a model formal d'algorisme. Aquesta idea es coneix com a

Tesi de Church-Turing
La noció intuïtiva d'algorisme es correspon amb la de màquina de Turing.

Aquesta tesi no és un teorema, en el sentit que no és objecte de demostració, ja que el concepte d'algorisme no es pot definir formalment. Una vegada admesa la tesi, no fem cap distinció entre els termes *algorisme* i *màquina de Turing*, i sovint descrivim una màquina de Turing mitjançant un algorisme.

Les raons que hem presentat justifiquen la decisió d'admetre el model, però no són les úniques que avalen aquesta tesi. La raó fonamental, tal com ja es va apuntar al capítol 1, és l'equivalència del model de Turing amb altres models proposats.

Entre les primeres propostes de caracterització del concepte d'algorisme es troben la formalització de les *funcions recursives* en termes de *λ-càlcul* per A. Church [11], la recursivitat de S. C. Kleene [24], la màquina de Turing descrita a [40] i un model de màquina molt similar a la de Turing, proposat per E. Post [29] el mateix any que A. Turing. Altres models equivalents proposats es basen en conceptes de programació més moderns, com ara programes imperatius (simplificacions de llenguatges d'alt nivell), màquines RAM (llenguatges de baix nivell) o circuits booleans. Remetem el lector la referència [34] per a un tractament amb profunditat dels diferents models de computació.

2.5 Problemes

2.1 Trobeu màquines de Turing que calculin les funcions següents:

1. El producte de dos naturals donats en binari
2. La divisió entera i el residu
3. L'n-èsim nombre de Fibonacci

2.2 Sigui M la TM següent:

```
var
    t : taula [0..N)[0..N) de booleà;
    c : taula [0..N) de booleà;
    es : booleà;
    size, max, i, j : nat
fi var;
entrada t;
c := (0, ..., 0);
max := 0;
mentre c ≠ (1, ..., 1) fer
    c := següent(c);
    size := 0;
    es := cert;
    per i := 0 fins N − 1 fer
        si c[i] llavors size := size + 1 fi si;
        per j := i + 1 fins N − 1 fer
            si c[i] ∧ c[j] llavors es := es ∧ t[i, j] fi si
        fi per
    fi per;
    si es ∧ (size > max) llavors max := size fi si
fi mentre;
retorna (max).
```

La funció següent d'una taula de booleans calcula el mot següent en ordre lexicogràfic.

1. Determineu la funció calculada per M, interpretant la taula t com la matriu d'adjacència d'un graf i la taula c com un subconjunt de vèrtexs del graf.
2. Trobeu una fita superior (tan ajustada com pugueu) del temps de càlcul de M.

2.3 Considereu els problemes enunciats al capítol **??**. Doneu algorismes per a ells i estimeu-ne el cost.

2.4 Formalitzeu els problemes següents:

1. Donada una màquina de Turing, determinar si el llenguatge reconegut és buit.
2. Donada una màquina de Turing, determinar si per a dues entrades diferents produeix la mateixa sortida.
3. Donada una màquina de Turing, determinar si per a alguna entrada dóna com a sortida un símbol diferent de blanc.
4. Donada una màquina de Turing, determinar si té un punt fix (és a dir, si per a alguna entrada dóna com a sortida ella mateixa).
5. Donada una màquina de Turing, determinar si el llenguatge reconegut és finit.
6. Donades dues màquines de Turing, determinar si computen la mateixa funció.

7. Donada una màquina de Turing, determinar si s'atura en llegir una cinta en blanc.
8. Donats una màquina de Turing, un mot d'entrada i un natural n, determinar si la màquina utilitza n o més cel·les en processar l'entrada.
9. Donats una màquina de Turing, un estat q i una entrada w, determinar si la màquina arriba a q a partir de la configuració inicial per a w.
10. Donada una màquina de Turing, determinar si té menys de 10 estats.
11. Donada una màquina de Turing, determinar si hi ha una altra màquina de Turing que computi la mateixa funció i que tingui el mateix alfabet de cinta, però que tingui menys de 10 estats.
12. Donada un màquina de Turing, determinar si hi ha una altra màquina de Turing que computi la mateixa funció.
13. Donada una màquina de Turing M, determinar si hi ha dues màquines de Turing N i O tals que
$$\mathsf{L}(N) \subsetneq \mathsf{L}(M) \subsetneq \mathsf{L}(O).$$

3 Computabilitat de funcions i decidibilitat de llenguatges

Als capítols anteriors hem introduït la màquina de Turing com un model general de computació i hem enunciat la tesi de Church-Turing. Acceptant aquesta tesi, obtenim una definició formal d'algorisme. En aquesta situació ja podem caracteritzar la classe de problemes que es poden resoldre algorísmicament.

En aquest capítol caracteritzem la classe de problemes *funcionals* que es poden resoldre algorísmicament i la classe de problemes *decisionals* que es poden resoldre algorísmicament. Un algorisme que resolgui un problema decisional, davant d'una entrada x, ha de poder determinar si x satisfà o no la propietat que defineix el problema. Tant en el cas dels problemes funcionals com en el dels decisionals, mostrarem que hi ha problemes que no tenen cap solució algorísmica. També veurem problemes decisionals que poden ser decidits *parcialment* per un algorisme. En aquest cas, l'algorisme, davant d'una entrada que satisfà la propietat que defineix el problema, s'atura i accepta. Si l'entrada no satisfà la propietat, aleshores l'algorisme pot no aturar-se. Veurem que problemes ben naturals com, per exemple, decidir si un programa que s'executa sobre una entrada retorna un determinat valor, són parcialment decidibles, però en canvi no són decidibles.

3.1 Computabilitat de funcions

Recordem que una funció s'anomena computable quan existeix una màquina de Turing que la computa. Si una funció computable f és definida sobre x, aleshores la màquina M que computa f, amb entrada x, ha de retornar $f(x)$. Si la funció f no és definida sobre x, aleshores M amb aquesta entrada no s'atura.

Com ja s'ha vist al capítol 2, totes les funcions aritmètiques senzilles són computables. A l'exemple que segueix dissenyem una TM per a la funció suma.

EXEMPLE 3.1 Sigui f la funció suma, $f(\langle x, y\rangle) = x + y$ per a tot $x, y \in \mathbb{N}$. Podem construir una TM amb cinc cintes c1, c2, c3, c4 i c5 que computi f. Considerem que c1 és la cinta d'entrada, c2, c3 i c4 són cintes de treball de manera que, a c2 es copiarà x, a c3 es copiarà y, a c4 es deixarà el resultat de la suma (en forma revessada). Finalment, s'escriurà el resultat a c5, la cinta de sortida. Suposem que $\langle x, y\rangle = x\texttt{\#}y$ on $x, y \in \{0\} \cup 1\{0, 1\}^*$.

1. Mentre el capçal 1 llegeix 0 o 1 copiar de c1 a c2; avançar cap a la dreta els capçals 1 i 2.

2. Si el capçal 1 llegeix #, aleshores avançar una posició cap a la dreta.

3. Mentre el capçal 1 llegeix 0 o 1 copiar de c1 a c3; avançar cap a la dreta els capçals 1 i 3.

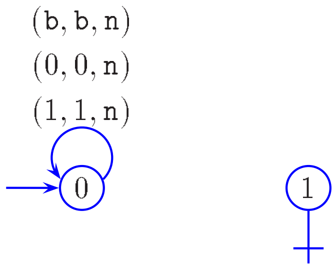


Fig. 3.1: *Màquina que computa la funció buida amb alfabet d'entrada* $\{0, 1\}$

4. Retrocedir alhora els capçals 2 i 3.

5. Inicialitzar el bit de transport (*carry*) a 0. (Els estats de la TM "recorden" si s'ha de propagar el transport o no).

6. Mentre els capçals 2 i 3 no llegeixin ▶:

    6.1. Sumar els bits llegits pels capçals 2 i 3, juntament amb el bit de transport; escriure el bit de menys pes de la suma a c4; actualitzar el transport amb el bit de més pes.

    6.2. Retrocedir els capçals 2 i 3; avançar el capçal 4.

7. Mentre el capçal 2 (3) llegeix ▶ i el capçal 3 (2) llegeix 0 o 1:

    7.1. Sumar el bit llegit pel capçal 3 (2) amb el bit de transport; escriure el bit de menys pes de la suma a c4; actualitzar el transport amb el bit de més pes.

    7.2. Retrocedir el capçal 3 (2); avançar el capçal 4.

8. Si el bit de transport és 1, aleshores escriure 1 a c4; altrament, retrocedir el capçal 4.

9. Mentre el capçal 4 llegeix 0 o 1, copiar de c4 a c5; retrocedir cap a l'esquerra el capçal 4 i avançar cap a la dreta el capçal 5. (Copiar a c5 el revessat del contingut de c4).

Considerem ara la funció buida (vegeu-ne la definició a la secció **??**). Aquesta funció té un interès teòric perquè apareix en moltes demostracions de teoremes. És fàcil veure que la funció buida és computable.

EXEMPLE 3.2 La TM M de la figura 3.1 computa la funció buida sobre el conjunt de partida $\{0, 1\}^*$; per a qualsevol entrada x, $M(x)\uparrow$.

EXERCICI 3.1 Demostreu que les funcions següents, en què $x, y \in \mathbb{N}$, són computables:

$$
\begin{aligned}
f(x) &= \lceil \mathsf{log}\ x \rceil \\
f(\langle x, y \rangle) &= x * y \\
f(\langle x, y \rangle) &= x \ \mathsf{mod}\ y \\
f(\langle x, y \rangle) &= \langle \mathsf{min}\ (x, y), \mathsf{max}\ (x, y) \rangle \\
f(x) &= \begin{cases} 1 & \text{si } x \text{ és primer} \\ 0 & \text{si no} \end{cases} \\
f(x) &= 2^x \\
f(x) &= x! \\
f(x) &= 2^{2^x}
\end{aligned}
$$

Els exemples anteriors consisteixen bàsicament en funcions totals les entrades de les quals són codificacions de nombres naturals. També podem considerar funcions en què alguns dels seus components d'entrada són descripcions de TM. Tot seguit presentem alguns exemples d'aquests tipus de funcions.

El primer exemple consisteix a transformar la TM d'entrada en una altra màquina que ha de satisfer uns requeriments determinats.

EXEMPLE 3.3 En el model de TM que hem introduït, es considera que un mot w és acceptat per una màquina M quan M, amb entrada w, assoleix l'estat acceptador, que és de parada. Podríem adoptar un criteri d'acceptació diferent, en què es considera que un mot w és acceptat per una màquina N quan N, amb entrada w, assoleix l'estat acceptador, que també és de parada, i el capçal ha d'estar posicionat a l'extrem esquerre de la cinta. Tot seguit veurem que qualsevol llenguatge reconegut per una TM pot ser reconegut per una TM amb el nou criteri d'acceptació. Donada una màquina M, transformarem M en una màquina N de manera que, per a tot $w \in \Sigma^*$, M accepta w si i només si N accepta w segons el nou criteri acceptador. Una transformació possible consisteix a afegir un nou estat, que és el nou estat acceptador, i modificar la funció de transició per tal que, en el moment en què s'assoleix l'antic estat acceptador q_{F}, el capçal retrocedeixi fins al primer símbol de la cinta i s'aturi en el nou estat acceptador $q_{\mathrm{F}'}$. D'aquesta manera, a partir de $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$ podem definir $N = (Q', \Sigma, \Gamma', \delta', q_0, q_{\mathrm{F}'})$ on

1. $\Gamma' = \Gamma$.
2. $Q' = Q \cup \{q_{\mathrm{F}'}\}$ i $q_{\mathrm{F}'}$ és el nou estat acceptador.
3. La funció de transició és la de M, afegint les transicions
   $\delta'(q_{\mathrm{F}}, a) = (q_{\mathrm{F}}, a, \mathtt{e})$, sempre que $a \neq \blacktriangleright$, i
   $\delta'(q_{\mathrm{F}}, \blacktriangleright) = (q_{\mathrm{F}'}, \blacktriangleright, \mathtt{n})$.

Tal com es construeix N, és fàcil veure que M, amb una entrada w qualsevol, s'atura en estat q_{F} si i només si N, amb entrada w, s'atura en estat $q_{\mathrm{F}'}$, i el capçal està posicionat sobre $\blacktriangleright$, sempre que M no pugui esborrar el símbol $\blacktriangleright$ d'inici de cinta. En cas que M pugui esborrar

$\blacktriangleright$, aleshores N ha de deixar igualment una marca a la primera cel·la de la cinta. Formalment, si existeix algun estat $q \in Q$ tal que $\delta(q, \blacktriangleright) = (p, a, m)$, aleshores N escriu una còpia a' de a a la primera cel·la. Per això s'han d'afegir a la definició de N les còpies dels símbols i també algunes transicions addicionals:

1. $\Gamma' = \Gamma \cup \{a' \mid a \in \Gamma\}$ de manera que, les còpies són símbols nous.
2. Les transicions que s'han d'afegir són
   $\delta'(q, \blacktriangleright) = (p, a', m)$, si $\delta(q, \blacktriangleright) = (p, a, m)$, per no perdre la marca d'inici de cinta,
   $\delta'(q, a') = (p, b', m)$, si $\delta(q, a) = (p, b, m)$, per mantenir la marca d'inici,
   $\delta'(q_{\mathrm{F}}, a) = (q_{\mathrm{F}}, a, \mathtt{e})$, sempre que $a \neq \blacktriangleright$ i $a \notin \Gamma' - \Gamma$,
   $\delta'(q_{\mathrm{F}}, \blacktriangleright) = (q_{\mathrm{F}'}, \blacktriangleright, \mathtt{n})$ i $\delta'(q_{\mathrm{F}}, a') = (q_{\mathrm{F}'}, a, \mathtt{n})$.

Notem que aquesta transformació de M en N és purament sintàctica, perquè la transformació es duu a terme sense fer cap consideració relativa als mots acceptats per M. Aleshores podem construir una TM que, amb una entrada x qualsevol, retorni l'índex y, que s'obté en transformar M_x tal com hem especificat:

> **entrada** x
> $(Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}}) := \mathsf{M}_x$;
> $Q' := Q \cup \{q_{\mathrm{F}'}\}$;
> $\delta' := \delta \cup \{(q_{\mathrm{F}}, a, q_{\mathrm{F}}, a, \mathtt{e}) \mid a \neq \blacktriangleright\} \cup \{(q_{\mathrm{F}}, \blacktriangleright, q_{\mathrm{F}'}, \blacktriangleright, \mathtt{n})\}$;
> $y := \mathsf{gödel}(Q', \Sigma, \Gamma, \delta', q_0, q_{\mathrm{F}'})$;
> **escriure** y.

Recordeu que a la secció **??** hem vist que, donat el número de Gödel x, podem obtenir de manera algorísmica la definició de la TM M_x corresponent. Aleshores, la primera línia d'aquest programa consisteix a obtenir les components de M_x. Utilitzem una assignació múltiple en comptes de fer tantes assignacions com elements té el tuple, per tal de presentar d'una manera més succinta l'especificació del programa.

Més endavant, al capítol 4, veurem més exemples de funcions computables en què es calculen transformacions de TM. Considerem ara alguns exemples de funcions computables però no totals, en què també alguna de les seves components d'entrada és el número de Gödel d'una màquina de Turing.

EXEMPLE 3.4 La funció definida a continuació és computable.

$$f_{\mathrm{HALT}}(\langle x, y\rangle) = \begin{cases} 1 & \text{si } \mathsf{M}_x(y)\downarrow \\ \text{indefinida}, & \text{altrament.} \end{cases}$$

Podem construir una TM M que, amb entrada $\langle x, y\rangle$, simuli la màquina M_x amb entrada y. Si aquesta s'atura, aleshores M ha de retornar el valor 1. Si M_x amb entrada y no s'atura, aleshores M amb entrada x tampoc no s'aturarà.

> M : **entrada** $\langle x, y\rangle$
> simular M_x amb entrada y;
> **escriure** 1.

És clar que M computa la funció f_{HALT}. Observeu que aquesta funció no és total. Si x és el número de Gödel d'una màquina que computa la funció buida, aleshores la funció f_{HALT} està indefinida en $\langle x, y\rangle$, per a qualsevol y. Fixeu-vos que la simulació de M_x en aquest cas no finalitza i per tant, M no s'atura.

EXERCICI 3.2 Demostreu que les funcions següents són computables i no són totals:

$$f_{\text{K}}(x) = \begin{cases} 1 & \text{si } \mathsf{M}_x(x)\downarrow \\ \text{indefinida,} & \text{altrament} \end{cases}$$

$$f_{\text{PERT}}(\langle x, y\rangle) = \begin{cases} 1 & \text{si } y \in \mathsf{L}(\mathsf{M}_x) \\ \text{indefinida,} & \text{altrament.} \end{cases}$$

Els dominis de les funcions $f_{\text{HALT}}, f_{\text{K}}$ i f_{PERT} coincideixen amb els llenguatges HALT, K i PERT, respectivament. Si considerem la prolongació de cadascuna d'aquestes funcions a una funció total, de manera que aquesta retorni 0 per als elements que no pertanyen al domini de la funció de partida, aleshores obtenim les funcions característiques dels llenguatges HALT, K i PERT. Aquests són els primers exemples de funcions no computables que presentem. Totes aquestes funcions comparteixen la particularitat següent: l'argument de la funció (o almenys un dels arguments de la funció) és interpretat com un número de Gödel i la propietat que defineix la funció depèn de certes propietats de la computació associada a la TM que té aquest número de Gödel. Aquestes funcions, juntament amb algunes eines que introduirem al capítol 4, ens permetran demostrar que problemes ben naturals com, per exemple, el problema de decidir si dues gramàtiques incontextuals són equivalents, o el problema de determinar si una gramàtica incontextual és ambigua, no es poden resoldre algorísmicament.

Per demostrar que $\mathcal{X}_{\text{K}}, \mathcal{X}_{\text{HALT}}$ i $\mathcal{X}_{\text{PERT}}$ no són computables considerem primer una funció que s'obté per aplicació de la tècnica de diagonalització de Cantor. A la proposició següent donem la definició formal d'aquesta funció i demostrem que aquesta no pot formar part de la llista de funcions computables.

PROPOSICIÓ 3.1 *La funció*

$$f(x) = \begin{cases} 1 & \textit{si } \mathsf{M}_x(x)\uparrow \\ \textit{indefinida,} & \textit{altrament} \end{cases}$$

no és computable.

DEMOSTRACIÓ Suposem que f fos una funció computable. Aleshores existiria un natural i tal que $\varphi_i = f$. Considerem ara f amb entrada i. D'una banda, tenim que si $i \in \mathsf{Dom} f$ aleshores $f(i) = 1$. Però, per la definició de f, tenim que $\mathsf{M}_i(i)\uparrow$ i, per tant, $i \notin \mathsf{Dom}\varphi_i = \mathsf{Dom} f$. D'altra banda, si $i \notin \mathsf{Dom} f$ aleshores, per la definició de f, $\mathsf{M}_i(i)\downarrow$. Amb tot això tenim ($i \in \mathsf{Dom} f \iff i \notin \mathsf{Dom} f$), que és una contradicció. □

Amb aquest resultat és fàcil veure que la funció característica de K no és computable.

TEOREMA 3.2 *La funció característica de* K *no és computable.*

DEMOSTRACIÓ Sigui $\mathcal{X}_{\mathrm{K}}$ la funció característica de K.

$$\mathcal{X}_{\mathrm{K}}(x) = \begin{cases} 1 & \text{si } \mathsf{M}_x(x)\downarrow \\ 0, & \text{altrament.} \end{cases}$$

Suposem que $\mathcal{X}_{\mathrm{K}}$ sigui computable. Considerem ara la funció f que hem definit al teorema anterior. Aleshores podem dissenyar una TM M que computi f utilitzant la màquina que computa $\mathcal{X}_{\mathrm{K}}$.

```
M : entrada x
    y := X_K(x);
    si y = 0
       llavors escriure 1
       altrament
         mentre cert fer
         fi mentre
    fi si.
```

És clar que M computa exactament la funció f. També és clar que és ben definida, perquè hem suposat que $\mathcal{X}_{\mathrm{K}}$ és computable. D'aquesta manera, obtenim que f també és computable. Aquesta contradicció invalida la suposició que $\mathcal{X}_{\mathrm{K}}$ sigui computable. □

Ara ja estem en condicions de demostrar que $\mathcal{X}_{\mathrm{HALT}}$ i $\mathcal{X}_{\mathrm{PERT}}$ tampoc no són computables.

COROL·LARI 3.3 *Les funcions característiques de* HALT *i* PERT *no són computables.*

DEMOSTRACIÓ La funció característica de K es pot veure com un cas particular de la funció característica de HALT, ja que, per a tot x, $x \in \mathrm{K}$ si i només si $\langle x, x\rangle \in \mathrm{HALT}$. Per tant, si la funció característica de HALT fos computable, aleshores la de K també ho seria.

Per demostrar que $\mathcal{X}_{\mathrm{PERT}}$ no és computable fem també un raonament a l'absurd. Suposem que $\mathcal{X}_{\mathrm{PERT}}$ fos computable. Definim la funció f com $f(\langle x, y\rangle) = \langle x', y\rangle$, en què $\mathsf{M}_{x'}$ s'obté a partir de M_x afegint a les transicions de parada una transició a l'estat acceptador. No és difícil veure que aquesta funció és computable i total.

Per la definició de $\mathsf{M}_{x'}$ tenim que $y \in \mathsf{L}(\mathsf{M}_{x'}) \Longleftrightarrow \mathsf{M}_x(y)\downarrow$.

Aleshores podem expressar la funció característica de HALT com a composició de la funció característica de PERT i la funció f: per a tot x, y, $\mathcal{X}_{\mathrm{HALT}}(\langle x, y\rangle) = \mathcal{X}_{\mathrm{PERT}}(f(\langle x, y\rangle))$. Com que les dues funcions són computables, aleshores $\mathcal{X}_{\mathrm{HALT}}$ seria computable, i d'aquesta manera s'obté una contradicció. Per tant, $\mathcal{X}_{\mathrm{PERT}}$ no és computable. □

Acabem aquesta secció posant un exemple d'una funció computable no total que no es pot prolongar a cap funció computable total.

EXEMPLE 3.5 Definim la funció següent:

$$f(x) = \begin{cases} t & \text{si } \mathsf{M}_x \text{ amb entrada } x \text{ s'atura en } t \text{ passos,} \\ \text{indefinida} & \text{si } \mathsf{M}_x(x)\uparrow. \end{cases}$$

Recordem que a la secció 2.3 hem demostrat que la funció $\mathsf{T}(M, w)$, que calcula el nombre de passos que efectua una màquina M amb entrada w, és computable. Considerem la TM següent:

> **entrada** x
> $y := \mathsf{T}(\mathsf{M}_x, x)$;
> **escriure** y.

És clar que aquesta TM computa exactament la funció f que acabem de definir. També és clar que és ben definida, perquè les funcions T i M són computables. Així doncs, f també és computable.

Observem que el domini de definició de f és $\mathsf{Dom}f = \{x \mid \mathsf{M}_x(x)\downarrow\} = \mathrm{K}$. ¿Existeix alguna prolongació total de f que sigui computable? Vegem que no. Sigui g una prolongació total de f. Si g fos computable, existiria una TM N que computaria g. Aquesta màquina seria d'aturada segura, perquè g és una funció total. Amb l'ajut de N podríem construir una altra màquina N' que computaria $\mathcal{X}_{\mathrm{K}}$, la funció característica del conjunt K. En efecte, la descripció de N' seria:

> N' : **entrada** x
> simular N amb entrada x;
> $t := N(x)$;
> $c :=$ configuració-inicial(M_x, x);
> **mentre** $(t > 0) \wedge (c$ no és terminal per a $\mathsf{M}_x)$ **fer**
> $c :=$ configuració-següent(M_x, c);
> $t := t - 1$
> **fi mentre**;
> **si** $(t = 0) \wedge (c$ és terminal per a $\mathsf{M}_x)$
> **llavors escriure** 1
> **altrament escriure** 0
> **fi si**.

És clar que N' és d'aturada segura, perquè N també ho és. També és clar que N' computa $\mathcal{X}_{\mathrm{K}}$, ja que podem afirmar que $\mathsf{M}_x(x)\downarrow$ si i només si M_x amb entrada x s'atura en exactament $g(x)$ passos. En efecte, si $\mathsf{M}_x(x)\downarrow$ llavors $x \in \mathsf{Dom}f$. Com que g una prolongació de f, aleshores $f(x) = g(x)$. I per la definició de f tenim que M_x amb entrada x s'atura en $g(x)$ passos. En canvi, si $\mathsf{M}_x(x)\uparrow$ llavors M_x amb entrada x no s'atura en $g(x)$ passos. En conseqüència, la condició avaluada per N' és precisament $\mathsf{M}_x(x)\downarrow$. Per això, $\mathsf{f}_{N'} = \mathcal{X}_{\mathrm{K}}$.

Com que al teorema 3.2 hem demostrat que $\mathcal{X}_{\mathrm{K}}$ no és computable, la màquina N' no pot existir. Per tant, la màquina N tampoc. Així doncs, cap prolongació total g no és computable.

3.2 Decidibilitat de llenguatges

En aquesta secció tractarem els problemes decidibles i en veurem alguns exemples. A continuació demostrarem l'existència de problemes indecidibles i també en presentarem alguns exemples.

Recordem que un llenguatge s'anomena *decidible* quan existeix una màquina de Turing d'aturada segura que el reconeix. Representem per REC la classe dels llenguatges decidibles fent ús de la notació clàssica.

Vegem alguns exemples de llenguatges decidibles.

Exemple 3.6 Al capítol 1 hem dissenyat dues TM d'aturada segura que reconeixen els llenguatges $\{0^n1^n0^n \mid n > 0\}$ i $\{0^n1^{n^2} \mid n > 0\}$. Així doncs, aquests dos llenguatges són decidibles.

Exercici 3.3 Demostreu que tot llenguatge finit és decidible.

Exercici 3.4 Demostreu que els llenguatges següents són decidibles:

$$\begin{aligned}
\textsc{parells} &= \{x \mid x \text{ és parell}\}\\
\textsc{senars} &= \{x \mid x \text{ és senar}\}\\
\textsc{primes} &= \{x \mid x \text{ és primer}\}\\
\textsc{composites} &= \{x \mid x \text{ és compost}\}\\
\textsc{fa-pert} &= \{\langle M, y\rangle \mid M \text{ és un autòmat finit i } y \in \mathsf{L}(M)\}
\end{aligned}$$

Exemple 3.7 Els llenguatges següents són decidibles:

$$\begin{aligned}
\textsc{t-halt} &= \{\langle x, y, t\rangle \mid \mathsf{M}_x \text{ amb entrada } y \text{ s'atura en } t \text{ passos o menys}\}\\
\textsc{s-halt} &= \{\langle x, y, s\rangle \mid \mathsf{M}_x \text{ amb entrada } y \text{ s'atura i utilitza } s \text{ cel·les com a molt}\}
\end{aligned}$$

En el cas de T-HALT, podem construir una TM que, amb entrada $\langle x, y, t\rangle$, simuli t passos, com a màxim, de M_x amb entrada y. Pel que fa a S-HALT, podem construir una TM M que, amb entrada $\langle x, y, s\rangle$, en el cas pitjor ha de simular M_x amb entrada y durant tants passos com configuracions *diferents* es poden formar, considerant el nombre d'estats de M_x i el nombre s de cel·les que es poden utilitzar com a màxim. Si després d'aquest nombre de passos no s'han utilitzat les s cel·les, aleshores M_x amb entrada y segur que no utilitzarà més cel·les, ja que ha repetit configuració, ha entrat en un bucle. En aquest cas, l'entrada no pertany a S-HALT. Si abans d'exhaurir aquest nombre de passos M_x s'atura o bé utilitza més de s cel·les, aleshores M para la simulació. En el primer cas M acceptaria l'entrada, mentre que en el segon M rebutjaria.

Una TM d'aturada segura que reconeix un llenguatge L s'atura davant d'una entrada qualsevol i "decideix" si aquesta entrada pertany o no a L. En aquest sentit, introduïm la definició següent.

Definició Diem que un problema decisional és *decidible* quan el llenguatge associat al problema és decidible. En cas contrari diem que el problema és *indecidible*.

Així doncs, la classe de problemes decidibles representa els problemes que es poden resoldre algorísmicament.

Considerem ara aquells problemes tals que el llenguatge associat és reconegut per una TM (que pot no ser d'aturada segura).

Recordem que un llenguatge L s'anomena *enumerable recursivament* quan existeix una màquina de Turing que el reconeix. Representem per RE la classe dels llenguatges enumerables recursivament. Usem e.r. com a abreviació de 'enumerable recursivament'.

La classe de problemes associada als llenguatges e.r. conté els problemes que abans hem denominat *parcialment* decidibles. En principi, podem garantir que existeix un algorisme que, almenys per a les entrades que satisfan la propietat que defineix el problema, s'atura. Tot seguit veurem alguns exemples de llenguatges e.r.

EXEMPLE 3.8 $\textsc{k} = \{x \mid \mathsf{M}_x(x)\downarrow\}$ és e.r. Per demostrar-ho considerem la TM que definim a continuació:

M : **entrada** x
  simular M_x amb entrada x;
  acceptar.

Per la construcció de M tenim que, donat un x, $M(x)\downarrow$ quan $\mathsf{M}_x(x)\downarrow$ i, en aquest cas, M accepta x. En canvi, si $\mathsf{M}_x(x)\uparrow$, aleshores $M(x)\uparrow$. Per tant, $\mathsf{L}(M) = \textsc{k}$.

EXEMPLE 3.9 $\textsc{halt} = \{\langle x, y\rangle \mid \mathsf{M}_x(y)\downarrow\}$ és enumerable recursivament. Per demostrar-ho considerem la TM següent:

M : **entrada** $\langle x, y\rangle$
  simular M_x amb entrada y;
  acceptar.

Per la construcció de M tenim que, donat un parell $\langle x, y\rangle$ qualsevol, M accepta x si i només si $\mathsf{M}_x(y)\downarrow$. Per tant, $\mathsf{L}(M) = \textsc{halt}$.

EXEMPLE 3.10 $\textsc{pert} = \{\langle x, y\rangle \mid y \in \mathsf{L}(\mathsf{M}_x)\}$ és enumerable recursivament. Per demostrar-ho considerem la TM següent:

M : **entrada** $\langle x, y\rangle$
  simular M_x amb entrada y;
  si M_x accepta y
    llavors acceptar
    altrament rebutjar
  fi si.

Per la construcció de M tenim que, donat un parell $\langle x, y\rangle$ qualsevol, M amb entrada $\langle x, y\rangle$ s'atura si i només si M_x amb entrada y s'atura. Si M_x accepta x llavors M accepta $\langle x, y\rangle$. Si M_x no accepta y, llavors M no accepta $\langle x, y\rangle$. Per tant, $\mathsf{L}(M) = \textsc{pert}$.

Els llenguatges e.r. també es poden caracteritzar de la manera següent:

TEOREMA 3.4 (Teorema de la projecció) *Un llenguatge A és* enumerable recursivament *si i només si existeix un llenguatge* decidible B *que permet expressar A en la forma*

$$A = \{x \mid \exists y \ \langle x, y\rangle \in B\}.$$

DEMOSTRACIÓ Primer, vegem que la condició és necessària. Sigui A un llenguatge e.r. i sigui M una TM que el reconeix. Un mot x qualsevol pertany a A si i només si M amb entrada x s'atura i accepta. Formalment tenim

$$\begin{aligned} A &= \{x \mid M \text{ accepta } x\} \\ &= \{x \mid \exists y \ M \text{ accepta } x \text{ en } y \text{ passos}\}. \end{aligned}$$

Definim el llenguatge $B = \{\langle x, y\rangle \mid M \text{ accepta } x \text{ en } y \text{ passos}\}$. No és difícil veure que B és decidible. Podem construir una màquina que reconeix B de la manera següent:

```
N : entrada ⟨x, y⟩
    c := configuració-inicial(M, x);
    mentre (y > 0) ∧ ( c no és terminal per a M) fer
        c := configuració-següent(M, c);
        y := y − 1
    fi mentre;
    si (y = 0) ∧ ( c és acceptadora per a M)
      llavors acceptar
      altrament rebutjar
    fi si.
```

N amb entrada $\langle x, y\rangle$ simula pas a pas M amb entrada x, calculant la configuració corresponent a cada pas. Quan ja ha simulat y passos, comprova si la configuració és acceptadora. Si és així aleshores accepta; si no, rebutja. Per tant, N és d'aturada segura i accepta si i només si $\langle x, y\rangle \in B$.

Un cop definit B, observem que podem expressar A com

$$A = \{x \mid \exists y \ \langle x, y\rangle \in B\}.$$

Demostrem tot seguit que la condició és suficient. Sigui A un llenguatge que es pot expressar com $A = \{x \mid \exists y \langle x, y\rangle \in B\}$, en què B és un llenguatge decidible. Sigui N una TM d'aturada segura que reconeix B. Podem construir una TM que reconegui A, que consisteix a fer una cerca del mot y adequat. Es tracta de donar a y tots els possibles valors seguint l'ordre lexicogràfic per longituds i, per cada un, simular N amb entrada $\langle x, y\rangle$. Com que N és d'aturada segura, totes les simulacions convergeixen. El procés es deté si hi ha algun parell $\langle x, y\rangle$ que és acceptat per N. Altrament, continua indefinidament.

```
M : entrada x
    y := λ; trobat := fals;
```

mentre $\neg trobat$ **fer**
  simular N amb entrada $\langle x, y \rangle$;
  si N accepta $\langle x, y \rangle$
    llavors $trobat := \mathsf{cert}$
    altrament $y := \mathsf{següent}(y)$
  fi si
fi mentre;
acceptar.

La funció $\mathsf{següent}(y)$, que ha estat definida a l'exemple 1.3 , computa el mot successor de y en l'ordre lexicogràfic per longituds.

Aleshores M amb entrada x s'atura i accepta només quan existeix y tal que $\langle x, y \rangle \in B$. Amb això tenim que $A = \mathsf{L}(M)$ i, per tant, A és e.r. □

EXEMPLE 3.11 El llenguatge $L = \{x \mid \mathsf{Dom}(\varphi_x) \neq \varnothing\}$ és enumerable recursivament. Per demostrar-ho apliquem el teorema de la projecció. Observem que podem expressar L com

$$\begin{aligned} L &= \{x \mid \exists y \, \mathsf{M}_x(y)\downarrow\} \\ &= \{x \mid \exists y, t \;\; \mathsf{M}_x \text{ amb entrada } y \text{ s'atura en } t \text{ passos}\} \end{aligned}$$

Ara definim el llenguatge $B = \{\langle x, \langle y, t \rangle\rangle \mid \mathsf{M}_x$ amb entrada y s'atura en t passos$\}$. Per demostrar que B és decidible construïm la TM següent:

N : **entrada** $\langle x, \langle y, t \rangle\rangle$
  $c := \mathsf{configuració\text{-}inicial}(\mathsf{M}_x, y)$;
  mentre $(t > 0) \wedge (\, c$ no és terminal per a $\mathsf{M}_x)$ **fer**
    $c := \mathsf{configuració\text{-}següent}(\mathsf{M}_x, c)$;
    $t := t - 1$
  fi mentre;
  si $(t = 0) \wedge (\, c$ és terminal per a $\mathsf{M}_x)$
    llavors acceptar
    altrament rebutjar
  fi si.

N amb entrada $\langle x, \langle y, t \rangle\rangle$ simula pas a pas la màquina M_x amb entrada y, computant la configuració corresponent a cada pas. Si després de simular t passos la configuració actual és terminal, aleshores accepta. Si no, rebutja. Així doncs, N és una TM d'aturada segura i N accepta $\langle x, \langle y, t \rangle\rangle$ si i només si $\langle x, \langle y, t \rangle\rangle \in B$.

Amb l'ajut d'aquest llenguatge B, podem expressar L com

$$L = \{x \mid \exists u \;\; \langle x, u \rangle \in B\},$$

on u representa el parell $\langle y, t \rangle$ anterior. Així doncs, aplicant el teorema de la projecció, tenim que L és e.r.

Per les definicions de les classes REC i RE, REC és una subclasse de RE; veurem tanmateix que aquesta inclusió és estricta. Noteu que als exemples de llenguatges decidibles no hem inclòs cap dels llenguatges següents: K, HALT, PERT. Aquests poden ser reconeguts per una TM, però demostrarem que no poden ser-ho per cap TM d'aturada segura. Remarquem la diferència entre HALT d'una banda, i T-HALT, S-HALT (definits a l'exemple 3.7) de l'altra. Els dos últims són llenguatges decidibles ja que podem dissenyar una TM de parada segura per a cadascun d'aquests llenguatges. La propietat comuna a tots dos és que només cal fer una simulació "controlada" per poder determinar si una entrada pertany o no al llenguatge en qüestió. En canvi, per determinar si $\langle x, y\rangle$ pertany a HALT, no disposem de cap informació que, d'entrada, ens permeti fitar el nombre de passos de la simulació de M_x amb entrada y, ni tampoc el nombre de cel·les que es poden utilitzar.

Els primers exemples de llenguatges indecidibles els obtenim utilitzant una caracterització dels decidibles en termes de funcions computables. Més endavant, quan examinem les propietats de tancament de les classes RE i REC, veurem exemples de llenguatges no e.r.

Recordem que a cada llenguatge L li podem associar la seva funció característica $\mathcal{X}_L$. Si considerem un llenguatge decidible L, no és difícil veure que podem modificar la màquina de parada segura que el reconeix per obtenir una TM que computa $\mathcal{X}_L$. Només cal afegir les transicions necessàries per escriure un 1 a la cinta de sortida quan la TM original arribi a l'estat acceptador i per escriure un 0 quan arribi a un estat de parada no acceptador. I en el sentit invers, és a dir, si suposem que $\mathcal{X}_L$ és una funció computable, aleshores es pot modificar la TM que computa la funció per tal d'obtenir la màquina que reconeix L. Només cal garantir que un cop ha escrit 1 a la cinta de sortida i arriba a un estat de parada, aquest últim ha de ser l'estat acceptador. Si ha escrit un 0, aleshores l'estat de parada no pot ser l'estat acceptador. Amb aquestes indicacions és fàcil demostrar la proposició següent, els detalls de la qual els deixem al lector.

PROPOSICIÓ 3.5 *Un llenguatge L és decidible si i només si la seva funció característica, $\mathcal{X}_L$, és computable.*

Aquesta caracterització dels llenguatges decidibles, juntament amb els resultats obtinguts a la secció anterior, permeten obtenir els primers exemples de llenguatges no decidibles o, el que és el mateix, els primers exemples de problemes indecidibles.

COROL·LARI 3.6 *Els llenguatges* K, HALT *i* PERT *no són decidibles.*

Aquests llenguatges són enumerables recursivament; per tant, podem dir que la inclusió de REC a RE és estricta.

COROL·LARI 3.7 $\mathsf{REC} \subsetneq \mathsf{RE}$.

3.3 Propietats de tancament

Diem que una classe de llenguatges és tancada respecte d'una operació quan el resultat d'aplicar aquesta operació a llenguatges de la classe considerada és també un llenguatge de la mateixa classe. En aquesta secció estudiarem algunes de les propietats de tancament de les classes presentades en aquest capítol, respecte d'operacions sobre llenguatges.

3.3.1 Reunió i intersecció

TEOREMA 3.8 *Les classes* RE *i* REC *són tancades respecte de la reunió i la intersecció.*

DEMOSTRACIÓ Primer demostrarem que la classe REC és tancada respecte de la reunió. REC està definida com el conjunt de llenguatges reconeguts per TM de parada segura.

Siguin A i B dos llenguatges reconeguts, respectivament, per les màquines M_A i M_B, ambdues de parada segura. Per demostrar que $A \cup B$ també és un llenguatge decidible dissenyem una TM M que consisteix bàsicament en la simulació de M_A i M_B.

M : **entrada** x
  simular M_A amb entrada x;
  si M_A accepta x
    llavors acceptar
    altrament
      simular M_B amb entrada x;
      si M_B accepta x
        llavors acceptar
        altrament rebutjar
      fi si
  fi si.

Com que M_A i M_B són d'aturada segura, aleshores M també és d'aturada segura. M accepta x si i només si $x \in A$ o $x \in B$ i, per tant, M reconeix $A \cup B$. Obtenim d'aquesta manera que, si $A, B \in$ REC, aleshores $A \cup B \in$ REC.

Considerem la classe RE. Els llenguatges d'aquesta classe també són reconeguts per TM, però RE es diferencia de les tres classes anteriors pel fet que la màquina no ha de ser forçosament d'aturada segura. En aquest cas, la demostració que RE és tancada respecte de la reunió també és constructiva, però la TM que hem dissenyat abans no és vàlida en aquest cas, ja que la màquina M_A no és d'aturada segura. Notem que, en cas que $x \notin A$ i $x \in B$, pot passar que la màquina M_A no s'aturi i, per tant, M tampoc no s'aturaria, cosa que no volem. Per aquest motiu la construcció de M pren com a base una simulació "pas a pas" de M_A i de M_B. Si una de les dues s'atura i accepta, aleshores M s'atura i accepta, altrament M o no s'atura o s'atura i rebutja.

La TM considerada és la següent:

M : **entrada** x
  $c_A :=$ configuració-inicial(M_A, x);
  $c_B :=$ configuració-inicial(M_B, x);
  trobat := (c_A és acceptadora per a M_A) $\vee$ (c_B és acceptadora per a M_B);

mentre $\neg trobat$ **fer**
  si $\neg(c_A$ és terminal) **llavors** $c_A :=$ configuració-següent(M_A, c_A) **fi si**;
  si $\neg(c_B$ és terminal) **llavors** $c_B :=$ configuració-següent(M_B, c_B) **fi si**;
  $trobat := (c_A$ és acceptadora per a $M_A) \vee (c_B$ és acceptadora per a $M_B)$
fi mentre;
acceptar.

Per la seva construcció tenim que $\mathsf{L}(M) = A \cup B$ i, per tant, $A \cup B \in \mathsf{RE}$.

La demostració que REC i RE són tancades respecte de la intersecció és molt similar a la de la reunió; només canvia l'operació booleana de la disjunció per la conjunció en els llocs que cal. □

3.3.2 Complementació

És fàcil veure que el complementari d'un llenguatge decidit per una TM determinista d'aturada segura M pot ser decidit per una TM M', que s'obté en modificar M de la manera següent: se substitueix l'estat acceptador per un d'aturada no acceptador i s'afegeix als estats d'aturada no acceptadors una transició a un nou estat acceptador. El determinisme de la TM garanteix la inversió del comportament acceptador quan es processa qualsevol entrada. Així doncs, obtenim el resultat següent.

TEOREMA 3.9 *La classe REC és tancada respecte de la complementació.*

En canvi, la classe RE no és tancada respecte de la complementació. El teorema següent, que es coneix com a teorema del complementari, ens permet demostrar aquesta propietat de la classe RE.

TEOREMA 3.10 (Teorema del complementari) *L i $\overline{L}$ són e.r. si i només si L és decidible.*

DEMOSTRACIÓ Si L és decidible aleshores $\overline{L}$ també ho és, en conseqüència, tant L com $\overline{L}$ pertanyen a RE.

Suposem ara que tant L com $\overline{L}$ són e.r. Siguin M_L i $M_{\overline{L}}$ dues TM tals que $\mathsf{L}(M_L) = L$ i $\mathsf{L}(M_{\overline{L}}) = \overline{L}$. Per demostrar que L és decidible dissenyem un algorisme en què se simulen les dues TM M_L i $M_{\overline{L}}$ "pas a pas". Si considerem un mot x, tenim que o $x \in L$ o $x \in \overline{L}$. Per tant, si simulem M_L amb entrada x i $M_{\overline{L}}$ amb entrada x, segur que una de les dues s'atura i (només una) accepta.

Considerem la TM següent:

M : **entrada** x
  $c_L :=$ configuració-inicial(M_L, x);
  $c_{\overline{L}} :=$ configuració-inicial$(M_{\overline{L}}, x)$;
  $trobat := (c_L$ és acceptadora per a $M_L) \vee (c_{\overline{L}}$ és acceptadora per a $M_{\overline{L}})$;
  mentre $\neg trobat$ **fer**

si $\neg(c_L$ és terminal) **llavors** $c_L :=$ configuració-següent(M_L, c_L) **fi si**;
si $\neg(c_{\overline{L}}$ és terminal) **llavors**
$c_{\overline{L}} :=$ configuració-següent$(M_{\overline{L}}, c_{\overline{L}})$ **fi si**;
$trobat := (c_L$ és acceptadora per a $M_L) \vee (c_{\overline{L}}$ és acceptadora per a $M_{\overline{L}})$
fi mentre;
si c_L és acceptadora per a M_L
llavors acceptar
altrament rebutjar
fi si.

Aquesta TM decideix L i, per tant, $L \in \mathsf{REC}$. □

Com a conseqüencia immediata, obtenim els primers exemples de llenguatges no e.r.

COROL·LARI 3.11 *$\overline{\text{K}}$, $\overline{\text{HALT}}$ i $\overline{\text{PERT}}$ no són enumerables recursivament.*

DEMOSTRACIÓ Si $\overline{\text{K}}$ ($\overline{\text{HALT}}$ o $\overline{\text{PERT}}$) fos e.r., pel teorema del complementari K (HALT o PERT) seria decidible. □

Així doncs, el complementari d'un llenguatge e.r. pot no ser e.r.

COROL·LARI 3.12 *La classe RE no és tancada respecte de la complementació.*

Donada una classe de llenguatges qualsevol $\mathcal{C}$, denotem per co-$\mathcal{C}$ la classe dels llenguatges L tals que $\overline{L} \in \mathcal{C}$. Amb aquesta notació tenim que una classe $\mathcal{C}$ és tancada respecte de complementació si i només si $\mathcal{C} = \text{co-}\mathcal{C}$.

Així doncs, tenim que:

$$\mathsf{REC} = \text{co-}\mathsf{REC}$$

I pel corol·lari 3.12,

$$\mathsf{RE} \neq \text{co-}\mathsf{RE}$$
$$\mathsf{REC} = \mathsf{RE} \cap \text{co-}\mathsf{RE}.$$

3.4 Problemes

3.1 Sigui f una funció computable i injectiva. És f^{-1} computable?

3.2 Considereu les funcions següents:

1. $f(x) = \begin{cases} 1 & \text{si } \exists n \geqslant 0 \ \ \mathsf{M}_n(x)\downarrow \\ \text{indefinida,} & \text{altrament.} \end{cases}$

2. $f(x) = \begin{cases} x & \text{si } \forall n \geqslant 0 \ \ \mathsf{M}_n(x)\downarrow \\ \text{indefinida,} & \text{altrament.} \end{cases}$

3. $f(x) = \begin{cases} 1 & \text{si } \exists n \geqslant 0 \;\; \mathsf{M}_x(n)\downarrow \\ \text{indefinida,} & \text{altrament.} \end{cases}$

4. $f(x) = \begin{cases} x & \text{si } \forall n \geqslant 0 \;\; \mathsf{M}_x(n)\downarrow \\ \text{indefinida,} & \text{altrament.} \end{cases}$

a) Són computables?

b) Són totals?

c) Quins són els seus abasts?

3.3 Demostreu que les funcions definides a continuació són computables i es poden prolongar a funcions computables i totals.

1. $f(x) = \begin{cases} 1 & \text{si } \mathsf{M}_x(x)\downarrow \\ \text{indefinida,} & \text{altrament.} \end{cases}$

2. $g(\langle x, t\rangle) = \begin{cases} t & \text{si } \mathsf{M}_x \text{ amb entrada } x \text{ s'atura en } t \text{ passos} \\ \text{indefinida,} & \text{altrament.} \end{cases}$

3.4 Demostreu que les funcions definides a continuació són computables però no tenen cap prolongació computable total.

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{si } \mathsf{M}_x(x)\downarrow \\ \text{indefinida,} & \text{altrament.} \end{cases}$$

$$g(x) = \begin{cases} \varphi_x(x) & \text{si } \mathsf{M}_x(x)\downarrow \\ \text{idefinida,} & \text{altrament.} \end{cases}$$

3.5 Sigui $f : \mathbb{N} \to \mathbb{N}$ una funció computable i decreixent al seu domini. Demostreu que $\mathsf{Im} f$ és decidible. Demostreu que, de fet, per a tot conjunt A, $f(A)$ és decidible. Segueix sent cert aquest resultat si f no és computable?

3.6 Sigui f una funció computable i creixent al seu domini. Sigui A un llenguatge decidible. Demostreu que, si f és total, llavors $f(A)$ és decidible.

3.7 Demostreu que per a tot conjunt $A \neq \varnothing$ es compleix:

1. A és enumerable recursivament si i només si és l'abast d'alguna funció computable.
2. A és enumerable recursivament si i només si és l'abast d'alguna funció total computable.
3. A és enumerable recursivament si i només si és el domini d'alguna funció computable.

3.8 Sigui $R = \{x \mid \mathsf{Dom}(\varphi_x) \in \mathsf{REC}\}$. Digueu si les afirmacions següents són certes o no i per què:

a) $R \subseteq \mathrm{K}$.

b) $R \cap \mathrm{K} = \varnothing$.

3.9 Definim la funció $f : \mathbb{N} \to \mathbb{N}$ com

$$f(n) = \begin{cases} \sum_{i=1}^{n} \mathsf{M}_i(n) & \text{si } \mathsf{M}_1(n)\downarrow \ \wedge\ \mathsf{M}_2(n)\downarrow \ \wedge\ \ldots \ \wedge\ \mathsf{M}_n(n)\downarrow \\ \text{indefinida,} & \text{altrament.} \end{cases}$$

i el conjunt $A = \{\langle n, m\rangle \mid f(n) = m\}$. Demostreu les afirmacions següents:

a) f és computable.

b) $\mathsf{Dom} f$ és finit.

c) A és decidible.

3.10 Demostreu que tot llenguatge decidible (enumerable recursivament) infinit es pot descompondre com a reunió de dos llenguatges decidibles (enumerables recursivament) infinits i disjunts.

3.11 Siguin A, B dos conjunts tals que $A \triangle B = (A \cup B) - (A \cap B)$ és decidible. Digueu si les afirmacions següents són certes o no i per què:

a) Si A és e.r., llavors B és e.r.

b) Si A és decidible, llavors B és decidible.

3.12 Sigui A un llenguatge e.r. Demostreu que $B = \bigcup_{x \in A} \mathsf{Dom}\ \varphi_x$ també és e.r.

3.13 Sigui A un llenguatge e.r. Demostreu que $B = \{x \mid \exists y\ (x, y) \in A\}$ també és e.r.

3.14 Demostreu que les classes REC i RE són tancades respecte de la concatenació i de l'estrella de Kleene.

4 Reductibilitat i completesa

En aquest capítol formalitzarem el concepte intuïtiu que un llenguatge és com a molt tan "difícil" computacionalment com un altre. Aquest concepte ens permetrà establir un preordre entre llenguatges. També ens proporcionarà una eina per demostrar propietats negatives de llenguatges, com per exemple la indecidibilititat i, fins i tot, la no-enumerabilitat recursiva. En aquest sentit, la demostració que un llenguatge indecidible és com a molt tan difícil com un cert llenguatge L implicarà, com veurem, que L també és indecidible.

Veurem com identificar, en algunes de les classes estudiades fins ara, els llenguatges que són maximals, segons aquest preordre. Per exemple, demostrarem que $\textsc{k}$ és un element maximal de RE i $\textsc{circuit-sat}$ és un element maximal de NP. Veurem que la dificultat d'aquests problemes maximals està relacionada amb la de tota la classe; qualsevol resultat positiu relatiu a aquests problemes maximals afecta a la classe sencera. En aquest sentit, demostrarem que si existís un algorisme que decidís $\textsc{circuit-sat}$ en temps polinòmic, aleshores també existiria un algorisme de temps polinòmic per a cadascun dels llenguatges dins de NP.

4.1 Reduccions

Intuïtivament, diem que un problema A es redueix a un problema B si podem *transformar* de manera efectiva les entrades de A en entrades de B. D'aquesta manera, si es disposa d'un algorisme que resol el problema B, es pot construir un algorisme que resol A. Nosaltres ens limitem a l'estudi de reduccions entre problemes decisionals i, per tant, parlem de reduccions entre llenguatges.

Si, per exemple, considerem els llenguatges $\textsc{k}$ i $\textsc{halt}$, no és difícil veure que podem transformar (*reduir*) les entrades del problema $\textsc{k}$ en entrades del problema $\textsc{halt}$. Donada una entrada x qualsevol, $x \in \textsc{k}$ si i només si $\langle x, x\rangle \in \textsc{halt}$, com ja hem vist al capítol ??. Aquesta transformació es pot formalitzar mitjançant la funció $f(x) = \langle x, x\rangle$, que és total i computable. Amb això tenim que si existís un algorisme que decidís $\textsc{halt}$, aleshores també existiria un algorisme que decidiria $\textsc{k}$. Aquest algorisme amb entrada x computaria la funció f amb entrada x i després simularia l'algorisme per a $\textsc{halt}$ sobre $f(x)$. Però aquest algorisme no pot existir perquè $\textsc{k}$ és indecidible. Aleshores, $\textsc{halt}$ és també indecidible com $\textsc{k}$.

Aquest tipus de funcions que transformen de manera efectiva entrades d'un problema A a entrades d'un altre problema B es coneix amb el nom de funcions de reducció. Tot seguit donem la definició formal d'aquests conceptes.

DEFINICIÓ Donats dos llenguatges A i B sobre un alfabet Σ, diem que A es *redueix* (o *és reductible*) a B, i ho representem per $A \leqslant_{\mathrm{m}} B$, quan existeix una funció $f : \Sigma^* \longrightarrow \Sigma^*$ total i computable tal que, per a tot $w \in \Sigma^*$,

$$w \in A \Longleftrightarrow f(w) \in B.$$

La funció f s'anomena *funció de reducció*.

El subíndex 'm' que apareix en el signe de la relació prové de l'anglès 'many-one', que en el context de funcions indica que elements diferents del domini poden tenir la mateixa imatge, és a dir, que no requerim que la funció f considerada sigui injectiva. També es pot donar que alguns elements del conjunt d'arribada no tinguin cap antiimatge.

Per la definició de reducció tenim que

$$A \leqslant_{\mathrm{m}} B \Longleftrightarrow \overline{A} \leqslant_{\mathrm{m}} \overline{B}.$$

DEFINICIÓ Donats dos llenguatges A i B sobre un alfabet Σ, diem que A i B són *recursivament equivalents* quan $A \leqslant_{\mathrm{m}} B$ i $B \leqslant_{\mathrm{m}} A$.

Seguint amb l'exemple, podem dir que $f(x) = \langle x, x \rangle$ és una funció de reducció de K a HALT i, per tant, K $\leqslant_{\mathrm{m}}$ HALT.

Tot seguit veiem que podem reduir HALT a PERT.

EXEMPLE 4.1 HALT $\leqslant_{\mathrm{m}}$ PERT.

Donada una entrada $\langle x, y \rangle$ de HALT, construïm una entrada $\langle x', y \rangle$ de PERT de manera que $\langle x, y \rangle \in$ HALT si i només si $\langle x', y \rangle \in$ PERT. A partir de $\mathsf{M}_x = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathrm{F}})$, definim la màquina $\mathsf{M}_{x'} = (Q', \Sigma, \Gamma, \delta', q_0, q_{\mathrm{F}'})$ de manera que

1. Afegim un nou estat que anomenem $q_{\mathrm{F}'}$, $Q' = Q \cup \{q_{\mathrm{F}'}\}$, i que és el nou estat acceptador.

2. L'estat inicial de $\mathsf{M}_{x'}$ és el mateix q_0.

3. La funció de transició és la de M_x afegint les transicions
   $\delta'(q, a) = (q_{\mathrm{F}'}, a, \mathtt{n})$ per a cada $q \in Q$ i $a \in \Gamma$ tal que $(q, a) \notin \mathsf{Dom}\delta$.

Recordem que M_x s'atura si assoleix una configuració en què no hi ha transició definida. D'aquesta manera, tenim que $\mathsf{M}_x(y)\downarrow \Longleftrightarrow$ $\mathsf{M}_{x'}$ amb entrada y accedeix a l'estat $q_{\mathrm{F}'} \Longleftrightarrow y \in \mathsf{L}(\mathsf{M}_{x'})$.

Definint $f(\langle x, y \rangle) = \langle x', y \rangle$ tenim que f és una funció computable i total i redueix HALT a PERT.

Com que la reducció és una relació entre llenguatges lligada al concepte de dificultat, és d'esperar que aquesta relació sigui transitiva: si A és com a molt tan difícil com B i B és com a molt tan difícil com C, aleshores A ha de ser com a molt tan difícil com C.

PROPOSICIÓ 4.1 *La relació $\leqslant_m$ és reflexiva i transitiva (defineix un preordre).*

DEMOSTRACIÓ Per veure que $\leqslant_{\mathrm{m}}$ és reflexiva només cal considerar la funció identitat. Aquesta redueix qualsevol llenguatge A a ell mateix i és computable en temps polinòmic.

Considerem la transitivitat de $\leqslant_{\mathrm{m}}$. Siguin A, B i C tres llenguatges tals que $A \leqslant_{\mathrm{m}} B$ i $B \leqslant_{\mathrm{m}} C$. Volem demostrar que $A \leqslant_{\mathrm{m}} C$. Siguin f i g funcions totals i computables que redueixen A a B i B a C, respectivament. Aleshores, per a tot x, $x \in A \iff f(x) \in B$ i $x \in B \iff g(x) \in C$. Per tant, $x \in A \iff f(x) \in B \iff g(f(x)) \in C$. Com que la composició de funcions computables i totals és una funció computable i total (proposició **??** i teorema **??**), tenim que $g \circ f$ és computable i total. Amb tot això podem deduir que $g \circ f$ redueix A a C. $\square$

EXERCICI 4.1 Demostreu que l'equivalència recursiva és, efectivament, una relació d'equivalència.

4.2 Propietats de les reduccions

Si analitzem la dificultat dels problemes relacionats per una reducció, $A \leqslant_{\mathrm{m}} B$, veiem que l'existència d'un algorisme que decideixi B garanteix l'existència d'un algorisme que decideixi A. Aquesta propietat vé expressada formalment en el teorema següent.

TEOREMA 4.2 *Siguin A i B dos llenguatges.*

1. *Si $A \leqslant_m B$ i $B \in \mathsf{REC}$, aleshores $A \in \mathsf{REC}$.*
2. *Si $A \leqslant_m B$ i $B \in \mathsf{RE}$, aleshores $A \in \mathsf{RE}$.*

DEMOSTRACIÓ
APARTAT 1. Sigui $B \in \mathsf{REC}$ i sigui M una TM de parada segura que decideix B. Com que $A \leqslant_{\mathrm{m}} B$, aleshores existeix una TM N, també d'aturada segura, que computa la funció de reducció f de A a B. Per construir la TM que decideix A utilitzem la propietat que, per a qualsevol x, $x \in A$ si i només si $f(x) \in B$. Considerem, doncs, la TM següent:

M' : **entrada** x
  simular N amb entrada x;
  $y := N(x)$;
  simular M amb entrada y;
  si M accepta y
    llavors acceptar
    altrament rebutjar
  fi si.

Per construcció, la màquina M' és d'aturada segura. Per a qualsevol entrada x, M' reconeix x si i només si $f(x) \in B$, que és equivalent a $x \in A$. Amb tot això tenim que M' és una TM que decideix A i, per tant, $A \in \mathsf{REC}$.

APARTAT 2. Sigui $B \in \mathsf{RE}$. Aleshores existeix una TM M que reconeix B que pot no ser d'aturada segura. Sigui N la TM que computa la funció de reducció f de A a B. Podem construir una TM M' tal com ho hem fet a l'apartat 1. Ara M' pot no aturar-se per a entrades x tals que $f(x) \notin B$, és a dir, $x \notin A$. Però per a x tals que $x \in A$, M' para i accepta. Amb tot això tenim que M' reconeix A i, per tant, $A \in \mathsf{RE}$.

□

El teorema anterior ens proporciona una eina fonamental per a la demostració de resultats negatius com la indecidibilitat i la no-enumerabilitat recursiva de llenguatges. El teorema ens diu que si considerem dos llenguatges qualssevol A i B, aleshores:

1. Si $A \leqslant_{\mathrm{m}} B$ i $A \notin \mathsf{REC}$, aleshores $B \notin \mathsf{REC}$.

2. Si $A \leqslant_{\mathrm{m}} B$ i $A \notin \mathsf{RE}$, aleshores $B \notin \mathsf{RE}$.

En la proposició següent veurem que un llenguatge decidible A sempre es pot reduir a qualsevol llenguatge B diferent del $\varnothing$ i de Σ^*. Podem definir una funció que amb entrada x o bé retorna un element de B si $x \in A$, o bé un element que no pertanyi a B en cas contrari. Com que A és decidible, podem utilitzar la TM que el decideix per determinar si un element x pertany a A o no.

PROPOSICIÓ 4.3 *Siguin A i B dos llenguatges. Si $A \in \mathsf{REC}$ i $B \neq \varnothing$ i $B \neq \Sigma^*$, aleshores $A \leqslant_m B$.*

DEMOSTRACIÓ
Si $A \in \mathsf{REC}$, aleshores existeix una TM que decideix A. Sigui M_A aquesta màquina. Com que $B \neq \varnothing$ i $B \neq \Sigma^*$, aleshores existeixen dos elements x_0 i x_1 tals que $x_0 \in B$ i $x_1 \notin B$.

Definim ara la funció

$$f(x) = \begin{cases} x_0 & \text{si } x \in A \\ x_1 & \text{si } x \notin A. \end{cases}$$

La funció f és total i per a qualsevol x satisfà $x \in A$ si i només si $f(x) \in B$. Com que M_A és una TM de parada segura, podem construir una TM que anomenem M, que simuli M_A sobre la mateixa entrada i que retorni x_0 o x_1 depenent de si M_A accepta o no, respectivament.

```
M : entrada x
    simular M_A amb entrada x;
    si M_A accepta x
        llavors escriure x_0
        altrament escriure x_1
    fi si.
```

És fàcil veure que M computa f. Amb tot això tenim que f és una funció de reducció de A a B.

□

Amb l'ajut d'aquesta propietat és fàcil demostrar que la reducció no satisfà la propietat simètrica ni l'antisimètrica.

Exercici 4.2 Demostreu que les relació $\leqslant_{\mathrm{m}}$ no és simètrica ni antisimètrica.

Exercici 4.3 Demostreu que les relació $\leqslant_{\mathrm{m}}$ no és total. (La definició de relació total es pot trobar a l'apèndix **??**.) [*Indicació.* Podeu considerar els llenguatges K i $\overline{\text{K}}$.]

4.3 Reduccions i indecidibilitat

En aquesta secció veiem primer alguns exemples de reduccions no trivials que ens permeten demostrar la indecidibilitat d'alguns llenguatges. Les transformacions en aquests casos no consisteixen en petites manipulacions de les entrades, com era el cas de la reducció de K a HALT o a l'inrevés. Moltes vegades s'ha de transformar un número de Gödel x en un altre número de Gödel y, de manera que aquesta transformació depèn de propietats de la computació de M_x sobre determinades entrades. En aquests casos, veiem que ens és molt útil l'aplicació del teorema s-m-n, que enunciem i demostrem a la subsecció que segueix.

En segon lloc, introduïm una tècnica derivada de les reduccions per demostrar que determinats tipus de llenguatges són o no són decidibles. Aquesta tècnica s'aplica a llenguatges d'un tipus particular, però de gran utilitat, que s'anomenen conjunts d'índexs. Si és així, podem aplicar el teorema de Rice, que proporciona la condició necessària i suficient que ha de satisfer un conjunt d'índexs recursiu.

4.3.1 Teorema s-m-n

Considerem TM que computen funcions de més d'una variable d'entrada. Si assignem un valor constant a algun dels arguments d'entrada, és natural pensar que la funció que en resulta també és computable. En un programa escrit en un llenguatge de programació qualsevol només caldria substituir la referència a la variable d'entrada pel valor constant corresponent. Donats el programa i els valors dels arguments que es volen fixar, podem obtenir de manera efectiva el programa que computa la mateixa funció havent fixat aquests valors. Tot seguit veiem com realitzar aquesta transformació en el model de la TM.

Considerem primer aquelles funcions que es defineixen a partir d'una altra funció a la qual s'assignen uns valors constants a algunes variables d'entrada.

Definició Sigui f una funció de $m+n$ variables d'entrada, amb $m, n \geqslant 1$. Per a cada m-tuple $(x_1, \ldots, x_m) \in \mathbb{N}^m$, podem definir la funció següent amb n variables d'entrada:

$$f^{x_1,\ldots,x_m}(x_{m+1}, \ldots, x_{m+n}) = \begin{cases} f(\langle x_1, \ldots, x_{m+n}\rangle) & \text{si } \langle x_1, \ldots, x_{m+n}\rangle \in \mathsf{Dom} f \\ \text{indefinida,} & \text{altrament.} \end{cases}$$

EXEMPLE 4.2 Considerem la funció suma d'enters definida per $f(\langle x, y\rangle) = x + y$. Si substituïm x per una constant c, obtenim una funció d'una sola variable, que podem anomenar f^c i que es defineix com $f^c(y) = c + y$. D'aquesta manera, considerant diferents constants c, obtenim una família de funcions $\{f^c\}_{c\in\mathbb{N}}$.

La segona definició fa referència a TM que computen funcions amb més d'un argument d'entrada. Considerem una TM M qualsevol que té com a entrada un tuple de $m + n$ elements per $m, n \geqslant 1$. Igual que abans, podem fixar els valors dels primers m elements d'entrada i, en aquest cas, construir una TM per cada m-tuple.

DEFINICIÓ Sigui M una TM que té com a entrada un tuple de $m + n$ elements, amb $m, n \geqslant 1$. Per a cada m-tuple $(x_1, \ldots, x_m) \in \mathbb{N}^m$, podem construir la TM següent:

$M^{x_1,\ldots,x_m}$: **entrada** $\langle x_{m+1}, \ldots, x_{m+n}\rangle$
simular M amb entrada $\langle x_1, \ldots, x_m, x_{m+1}, \ldots, x_{m+n}\rangle$;
escriure $M(\langle x_1, \ldots, x_m, x_{m+1}, \ldots, x_{m+n}\rangle)$.

EXEMPLE 4.3 Si M és una TM que computa la funció suma de dos enters, aleshores per a cada valor c podem construir una TM M^c que computa f^c:

M^c : **entrada** y
simular M amb entrada $\langle c, y\rangle$;
$z := M(\langle c, y\rangle)$;
escriure z.

De les dues definicions anteriors es dedueix immediatament la proposició següent.

PROPOSICIÓ 4.4 *Sigui f una funció de $m + n$ variables, amb $m, n \geqslant 1$. Si M és una* TM *que computa f, és a dir, si $f = \mathsf{f}_M$, aleshores tenim que, per a tot $(x_1, \ldots, x_m) \in \mathbb{N}^m$,*

$$f^{x_1,\ldots,x_m} = \mathsf{f}_{M^{x_1,\ldots,x_m}}$$

EXEMPLE 4.4 Seguim amb l'exemple de la suma d'enters. Observeu que la construcció de cadascuna de les TM M^c consisteix en una transformació de la TM que computa la funció suma, i que depèn de la constant c. Per cada valor de c tenim una TM sintàcticament diferent. No és difícil veure que la funció que obté el número de Gödel de la màquina M^c a partir del valor c és una funció computable. L'algorisme que la computa és el següent (hem canviat el símbol c pel tradicional símbol x per referir-nos a la variable d'aquesta funció):

entrada x
M^x := "**entrada** y
simular M amb entrada $\langle x, y\rangle$;
$z := M(\langle x, y\rangle)$;
escriure z";
$t := \mathsf{gödel}(M^x)$;
escriure t.

Cal remarcar que l'algorisme no executa el programa escrit entre cometes sinó que el *construeix* a partir de la descripció de la màquina M i de x. Un cop especificada la TM M^x, el càlcul del seu número de Gödel és efectiu, és a dir, la funció $\mathsf{gödel}$ és computable, com ja s'ha explicat a la secció **??**.

Anomenem s aquesta funció, és a dir, $s(x) = \mathsf{gödel}(M^x)$. Remarquem que s depèn de M en el sentit que a cada TM M li correspon una funció s específica. Remarquem, finalment, que la funció s és computable i total.

Podem resumir la relació entre les funcions que hem definit partint de l'exemple de la suma d'enters, així:

$x + y = f(\langle x, y\rangle) = \mathsf{f}_M(\langle x, y\rangle) = f^x(y) = \mathsf{f}_{M^x}(y) = \varphi_{\mathsf{gödel}(M^x)}(y) = \varphi_{s(x)}(y)$

La possibilitat de transformar de manera efectiva una TM qualsevol que computa una funció de més d'una variable d'entrada en una altra TM que computa la mateixa funció, però havent fixat el valor d'algunes de les seves variables, s'enuncia al teorema que es coneix amb el nom de *teorema s-m-n* o *teorema de la parametrització.*

TEOREMA 4.5 (Teorema s-m-n) *Sigui M una* TM *que computa una funció f de $m + n$ variables d'entrada, amb $m, n \geqslant 1$. Aleshores $\{f^{x_1,\dots,x_m}\}_{(x_1,\dots,x_m)\in\mathbb{N}^m}$ és una família de funcions computables, i existeix una funció computable i total s, pròpia de cada M, tal que, per a tot m-tuple $(x_1, \dots , x_m) \in \mathbb{N}^m$, es verifica*

$$f^{x_1,\dots,x_m} = \varphi_{s(\langle x_1,\dots,x_m\rangle)}$$

o, formulat altrament,

$$M^{x_1,\dots,x_m} = \mathsf{M}_{s(\langle x_1,\dots,x_m\rangle)}.$$

DEMOSTRACIÓ Considerem la família de TM $\{M^{x_1,\dots,x_m}\}_{(x_1,\dots,x_m)\in\mathbb{N}^m}$. Per la proposició anterior sabem que, per a tot $(x_1, \dots , x_m) \in \mathbb{N}^m$, $f^{x_1,\dots,x_m} = \mathsf{f}_{M^{x_1,\dots,x_m}}$ i, per tant, la família $\{f^{x_1,\dots,x_m}\}_{(x_1,\dots,x_m)\in\mathbb{N}^m}$ és una família de funcions computables. Construïm la TM següent:

N : **entrada** $\langle x_1, \dots , x_m\rangle$
  $M^{x_1,\dots,x_m}$:= "**entrada** $\langle x_{m+1}, \dots , x_{m+n}\rangle$;
    simular M amb entrada $\langle x_1, \dots , x_m, x_{m+1}, \dots , x_{m+n}\rangle$;
    escriure $M(\langle x_1, \dots , x_m, x_{m+1}, \dots , x_{m+n}\rangle)$";
  $z := \mathsf{gödel}(M^{x_1,\dots,x_m})$;
  escriure z.

Anomenem s la funció computada per N. Per la construcció de N, tenim que s és una funció total i que, per a cada m-tuple $(x_1, \dots , x_m) \in \mathbb{N}^m$, $s(\langle x_1, \dots , x_m\rangle)$ és igual al número de Gödel de la màquina $M^{x_1,\dots,x_m}$. Per tant, s satisfà que, per a tot $(x_1, \dots , x_m) \in \mathbb{N}$,

$$f^{x_1,\dots,x_m} = \varphi_{s(\langle x_1,\dots,x_m\rangle)}.$$

□

Aquest resultat permet transformar una màquina M que computa una funció de $m+n$ variables d'entrada en una altra màquina $M^{x_1,\dots,x_m}$ que computa la mateixa funció quan s'ha fixat el valor de m de les seves variables (m paràmetres). Aquesta transformació és efectiva, és a dir, existeix una funció computable s que, donats els valors dels m paràmetres $x_1, \dots, x_m$, ens retorna la descripció (número de Gödel) de $M^{x_1,\dots,x_m}$. D'aquí ve el nom de teorema s-m-n amb què es coneix.

EXEMPLE 4.5 En el model de TM que hem introduït, es considera que un mot és acceptat per una màquina M quan M amb entrada x assoleix l'estat final, que és de parada. Imaginem ara que volem transformar cada màquina M en una màquina M', de manera que si M para i accepta, aleshores M' para i retorna 1. Si M para i no accepta, aleshores M' para i retorna un 0. I finalment, si M no para, aleshores M' tampoc. Aquesta transformació es pot realitzar de manera efectiva, és a dir, existeix una funció computable i total s tal que, per a qualsevol descripció x d'una TM, $s(x)$ ens retorna la descripció d'una TM equivalent a M_x en tant que acceptadora de llenguatges, però amb la propietat que acabem de descriure. Per demostrar-ho construïm primer la TM M següent:

M : **entrada** $\langle x, y\rangle$
  simular M_x amb entrada y;
  si M_x accepta y
    llavors escriure 1
    altrament escriure 0
  fi si.

Per la seva construcció tenim que M amb entrada $\langle x, y\rangle$ simula M_x amb entrada y i, en cas que $\mathsf{M}_x(y)\downarrow$, aleshores retorna 1 o 0 segons si M_x accepta o rebutja, respectivament. Si fixem el valor del primer element d'entrada, aleshores per cada x tenim una TM M^x que es comporta com esperem:

$$M^x(y) = \begin{cases} 1 & \text{si } \mathsf{M}_x \text{ accepta } y \\ 0 & \text{si } \mathsf{M}_x(y)\downarrow \text{ i } \mathsf{M}_x \text{ no accepta } y \\ \text{indefinida} & \text{si } \mathsf{M}_x(y)\uparrow. \end{cases}$$

Pel teorema s-m-n sabem que existeix una funció s computable i total tal que $M^x = \mathsf{M}_{s(x)}$ i, per tant, s amb entrada x retorna la descripció de la TM esperada. Observeu que, si M_x és un TM de parada segura, aleshores $\mathsf{M}_{s(x)}$ computa la funció característica del llenguatge acceptat per M_x.

Ara ja estem en condicions de veure l'aplicabilitat del teorema s-m-n per construir reduccions entre problemes en què es requereix la transformació de descripcions de TM.

EXEMPLE 4.6 $\mathrm{K} \leqslant_{\mathrm{m}} \{x \mid \varphi_x \text{ és la funció identitat}\}$.

En aquest cas, volem transformar una TM, que s'atura quan llegeix com a entrada el seu propi número de Gödel, en una màquina que computa la funció identitat.

Considerem la màquina següent:

M : **entrada** $\langle x, y \rangle$
    simular M_x amb entrada x;
    escriure y.

Per la construcció de M tenim que, si $x \in \mathrm{K}$, aleshores, per a tot y, $M(\langle x, y \rangle) = y$ i, per tant, $M^x(y) = y$. En canvi, si $x \notin \mathrm{K}$, aleshores, per a tot y, $M(\langle x, y \rangle)\uparrow$ i, per tant, $M^x(y)\uparrow$.

Pel teorema s-m-n sabem que existeix una funció computable i total s que amb entrada x retorna el número de Gödel de M^x, formalment, $\mathsf{M}_{s(x)} = M^x$. Per tant, si $x \in \mathrm{K}$ tenim que, per a tot y, $\mathsf{M}_{s(x)}(y) = y$ i amb això $s(x) \in \{x \mid \varphi_x$ és la funció identitat$\}$. En canvi, si $x \notin \mathrm{K}$, aleshores, per a qualsevol y, $\mathsf{M}_{s(x)}(y)\uparrow$ i, per tant, $s(x) \notin \{x \mid \varphi_x$ és la funció identitat$\}$.

Per tant, s redueix K a $\{x \mid \varphi_x$ és la funció identitat$\}$.

Exemple 4.7 $\overline{\mathrm{K}} \leqslant_{\mathrm{m}} \{x \mid \varphi_x$ és la funció identitat$\}$.

En aquest cas, volem transformar una TM, que no s'atura quan llegeix com a entrada el seu propi número de Gödel, en una màquina que computa la funció identitat.

Considerem la màquina següent:

M : **entrada** $\langle x, y \rangle$
    $c :=$ configuració-inicial(M_x, x);
    $t := y$;
    mentre $(t > 0) \wedge (c$ no és terminal per a $\mathsf{M}_x)$ **fer**
        $c :=$ configuració-següent(M_x, c);
        $t := t - 1$
    fi mentre;
    si c és terminal per a M_x
      llavors escriure $(y + 1)$
      altrament escriure y
    fi si.

Per la construcció de M tenim que, si $x \in \overline{\mathrm{K}}$, aleshores, per a tot y, $M(\langle x, y \rangle) = y$ i, per tant, per a tot y, $M^x(y) = y$. En canvi, si $x \notin \overline{\mathrm{K}}$, aleshores existeix y tal que $\mathsf{T}(\mathsf{M}_x, x) = y$, de manera que $M(\langle x, y \rangle) = y + 1$ i, per tant, $M^x(y) = y + 1$.

Pel teorema s-m-n sabem que existeix una funció computable i total s que amb entrada x retorna el número de Gödel de M^x, és a dir, $\mathsf{M}_{s(x)} = M^x$. Per tant, si $x \in \overline{\mathrm{K}}$, aleshores, per a tot y, $\mathsf{M}_{s(x)}(y) = y$ i amb això $s(x) \in \{x \mid \varphi_x$ és la funció identitat$\}$. Si $x \notin \overline{\mathrm{K}}$, aleshores existeix y tal que $\mathsf{M}_{s(x)}(y) = y + 1$ i, per tant, $s(x) \notin \{x \mid \varphi_x$ és la funció identitat$\}$.

Aleshores, s redueix $\overline{\mathrm{K}}$ a $\{x \mid \varphi_x$ és la funció identitat$\}$.

Exemple 4.8 HALT $\leqslant_{\mathrm{m}}$ K.

En aquest cas, volem transformar una entrada del tipus $\langle x, y \rangle$, en què x és el número de Gödel d'una TM i y una entrada, en t, un número de Gödel d'una altra TM, de manera que $\mathsf{M}_x(y)\downarrow \Longleftrightarrow \mathsf{M}_t(t)\downarrow$. Per això, construïm la màquina de Turing següent:

M : **entrada** $\langle x, y, z\rangle$
  simular M_x amb entrada y;
  escriure z.

Si considerem $M^{x,y}$, aleshores, per a tot z, $M^{x,y}(z)\downarrow \Longleftrightarrow \mathsf{M}_x(y)\downarrow$. Pel teorema s-m-n, existeix una funció computable i total s que a cada parell $\langle x, y\rangle$ li associa el número de Gödel de $M^{x,y}$. Aleshores, si $\langle x, y\rangle \in$ HALT tenim que, per a tot z, $M^{x,y}(z))\downarrow$ i, per tant, $\mathsf{M}_{s(\langle x,y\rangle)}(z)\downarrow$. En particular, $\mathsf{M}_{s(\langle x,y\rangle)}(s(\langle x,y\rangle))\downarrow$, i amb això $s(\langle x,y\rangle) \in$ K. En canvi, si $\langle x, y\rangle \notin$ HALT, aleshores, per a tot z, $M^{x,y}(z)\uparrow$ i, per tant, $\mathsf{M}_{s(\langle x,y\rangle)}(z)\uparrow$. En particular, $\mathsf{M}_{s(\langle x,y\rangle)}(s(\langle x,y\rangle))\uparrow$ i amb això tenim que $s(\langle x,y\rangle) \notin$ K.

De tot això es dedueix que s redueix HALT a K. Com que hem vist al principi d'aquest capítol que K es redueix a HALT, podem concloure que els dos problemes són recursivament equivalents.

4.3.2 Conjunts d'índexs. Teorema de Rice

Les reduccions a partir de llenguatges no recursius constitueixen una eina fonamental per demostrar la no-recursivitat de llenguatges. Tot seguit veurem que, per a un determinat tipus de llenguatges anomenats *conjunts d'índexs*, disposem d'una altra eina derivada de les reduccions, però que, per aplicar-la, no ens cal construir cap reducció.

Els *conjunts d'índexs* són subconjunts de $\mathbb{N}$ (o de Σ^*) de la forma $\{x \in \mathbb{N} \mid P(\varphi_x)\}$. La propietat P, que determina la pertinença d'un element x al conjunt, fa referència a la funció computada per la màquina M_x. Dit d'una altra manera, els elements són descripcions (sintàctiques) de TM però la seva pertinença depèn de propietats relatives a la funció computada per la TM donada (propietats que en podríem dir semàntiques) i no del seu índex o número de Gödel. Per exemple, si considerem els dos conjunts següents,

$A = \{x \in \mathbb{N} \mid \mathsf{M}_x \text{ consta de 10 estats}\}$ i
$B = \{x \in \mathbb{N} \mid \varphi_x \text{ és computable per } \mathsf{M}_{100}\}$,

és clar que la propietat que defineix quins elements pertanyen A depèn directament de l'*estructura sintàctica* de M_x. En canvi, la pertinença a B depèn de la *funció computada* per M_x. Veurem que B és un conjunt d'índexs mentre que A no ho és.

Per tal de definir formalment els conjunts d'índexs, considerem primer la definició següent:

DEFINICIÓ Donada una funció f, definim la família d'índexs de f, que es representa per $\mathsf{Ind}(f)$, com el conjunt de naturals x tals que $\varphi_x = f$. És a dir,

$$\mathsf{Ind}(f) = \{x \mid f = \varphi_x\}.$$

Dit d'una altra manera, la família d'índexs d'una funció f està formada per totes les descripcions de les TM que computen la funció f. Si considerem la funció computada per una TM M_x, aleshores la seva família d'índexs conté totes les descripcions de les TM equivalents en tant que computadores de funcions. Formalment, $\varphi_x = \varphi_y \iff y \in \mathsf{Ind}(\varphi_x)$

EXERCICI 4.4 Per què la família d'índexs d'una funció o bé és infinita o bé és buida?

DEFINICIÓ Sigui $A \subseteq \mathbb{N}$. Diem que A és un *conjunt d'índexs* quan $\forall x \in A\ \mathsf{Ind}(\varphi_x) \subseteq A$.

Si un conjunt d'índexs A conté la descripció d'una TM, aleshores també conté totes les descripcions de TM equivalents. Podem caracteritzar aquesta propietat de la manera següent:

$$A \text{ és un conjunt d'índexs} \iff \forall x, y\ (x \in A \ \wedge\ \varphi_x = \varphi_y \implies y \in A)$$

EXERCICI 4.5 Demostreu que el complementari d'un conjunt d'índexs és també un conjunt d'índexs.

EXEMPLE 4.9 $A = \{x \in \mathbb{N} \mid \mathsf{M}_x \text{ té 10 estats}\}$ no és un conjunt d'índexs.

Sigui $x \in A$ i, per tant, M_x té 10 estats. És clar que podem afegir a M_x tants estats com vulguem sense canviar per això el resultat de la seva computació. Per exemple, hi podem afegir un nou estat que no és accessible des del seu estat inicial. Sigui M_y aquesta nova màquina. Aleshores, $\varphi_y = \varphi_x$ i, en canvi, $y \notin A$.

EXEMPLE 4.10 $B = \{x \in \mathbb{N} \mid \varphi_x = \varphi_{100}\}$ és un conjunt d'índexs.

Sigui $x \in B$ i, per tant, $\varphi_x = \varphi_{100}$. Per a qualsevol y tal que $\varphi_x = \varphi_y$ tenim també que $\varphi_y = \varphi_{100}$. Aleshores, B és un conjunt d'índexs. Fixeu-vos que $B = \mathsf{Ind}(\varphi_{100})$.

EXEMPLE 4.11 $C = \{x \in \mathbb{N} \mid \varphi_x \text{ és la funció identitat}\}$ és un conjunt d'índexs.

Sigui $x \in C$ i, per tant, φ_x és la funció identitat. Per a qualsevol y tal que $\varphi_x = \varphi_y$ tenim també que φ_y és la funció identitat. Aleshores, C és un conjunt d'índexs. Fixeu-vos que $C = \mathsf{Ind}(f)$, en què $f(x) = x$ per a tot x.

EXEMPLE 4.12 $D = \{x \in \mathbb{N} \mid \varphi_x \text{ és total}\}$ és un conjunt d'índexs. Sigui $x \in D$ i, per tant, φ_x és una funció total. Per a qualsevol y tal que $\varphi_x = \varphi_y$ tenim també que $\mathsf{Dom}\varphi_x = \mathsf{Dom}\varphi_y$ i, per tant, φ_y és una funció total. Aleshores, D és un conjunt d'índexs. Fixeu-vos que D conté la família d'índexs de la funció identitat, de la funció $f(x) = 1$ per a tot x, d'entre moltes d'altres famílies.

EXEMPLE 4.13 K no és un conjunt d'índexs.

Intuïtivament es veu que la propietat que defineix el llenguatge no depèn només de la funció que computa la màquina M_x, sinó que també depèn de la descripció sintàctica d'aquesta. Si podem construir una TM que només pari amb entrada la seva pròpia descripció, aleshores qualsevol altra descripció d'una TM equivalent no forma part de K. Formalment, si existeix x_0 tal que

$\mathsf{Dom}(\varphi_{x_0}) = \{x_0\}$, aleshores, per a qualsevol $y \neq x_0$ tal que $\varphi_y = \varphi_{x_0}$, tenim que $y \notin \mathrm{K}$, i, per tant, K no és un conjunt d'índexs. Per demostrar l'existència de x_0 podríem utilitzar el teorema de recursió (que queda fora de l'abast del nostre curs). El lector que hi estigui interessat pot consultar [21, 35].

Ara ja estem en condicions d'enunciar i demostrar el teorema que ens proporciona la condició necessària i suficient que caracteritza els conjunts d'índexs recursius.

TEOREMA 4.6 (Teorema de Rice) *Sigui A un conjunt d'índexs. A és recursiu si i només si $A = \varnothing$ o $A = \mathbb{N}$.*

DEMOSTRACIÓ Si $A = \varnothing$ o $A = \mathbb{N}$, aleshores és clar que A és recursiu.

Suposem que $A \neq \varnothing$ i $A \neq \mathbb{N}$. Sigui $c \in \mathbb{N}$ el número de Gödel d'una TM que no s'atura per a cap entrada, és a dir, $\mathsf{Dom}(\varphi_c) = \varnothing$.

Tot seguit fem un tractament per casos segons si c pertany o no a A.

CAS 1: $c \in A$.
En aquest cas, veiem que podem reduir $\overline{\mathrm{K}}$ a A. Per a això, considerem un element $b \notin A$ (recordem que $A \neq \mathbb{N}$) i definim la TM següent:

M : **entrada** $\langle x, y\rangle$
simular M_x amb entrada x;
simular M_b amb entrada y;
escriure $\mathsf{M}_b(y)$.

Per la construcció de M tenim que si $x \in \mathrm{K}$ aleshores, per a tot y, $M^x(y)\downarrow \Longleftrightarrow \mathsf{M}_b(y)\downarrow$, i en cas que $\mathsf{M}_b(y)\downarrow$ aleshores $M^x(y) = \mathsf{M}_b(y)$. Si $x \notin \mathrm{K}$ aleshores, per a tot y, $M^x(y)\uparrow$.

Pel teorema s-m-n sabem que existeix una funció s computable i total tal que $M^x = \mathsf{M}_{s(x)}$. Amb això tenim que si $x \in \mathrm{K}$ aleshores, $\varphi_{s(x)} = \varphi_b$. Si $x \notin \mathrm{K}$, aleshores per a tot y, $\mathsf{M}_{s(x)}(y)\uparrow$ i, per tant, $\varphi_{s(x)} = \varphi_c$

Com que A és un conjunt d'índexs, $c \in A$ i $b \notin A$, aleshores si $x \in \mathrm{K}$ tenim que $s(x) \notin A$. I si $x \notin \mathrm{K}$, llavors $s(x) \in A$.

D'aquesta manera, obtenim que s redueix $\overline{\mathrm{K}}$ a A i, per tant, A no és recursiu (ni enumerable recursivament).

CAS 2: $c \notin A$.
En aquest cas, només cal observar que $\overline{A}$ compleix les condicions del cas 1: $c \in \overline{A}$ i $\overline{A} \neq \mathbb{N}$. Per tant, $\overline{\mathrm{K}}$ es redueix a $\overline{A}$, d'aquesta manera s'obté que A no és recursiu. □

De la reducció construïda al primer cas de la demostració anterior es dedueix el resultat següent:

COROL·LARI 4.7 *Sigui A un conjunt d'índexs no recursiu. Si existeix $x \in A$ tal que $\mathsf{Dom}(\varphi_x) = \varnothing$, aleshores A no és enumerable recursivament.*

EXEMPLE 4.14 $B = \{x \in \mathbb{N} \mid \varphi_x = \varphi_{100}\}$ és un conjunt d'índexs no recursiu. Per la definició de B tenim que $100 \in B$ i, per tant, $B \neq \varnothing$. D'altra banda, si $B = \mathbb{N}$ llavors totes les funcions computables coincidirien amb φ_{100}, i això és clarament fals.

A continuació, apliquem el teorema de Rice als conjunts d'índexs dels exemples anteriors.

EXEMPLE 4.15 $C = \{x \mid \varphi_x \text{ és la funció identitat}\}$ és un conjunt d'índexs no recursiu.

Com que la funció identitat és una funció computable, existeix x tal que M_x computa la funció identitat i satisfà $x \in C$.

D'altra banda, si considerem la funció successor, és a dir, $f(x) = x + 1$, que és computable, tenim que qualsevol número de Gödel y d'una TM que la computi satisfà $y \notin C$.

Aleshores, com que C és un conjunt d'índexs diferent de $\varnothing$ i de $\mathbb{N}$, tenim que C no és recursiu.

EXEMPLE 4.16 $D = \{x \mid \varphi_x \text{ és total}\}$ és un conjunt d'índexs no recursiu.

Com que la funció identitat és una funció total computable, existeix x tal que M_x computa la funció identitat i satisfà $x \in D$.

D'altra banda, si considerem la funció buida, que és computable, com hem vist a la secció **??**, tenim que qualsevol número de Gödel y d'una TM que la computi satisfà $y \notin D$.

Aleshores, com que D és un conjunt d'índexs diferent de $\varnothing$ i de $\mathbb{N}$, tenim que D no és recursiu.

EXERCICI 4.6 Considerem ara el conjunt de TM com a reconeixedores de llenguatges. Recordem que a la secció **??** hem representat per L_x el llenguatge reconegut per M_x. Donat un llenguatge L, definim la família d'índexs de L com

$$\mathsf{Ind}(L) = \{x \mid L = \mathsf{L}_x\}.$$

Sigui $A \subseteq \mathbb{N}$. Diem que A és un *conjunt d'índexs de llenguatges* quan, per tot $x \in A$ $\mathsf{Ind}(\mathsf{L}_x) \subseteq A$. Demostreu el teorema de Rice per a aquests conjunts d'índexs, és a dir, demostreu que un conjunt A d'índexs de llenguatges és recursiu si i només si $A = \varnothing$ o $A = \mathbb{N}$.

4.4 Llenguatges e.r. complets

DEFINICIÓ Diem que una classe de llenguatges $\mathcal{C}$ és tancada respecte d'una reducció $\leqslant$, quan per a tot llenguatge A, si $A \leqslant B$ i $B \in \mathcal{C}$, aleshores $A \in \mathcal{C}$.

Observeu que amb aquesta definició podem presentar el teorema 4.2 en la forma següent:

TEOREMA 4.8 *Les classes REC i RE són tancades respecte de $\leqslant_m$.*

Si $\mathcal{C}$ és una classe de llenguatges tancada respecte d'una reducció $\leqslant$ (que és reflexiva i transitiva), com que $\leqslant$ preordena els elements de $\mathcal{C}$, és natural pensar en l'existència d'elements maximals d'aquesta classe, els elements "més difícils" de $\mathcal{C}$.

DEFINICIÓ Sigui $\mathcal{C}$ una família de llenguatges i sigui $\leqslant$ una reducció.

Diem que un llenguatge L és *$\mathcal{C}$-difícil* respecte de la reducció $\leqslant$ quan, per a tot $L' \in \mathcal{C}$, es compleix que $L' \leqslant L$. Diem que un llenguatge L és *$\mathcal{C}$-complet* respecte de la reducció $\leqslant$ quan L és *$\mathcal{C}$-difícil* respecte d'aquesta reducció i $L \in \mathcal{C}$.

Utilitzem les abreviacions 'RE-complet' i 'RE-difícil' per designar RE-complet i RE-difícil respecte de $\leqslant_{\mathrm{m}}$.

PROPOSICIÓ 4.9 *Si un llenguatge L' és RE-difícil i $L' \leqslant_m L$, aleshores L és RE-difícil.*

DEMOSTRACIÓ Si L' és RE-difícil, aleshores tot $L'' \in \mathsf{RE}$ satisfà $L'' \leqslant_{\mathrm{m}} L'$. Com que $L' \leqslant_{\mathrm{m}} L$, aplicant la propietat transitiva de les reduccions tenim que tot llenguatge $L'' \in \mathsf{RE}$ satisfà $L'' \leqslant L$ i, per tant, L és RE-difícil respecte de $\leqslant_{\mathrm{m}}$. □

Notem que els llenguatges RE-complets són recursivament equivalents.

Els llenguatges complets per a una classe $\mathcal{C}$ representen el conjunt de llenguatges maximals de $\mathcal{C}$ segons la relació de reducció. En aquest sentit, qualsevol propietat relativa a un llenguatge complet de $\mathcal{C}$ podem dir que "afecta" tots els llenguatges de $\mathcal{C}$.

PROPOSICIÓ 4.10 *Siguin $\mathcal{C}$ i $\mathcal{C}'$ dues classes de llenguatges i sigui $\mathcal{C}'$ tancada respecte d'una reducció $\leqslant$. Si L és $\mathcal{C}$-difícil respecte de la reducció i $L \in \mathcal{C}'$, aleshores $\mathcal{C} \subseteq \mathcal{C}'$.*

DEMOSTRACIÓ

Com que L és $\mathcal{C}$-difícil, aleshores tot llenguatge $L' \in \mathcal{C}$ satisfà $L' \leqslant L$. Si $L \in \mathcal{C}'$, com que $\mathcal{C}'$ és tancada respecte de $\leqslant$ tenim que $\forall L', L' \in \mathcal{C} \Rightarrow L' \in \mathcal{C}'$ i, per tant, $\mathcal{C} \subseteq \mathcal{C}'$. □

Vist aquest resultat, podem dir que els problemes complets copsen la dificultat de la classe. La classificació de problemes complets per a determinades classes, és una tècnica molt utilitzada per obtenir resultats negatius. Del fet que $\mathsf{REC} \subsetneq \mathsf{RE}$ deduïm, utilizant la proposició anterior, el resultat següent:

COROL·LARI 4.11 *Si L és un llenguatge RE-complet aleshores $L \notin \mathsf{REC}$.*

Tot seguit demostrem que problemes ja ben coneguts pel lector són RE-complets. El nostre objectiu principal és aprendre a distingir els problemes decidibles dels indecidibles.

TEOREMA 4.12 $\textsc{k}$ *és RE-complet.*

DEMOSTRACIÓ A l'exemple 3.8 hem demostrat que $\textsc{k} \in \mathsf{RE}$. Ara cal demostrar que $\textsc{k}$ és RE-difícil. Sigui L un llenguatge tal que $L \in \mathsf{RE}$. Sigui M la TM que reconeix L.

Volem transformar una entrada x de M en un número de Gödel y d'una TM, de manera que M accepta $x \Longleftrightarrow \mathsf{M}_y(y)\downarrow$. Amb aquest fi, construïm la màquina de Turing següent:

```
N : entrada ⟨x, z⟩
    simular M amb entrada x;
    si M accepta x
       llavors escriure z
       altrament
```

mentre cert **fer**
fi mentre
fi si.

Per la construcció de N tenim que si $x \in \mathsf{L}(M)$, aleshores per a tot z, $N(\langle x, z\rangle) = z$ i, per tant, $N^x(z) = z$. I si $x \notin \mathsf{L}(M)$, és a dir, tant si $M(x)\!\downarrow$ i M no accepta com si $M(x)\!\uparrow$, es té que, per a tot z, $N(\langle x, z\rangle)\!\uparrow$ i, per tant, $N^x(z)\!\uparrow$.

Aplicant el teorema s-m-n sabem que existeix una funció computable i total s tal que, per a tot x, $N^x = \mathsf{M}_{s(x)}$. Aleshores, si $x \in \mathsf{L}(M)$ tenim que, per a tot z, $\mathsf{M}_{s(x)}(z)\!\downarrow$ i en particular $\mathsf{M}_{s(x)}(s(x))\!\downarrow$. Per tant, $s(x) \in$ K. I si $x \notin \mathsf{L}(M)$ aleshores per a tot z, $\mathsf{M}_{s(x)}(z)\!\uparrow$ i en particular $\mathsf{M}_{s(x)}(s(x))\!\uparrow$. Per tant $s(x) \notin$ K.

Amb tot això, tenim que s és una funció que redueix L a K. □

COROL·LARI 4.13 *Els problemes* HALT *i* PERT *són* RE*-complets.*

DEMOSTRACIÓ A la secció 3.2 hem vist que HALT, PERT $\in$ RE. Com que K és RE-difícil i acabem de veure a la secció 4.1 que K $\leqslant_{\mathrm{m}}$ HALT $\leqslant_{\mathrm{m}}$ PERT, resulta que HALT i PERT també són RE-difícils. □

4.5 Problemes

4.1 Classifiqueu com a recursius, enumerables recursivament o cap de les dues coses els problemes següents:

1. $\{x \mid \exists y, z \ \ \varphi_x(y) = z\}$
2. $\{x \mid \exists y \ \ \varphi_x(y) = y\}$
3. $\{x \mid \varphi_x(x) = x\}$
4. $\{x \mid \exists y \ \ (\mathsf{M}_x(y)\!\downarrow\ \wedge \mathsf{M}_y(y)\!\downarrow\ \wedge \mathsf{M}_x(y) = \mathsf{M}_y(y))\}$
5. $\{x \mid \mathsf{Dom}(\varphi_x)$ és recursiu$\}$
6. $\{x \mid \mathsf{Dom}(\varphi_x)$ no és recursiu$\}$
7. $\{x \mid \mathsf{Im}(\varphi_x)$ és recursiu$\}$
8. $\{x \mid \mathsf{Im}(\varphi_x)$ no és recursiu$\}$
9. $\{x \mid \mathsf{Dom}(\varphi_x)$ és e.r.$\}$
10. $\{x \mid x < 666 \ \wedge \ \mathsf{Dom}(\varphi_x)$ és recursiu$\}$
11. $\{x \mid x \geqslant 666 \ \wedge \ \mathsf{Dom}(\varphi_x)$ és e.r.$\}$
12. $\{x \mid$ existeix una extensió computable total de $\varphi_x\}$
13. $\{x \mid \forall y \geqslant x \ \ \varphi_y$ és constant$\}$

14. $\{x \mid \exists y \geqslant x \ \ \varphi_y \text{ és constant}\}$
15. $\{x \mid \varphi_x \text{ és injectiva}\}$
16. $\{x \mid \varphi_x \text{ no és injectiva}\}$
17. $\{x \mid \varphi_x \text{ és exhaustiva}\}$
18. $\{x \mid \varphi_x \text{ no és exhaustiva}\}$
19. $\{x \mid \varphi_x \text{ és bijectiva}\}$
20. $\{x \mid \varphi_x \text{ no és bijectiva}\}$
21. $\{x \mid \mathsf{L}(\mathsf{M}_x) \text{ és finit}\}$
22. $\{x \mid \mathsf{L}(\mathsf{M}_x) \text{ és infinit}\}$
23. $\{x \mid \exists y \ \ \mathsf{Dom}(\varphi_x) \subseteq \mathsf{Dom}(\varphi_y)\}$
24. $\{x \mid \mathsf{Dom}(\varphi_x) \subseteq \{2n \mid n \geqslant 0\}\}$
25. $\{x \mid \mathsf{Dom}(\varphi_x) \cap \{2^n \mid n \geqslant 0\} \neq \varnothing\}$
26. $\{\langle x, y\rangle \mid \mathsf{Dom}(\varphi_x) = \mathsf{Dom}(\varphi_y)\}$
27. $\{\langle x, y\rangle \mid \mathsf{Dom}(\varphi_x) = \{2z \mid z \in \mathsf{Im}(\varphi_y)\}\}$
28. $\{x \mid \mathsf{Dom}(\varphi_x) = \mathbb{N}\}$
29. $\{x \mid \mathsf{Dom}(\varphi_x) = \varnothing\}$
30. $\{x \mid \mathsf{Dom}(\varphi_x) \neq \varnothing\}$
31. $\{x \mid \mathsf{Dom}(\varphi_x) \text{ és infinit}\}$
32. $\{x \mid \mathsf{Im}(\varphi_x) = \mathbb{N}\}$
33. $\{x \mid \mathsf{Im}(\varphi_x) = \varnothing\}$
34. $\{x \mid \mathsf{Im}(\varphi_x) \neq \varnothing\}$
35. $\{x \mid \mathsf{Im}(\varphi_x) \text{ és infinit}\}$

4.2 Considerem la funció f definida per:

$$f(x) = \begin{cases} 1 & \text{si } \exists y \leqslant 400 \ \ \varphi_y = \varphi_x \\ \text{indefinida,} & \text{altrament.} \end{cases}$$

Demostreu les afirmacions següents:

a) $\mathsf{Dom} f$ no és finit.

b) $\mathsf{Dom} f$ no és recursiu, perquè $\mathrm{K} \leqslant_{\mathrm{m}} \mathsf{Dom} f$ o $\overline{\mathrm{K}} \leqslant_{\mathrm{m}} \mathsf{Dom} f$.

4.3 Demostreu les afirmacions següents:

a) $A = \{x \mid \forall y \ \ \varphi_x(y) \in \{0,1\}\}$ no és recursiu ni enumerable recursivament.

b) $B = \{x \mid \forall y \ \ \varphi_x(y) \in \{0,1\} \ \wedge \ (\varphi_x(y) = 1 \Rightarrow \mathsf{M}_y(y)\!\downarrow)\}$ no és recursiu ni enumerable recursivament.

c) $C = \{x \mid \forall y \ \ \varphi_x(y) \in \{0,1\} \ \wedge \ (\varphi_x(y) = 1 \Longleftrightarrow \mathsf{M}_y(y)\!\downarrow)\}$ és recursiu.

4.4 Demostreu les propietats següents:

a) K no es pot reduir a $\overline{\mathrm{K}}$.

b) $\mathrm{K} \leqslant_{\mathrm{m}} \mathrm{K} \times \overline{\mathrm{K}}$ i $\overline{\mathrm{K}} \leqslant_{\mathrm{m}} \mathrm{K} \times \overline{\mathrm{K}}$.

c) $\mathrm{K} \times \overline{\mathrm{K}}$ no és e.r. i el seu complementari tampoc no ho és.

5 Alguns problemes indecidibles

En aquest capítol estudiem la indecidibilitat d'alguns problemes clàssics de la teoria de la calculabilitat. Si ens atenim a l'ordre cronològic, el primer que va aparèixer va ser el *problema dels mots*, formulat per Axel Thue l'any 1914 i resolt per Emile Post el 1947. També veiem el *problema de la correspondència* formulat pel mateix E. Post el 1946. Molt relacionat amb el primer tenim el *problema de la pertinença en gramàtiques de tipus 0*, estudiat per Noam Chomsky el 1959. Finalment, els problemes sobre gramàtiques incontextuals es formulen el 1961.

Comencem amb el problema dels mots de Thue i, a continuació, demostem l'equivalència entre aquest problema i el de la pertinença en gramàtiques de tipus 0, que és immediata. El problema de la correspondència de Post ens servirà d'eina per demostrar la indecidibilitat d'alguns problemes associats a gramàtiques incontextuals.

5.1 El problema dels mots de Thue

Vint anys abans que Kurt Gödel establís els seus primers resultats sobre problemes indecidibles, el matemàtic noruec Axel Thue intuïa la indecidibilitat d'alguns problemes de formulació molt senzilla, com ara el que presentem a continuació.

En el context d'aquesta secció, entenem per *regla de reescriptura* un parell ordenat (u, v) de mots sobre algun alfabet Σ de referència. Diem que passem d'un mot w_1 a un altre w_2, tots dos de Σ^*, per aplicació d'aquesta regla quan existeixin un prefix α i un sufix β comuns a w_1 i w_2 tals que $w_1 = \alpha u \beta$ i $w_2 = \alpha v \beta$. Ho representem de la forma $w_1 \rightarrow w_2$.

> Mots (WP)
> Donats una llista finita R de regles de reescriptura (un conjunt finit de parells de mots (u_i, v_i)) i dos mots u i v, determinar si és possible passar de u a v utilitzant les regles de reescriptura (un seguit de substitucions d'algun submot u_i pel seu corresponent v_i).

EXEMPLE 5.1 Considerem dues entrades del problema dels mots formades per un mateix conjunt de regles; en el primer cas, no hi ha possibilitat de passar del mot inicial al mot final; en el segon cas, la possibilitat existeix.

regles	$a \to aa \quad bb \to b \quad aaaaaa \to bbbb$
mots	inici b final a
solució	no n'hi ha

regles	$a \to aa \quad bb \to b \quad aaaaaa \to bbbb$
mots	inici aba final ba
solució	$aba \to aaba \to aaaba \to aaaaba \to aaaaaba \to aaaaaaba$ $\to bbbbba \to bbbba \to bbba \to bba \to ba$

Per tal de demostrar la indecidibilitat del problema dels mots, fem una reducció a partir del problema de la pertinença en màquines de Turing. Recordem que PERT $= \{\langle M, x\rangle \mid x \in \mathsf{L}(M)\}$.

TEOREMA 5.1 PERT $\leqslant_m$ WP

DEMOSTRACIÓ Sigui $\langle M, x\rangle$ una entrada de PERT en què $x \in \Sigma^*$ i $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$. Sabem que $x \in \mathsf{L}(M)$ sii $\blacktriangleright q_0 x \vdash^* \alpha q_F \beta$, per a alguns $\alpha, \beta \in \Gamma^*$, és a dir, hi ha una computació vàlida que comença a l'estat inicial i que acaba a l'estat acceptador.

A partir de M i x, construïm una entrada de WP sobre l'alfabet format per la reunió de Γ, Q i un símbol `#` que no pertany a Γ.

Regles de reescriptura:

per a tot $q, p \in Q$, tals que $q \neq q_F$ i per a tot $a, b, c \in \Gamma$

$$
\begin{array}{ll}
(qa, bp) & \text{si } \delta(q,a) = (p, b, \mathtt{d}) \\
(cqa, pcb) & \text{si } \delta(q,a) = (p, b, \mathtt{e}) \\
(q\#, bp\#) & \text{si } \delta(q,\mathtt{b}) = (p, b, \mathtt{d}) \\
(cq\#, pcb\#) & \text{si } \delta(q,\mathtt{b}) = (p, b, \mathtt{e})
\end{array}
$$

per a tot $a \in \Gamma$

$$
\begin{array}{l}
(aq_F, q_F) \\
(q_F a, q_F)
\end{array}
$$

Mot inicial: $u = \blacktriangleright q_0 x\#$ Mot final: $v = q_F\#$

Els mots que anem obtenint a cada aplicació d'una de les regles de reescriptura descriuen configuracions de la TM M, començant per la configuració inicial. Això és exactament així fins que eventualment s'arriba a una configuració que inclou l'estat acceptador. A partir d'aquell moment, el procés de reescriptura permet, mitjançant les dues últimes regles, esborrar tots els símbols que acompanyen q_F. És clar que, en aquestes condicions, si hi ha una seqüència de passos que permet a M passar de la configuració inicial a una d'acceptadora, llavors és possible obtenir el mot final a partir de l'inicial mitjançant les regles de reescriptura definides. Recíprocament, si el procés de reescriptura permet arribar al mot $q_F\#$, forçosament s'ha d'haver passat per un mot que descriu una configuració que conté q_F i que correspon, doncs, a un procés acceptador de M. □

EXEMPLE 5.2 Considerem la TM descrita a la figura 5.1 i ens proposem construir l'entrada del WP associat.

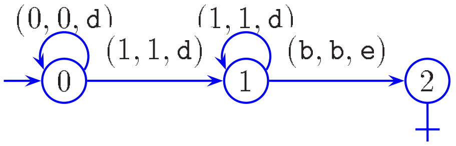


Fig. 5.1: *Graf de transicions d'una* TM

Per a cada mot $x \in \Sigma^*$, l'entrada del WP que correspon a l'entrada $\langle M, x\rangle$ de PERT, obtinguda a partir de la construcció del teorema anterior, és la següent:

$q_0 0 \to 0 q_0$	$q_0 1 \to 1 q_1$	$q_1 1 \to 1 q_1$	
$0 q_1 \mathtt{b} \to q_2 0 \mathtt{b}$ $0 q_1 \# \to q_2 0 \#$	$1 q_1 \mathtt{b} \to q_2 1 \mathtt{b}$ $1 q_1 \# \to q_2 1 \#$	$\mathtt{b} q_1 \mathtt{b} \to q_2 \mathtt{bb}$ $\mathtt{b} q_1 \# \to q_2 \mathtt{b} \#$	$\blacktriangleright q_1 \mathtt{b} \to q_2 \blacktriangleright \mathtt{b}$ $\blacktriangleright q_1 \# \to q_2 \blacktriangleright \#$
$0 q_2 \to q_2$	$1 q_2 \to q_2$	$\mathtt{b} q_2 \to q_2$	$\blacktriangleright q_2 \to q_2$
$q_2 0 \to q_2$	$q_2 1 \to q_2$	$q_2 \mathtt{b} \to q_2$	$q_2 \blacktriangleright \to q_2$
	$u = \blacktriangleright q_0 x \#$	$v = q_2 \#$	

En el cas del mot 011, els processos de càlcul i reescriptura són els que es descriuen a continuació:

Càlcul	$\blacktriangleright q_0 011 \vdash \blacktriangleright 0 q_0 11 \vdash \blacktriangleright 01 q_1 1 \vdash \blacktriangleright 011 q_1 \vdash \blacktriangleright 01 q_2 1$
Reescriptura	$\blacktriangleright q_0 011\# \to \blacktriangleright 0 q_0 11\# \to \blacktriangleright 01 q_1 1\# \to \blacktriangleright 011 q_1 \# \to \blacktriangleright 01 q_2 1\# \to$ $\blacktriangleright 0 q_2 1\# \to \blacktriangleright q_2 1\# \to q_2 1\# \to q_2 \#$

5.2 Gramàtiques de tipus 0

Una gramàtica de tipus 0 és una estructura de la forma $G = (V, \Sigma, P, S)$, en què V és un alfabet els símbols del qual s'anomenen *variables*; Σ és un altre alfabet els símbols del qual s'anomenen *terminals*; $S \in V$ és una variable, anomenada *inicial*; i P és un subconjunt finit de parells de $(V \cup \Sigma)^* \times (V \cup \Sigma)^*$, anomenats *produccions* i que representem en la forma $\alpha \to \beta$, amb $\alpha, \beta \in (V \cup \Sigma)^*$[1].

Diem que entre dos mots $u, v \in (V \cup \Sigma)^*$ hi ha una *derivació directa* quan podem expressar u i v en la forma $u = \gamma\alpha\delta$ i $v = \gamma\beta\delta$, de manera que $\alpha \to \beta$ sigui una producció de la gramàtica. Anàlogament amb el cas de les gramàtiques incontextuals, representem per $u \Rightarrow v$ la derivació directa entre u i v. També anàlogament amb aquell cas, diem que v

[1] De vegades, es defineixen les produccions de les gramàtiques de tipus 0 amb la condició addicional que el primer component contingui almenys una varible, és a dir, que $\alpha \in (V \cup \Sigma)^* - \Sigma^*$. És fàcil demostrar que les dues definicions són equivalents, en el sentit que qualsevol llenguatge generat per una gramàtica d'una de les dues formes pot ser-ho també per una de les de l'altra forma.

deriva de u, i ho representem per $u \stackrel{*}{\Rightarrow} v$, quan és possible passar de u a v per una seqüència finita (eventualment buida) de derivacions directes.

Diem que un mot $w \in \Sigma^*$ és *generat* per G quan existeix alguna derivació de la forma $S \stackrel{*}{\Rightarrow} w$. El conjunt de mots generats per una gramàtica G l'anomenem *llenguatge generat* per G i el representem per $\mathsf{L}(G)$. Un llenguatge s'anomena *de tipus 0* quan existeix una gramàtica de tipus 0 que el genera.

Anomenem *problema de la pertinença* en gramàtiques de tipus 0 el següent:

Pertinença en gramàtiques de tipus 0 (G0-PERT)
Donat un parell (G, w), format per una gramàtica G de tipus 0 i un mot w, determinar si $w \in \mathsf{L}(G)$, és a dir, si w és generat per G.

És clar que es tracta d'un cas particular del problema dels mots estudiat anteriorment. En efecte, només cal considerar les produccions de la gramàtica com a regles de reescriptura (en un alfabet que inclogui variables i terminals) i prendre com a parell de mots inicial i final el parell (S, w). Però, amb vista a demostrar la indecidibilitat del problema de la pertinença en gramàtiques, el que necessitem és la reducció recíproca. Vegem-la a continuació.

TEOREMA 5.2 *El problema de la pertinença en gramàtiques de tipus 0 és indecidible.*

DEMOSTRACIÓ Es tracta de partir d'una entrada del problema dels mots i construir a partir d'ella una entrada del de la pertinença en gramàtiques. Sigui (R, u, v) l'entrada del problema dels mots, en què R representa un conjunt finit de regles de reescriptura sobre un cert alfabet Σ, i en què u i v de Σ^* són els mots inicial i final. Sigui S un símbol que no pertany a Σ i considerem la gramàtica $G = (\{S\}, \Sigma, P, S)$, on el conjunt P de produccions conté totes les regles de R i, a més, la producció addicional $S \to u$. En aquestes condicions, considerem el problema de pertinença $v \in \mathsf{L}(G)$. És immediat constatar que aquesta relació és certa si i només si podem obtenir v a partir de u per aplicació de les regles de R. □

De fet, aquest resultat s'hauria pogut obtenir com a conseqüència immediata de la caracterització següent: els llenguatges de tipus 0 són exactament els llenguatges enumerables recursivament. És a dir, un llenguatge pot ser generat per una gramàtica de tipus 0 si i solament si hi ha alguna TM que el reconeix. Aquest fet, la demostració del qual no és molt difícil, el deixem com a exercici.

EXERCICI 5.1 Demostreu que els llenguatges de tipus 0 són exactament els llenguatges enumerables recursivament.
[*Esbós de demostració.* En un dels sentits, el de construir una TM a partir d'una gramàtica de tipus 0, es tracta de definir una NTM amb dues cintes, que conserva el mot d'entrada en una de les cintes i que al començament escriu el símbol inicial S a la segona cinta. La màquina pot efectuar indeterministament substitucions d'un submot a la segona cinta d'acord amb les produccions de la gramàtica. Si es produeix una coincidència entre les dues cintes, la màquina accepta. La construcció recíproca d'una gramàtica que correspongui a una TM donada segueix les mateixes idees donades a la secció precedent, que consisteixen a associar una producció a cada transició de la TM. Convé que les variables de la gramàtica constin de dos components. Inicialment és possible generar qualsevol mot de la forma $q_0[a_1, a_1][a_2, a_2] \ldots [a_n, a_n]$. A continuació, sobre el

segon component es va reproduint el procés de transicions fins a arribar, eventualment, a l'estat acceptador. En aquest punt es procedeix a esborrar tot el que no sigui el mot guardat als primers components.]

5.3 El problema de la correspondència de Post

Aquest problema és especialment interessant per dues raons. Primer, perquè constitueix un exemple de problema indecidible d'extrema sezillesa. No es requereix cap preparació matemàtica per abordar-lo i pot ser utilitzat per il·lustrar el significat del que són els problemes indecidibles a qualsevol persona no experta en lògica. En segon lloc, perquè permet, per via de reduccions, demostrar fàcilment la indecidibilitat de molts altres problemes.

> Correspondència de Post (PCP)
> Donades dues llistes d'igual longitud $A = (x_1, \ldots, x_n)$ i $B = (y_1, \ldots, y_n)$, de mots sobre un cert alfabet Σ, determinar si hi ha una seqüència finita no buida d'enters compresos entre 1 i n, $(i_1, \ldots, i_r)$, amb $r \geqslant 1$ tal que $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$.

Donem a continuació un parell d'exemples d'entrades del PCP, la primera admet solució i la segona no.

EXEMPLE 5.3 Considerem l'entrada següent d'un PCP:

	A	B
1	01	011
2	10	010
3	001	01
4	1011	10

Es tracta d'una entrada que té solució. En efecte, la seqüència d'eleccions $(1,2,2,1,3,4,3)$ dóna en ambdues llistes el mot 011010010011011001.

EXEMPLE 5.4 Considerem ara aquesta altra entrada:

	A	B
1	101	0100
2	100	10
3	0110	01
4	1010	10

Aquesta entrada no té cap solució. En efecte, els components de la llista A dels parells 1, 3 i 4 tenen un 1 de més que els components respectius de la llista B. Com que el parell 2 no compensa aquest efecte, no poden ser utilitzats. I és clar que amb el parell 2, tot sol, no fem res.

Observem que la concatenació de solucions és solució. Per tant, si una entrada té solució, en té infinites. El primer dels dos exemples precedents posa de manifest que el PCP és un problema que pertany a la classe RE, ja que el procés de verificació d'una solució és computable. En altres paraules, podem dissenyar una NTM que reconeix el conjunt d'entrades del PCP que tenen solució. Es tracta de considerar una màquina amb tres cintes, una que conté les dues llistes que conformen l'entrada donada i dues cintes més de treball. De manera indeterminista, la màquina efectua una seqüència d'eleccions adequada. Utilitzem les dues cintes de treball per anar escrivint en cada una d'elles el mot que correspon a cada una de les llistes definides a l'entrada donada. Finalment, comprovem que les dues cintes contenen el mateix mot.

En canvi, per al segon exemple hem utilitzat una demostració *ad hoc*. És clar que altres exemples requeriran demostracions de tipus diferent. Això és característic dels problemes indecidibles. Demostrem a continuació que el PCP ho és.

La demostració es basa en una reducció a partir del problema dels mots de Thue. Però, en comptes de fer directament la reducció WP $\leqslant_{\mathrm{m}}$ PCP, fem la demostració en dos passos, introduint prèviament un problema auxiliar, que ens simplificarà les coses.

Correspondència de Post modificada (MPCP)
Donades dues llistes d'igual longitud $A = (x_1, \ldots, x_n)$ i $B = (y_1, \ldots, y_n)$ de mots sobre un cert alfabet Σ, determinar si hi ha una seqüència finita d'enters compresos entre 1 i n, $(i_1, \ldots, i_r)$, eventualment buida, tal que $x_1 x_{i_1} \cdots x_{i_r} = y_1 y_{i_1} \cdots y_{i_r}$.

Observem que l'única diferència amb el problema de Post original és el requeriment que la seqüència ha de començar per un parell determinat (el primer de la llista).

PROPOSICIÓ 5.3 WP $\leqslant_m$ MPCP

DEMOSTRACIÓ Sigui (R, u, v) una entrada del problema dels mots, en què el conjunt de regles de reescriptura és de la forma $R = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ sobre un cert alfabet Σ, i en què u i v de Σ^* són els mots inicial i final. Sigui # un símbol que no pertany a Σ. Construïm, com a entrada del MPCP, el parell (A, B) de llistes següent.

A	B	
#	#u#	
u_i	v_i	per a tot $(u_i, v_i) \in R$
a	a	per a tot $a \in \Sigma$
#	#	
#v#	#	

L'ordre és irrellevant excepte per al parell de mots en la primera posició. La idea conductora d'aquesta construcció és formar un mot de la forma $\#x_1\#x_2\#\ldots\#x_m\#$, en què cada submot x_{i+1} s'obtingui del seu predecessor x_i per aplicació d'alguna regla de reescriptura. A més, tindrem $x_1 = u$ i $x_m = v$. Durant tot el procés de construcció, tret de l'últim pas, l'elaboració d'aquest mot a partir de la llista B va un pas avançat respecte del de la llista

A. Els parells de la forma (a, a), per a tot a de Σ, serveixen per completar els submots x_i un cop feta la reescriptura. Quan el mot construït amb la llista B arriba eventualment a la forma $\dots \#v$, s'utilitza l'últim parell per completar el mot sencer.

Les idees expressades al paràgraf precedent demostren que si l'entrada del WP té solució, també la té la del MPCP. Recíprocament, si les llistes (A, B) considerades admeten alguna solució, com que aquesta forçosament ha de començar pel primer parell, s'introdueix en la construcció que es fa amb mots de B un símbol # de més respecte de la que es fa amb mots de A. Així doncs, forçosament cal introduir en algun moment el parell $(\#v\#, \#)$, que és l'únic que permet reequilibrar el nombre d'aquests símbols. Però això implica que és possible obtenir v a partir de u en el WP donat. □

PROPOSICIÓ 5.4 MPCP $\leqslant_m$ PCP

DEMOSTRACIÓ Sigui (A, B) una entrada del MPCP, en què $A = (x_1, \dots, x_n)$ i $B = (y_1, \dots, y_n)$ són llistes de mots sobre un alfabet Σ. Suposem que aquests mots són de la forma $x_i = a_{i1} \cdots a_{ik_i}$ i $y_i = b_{i1} \cdots b_{ir_i}$ per a tot i tal que $1 \leqslant i \leqslant n$. Siguin \$ i # dos símbols que no pertanyen a Σ. Com a entrada del PCP construïm un nou parell de llistes (C, D) en què $C = (v_0, \dots, v_{n+1})$ i $D = (w_0, \dots, w_{n+1})$. Les noves llistes són:

	C	D
	$v_0 = \#a_{11}\# \cdots \#a_{1k_1}\#$	$w_0 = \#b_{11}\# \cdots \#b_{1r_1}$
per a $1 \leqslant i \leqslant n$	$v_i = a_{i1}\# \cdots \#a_{ik_i}\#$	$w_i = \#b_{i1}\# \cdots \#b_{ir_i}$
	\$	#\$

És clar que si $x_1 x_{i_1} \dots x_{i_k} = y_1 y_{i_1} \dots y_{i_k}$ és una solució de (A, B), tenim que

$$v_0 v_{i_1} \dots v_{i_k}\$ = w_0 w_{i_1} \dots w_{i_k}\#\$,$$

ja que la segona seqüència s'obté de la primera intercalant el símbol # entre cada dos símbols de Σ, afegint-hi # al començament i #\$ al final. I això constitueix una solució de (C, D).

Recíprocament, tota solució minimal de (C, D) ha de ser de la forma

$$v_0 v_{i_1} \dots v_{i_k}\$ = w_0 w_{i_1} \dots w_{i_k}\#\$,$$

perquè només la línia d'índex 0 conté dos mots que comencen amb el mateix símbol i només l'última línia conté dos mots que acaben amb el mateix símbol. Aleshores, és immediat veure que (A, B) admet la solució $x_1 x_{i_1} \dots x_{i_k} = y_1 y_{i_1} \dots y_{i_k}$. □

Com a conseqüència de les dues proposicions anteriors podem enunciar el teorema següent.

TEOREMA 5.5 *El problema de la correspondència de Post és indecidible.*

5.4 Problemes sobre gramàtiques incontextuals

Considerem les famílies de llenguatges més importants que s'estudien en teoria de llenguatges: finits, regulars, incontextuals ienumerables recursivament. De les primeres se'n dóna una descripció a l'apèndix **??**. Per a cada una d'elles, els llenguatges que la integren poden ser descrits mitjançant descriptors finits específics de la família considerada: llistes de mots per als llenguatges finits; autòmats finits, gramàtiques regulars o expressions regulars per als llenguatges regulars; gramàtiques incontextuals o autòmats amb pila per als llenguatges incontextuals; màquines de Turing per als enumerables recursivament. En aquesta progressió, cada família de llenguatges conté l'anterior. A mesura que creix l'amplitud de la família, augmenta el nombre de problemes que són indecidibles per a la família considerada. En el cas dels llenguatges regulars, els més senzills que s'estudien en teoria de llenguatges formals, la majoria dels problemes interessantssón decidibles. A l'altre extrem, el dels llenguatges RE, hem vist, en estudiar el teorema de Rice, que qualsevol propietat semàntica que no sigui trivial és indecidible.

En el cas dels llenguatges incontextuals (als quals ens referirem sovint com a CFL, igual que farem servir els noms CFG i PDA per referir-nos als seus descriptors, gramàtiques o autòmats, respectivament), coneixem algorismes que permeten resoldre'n alguns problemes bàsics. Recordem-los a continuació.

Buidor en incontextuals (CFG-EMPTINESS)
Donada una CFG, determinar si genera el conjunt buit.

Es tracta de saber si un CFL considerat conté o no algun mot, cosa que equival a dir, donada una CFG que generi el llenguatge, si el símbol inicial de la gramàtica és *fecund* o no. Els textos que tracten de models bàsics de computació (per exemple [9]) solen incloure algun algorisme per resoldre aquest problema que apareix en els processos de normalització de gramàtiques.

Finitud en incontextuals (CFG-FINITENESS)
Donada una CFG, determinar si genera un conjunt finit.

Es tracta de saber si un CFL considerat és finit o infinit, cosa que equival a decidir si un graf determinat està o no exempt de cicles. Es parteix d'una gramàtica *depurada* que generi el llenguatge (tota gramàtica pot ser depurada efectivament, és a dir, es disposa d'algorismes per eliminar les produccions nul·les, les produccions unàries i els símbols inútils) i associar a aquesta gramàtica un graf dirigit que té per vèrtexs les variables de la gramàtica i que té un arc entre dos vèrtexs si hi ha algua producció en què apareguin, a costat i costat, les variables associades a aquests dos vèrtexs.

Pertinença en incontextuals (CFG-PERT)
Donats una CFG i un mot, determinar si el mot és generat per la gramàtica.

És fàcil construir algorismes elementals (però de cost exponencial) que parteixen d'una gramàtica depurada i consideren totes les possibles derivacions que no ultrapassen la longitud del mot. Però també hi ha algorismes més eficients que resolen aquest problema (l'algorisme anomenat CYK té complexitat $O(n^3)$; pot trobar-se a [20]).

Tanmateix, hi ha una colla de propietats que són indecidibles en la família dels CFL. Dediquem aquesta secció a presentar-ne algunes i a demostrar la seva indecidibilitat. Estudiem, en primer lloc, un grup de propietats que, tot i ser indecidibles en el cas general dels CFL, sí que són decidibles si ens restringim a la subfamília dels llenguatges incontextuals que són deterministes (per als quals usem com a descriptors els seus reconeixedors, els autòmats amb pila deterministes o DPDA). Ens referim a aquests llenguatges com a DCFL.

5.4.1 Problemes decidibles en DCFL

Comencem presentant quatre problemes que tenen la particularitat de ser recursivament equivalents, és a dir, poden ser reduïts l'un a l'altre sigui quin sigui el parell considerat. Veurem, més endavant, que tots ells són decidibles sobre DCFL i indecidibles sobre CFL en general. És a dir, són decidibles si els descriptors que utilitzem per definir els CFL són DPDA, però són indecidibles si utilitzem CFG o PDA generals. Donem a continuació la definició d'aquests problemes suposant, per tal de concretar-ne l'entrada, que la descripció dels CFL es fa mitjançant CFG i que la definició dels llenguatges regulars es fa mitjançant DFA (autòmats finits deterministes). És clar, tanmateix, que podem considerar indistintament una CFG o un PDA per tal de descriure un CFL, perquè disposem de mètodes per construir qualsevol d'aquests dos descriptors a partir de l'altre.

> Igualtat amb Σ^* (EQTOT)
> Donada una CFG que genera un llenguatge incontextual L i un alfabet Σ, determinar si $L = \Sigma^*$.

> Igualtat amb regular (EQREG)
> Donada una CFG que genera un llenguatge incontextual L i un DFA que reconeix un llenguatge regular R, determinar si $L = R$.

> Inclusió d'un regular (INREG)
> Donada una CFG que genera un llenguatge incontextual L i un DFA que reconeix un llenguatge regular R, determinar si $R \subseteq L$.

> Inclusió de Σ^* (INTOT)
> Donada una CFG que genera un llenguatge incontextual L i un alfabet Σ, determinar si $\Sigma^* \subseteq L$.

PROPOSICIÓ 5.6 *Els quatre problemes* EQTOT, EQREG, INREG *i* INTOT *són recursivament equivalents.*

DEMOSTRACIÓ Fem successivament la reducció de cada problema al següent i de l'últim al primer.

a) EQTOT $\leqslant_{\mathrm{m}}$ EQREG.

Sigui (G, Σ) l'entrada d'EQTOT. N'hi ha prou de construir un DFA M que reconegui Σ^* i prendre com a entrada d'EQREG el parell (G, M).

b) EQREG $\leqslant_{\mathrm{m}}$ INREG.

Sigui (G, M) l'entrada d'EQREG, en què G és una CFG i M és un DFA. Sigui $L = \mathsf{L}(G)$ i $R = \mathsf{L}(M)$. Tenim un algorisme per saber si $L \subseteq R$. En efecte, es tracta de trobar el complementari de R (per via d'autòmats finits deterministes), fer la intersecció de $\overline{R}$ amb L (per via d'autòmat finit i autòmat amb pila) i determinar si aquesta intersecció és buida (per via de gramàtica que genera $L \cap \overline{R}$). És clar que $L \subseteq R \iff L \cap \overline{R} = \varnothing$. Podem construir ara l'entrada (G', M') d'INREG que correspon a (G, M) de la manera següent: si $L \subseteq R$, llavors fem $G' := G$ i $M' := M$; altrament fem G' i M' tals que $\mathsf{L}(G') = \varnothing$ i $\mathsf{L}(M') = \{a\}^*$. Tenim en qualsevol cas que $L = R \iff \mathsf{L}(M') \subseteq \mathsf{L}(G')$.

c) INREG $\leqslant_{\mathrm{m}}$ INTOT.

Sigui (G, M) l'entrada d'INREG. Sigui $L = \mathsf{L}(G)$ i $R = \mathsf{L}(M)$. Podem construir l'entrada (G', Σ) d'INTOT que correspon a (G, M) de la manera següent: siguin Σ_L i Σ_R els alfabets sobre els quals estan definits L i R, respectivament (aquests alfabets formen part dels descriptors respectius). Construïm G' tal que $\mathsf{L}(G') = L \cup \overline{R}$ i fem $\Sigma := \Sigma_L \cup \Sigma_R$. Aquestes construccions són efectives, com és immediat de verificar. I és clar que $R \subseteq L \iff \Sigma^* \subseteq \mathsf{L}(G')$.

d) INTOT $\leqslant_{\mathrm{m}}$ EQTOT.

Sigui (G, Σ) l'entrada d'INTOT. Prenem com a entrada d'EQTOT el parell (G', Σ), en què G' és tal que $\mathsf{L}(G') = \mathsf{L}(G) \cap \Sigma^*$. Podem fer la construció de G' tant indirectament, per via de PDA, com directament, per via de CFG. És obvi que $\Sigma^* \subseteq \mathsf{L}(G) \iff \mathsf{L}(G') = \Sigma^*$. □

TEOREMA 5.7 *Els quatre problemes* EQTOT, EQREG, INREG *i* INTOT *són decidibles per a* DCFL.

DEMOSTRACIÓ N'hi ha prou de demostrar que qualsevol dels quatre problemes és decidible. Elegim INTOT. Sigui (M, Σ) una entrada d'INTOT, en la qual M és un DPDA que reconeix un cert llenguatge L. Sigui Σ_L l'alfabet sobre el qual està definit L i que és un component estructural del DPDA. És clar que $\Sigma^* \subseteq L \iff \Sigma^* - L = \varnothing$. Però cal anar amb compte de calcular $\Sigma^* - L$ correctament. Hem de fer la intersecció de Σ^* amb el complementari de L *respecte de* $(\Sigma \cup \Sigma_L)^*$. Així doncs, es tracta de modificar primer el DPDA ampliant-ne l'alfabet i, a continuació, aplicar la construcció de l'autòmat complementari definida en el model dels DPDA. Finalment, cal fer la intersecció amb Σ^*. Una manera de saber si l'autòmat que en resulta reconeix el llenguatge buit és construir una gramàtica equivalent i aplicar-hi l'algorisme de la buidor. □

Hi ha d'altres problemes que també són decidibles per a DCFL i indecidibles en el cas general de CFL. N'és un el problema de la *regularitat*: donat un PDA, saber si el llenguatge que reconeix és regular. Un altre és el problema de la *igualtat*: donats dos PDA, saber si són equivalents, és a dir, si defineixen un mateix llenguatge. La demostració de la decidibilitat del primer en el cas de DCFL és deguda a Stearns [38], mentre que la del segon és deguda a Sénizergues [37]. Es tracta, en els dos casos, de demostracions força complicades que queden fora de l'abast d'aquest text.

5.4.2 Problemes indecidibles en CFL

En aquesta subsecció utilitzem el problema de la correspondència de Post per demostrar la indecidibilitat d'alguns problemes sobre gramàtiques i llenguatges incontextuals. Bàsicament ens centrarem en els problemes següents i en algunes variants dels mateixos.

Ambigüitat de gramàtiques (CFG-AMBIGUITY)
Donada una CFG, determinar si és ambigua.

Equivalència entre CFG (CFG-EQUIV)
Donades dues CFG, determinar si generen el mateix llenguatge.

Inclusió entre DCFL (DCFL-INCLUSION)
Donats dos autòmats amb pila deterministes, determinar si el llenguatge reconegut per un d'ells està inclòs en el reconegut per l'altre.

Inclusió entre CFL (CFL-INCLUSION)
Donades dues CFG, determinar si el llenguatge generat per una d'elles està inclòs en el generat per l'altra.

Comencem associant a cada entrada del PCP una terna de gramàtiques que ens serviran de base per efectuar les reduccions adequades als problemes considerats.

Considerem dues llistes d'igual longitud, $A = (x_1, \dots, x_n)$ i $B = (y_1, \dots, y_n)$, de mots sobre un cert alfabet Σ_0. Considerem un alfabet Σ_n format per n símbols auxiliars $\{a_1, \dots a_n\}$, cap dels quals no pertany a Σ_0. Fem $\Sigma = \Sigma_0 \cup \Sigma_n$. Associem a cada una d'aquestes llistes A i B les gramàtiques G_A i G_B definides de la manera següent:

- $G_A = (\{X_A\}, \Sigma, P_A, X_A)$
- $G_B = (\{X_B\}, \Sigma, P_B, X_B)$

on P_A i P_B contenen, respectivament i per a tot i, $1 \leqslant i \leqslant n$, les produccions següents:

- P_A: $X_A \to x_i X_A a_i \mid x_i a_i$
- P_B: $X_B \to y_i X_B a_i \mid y_i a_i$

Considerem, a més, la gramàtica següent, que correspon a la reunió dels llenguatges generats per G_A i G_B:

$$G_{A,B} = (\{S, X_A, X_B\}, \Sigma, P, S)$$

on P està format per totes les produccions de P_A i P_B i, a més, les dues produccions $S \to X_A \mid X_B$.

Representem per L_A i L_B els llenguatges generats per G_A i G_B, respectivament. Observem que tant L_A com L_B són llenguatges *deterministes* (no així $L_A \cup L_B$, generat per $G_{A,B}$, que no té per què ser-ho). Aquest fet serà utilitzat més endavant per posar de manifest que alguns dels problemes estudiats són indecidibles també per a DCFL.

Amb les notacions que acabem d'introduir, podem establir la proposició següent:

PROPOSICIÓ 5.8 *Els enunciats següents són equivalents:*

1. *El parell de llistes (A, B) admet solució com a entrada del* PCP.
2. $L_A \cap L_B \neq \varnothing$.
3. *La gramàtica $G_{A,B}$ és ambigua.*

DEMOSTRACIÓ Fem un procés de demostració circular.

a) $1 \Rightarrow 2$.

Suposem que (A, B) admet, com a solució, la seqüència $(i_1, \ldots, i_r)$. Això vol dir que $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$. Ara bé, el mot $x_{i_1} \cdots x_{i_r} a_{i_r} \cdots a_{i_1}$ és generat per G_A, mentre que el mot $y_{i_1} \cdots y_{i_r} a_{i_r} \cdots a_{i_1}$ ho és per G_B. I es té $x_{i_1} \cdots x_{i_r} a_{i_r} \cdots a_{i_1} = y_{i_1} \cdots y_{i_r} a_{i_r} \cdots a_{i_1}$.

b) $2 \Rightarrow 3$.

Sigui $w \in L_A \cap L_B$. Tenim $X_A \overset{*}{\Rightarrow} w$ i $X_B \overset{*}{\Rightarrow} w$, la primera derivació en G_A i la segona en G_B. Però aleshores tenim en $G_{A,B}$ dues derivacions diferents de w, que són $S \Rightarrow X_A \overset{*}{\Rightarrow} w$ i $S \Rightarrow X_B \overset{*}{\Rightarrow} w$.

c) $3 \Rightarrow 1$.

Com que les gramàtiques G_A i G_B són inambigües, l'única possibilitat que algun mot w tingui dues derivacions diferents en $G_{A,B}$ és que aquestes dues derivacions tinguin la forma $S \Rightarrow X_A \overset{*}{\Rightarrow} w$ i $S \Rightarrow X_B \overset{*}{\Rightarrow} w$. Això comporta que w és de la forma

$$w = x_{i_1} \cdots x_{i_r} a_{i_r} \cdots a_{i_1} = y_{j_1} \cdots y_{j_s} a_{j_s} \cdots a_{j_1}$$

i d'aquí resulta que $r = s$ i que $(i_1, \ldots, i_r) = (j_1, \ldots, j_s)$, cosa que constitueix una solució del parell de llistes (A, B) del PCP. □

La proposició anterior ens posa en condicions de fer les primeres demostracions de problemes indecidibles sobre CFL.

TEOREMA 5.9 *El problema* Ambigüitat de gramàtiques *és indecidible.*

DEMOSTRACIÓ La construcció de la gramàtica $G_{A,B}$ a partir del parell de llistes (A, B), acompanyada de l'equivalència entre els enunciats 1 i 3 de la proposició anterior, constitueix una reducció del PCP al problema CFG-AMBIGUITY. □

TEOREMA 5.10 *Els quatre problemes* EQTOT, EQREG, INREG *i* INTOT *enunciats a la subsecció precedent són indecidibles per a* CFL *generals.*

DEMOSTRACIÓ Com que els llenguatges L_A i L_B són DCFL, els seus complementaris $\overline{L}_A$ i $\overline{L}_B$ també ho són. Considerem $L = \overline{L}_A \cup \overline{L}_B$. Es tracta d'un CFL ja que és reunió de dos CFL, tot i que no té per què ser determinista com ells. Remarquem que podem construir de manera efectiva una gramàtica que generi L. Ara bé, és evident que

$$L_A \cap L_B = \varnothing \iff L = \Sigma^*.$$

Si el problema EQTOT (el de la igualtat entre un cert llenguatge L i un cert Σ^* donats) fos decidible, podríem decidir si $L_A \cap L_B = \varnothing$ i, per tant, en virtut de l'equivalència entre els enunciats 1 i 2 de la proposició anterior, podríem decidir també si el parell de llistes (A, B) és solució del PCP. Això demostra que EQTOT és indecidible per a CFL generals. L'equivalència recursiva d'EQTOT amb EQREG, INREG i INTOT demostra que tots ells són indecidibles en aquest cas. □

Remarquem que en la demostració precedent hem fet el pas del problema $L_A \cap L_B = \varnothing$ al seu complementari $L_A \cap L_B \neq \varnothing$. Tots dos problemes són indecidibles, però mentre que el segon és enumerable recursivament, el primer no ho és.

COROL·LARI 5.11 *El problema* **Equivalència de** CFG *és indecidible.*

DEMOSTRACIÓ Si aquest problema fos decidible, també ho seria el EQREG (igualtat entre un CFL i un llenguatge regular), que n'és un cas particular. □

TEOREMA 5.12 *El problema de la inclusió entre* DCFL *és indecidible. És a dir, donats dos* DCFL*, L_1 i L_2, és indecidible saber si $L_1 \subseteq L_2$.*

DEMOSTRACIÓ Com que els llenguatges L_A i L_B són DCFL, els seus complementaris també ho són. Ara bé, és immediat que

$$L_A \cap L_B = \varnothing \iff L_A \subseteq \overline{L}_B.$$

Per tant, si la inclusió entre DCFL fos decidible, aplicant-ho al cas de L_A i $\overline{L}_B$ resoldríem l'entrada (A, B) del PCP. □

COROL·LARI 5.13 *El problema* **Inclusió entre** CFL *és indecidible.*

La proposició 5.8 ens ha permès deduir la indecidibilitat d'uns quants problemes referents a DCFL i a CFL. Dediquem la resta d'aquesta subsecció al problema de la intersecció de llenguatges. Sabem que la intersecció de dos CFL, fins i tot si són deterministes, pot no ser CFL. La qüestió és si podem saber *quan* aquesta interesecció és CFL (o DCFL) i quan no ho és. Els problemes següents, que tractarem a continuació, estan estretament relacionats amb aquest plantejament.

Intersecció determinista de DCFL (DCFL-INTERSECTION-DCFL)
Donats dos autòmats amb pila deterministes, determinar si la intersecció dels llenguatges que reconeixen és també un DCFL.

Reunió determinista de DCFL (DCFL-UNION)
Donats dos autòmats amb pila deterministes, determinar si la reunió dels llenguatges que reconeixen és també un DCFL.

Intersecció incontextual de DCFL (DCFL-INTERSECTION-CFL)
Donats dos autòmats amb pila deterministes, determinar si la intersecció dels llenguatges que reconeixen és un CFL.

Intersecció incontextual de CFL (CFL-INTERSECTION)
Donades dues CFG, determinar si la intersecció dels llenguatges que generen també és un CFL.

Complementació incontextual (CFL-COMPLEMENT)
Donada una CFG, determinar si el complementari del llenguatge que genera també és un CFL.

Demostrarem que tots aquests problemes són també indecidibles, però per a això necessitem reprendre els instruments que hem definit en la proposició 5.8 i afegir-ne de nous.

Considerem de nou el parell de llistes $A = (x_1, \dots, x_n)$ i $B = (y_1, \dots, y_n)$ sobre l'alfabet Σ_0 introduïdes al començament d'aquesta subsecció, com també els alfabets Σ_n i Σ i les gramàtiques $G_A = (\{X_A\}, \Sigma, P_A, X_A)$ i $G_B = (\{X_B\}, \Sigma, P_B, X_B)$ que s'hi defineixen. Sigui $\#$ un símbol nou que no pertany a Σ. Considerem els llenguatges següents:

- $L_1 = \{w\#w^{\mathsf{R}} \mid w \in \Sigma_0^*\Sigma_n^*\}$
- $L_2 = \{w\#v^{\mathsf{R}} \mid w \in L_A \ \wedge \ v \in L_B\}$

Cal remarcar que tant L_1 com L_2 són DCFL. Això comporta que també són DCFL els seus complementaris, $\overline{L_1}$ i $\overline{L_2}$. En conseqüència, el llenguatge $L = \overline{L_1} \cup \overline{L_2} = \overline{L_1 \cap L_2}$ és un CFL.

PROPOSICIÓ 5.14 *Els enunciats següents són equivalents:*

1. *El parell de llistes (A, B) admet solució com a entrada del* PCP.
2. *$L_1 \cap L_2$ no és* CFL.

3. $L_1 \cap L_2$ *no és* DCFL.

4. $L_1 \cap L_2 \neq \varnothing$.

DEMOSTRACIÓ Fem, com sempre, un procés de demostració circular.

a) $1 \Rightarrow 2$.

Suposem que (A, B) admet, com a solució, la seqüència $(i_1, \dots, i_r)$. Així doncs, $x_{i_1} \cdots x_{i_r} = y_{i_1} \cdots y_{i_r}$. Fem $u = x_{i_1} \cdots x_{i_r}$ i $z = a_{i_r} \cdots a_{i_1}$. Tenim que $uz\#z^{\mathsf{R}}u^{\mathsf{R}} \in L_1 \cap L_2$. Però el fet que u sigui un mot solució comporta que per a qualsevol nombre $n \geqslant 1$, el mot u^n també és solució. Per la mateixa raó, resulta $u^n z^n\#(z^{\mathsf{R}})^n(u^{\mathsf{R}})^n \in L_1 \cap L_2$. Per tant, podem escriure la igualtat

$$(L_1 \cap L_2) \cap u^* z^* \# (z^{\mathsf{R}})^* (u^{\mathsf{R}})^* = \{u^n z^n \# (z^{\mathsf{R}})^n (u^{\mathsf{R}})^n \mid n \geqslant 1\}.$$

És fàcil veure (pel lema de bombament) que aquest llenguatge no és CFL. Per tant, $L_1 \cap L_2$ tampoc no pot ser-ho.

b) $2 \Rightarrow 3$.

Aquesta implicació és immediata, ja que tot DCFL és CFL.

c) $3 \Rightarrow 4$.

Aquesta implicació també és immediata, ja que el llenguatge buit és DCFL.

d) $4 \Rightarrow 1$.

De la definició de L_1 i L_2 es desprèn que un mot que pertanyi als dos llenguatges ha de ser de la forma $w\#w^{\mathsf{R}}$, amb $w \in L_A \cap L_B$. És a dir, $L_1 \cap L_2 = \{w\#w^{\mathsf{R}} \mid w \in L_A \cap L_B\}$. Així doncs, $L_1 \cap L_2 \neq \varnothing \iff L_A \cap L_B \neq \varnothing$. I ja sabem per la proposició 5.8 que això equival al fet que el parell (A, B) tingui solució. □

Les proposicions 5.8 i 5.14 tenen una estructura semblant. Totes dues estableixen que la intersecció de dos llenguatges, construïts a partir d'unes llistes A i B en correspondència, és no buida si i només si la correspondència té solució. Però els llenguatges L_A i L_B de la proposició 5.8 són massa senzills i no permeten assegurar que la seva intersecció sigui no incontextual (en cas de ser no buida) com passa amb els L_1 i L_2 de la proposició 5.14.

TEOREMA 5.15 *És indecidible determinar si la intersecció de dos* DCFL *és un* DCFL.

DEMOSTRACIÓ Com que els llenguatges L_1 i L_2 de la proposició 5.14 són DCFL, si fos decidible determinar si la intersecció de dos DCFL és DCFL, en virtut de l'equivalència entre els enunciats 1 i 3, també ho seria el PCP. □

COROL·LARI 5.16 *És indecidible determinar si la reunió de dos* DCFL *és un* DCFL.

DEMOSTRACIÓ Si aquesta reunió fos decidible, podríem saber si la intersecció de dos DCFL, L' i L'', és DCFL determinant si ho és la reunió de $\overline{L'}$ i $\overline{L''}$ □

TEOREMA 5.17 *És indecidible determinar si la intersecció de dos* DCFL *és un* CFL.

DEMOSTRACIÓ La demostració és anàloga a la del teorema 5.15, però ara en virtud de l'equivalència entre els enunciats 1 i 2 de la proposició 5.14. □

COROL·LARI 5.18 *És indecidible determinar si la intersecció de dos* CFL *és un* CFL.

DEMOSTRACIÓ Si la intersecció de dos CFL fos decidible, també ho seria la de dos DCFL, que n'és un cas particular. □

TEOREMA 5.19 *És indecidible determinar si el complementari d'un* CFL *és un* CFL.

DEMOSTRACIÓ Ja hem remarcat, en definir els llenguatges L_1 i L_2, que el llenguatge $L = \overline{L}_1 \cup \overline{L}_2$ és incontextual. Com que el seu complementari és $\overline{L} = L_1 \cap L_2$, la construcció de L a partir del parell (A, B) ens dóna, en virtud de l'equivalència entre els enunciats 1 i 2 de la proposició 5.14, una reducció del PCP a CFL-COMPLEMENT. □

5.5 Problemes

5.1 Trobeu una reducció directa del problema de la correspondència de Post al problema dels mots de Thue.

[*Indicació.* Podeu seguir una idea semblant a la construcció de les dues gramàtiques de la secció 5.4.2.]

5.2 Trobeu una reducció directa del problema de la correspondència de Post (PCP) al Problema modificat de la correspondència de Post (MPCP).

5.3 Demostreu que el PCP és decidible quan està definit sobre un alfabet uniliteral.

5.4 Demostreu que els llenguatges WP, G0-PERT i PCP, tots els quals sabem que corresponen a problemes indecidibles, són RE-complets.

5.5 Demostreu que els llenguatges EQTOT, EQREG, INREG i INTOT, tots els quals sabem que corresponen a problemes indecidibles quan es defineixen sobre llenguatges incontextuals, són co-RE-complets.

5.6 Demostreu que els problemes següents són indecidibles.

1. Donats una CFG G i un nombre natural k, determinar si G genera tots els mots de longitud més gran o igual que k.
2. Donades dues CFG, G_1 i G_2, determinar si els llenguatges que generen són complementaris.
3. Donades dues CFG, G_1 i G_2, determinar si el llenguatge generat per la primera està inclòs en el generat per la segona.
4. Donades dues CFG, G_1 i G_2, sobre un cert alfabet Σ, determinar si la reunió dels llenguatges que generen és Σ^*.

Demostreu que tots aquests problemes són decidibles per a llenguatges regulars.

5.7 Un autònat amb pila bidireccional (2DPDA) és un autòmat amb pila al qual li és permès de moure el capçal a dreta i esquerra, sense permetre-li escriure sobre la cinta d'entrada (vegeu [9] per a més detalls).

Demostreu que els problemes EQTOT, EQREG, INREG i INTOT són indecidibles per a llenguatges reconeguts per 2DPDA. Deduïu-ne que també són indecidibles els pronblemes de la buidor, la inclusió i la igualtat entre llenguatges d'aquesta classe.

[*Indicació*: Comenceu demostrant que la família de ls llenguatges reconeguts per 2DPDA és una àlgebra de Boole (amb les operacions de reunió, intersecció i complementació). Verifiqueu que els llenguatges L_A i L_B de la proposició 5.8 pertanyen a aquesta família.]

Bibliografia

[1] K. Agrawal, N. Kayal i N. Saxena. Primes in P. *43th IEEE Symposium on Foundations of Computer Science*, pàg. 302–309, 2002.

[2] K. Apple i W. Haken. "Every planar map is four colorable. Part I: Discharging", *Illinois J. Math.*, 21:429–490, 1977.

[3] K. Apple, W. Haken i J. Koch. "Every planar map is four colorable. Part II: Reducibility", *Illinois J. Math.*, 21:491–567, 1997.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kan, A. Marchetti-Spaccamela i M. Protasi. *Complexity and approximation. Combinatorial Optimization problems and their approximability properties.* Springer-Verlag, 1999.

[5] J. L. Balcázar, J. Díaz i J. Gabarró. *Structural Complexity I.* Springer-Verlag, Heidelberg, 2a edició, 1995.

[6] J. L. Balcázar. *Programación metódica.* McGraw Hill, 1993.

[7] J. M. Basart. *Grafs: fonaments i algorismes.* Publicacions de la UAB, 1994.

[8] N. Bourbaki. *Eléments de Mathématique. Théorie des ensembles.* Hermann, París, 1970.

[9] R. Cases i L. Màrquez. *Llenguatges, gramàtiques i autòmats. Curs bàsic.* Edicions UPC, 3a edició, 2000.

[10] J. Castro, F. Cucker, X. Messeguer, A. Rubio, L. Solano i B. Valles. *Iniciació a la programació.* McGraw Hill, 1992.

[11] A. Church. "An unsolvable problem of elementary number theory", *American Journal of Mathematics*, 56:345–363, 1936.

[12] A. Cobham. "The intrinsic computational difficulty of functions". *International Congress for Logic Methodology and Philosophy of Science.* North Holland, Amsterdam, pàg. 24–30, 1964.

[13] E. G. Coffman Jr., M. R. Garey i D. S. Johnson. "Approximation algorithms for bin-packing. A survey". D. S. Hochbaum ed., *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, Boston, pàg. 46–93, 1997.

[14] T. H. Cormen, C. E. Leiserson i R. L. Rivest. *Introduction to algorithms.* MIT Press, 1990.

[15] J. Díaz. L'algorisme de primalitat AKS. *Bulletí de la Societat Catalana de Matematiques*, Institut d'Estudis Catalans, 17(2):21–28, 2002.

[16] J. Díaz, M. Serna, P. Spirakis i J. Torán. *Paradigms for fast parallel approximability*. Cambridge University Press, 1997.

[17] J. Edmonds. "Paths, trees, and flowers". *Canad. J. Math.*, 17:449–467, 1965.

[18] M. R. Garey i D. S. Johnson. *Computers and Intractability. A guide to the Theory of NP-completeness*. Freeman, 1979.

[19] R. Greenlaw, H. Hoover i L. Ruzzo. *Limits to parallel computing*. Oxford University Press, Nova York, 1993.

[20] J. E. Hopcroft, R. Motwani i J. D. Ullman. *Introduction to automata theory, languages, and computation, 2nd edition*. Addison-Wesley, Reading, Mass., 2000.

[21] J. E. Hopcroft i J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, Mass., 1979.

[22] T. R. Jensen i B. T. Toft. *Graph Coloring Problems*. John Wiley & Sons, 1995.

[23] R. M. Karp. "Reducibility among combinatorial problems". R. E. Miller i J. W. Thatcher, ed., *Complexity of Computer Computations*. Plenum Press, Nova York, pàg. 85–103, 1972.

[24] S. C. Kleene. "General recursive functions of natural numbers". *Mathematische analen*, 112:727–742, 1936.

[25] Z. Michalewicz i D. B. Fogel. *How to solve it: modern heuristics*. Springer-Verlag, Berlín, 2000.

[26] Y. N. Moschovakis. *Notes on Set Theory*. Springer-Verlag, Nova York, 1994.

[27] R. Motwani i P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[28] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[29] E. Post. "Finite combinatory processes: Formulation I", *Journal of Symbolic Logic*, 1:103–105, 1936.

[30] V. Pratt. "Every prime has a succint certificate", *SIAM J. Comp.*, 4(3):214–220, 1975.

[31] M. Rabin. "Mathematical theory of automata". *Proceedings of Symposium in Applied Mathematics*, vol. 19, pàg. 153–157. American Mathematical Society, 1966.

[32] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, Nova York, 1967.

[33] S. Sahni i T. Gonzalez. "P-complete approximation problems", *Journal of the ACM*, 23:555–565, 1976.

[34] J. E. Savage. *Models of Computation. Exploring the power of computing*. Addison Wesley, 1998.

[35] M. Sipser. *Introduction to the Theory of Computation*. PWS, 1997.

[36] S. S. Skiena. *The algorithm design manual*. Springer-Verlag, 1998.

[37] G. Sénizergues. "$L(a) = L(b)$? decidability results from complete formal systems", *Theoretical Computer Science*, 251(1–2):1–166, 2001.

[38] R. Stearns. "A regularity test for pushdown machines", *Information and Control*, 11(3):323–340, 1967.

[39] L. J. Stockmeyer i A. R. Meyer. "Word problems requiring exponential time: preliminary report". *Fifth Annual ACM Symposium on Theory of Computing* (Austin, Texas, 1973). ACM, Nova York, pàg. 1–9, 1973.

[40] A. M. Turing. "On computable numbers, with an application to the Entscheindungsproblem", *Proceedings*, London Mathematical Society, 42:235–265, 1936. Correcció al vol. 43, pàg. 544–546.