UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Sobre la decidibilitat de problemes computacionals

Autor: Aniol Garcia i Serrano

Director: Dr. Juan Carlos Martínez Alonso
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 15 de gener de 2024

Abstract

The proposition in 1928 of David Hilbert's *Entscheidungsproblem*, led to the development of the theory of computability, which seeks to understand what can and cannot be computed. This work aims to be an introduction to this field. During its development, the fundamental concepts of automata theory and formal languages will be shown, along with Turing machines and the Church-Turing thesis, to finally see that there are undecidable problems and show some examples of them.

Resum

A partir del 1928, quan David Hilbert enuncia l'*Entscheidungsproblem*, es va desenvolupar la teoria de la computabilitat, que busca entendre què pot ser computat i què no. Aquest treball pretén ser una introducció en aquest camp. Durant el treball es veuen els conceptes fonamentals de la teoria d'autòmats i llenguatges formals, continuant amb les màquines de Turing i la tesi de Church-Turing, per acabar veient que hi ha problemes indecidibles i mostrar-ne alguns exemples.

2020 Mathematics Subject Classification. 03B25, 03D03, 03D05, 03D10, 20F10, 20M05, 68Q04, 68Q42, 68Q45

Agraïments

En primer lloc, voldria agrair a en Juan Carlos Martínez la seva confiança en acceptar dirigir el treball, i el guiatge, l'acompanyament i l'atenció al detall que m'ha ofert en tot moment.

També el meu agraïment al professor Santiago Zarzuela que, tot i que malauradament no ha estat possible, va accedir a la proposta de fer un treball conjunt de lògica i àlgebra. Gràcies de totes maneres per tenir-me present i pel material que m'ha fet arribar.

Finalment, donar les gràcies també a la meva família, per la seva infinita paciència i la seva dedicació.

Gràcies a tots!

Índex

Capítol 0

Introducció

Als inicis del segle XX, diversos matemàtics es van interessar sobre la qüestió de què pot ser computat per una màquina i què no. Aquesta recerca va suposar el desenvolupament de la teoria de la computablilitat, que és el pilar fonamental de la computació i informàtica modernes, i va influenciar profundament en la comprensió dels límits del raonament i sistemes matemàtics. Al bell mig d'aquesta teoria s'hi troba el concepte d'algorisme, un procediment que detalla pas a pas el procediment per resoldre un problema. Durant la dècada del 1930 matemàtics com Alan Turing, Alonzo Church, Kurt Gödel, Stephen Klenee, Rózsa Péter o Emil Post proposen diferents maneres de formalitzar aquest concepte (a partir de màquines de Turing, càlcul λ, funcions recursives generals...) i acaben demostrant que aquests sistemes són, en última instància, equivalents. Això culmina en la tesi de Church-Turing, que diu que tot algorisme pot ser computat per una màquina de Turing. Les discussions coetànies en fonaments de les matemàtiques porten a Hilbert a replantejar en el seu segon problema tres qüestions:

1. Són les matemàtiques *completes*?
2. Són les matemàtiques *consistents*?
3. Són les matemàtiques *decidibles*?

La publicació dels teoremes de la incompletesa de Gödel donaven resposta negativa a la primera qüestió, però quedaven obertes la qüestió de la consistència, que seria resolta per Gentzen el 1936, i l'*Entscheidungsproblem*, el problema de la decisió. Així, aquests lògics investiguen la noció de problemes indecidibles: problemes de decisió pels quals no hi ha cap algorisme que pugui donar-ne una resposta. Un dels problemes de decisió més famosos és el problema de la parada, demostrat indecidible per Turing el 1936. Aquest resultat té implicacions importants tant en la teoria com en la pràctica de la computació, demostrant l'existència de grans limitacions sobre el que pot ser computat.

Aquest treball pretén, en certa manera, seguir el camí que ens ha portat fins al descobriment dels problemes indecidibles.

El primer capítol, doncs, és introductori i dona les nocions de teoria d'autòmats

i gramàtiques formals que s'utilitzen durant el treball. S'introdueix el concepte de llenguatge, que serà fonamental durant el transcurs del treball, i es dona un generador i un reconeixedor de llenguatges. En l'últim apartat es demostren els teoremes d'extracció, veient que els models computacionals exposats no poden ser universals.

En el segon capítol es tracten les màquines de Turing, tot mostrant les enormes capacitats de còmput que sorgeixen d'un sistema senzill, per acabar enunciant la tesi de Church-Turing.

Els dos darrers capítols tracten ja de problemes de decisió. En el primer es donen idees de com utilitzar les eines desenvolupades per demostrar la decidibilitat de múltiples problemes, mentre que en el segon es fa l'invers, demostrant la indecidibilitat, entre altres, del problema de la parada. No ens aturem, però, amb el problema de la parada, sinó que també es desenvolupen sistemes i es mostren els problemes indecidibles més utilitzats per fer reduccions d'altres problemes. Finalment, en aquest mateix capítol, ens allunyem dels problemes purament computacionals per veure problemes indecidibles en l'àlgebra, mitjançant quatre pinzellades de semigrups i sistemes semi-Thue, per acabar donant idees de la demostració del problema de la paraula per semigrups i grups, problemes que tenen grans implicacions en camps com la topologia algebraica.

Finalment, s'afegeix un apèndix on hi ha més resultats i demostracions referents als capítols 1 i 2. Aquestes nocions no s'utilitzen durant el treball, però afegides sobretot al primer capítol, donen una visió molt més completa de la teoria d'autòmats i gramàtiques.

Capítol 1

Autòmats, gramàtiques i llenguatges

1.1 Llenguatges formals

Comencem per un **alfabet**, que no és res més que un conjunt finit de símbols. L'exemple potser més evident és el de l'alfabet $\{a, b, \ldots, z\}$ que estem acostumats a utilitzar, però també ho són les xifres $\{0, 1, \ldots, 9\}$ i qualsevol subconjunt d'aquest, com el de l'alfabet binari $\{0, 1\}$, que utilitzarem sovint en aquest text. De fet, els símbols d'un alfabet podrien ser qualsevol cosa, tot i que normalment utilitzem només caràcters imprimibles.

Una **paraula** sobre un alfabet Σ és una seqüència finita de símbols de Σ. Efectivament la paraula pot ser buida, que normalment escriurem com a λ (i per tant no utilitzarem λ com a símbol de l'alfabet). El conjunt de totes les possibles paraules sobre un alfabet Σ, incloent-hi la paraula buida, es denota per Σ^*. La llargada d'una paraula $x \in \Sigma^*$, representada per $|x|$, és definida pel nombre de símbols que conté contant repeticions. Dues paraules d'un alfabet es poden combinar per formar-ne una altra en l'operació de **concatenació**. Siguin $x, y \in \Sigma^*$, $x = a_1 a_2 \cdots a_n$ i $y = b_1 b_2 \cdots b_m$, amb $a_i \in \Sigma$ i $b_j \in \Sigma$. Aleshores, diem que $x \circ y = a_1 \cdots a_n b_1 \cdots b_m \in \Sigma^*$. Per simplificar la notació, quan no hi hagi possible confusió pel context, utilitzarem la notació xy en lloc de $x \circ y$.

Per concatenar una paraula amb ella mateixa, utilitzarem la notació exponencial, escrivint

$$x^i = \begin{cases} \lambda & \text{si } i = 0 \\ x^{i-1} \circ x & \text{si } i \geq 1 \end{cases}$$

Anomenem **llenguatge** a qualsevol conjunt de paraules d'un alfabet. És a dir, si Σ és un alfabet, aleshores $\forall L \subseteq \Sigma^*$, L és un llenguatge.

Exemple 1.

1. *Per a qualsevol alfabet Σ, els conjunts $\emptyset$, Σ, Σ^* són llenguatges. Cal tenir*

present que $\emptyset \neq \{\lambda\}$, el primer és un conjunt amb 0 elements i el segon és un conjunt amb la paraula buida com a únic element.

2. *Prenent l'alfabet català, el conjunt de paraules vàlides en català és un llenguatge.*
3. *Prenent $\Sigma = \{0, 1\}$, el conjunt de cadenes amb el mateix nombre de 0 que d'1 també és un llenguatge. Notem que, mentre que l'alfabet només té dos elements, el llenguatge descrit és infinit numerable*

Com que un llenguatge no deixa de ser un conjunt, totes les operacions pròpies d'aquests (unió, intersecció, complementari, diferència...) són vàlides. Siguin L_1 i L_2 llenguatges sobre un alfabet Σ. Definim:

- **Unió**: $L_1 \cup L_2 = \{x \mid x \in L_1 \vee x \in L_2\}$
- **Intersecció**: $L_1 \cap L_2 = \{x \mid x \in L_1 \wedge x \in L_2\}$
- **Complementari**: $\overline{L_1} = \{x \in \Sigma^* \mid x \notin L_1\}$
- **Diferència**: $L_1 \setminus L_2 = \{x \in L_1 \mid x \notin L_2\}$

A aquestes hi afegim les següents operacions:

- **Concatenació**: $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$, és a dir, el llenguatge de paraules formades a partir de la concatenació d'una paraula d'L_1 i una d'L_2. De manera anàloga a la concatenació de paraules, utilitzarem la notació exponencial per indicar la concatenació d'un llenguatge amb si mateix:

$$L^i = \begin{cases} \{\lambda\} & \text{si } i = 0 \\ L^{i-1} \circ L & \text{si } i \leq 1 \end{cases}$$

- **Clausura (de Kleene)**: $L^* = \{x_1 \circ x_2 \circ \cdots \circ x_n \mid n \geq 0 \wedge x_1, \ldots, x_n \in Ł\}$, és a dir, el conjunt de totes les possibles concatenacions de paraules d'L o, de manera equivalent, el conjunt de totes les paraules finites formades a partir de símbols de Σ.

A les operacions d'**unió**, **concatenació** i **clausura** les anomenem **operacions regulars**.

Si un llenguatge és finit, el podem definir totalment donant tots els seus elements. Normalment, però, ens interessaran llenguatges infinits, que els definirem mitjançant generadors que els descriuen. El següent exemple defineix, sobre l'alfabet binari, el llenguatge infinit de paraules formades per n zeros seguits d'n uns: $\{0^n 1^n \mid n \geq 1\}$

1.2 Autòmats finits

Però no havíem de parlar de computació? Per què ens interessen els llenguatges? Doncs bé, el primer model de computació que tractarem són els **autòmats finits**

deterministes i indeterministes, que tenen molt a veure amb un tipus de llenguatges anomenats **llenguatges regulars**.

1.2.1 Autòmats deterministes

Els autòmats finits són un model computacional on es té una màquina amb un conjunt d'estats i una cinta associada on hi ha una paraula sobre un cert alfabet Σ. A cada pas de càlcul es llegeix, de manera seqüencial, un únic caràcter de la paraula, es calcula l'estat següent en funció del caràcter llegit i l'estat actual i es passa a la següent cel·la de la cinta, repetint el procés fins a haver llegit tota la paraula. Aquest model computacional no dona cap sortida, només direm que l'autòmat **accepta** o **rebutja** la paraula segons l'estat en què hagi quedat en llegir i computar l'últim caràcter.

Definició 1. *Un autòmat determinista és una estructura $M = (K, \Sigma, \delta, q_0, F)$ on:*

- *K és un conjunt finit i no buit d'estats*
- *Σ és l'alfabet d'entrada*
- *$\delta\colon K \times \Sigma \to K$ és la funció de transició que, donat el caràcter d'entrada i estat actual, determina el següent estat*
- *$q_0 \in K$ és l'estat inicial*
- *$F \subseteq K$ és el conjunt d'estats acceptadors*

Exemple 2. *Podem considerar l'autòmat $M_D = \{\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \delta, q_0, \{q_0, q_2, q_3\}\}$ amb δ definida per*

$$\begin{aligned} \delta(q_0, a) &= q_1 & \qquad \delta(q_0, b) &= q_4 \\ \delta(q_1, a) &= q_4 & \qquad \delta(q_1, b) &= q_2 \\ \delta(q_2, a) &= q_3 & \qquad \delta(q_2, b) &= q_4 \\ \delta(q_3, a) &= q_1 & \qquad \delta(q_3, b) &= q_2 \\ \delta(q_4, a) &= q_4 & \qquad \delta(q_4, b) &= q_4 \end{aligned}$$

Sovint representarem M mitjançant un graf on:

- Els nodes són els estats
- Es marca l'estat inicial amb una fletxa incident al node
- Es marquen els estats finals amb un doble cercle o una creu
- Si $\delta(q, a) = p$, es fa una aresta dirigida que va des del node representant q fins al node representant p mitjançant una aresta amb etiqueta a.

Exemple 3. *Podem representar l'autòmat M_D de l'exemple 2 pel següent graf:*

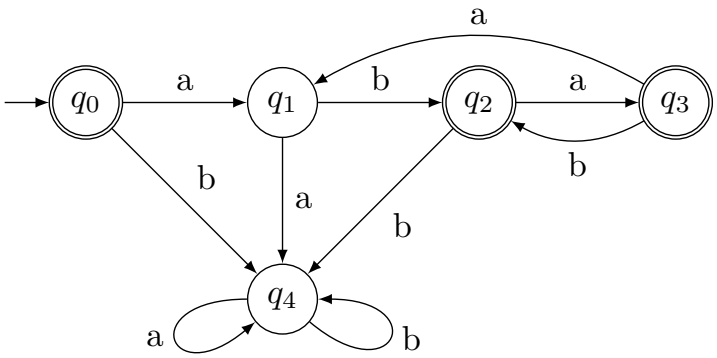


Figura 1.1: Exemple de representació d'autòmat determinista

Si en un pas de còmput l'autòmat es troba en l'estat $p \in K$ i la paraula que queda per llegir és $x \in \Sigma^*$, diem que $px \in K\Sigma^*$ és una **configuració de M**[1]. A més, si px i qy, amb $x = a_1 \dots a_n$, són configuracions de M, direm que px **produeix** qy **en un pas** si $\delta(p, a_1) = q$ i $y = a_2 \dots a_n$, i ho escriurem $px \vdash_M qy$. Si es pot passar d'una configuració px a una configuració qy amb un nombre finit de passos de càlcul, direm que px **produeix** qy i ho escriurem $px \vdash_M^* qy$.

Diem que una paraula $x \in \Sigma^*$ és **reconeguda o acceptada** per un autòmat M si existeix un estat $q \in F$ tal que $q_0 x \vdash_M^* q$, és a dir, que havent llegit tota la paraula l'autòmat acaba en un estat d'F. Si l'autòmat llegeix tota la paraula i acaba en un estat que no és d'F, direm que la paraula **no ha estat reconeguda o ha estat rebutjada**. Aleshores, podem definir el **llenguatge reconegut per** M com

$$L(M) = \{x \in \Sigma^* \mid x \text{ és reconeguda per } M\}$$

.

Exemple 4. *El llenguatge acceptat per l'autòmat M_D de l'exemple 2 és*

$$L(M_D) = (aba \cup ab)^*$$

1.2.2 Autòmats indeterministes

Definició 2. *Un autòmat indeterminista és una estructura $M = (K, \Sigma, \Delta, q_0, F)$ on:*

- *K és un conjunt finit i no buit d'estats*
- *Σ és l'alfabet d'entrada*
- *$\Delta\colon K \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(K)$ és la funció de transició, on $\mathcal{P}(K)$ denota el conjunt de les parts de K (i.e. el conjunt de tots els subconjunts de K)*
- *$q_0 \in K$ és l'estat inicial*
- *$F \subseteq K$ és el conjunt d'estats finals*

[1] A vegades escriurem una configuració px com (p, x), per facilitar la lectura

Notem que la definició és igual que pels autòmats deterministes excepte per la funció de transició. En els autòmats deterministes es fa exactament una transició per a cada símbol d'entrada que es llegeix, mentre que els indeterministes permeten fer zero, una o més transicions per cada símbol llegit. A més, es permeten transicions amb la paraula buida, passant d'un estat a un altre sense llegir cap símbol d'entrada. Per aquest motiu el conjunt d'arribada de la funció de transició Δ és $\mathcal{P}(K)$, ja que per un estat i un símbol d'entrada hi poden haver múltiples estats d'arribada.

Les definicions de **configuració**, **produeix en un pas de còmput**, **produeix**, **reconeix** i **llenguatge reconegut** són anàlogues a les dels autòmats deterministes.

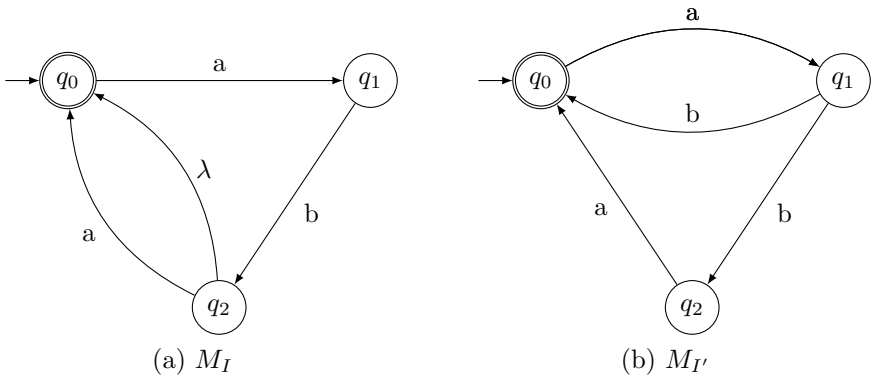
Exemple 5. *Considerem l'autòmat* $M_I = \{\{p_0, p_1, p_2\}, \{a, b\}, \Delta, p_0, \{p_0\}\}$*, amb*

$$\begin{aligned}\Delta(p_0, a) &= \{p_1\}\\ \Delta(p_1, b) &= \{p_2\}\\ \Delta(p_2, a) &= \{p_0\}\\ \Delta(p_2, \lambda) &= \{p_0\}\end{aligned}$$

Observi's que la transició λ *entre* p_2 *i* p_0 *es pot eliminar si s'afegeix una transició doble: sigui* $M_{I'} = \{\{p_0, p_1, p_2\}, \{a, b\}, \Delta', p_0, \{p_0\}\}$ *amb*

$$\begin{aligned}\Delta'(p_0, a) &= \{p_1\}\\ \Delta'(p_1, b) &= \{p_2, p_0\}\\ \Delta'(p_2, a) &= \{p_0\}\end{aligned}$$

Aleshores M_I *i* $M_{I'}$ *són autòmats equivalents en el sentit que reconeixen el mateix llenguatge. De fet, el llenguatge que reconeixen és precisament* $L = (aba \cup ab)^*$*, el mateix que l'autòmat determinista* M_D *de l'exemple 2, però aquests només necessiten 3 estats per fer-ho.*



(a) M_I (b) $M_{I'}$

Figura 1.2: Representacions d'autòmats indeterministes equivalents

Malgrat que sembli que amb les transicions λ i permetent fer diferents nombres de transicions aquest model és més potent que els autòmats deterministes, veurem que de fet aquests dos models són equivalents.

Necessitem, però, el concepte de **clausura d'un estat**, que fa referència a les transicions amb λ i que no teníem en el cas d'autòmats deterministes.

Definició 3. *Sigui $M = (K, \Sigma, \Delta, q_0, F)$ un autòmat indeterminista i sigui $p \in K$ un estat d'M. Definim la clausura de p com*

$$\Lambda(p) = \{q \in K \mid p\lambda \vdash_M^* q\}$$

és a dir, el conjunt d'estats a què es pot arribar des de l'estat q mitjançant una o més transicions λ, incloent-hi el mateix estat q.

Teorema 1. *Per a tot llenguatge L existeix un autòmat determinista M_D tal que $L = L(M_D)$ si i només si existeix un autòmat indeterminista M_I tal que $L(M_I) = L$.*

Demostració.

- $\boxed{\Longrightarrow}$: Sigui $M_D = (K, \Sigma, \delta, q_0, F)$. Prenent $M_I = (K, \Sigma, \Delta, q_0, F)$ amb

  $$\begin{aligned} \Delta\colon K \times \Sigma &\longrightarrow \mathcal{P}(K) \\ qx &\longmapsto \{\delta(q, x)\} \end{aligned}$$

  Fent la identificació $\{\delta(q, x)\} \mapsto \delta(q, x)$ tenim que les funcions de transició Δ i δ són equivalents, tal com la resta de components de M_I i M_D, de manera que els autòmats són idèntics i, per tant, reconeixen el mateix llenguatge.

- $\boxed{\Longleftarrow}$: La idea d'aquesta demostració és pensar que l'autòmat indeterminista M_I, en lloc d'estar en un únic estat després de llegir una certa entrada, està alhora en tot el conjunt d'estats als quals es pot arribar havent llegit l'entrada. Amb aquesta idea es pot generar un autòmat determinista els estats del qual siguin tots els possibles conjunts d'estats de l'indeterminista, i modelar δ en funció del conjunt d'estats a què es pugui arribar llegint un símbol.

  Sigui $M_I = (K, \Sigma, \Delta, q_0, F)$ un autòmat indeterminista. Construirem un autòmat determinista $M_D = (K', \Sigma, \delta, q_0', F')$ on:

  - $K' = \mathcal{P}(K)$ ja que cada estat de M_D és un conjunt d'estats de M_I
  - Si $X \in K'$ i $a \in \Sigma$, definim

    $$\begin{aligned} \delta(X, a) &= \bigcup \{\Lambda(q) \mid \exists p \in X \text{ tal que } q \in \Delta(p, a)\} \\ &= \{q \in K \mid q \in \Lambda\left(\Delta(r, a)\right) \text{ per algun } r \in X\} \end{aligned}$$

    Observem que això està ben definit, ja que el conjunt d'arribada de δ és K', que és precisament $\mathcal{P}(K)$. Notem, també, que és possible que $\delta(X, a) = \emptyset$. Això, però, no suposa cap problema ni que δ no estigui ben definida, ja que $\emptyset \in K' = \mathcal{P}(K)$. Aquest conjunt es pot obtenir de la següent manera:

1. Prendre tots els estats $q \in K$ pels quals existeix $p \in X$ tal que $q \in \Delta(p, a)$
2. Calcular $\Lambda(q)$ per a tots els estats q obtinguts en el pas (1)
3. Prendre la unió de tots els conjunts $\Lambda(q)$ del pas (2)

⋆ $q'_0 = \Lambda(q_0)$

⋆ $F' = \{X \in K' \mid X \cap F \neq \emptyset\}$

Ara ens cal veure que, efectivament, M_D és un autòmat determinista i que és equivalent a M_I. Que M_D és determinista és evident per construcció i amb les observacions sobre δ ja fetes. Per veure que els autòmats són equivalents, ens cal demostrar que per a qualsevol paraula $w \in \Sigma^*$ i per a qualssevol estats $p, q \in K$ tenim que

$$qw \vdash^*_{M_I} p\lambda \iff \Lambda(q)w \vdash^*_{M_D} P\lambda \tag{1.1}$$

per a algun $P \in \mathcal{P}(K)$ tal que $p \in P$.

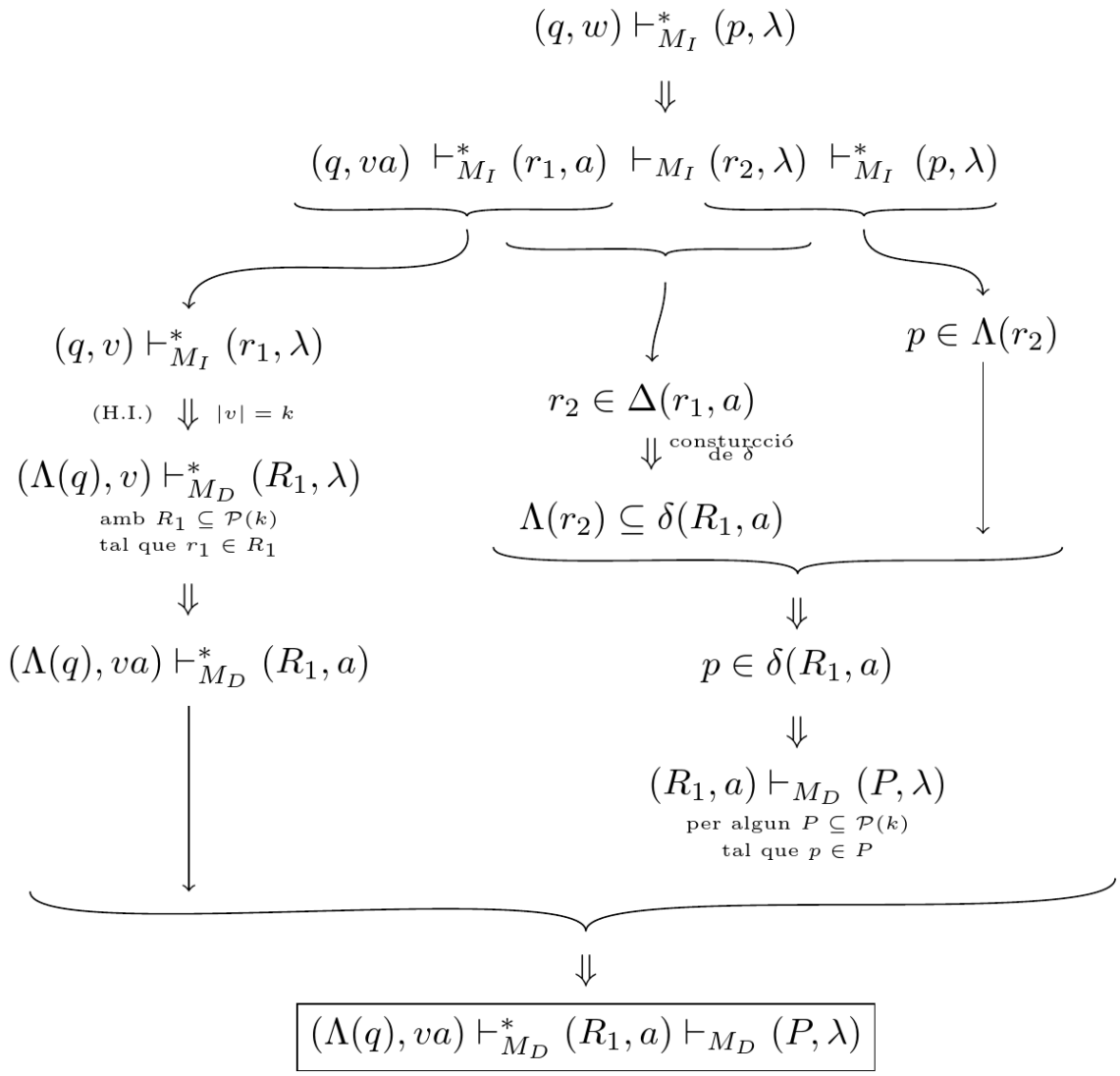
Demostrant aquesta proposició, el teorema se segueix: per veure que M_D i M_I són equivalents cal veure que els llenguatges que reconeixen són equivalents. Prenent $w \in \Sigma^*$, tenim que $w \in L(M_I) \underset{(def)}{\iff} q_0 w \vdash^*_{M_I} f\lambda$ per a algun $f \in F \underset{(1.1)}{\iff} \Lambda(q_0)w \vdash^*_{M_D} P\lambda$ per a algun $P \in K'$ que, per definició, és $w \in L(M_I)$.

Per demostrar la proposició utilitzarem inducció sobre $|w|$:

⋆ $\boxed{|w| = 0}$: O, de manera equivalent, $w = \lambda$. Cal veure doncs que $q\lambda \vdash^*_{M_I} p\lambda \iff \Lambda(q)\lambda \vdash^*_{M_D} P\lambda$ per a algun $P \in \mathcal{P}(K)$ tal que $p \in P$. Ara bé, tenim que $q\lambda \vdash^*_{M_I} p\lambda \iff p \in \Lambda(q)$. D'altra banda, com que M_D és un autòmat determinista, $\Lambda(q)\lambda \vdash^*_{M_D} P\lambda \iff P = \Lambda(q)$ per a algun P tal que $q \in P \iff p \in \Lambda(q)$.

⋆ $\boxed{|w| = k + 1}$: Suposem que la proposició és certa per a paraules de llargada com a màxim k per a alguna $k \geq 0$, i vegem que és certa per a paraules de longitud $k + 1$. Sigui $w = va$, amb $v \in \Sigma^*$ tal que $|v| = k$, i sigui $a \in \Sigma$.

$\implies$ Suposem $qw \vdash^*_{M_I} p\lambda$. Aleshores existeixen estats r_1 i r_2 tals que



$\Longleftarrow$ Suposem que $\Lambda(q)va \vdash^*_{M_D} R_1 a \vdash_{M_D} P\lambda$ per a algun P tal que $p \in P$ i algun R_1 tal que $\delta(R_1, a) = P$. Per definició, $\delta(R_1, a) = \cup\{\Lambda(r_2) \mid \exists r_1 \in R_1 \text{ tal que } r_2 \in \Delta(r_1, a)\}$. Com que $p \in P = \delta(R_1, a)$, existeix algun r_2 tal que $p \in \Lambda(r_2)$, i per a algun $r_1 \in R_1$ es té que $r_2 = \Delta(r_1, a)$. Aleshores tenim que $r_2\lambda \vdash^*_{M_I} p\lambda$. Per hipòtesi d'inducció, $qv \vdash^*_{M_I} r_1\lambda$ i, per tant $qva \vdash^*_{M_I} r_1 a \vdash_{M_I} r_2\lambda \vdash^*_{M_I} p\lambda$, que és el que volíem veure.

□

Així doncs, hem demostrat que els autòmats deterministes i indeterministes són equivalents i ho hem fet amb una demostració constructiva, de manera que també tenim un procediment per passar d'un a l'altre. D'ara endavant, quan ens referim a **autòmats finits** ens estarem referint indistintament a deterministes o indeterministes.

Els llenguatges que són reconeguts pels autòmats finits s'anomenen **llenguatges regulars** i es poden descriure també utilitzant les **expressions regulars**, que representen una gramàtica formal. A l'apèndix A.1 se'n pot trobar la definició i la demostració de l'equivalència entre llenguatges regulars i els generats mitjançant expressions regulars.

Proposició 1. *La classe dels llenguatges reconeguts per autòmats finits és tancada respecte a les operacions d'unió, concatenació i clausura de Kleene. En altres paraules, si tenim dos autòmats finits M_1 i M_2, aleshores existeixen autòmats $M_\cup$, $M_\circ$ i M_* tals que $L(M_\cup) = L(M_1) \cup L(M_2)$, $L(M_\circ) = L(M_1) \circ L(M_2)$ i $L(M_*) = L(M_1)^*$.*

Demostració. Siguin $M_1 = (K_1, \Sigma, \Delta_1, q_1, F_1)$ i $M_2 = (K_2, \Sigma, \Delta_2, q_2, F_2)$ dos autòmats indeterministes qualssevol.

- **Unió**: Volem construir un autòmat indeterminista M tal que $L(M) = L(M_1) \cup L(M_2)$. Per fer-ho, mantindrem els dos autòmats originals en paral·lel, afegint un estat inicial amb una transició λ cap a cadascun dels estats inicials de M_1 i M_2, fent que al primer pas de càlcul es triï si la paraula és de $L(M_1)$ o $L(M_2)$. Els estats acceptadors d'M seran la unió dels d'M_1 i M_2, permetent que es reconeguin les paraules tant d'un llenguatge com de l'altre. Definim $M = (K, \Sigma, \Delta, s, F)$ on:
  - $K = K_1 \cup K_2 \cup \{s\}$
  - $$\Delta(p,a) = \begin{cases} \Delta_1(p,a) & \text{si } p \in K_1 \\ \Delta_2(p,a) & \text{si } p \in K_2 \\ \{q_1, q_2\} & \text{si } p = s \text{ i } a = \lambda \\ \emptyset & \text{si } p = s \text{ i } a \neq \lambda \end{cases}$$
  - $F = F_1 \cup F_2$
- **Concatenació**: Volem construir un autòmat indeterminista M tal que $L(M) = L(M_1) \circ L(M_2)$. Per fer-ho posarem M_1 i M_2 seguits, afegint una transició λ des de cadascun dels estats acceptadors de F_1 cap a l'estat inicial de M_2 q_2. Així, en acabar de reconèixer una paraula de $L(M_1)$ es pot continuar reconeixent una paraula de $L(M_2)$. Formalment, $M = (K_1 \cup K_2, \Sigma, \Delta, q_1, F_2)$, amb
$$\Delta(p,a) = \begin{cases} \Delta_1(p,a) & \text{si } p \in K_1 \\ \Delta_2(p,a) & \text{si } p \in K_2 \\ \Delta_1(p,a) \cup \{q_2\} & \text{si } p \in F_1 \text{ i } a = \lambda \end{cases}$$
- **Clausura de Kleene**: Volem construir un autòmat indeterminista M tal que $L(M) = L(M_1)^*$. Per fer-ho, afegirem una transició λ des dels estats d'F_1 cap a l'estat q_1, de manera que en acabar de reconèixer una paraula pugui tornar a començar el procés de reconèixer-ne una altra. A més, per permetre acceptar la paraula buida λ, afegirem un estat que serà acceptador i inicial

amb una transició λ a q_1. Així, tindrem $M = (K \cup \{s\}, \Sigma, \Delta, s, F \cup \{s\})$ amb

$$\Delta(p,a) = \begin{cases} \{q_1\} & \text{si } p = s \text{ i } a = \lambda \\ \emptyset & \text{si } p = s \text{ i } a \neq \lambda \\ \Delta_1(p,a) \cup \{q_1\} & \text{si } p \in F \text{ i } a = \lambda \\ \Delta_1(p,a) & \text{en cas contrari} \end{cases}$$

□

Observació. *També es pot demostrar que és tancada per les operacions de complement i intersecció, però aquestes no les farem servir.*

1.3 Gramàtiques incontextuals

En la secció anterior hem definit els llenguatges regulars, però aquests llenguatges són molt limitats. Més endavant podrem demostrar que, per exemple, el llenguatge

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

no és regular. Aquest llenguatge pertany a la classe de **llenguatges incontextuals**, que definirem a continuació. En aquest cas, definirem els llenguatges incontextuals a partir d'una **gramàtica incontextual**, que actua com a generadora del llenguatge, definint les regles que fan que sigui vàlid.

Definició 4. *Una **gramàtica incontextual** és una estructura $G = (V, \Sigma, P, S)$ on:*

- *V és un alfabet, els símbols del qual anomenarem **variables***
- *Σ és un alfabet de símbols disjunt de V, els elements del qual anomenarem **terminals***
- *$P \subseteq V \times (V \cup \Sigma)^*$ els elements del qual anomenarem **produccions***
- *$S \in V$ és la variable inicial*

Per a $A \in V$ i $x \in (V \cup \Sigma)^*$, si $(A, x) \in P$, escriurem $A \longrightarrow x$. En una paraula $w \in (V \cup \Sigma)^*$ que contingui A, aplicar una producció $A \longrightarrow x$ consisteix a canviar una de les aparicions de A en w per x. Per simplificar la notació, si tenim les produccions $(A, x_1), \ldots, (A, x_n) \in P$, escriurem $A \longrightarrow x_1 | \ldots | x_n$.

Definició 5. *Sigui $G = (V, \Sigma, P, S)$ una gramàtica incontextual. Si $u, v \in (V \cup \Sigma)^*$, direm que v **es deriva (o és una derivació) de** u **en un pas** i escriurem $v \Rightarrow_G u$ si podem obtenir v aplicant una producció de P sobre u. Direm que v **es deriva (o és una derivació) de** u i ho escriurem $v \Rightarrow_G^* u$ si es pot obtenir v aplicant un nombre finit (possiblement 0) de produccions de P sobre u. Direm que v **es deriva de** u **per l'esquerra en un pas** si v es pot derivar de u en un pas substituint la primera variable de u (i.e. la variable situada més a l'esquerra de la paraula u). Respectivament, direm que v **es deriva de** u **per la dreta en un pas***

si v es pot derivar de u en un pas substituint l'última variable de u (i.e. la variable situada més a la dreta de u). Escriurem v es deriva de u per l'esquerra en un pas (respectivament dreta) com $u \overset{L}{\Rightarrow} v$ (respectivament $u \overset{R}{\Rightarrow} v$). Direm també que v es deriva de u per l'esquerra (resp. dreta) si existeixen paraules $u_1, \dots u_n \in V^*$ tals que $u \overset{L}{\Rightarrow} u_1 \overset{L}{\Rightarrow} \dots \overset{L}{\Rightarrow} u_n \overset{L}{\Rightarrow} v$ (resp. amb $\overset{R}{\Rightarrow}$), i ho escriurem $u \overset{L}{\Rightarrow}^* v$ (resp. $u \overset{R}{\Rightarrow}^* v$).

Exemple 6. *Podem donar una gramàtica incontextual que generi, per exemple, totes les expressions aritmètiques sintàcticament correctes involucrant les operacions $+$ i $*$. Sigui $G = \{V, \Sigma, P, E\}$ on*

- $V = \{E, T, F\}$
- $\Sigma = \{+, *, (,), id\}$
- *Les produccions de P són*

$$E \longrightarrow E + T \tag{P1}$$
$$E \longrightarrow T \tag{P2}$$
$$T \longrightarrow T * F \tag{P3}$$
$$T \longrightarrow F \tag{P4}$$
$$F \longrightarrow (E) \tag{P5}$$
$$F \longrightarrow id \tag{P6}$$

Les variables E, T i F representen expressions, termes i factors respectivament i el terminal id podria representar qualsevol valor numèric. Aleshores aquesta gramàtica permet generar, per exemple, la paraula $(id + id) * id$, però no permet generar expressions sintàcticament incorrectes com $*id+)($.

Exemple 7. *Prenent la gramàtica G de l'exemple 6, podem donar la següent derivació de la paraula $(id + id) * id$:*

$$E \underset{P2}{\overset{L}{\Rightarrow}} T \underset{P3}{\overset{L}{\Rightarrow}} T * F \underset{P6}{\overset{R}{\Rightarrow}} T * id \underset{P4}{\overset{L}{\Rightarrow}} F * id \underset{P5}{\overset{L}{\Rightarrow}} (E) * id \underset{P1}{\overset{L}{\Rightarrow}} (E + T) * id \underset{P4}{\overset{R}{\Rightarrow}} (E + F) * id$$
$$\underset{P6}{\overset{R}{\Rightarrow}} (E + id) * id \underset{P2}{\overset{L}{\Rightarrow}} (T + id) * id \underset{P4}{\overset{L}{\Rightarrow}} (F + id) * id \underset{P6}{\overset{L}{\Rightarrow}} (id + id) * id$$

Definició 6. *Sigui $G = (V, \Sigma, P, S)$ una gramàtica incontextual. Definim el **llenguatge generat per G** com*

$$L(G) = \{x \in \Sigma^* \mid S \Rightarrow_G^* x\}$$

*Direm que un llenguatge L és **incontextual** si existeix una gramàtica incontextual G tal que $L = L(G)$.*

Una eina que resulta molt útil quan es tracta amb gramàtiques incontextuals són els arbres de derivació, que permeten veure quines derivacions s'utilitzen per anar de la variable inicial a una paraula qualsevol. A més, en tractar-se d'arbres en el sentit de teoria de grafs, ens podem servir de les seves propietats per demostrar característiques dels llenguatges incontextuals.

Definició 7. *Sigui $G = (V, \Sigma, P, S)$ una gramàtica incontextual. Si $x \in L(G)$ a la derivació $S \Rightarrow_G^* x$ se li pot assignar un arbre de la següent manera:*

1. *S és l'arrel de l'arbre.*
2. *Cada node intern de l'arbre correspon a una variable que ha estat substituïda durant el procés de derivació.*
3. *Les fulles són terminals que formen la paraula x*

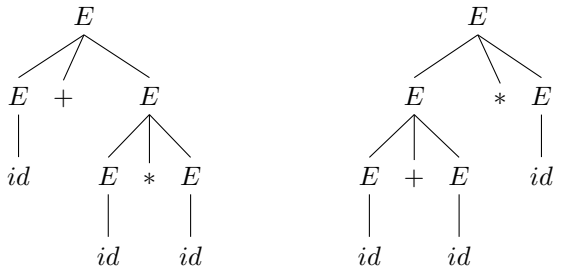
A un arbre generat d'aquesta manera l'anomenarem ***arbre de derivació****.*

Definició 8. *Una gramàtica incontextual G és* ***ambigua*** *si $\exists w \in L(G)$ tal que w té dos (o més) arbres de derivació diferents.*

Exemple 8. *El llenguatge de les expressions aritmètiques que genera la gramàtica G de l'exemple 6 es pot generar a partir d'una gramàtica G' molt més simple amb el mateix alfabet, una única variable E i les produccions*

$$E \longrightarrow E + E, \qquad E \longrightarrow E * E, \qquad E \longrightarrow (E), \qquad E \longrightarrow id.$$

Considerem ara l'expressió $id + id * id$. Aquesta expressió admet els següents arbres de derivació:



(a) (b)

Figura 1.3: Arbres de derivació d'$id + id * id$ en G'

Encara que els llenguatges que generen són els mateixos, l'estructura que segueixen les derivacions és molt diferent. En el cas de G mitjançant les distincions entre expressions, termes i factors s'assegura sempre la precedència de l'operació de multiplicació davant l'operació de suma. G', d'altra banda, permet generar expressions que admeten més d'una derivació tal com es veu a la figura 1.3: la derivació de (a) respecta la precedència del producte, mentre que la que s'utilitza a (b) no. Així, malgrat que el llenguatge generat sigui el mateix, G' és ambigua mentre que G no ho és.

1.3.1 Forma normal de Chomsky

En la definició de gramàtica incontextual no s'ha posat cap limitació a la forma de les produccions, de manera que es poden fer tan complexes com es vulgui. A vegades, però, és convenient representar les gramàtiques d'una manera més senzilla però que resulti equivalent. Una d'aquestes formes és la forma normal de Chomsky que, a més d'una representació senzilla de la gramàtica, aporta propietats que resultaran molt útils en capítols posteriors.

Definició 9. *Diem que una gramàtica incontextual* $G = (V, \Sigma, P, S)$ *està en* ***forma normal de Chomsky*** *si totes les produccions de* P *són de la forma*

$$A \longrightarrow BC$$

o bé de la forma

$$A \longrightarrow a$$

on $A \in V$ *és una variable qualsevol,* $B, C \in V \backslash \{S\}$ *són variables qualssevol diferents de la variable inicial i* $a \in \Sigma$ *és un terminal qualsevol. A part d'aquestes, també es permet la producció* $S \longrightarrow \lambda$*.*

Proposició 2. *Tot llenguatge incontextual està generat per una gramàtica incontextual en forma normal de Chomsky.*

Per fer la demostració de la proposició anterior donarem un procediment per convertir qualsevol gramàtica incontextual a una equivalent en forma normal de Chomsky.

Demostració. Sigui $G = (V, \Sigma, P, S)$ una gramàtica incontextual. Generarem $G' = (V', \Sigma, P', S')$ tal que G' estigui en forma normal de Chomsky i que $L(G) = L(G')$ mitjançant els següents passos:

1. Inicialment a V' hi ha tots els elements de V més un nou estat inicial S' i a P' hi ha totes les produccions de P.

2. Afegim a P' la producció $S' \longrightarrow S$ per tal d'assegurar que la variable inicial no aparegui a la part dreta de cap producció.

3. Eliminem de P' tota producció del tipus $A \longrightarrow \lambda$ amb $A \in V' \backslash \{S\}$. Aleshores, per a tota producció de P' que contingui la variable A a la banda dreta, s'afegeixen noves produccions eliminant una única instància de A. És a dir, si hi havia una producció $B \longrightarrow xAy$ (amb $x, y \in \Sigma \cup V$), s'afegeix la producció $B \longrightarrow xy$; si hi ha $B \longrightarrow xAyAz$, s'afegeixen $B \longrightarrow xyAz$ i $B \longrightarrow xyAz$ i continuant el procés a partir d'aquestes noves produccions, també $B \longrightarrow xyz$. En cas que hi hagués la producció $B \longrightarrow A$, s'afegiria $B \longrightarrow \lambda$ només en cas que no s'hagués eliminat a l'inici del pas. Es repeteix aquest pas fins que no quedi cap producció amb λ que no contingui la variable inicial S.

4. Eliminem de P' tota producció del tipus $A \longrightarrow B$. Després, per cada producció amb forma $B \longrightarrow x$ ($x \in \Sigma \cup V$) que aparegui, afegim a P' la producció $A \longrightarrow x$, sempre que aquesta no s'hagi eliminat en aquest pas o l'anterior. Es repeteix aquest pas fins que no quedi cap producció d'una variable a una altra.

5. En aquest punt, les produccions que queden són de la forma $A \longrightarrow u_1 u_2 \dots u_k$, amb $u_i \in \Sigma \cup V$). Per a cada producció d'aquesta forma i amb $k \geq 3$, substituïm la producció per les produccions $A \longrightarrow u_1 A_1$, $A_1 \longrightarrow u_2 A_2$, ... i $A_{k-2} \longrightarrow u_{k-1} u_k$, amb A_i noves variables. Com que u_i poden ser terminals, per acabar de complir amb les produccions de la forma normal de Chomsky, substituïm cada terminal u_i per una nova variable U_i i afegim la producció $U_i \longrightarrow u_i$.

□

Exemple 9. *Agafem la gramàtica incontextual* $G = \{\{S\}, \{a, b\}, P, S\}$ *amb les següents derivacions a* P*:*

$$S \longrightarrow aSb \mid ab$$

que genera el llenguatge $L(G) = \{a^n b^n \mid n \geq 1\}$*. Apliquem el procediment per obtenir una gramàtica* G' *equivalent a* $G = \{V', \{a, b\}, P', S'\}$ *en forma normal de Chomsky:*

1. *Aplicant el primer pas, tenim que* $V' = \{S', S\}$ *i que a* P' *hi ha la producció*

$$S \longrightarrow aSb \mid ab$$

2. *Afegim a les dues produccions de* P' *la producció* $S' \longrightarrow S$*.*

3. *No hi ha cap producció cap a la paraula buida, de manera que el pas 3 no fa variar* P'*.*

4. *Pel pas 4, eliminem la producció* $S' \longrightarrow S$ *i hi afegim les produccions corresponents. El conjunt de produccions resultant és*

$$S \longrightarrow aSb \mid ab$$
$$S' \longrightarrow aSb \mid ab$$

5. *A la primera part del pas 5, substituïm la producció* $S' \longrightarrow aSb$ *per* $S' \longrightarrow aT$ *i* $T \longrightarrow Sb$*, amb* T *una nova variable i* $S \longrightarrow aSb$ *per* $S \longrightarrow aT$ *i* $T \longrightarrow Sb$*. Mitjançant la segona part de 5, afegim les produccions* $A \longrightarrow a$ *i* $B \longrightarrow b$ *i modifiquem les anteriors per reflectir aquest canvi, obtenint*

$$S' \longrightarrow AT \mid AB$$
$$T \longrightarrow SB$$
$$S \longrightarrow AT \mid AB$$
$$A \longrightarrow a$$
$$B \longrightarrow b$$

Ara la gramàtica resultant G' *sí que està en forma normal de Chomsky i* $L(G') = L(G)$*.*

Observem que si $G = (V, \Sigma, P, S)$ és una gramàtica incontextual en la forma normal de Chomsky i $x \in L(G)$, l'arbre de derivació de $S \Rightarrow_G^* x$ és un arbre binari, ja que totes les produccions tenen dues variables.

Ara, doncs, podem demostrar les següents propietats:

Proposició 3. *Sigui $G = (V, \Sigma, P, S)$ una gramàtica incontextual en la forma normal de Chomsky i $x \in L(G)$ amb $|x| = n$. La derivació $S \Rightarrow_G^* x$ té exactament $2n - 1$ passos.*

Demostració. L'única manera d'obtenir terminals és a partir de les produccions de la forma $A \longrightarrow a$, de manera que per obtenir una cadena de n terminals necessàriament s'ha utilitzat n produccions d'aquest tipus. Per poder fer aquestes produccions ha calgut tenir n variables, que només es poden aconseguir mitjançant $n-1$ produccions de la forma $A \longrightarrow BC$. En total, doncs, s'ha fet servir $n - 1 + n = 2n - 1$ produccions. □

Proposició 4. *Sigui $G = (V, \Sigma, P, S)$ una gramàtica incontextual en la forma normal de Chomsky. Un arbre de derivació de G amb un camí màxim de n arestes pot generar una paraula de com a màxim 2^{n-1} símbols.*

Demostració. Ho demostrarem per inducció sobre la longitud del camí màxim:

- $\boxed{n = 1}$: Si el camí màxim d'un arbre de derivació és 1, aleshores l'única producció que s'ha pogut aplicar és de la forma $S \longrightarrow a$, amb $a \in Sigma$. Aleshores la màxima paraula que pot generar és la formada per una instància del terminal a, que té longitud $1 = 2^{1-1}$.
- $\boxed{n \implies n+1}$: Suposem demostrada la proposició per a arbres de derivació amb camins màxims de longitud $n \geq 1$ i vegem que també es compleix per a camins de longitud $n + 1$. Suposem, doncs, que tenim un arbre de derivació amb camí màxim de longitud $n + 1$. Com que G està en forma normal de Chomsky i $n \geq 2$, aleshores la primera derivació de l'arbre és del tipus $A \longrightarrow BC$, amb B, C variables. Prenem, doncs, els subarbres que tenen B i C com a arrel. Com que la longitud del camí màxim és $n + 1$, cap d'aquests dos arbres pot contenir un camí de longitud superior a n. Aplicant la hipòtesi d'inducció, tenim que cadascun d'aquests dos arbres pot generar una paraula de, com a màxim, longitud 2^{n-1}. Així, l'arbre de derivació inicial pot generar una paraula de com a màxim longitud $2^{n-1} + 2^{n-1} = 2 \times 2^{n-1} = 2^n = 2^{(n+1)-1}$, que és el que volíem veure.

□

Notem que el concepte de llenguatge incontextual l'hem definit a partir d'una gramàtica formal, mentre que en el cas de llenguatges regulars s'han definit a partir d'un autòmat que els reconeix. Els llenguatges incontextuals també estan estretament relacionats a un tipus d'autòmat que els reconeix: els autòmats amb pila. La definició formal d'aquest tipus d'autòmats i la demostració d'equivalència es troba a l'apèndix A.2.

Una de les principals aplicacions de les gramàtiques incontextuals consisteix en la definició formal de la sintaxi dels llenguatges de programació. Fent tal definició es pot obtenir un autòmat amb pila determinista que reconegui el llenguatge, que és precisament el que es fa en els analitzadors sintàctics de la majoria de compiladors. És necessari que les gramàtiques que s'usen no siguin ambigües, ja que en cas contrari hi hauria instruccions vàlides del llenguatge que el compilador podria interpretar de maneres diferents, obtenint programes amb el mateix codi font que puguin donar resultats diferents. Més endavant veurem, però, que no és gens fàcil comprovar si una gramàtica qualsevol és ambigua o no. En aquest capítol hem demostrat també un resultat important pel disseny de compiladors i és que, a partir de la relació entre la longitud d'una paraula i la profunditat de l'arbre de derivació d'una gramàtica en forma normal de Chomsky, hi ha un algorisme que determina en temps polinòmic si la paraula pertany o no a la gramàtica. Això implica que si les gramàtiques utilitzades per descriure la sintaxi dels llenguatges de programació és en forma normal de Chomsky (que hem vist que sempre ho pot ser), aleshores els compiladors poden determinar en temps polinòmic si un programa és vàlid sintàcticament o no.

1.4 Teoremes d'extracció

Els models computacionals que hem vist fins ara no tenen una semblança aparent al que considerem ordinadors. És evident que poden fer un cert tractament de dades i poden reconèixer llenguatges, però aquestes capacitats no les associem a computació arbitrària. De fet, ni tan sols són universals pel que fa al tractament de llenguatges, sinó que estan limitats a llenguatges regulars (en el cas d'autòmats finits) i llenguatges incontextuals (en el cas d'autòmats amb pila). A continuació es presenten uns resultats, anomenats teoremes d'extracció, que donen una manera de distingir els llenguatges regulars (resp. incontextuals) d'aquells que no ho són.

1.4.1 Per a llenguatges regulars

Teorema 2. *Sigui L llenguatge regular infinit. Aleshores existeix $n_L \in \mathbb{N}$ (depenent de L) tal que per a tota paraula $w \in L$ amb $|w| \geq n_L$ existeix una representació de la paraula $w = xyz$ tal que:*

1. $|xy| \leq n$
2. $y \neq \lambda$
3. $xy^i z \in L \ \forall i \geq 0$

Demostració. Com que L és un llenguatge regular, existeix un autòmat determinista $M = (K, \Sigma, \delta, q, F)$ que reconeix L. Sigui n el nombre d'estats en K i considerem una paraula $w \in L(M)$ tal que $|w| \geq n$. Suposem que $w = a_1 \dots a_k$, amb $a_i \in \Sigma \ \forall i = 1 \dots k$. Tenim que $k \geq n$, de manera que per reconèixer la paraula w necessàriament M ha de passar dues vegades per un mateix estat de K. Per a tot $m \leq k$, sigui q_m l'estat en què es troba M després d'haver llegit els m primers

símbols de w. Aleshores, existeixen i, j amb $i < j \leq n$ tals que $q_i = q_j$. Definim, doncs

- $x = a_1 \dots a_i$
- $y = a_{i+1} \dots a_j$
- $z = a_{j+1} \dots a_k$

Com que $j \leq n$, tenim que $|xy| \leq n$; com que $i < j$, $y \neq \lambda$. Finalment, com que $q_i = q_j$ es pot repetir el fragment y tantes vegades com es vulgui, obtenint que $xy^i z \in L \; \forall i \geq 0$. □

El teorema d'extracció dona les condicions necessàries per tal que un llenguatge sigui regular, però el podem utilitzar per demostrar que un llenguatge no és regular com en el següent exemple:

Exemple 10. *Anem a veure que el llenguatge $L = \{a^n b^n \mid n \geq 1\}$ no és regular. Suposant que sí que ho és, podem aplicar el teorema d'extracció per a un cert n. Considerem la paraula $w = a^n b^n$ i busquem una representació $w = xyz$. Sabem que $y \neq \lambda$ i que $|xy| \leq n$, de manera que necessàriament $y = a^k$ per a algun $0 < k < n$. Aleshores, s'hauria de complir que $xy^i z \in L \; \forall i \geq 0$, però clarament això no és cert, ja que $xy^i z$ té el mateix nombre de b que w però si $i > 1$ el nombre de a augmenta.*

1.4.2 Per a llenguatges incontextuals

Teorema 3. *Sigui L un llenguatge incontextual infinit. Aleshores existeix $n_L \in \mathbb{N}$ tal que, per a tota paraula $w \in L$ amb $|w| \geq n_L$, existeix una representació $w = uvxyz$ tal que:*

- $|vxy| \leq n_L$
- $vy \neq \lambda$ *(és a dir, v i y no poden ser les dues buides)*
- $uv^i xy^i z \in L \; \forall i \geq 0$

Demostració. Sigui G una gramàtica incontextual tal que $L = L(G)$. Per la proposició 2 podem assumir sense pèrdua de generalitat que G està en forma normal de Chomsky. Considerem l'arbre de derivació de w. Per la proposició 4, tenim que si la longitud de w fos més gran que 2^n, aleshores a l'arbre de derivació de w hi hauria un camí de longitud $\geq n + 1$.

Sigui ara n el nombre de variables de G i sigui $k = 2^n$. Suposem que $|w| > k$. Aleshores, l'arbre de derivació de w té algun camí de longitud $\geq n + 1$. Sobre aquest camí, considerem ara el subcamí de longitud $n+1$ des de la fulla. En aquest subcamí l'última producció serà del tipus $A \longrightarrow a$, amb A una variable i a un terminal i les n produccions restants seran de variables. En aquestes n produccions hi intervenen $n+1$ variables (les produccions són les arestes de l'arbre i les variables són els nodes, si hi ha n arestes necessàriament hi ha $n+1$ nodes), però com que n està definida com el nombre de variables de G, necessàriament alguna variable està repetida. És a dir, si $A_1 A_2 \dots A_{n+1}$ és la seqüència de variables, necessàriament

$\exists j, k \in \{1, \ldots, A_{n+1}\}$ amb $j < k$ tal que $A_j = A_k = A$ aleshores la derivació de w es pot escriure com

$$S \Rightarrow_G^* uA_jz \Rightarrow_G^* uvA_kyz \Rightarrow_G^* uvxyz = w$$

Considerem ara el subarbre corresponent a la derivació $A_j \Rightarrow_G^* vA_ky \Rightarrow_G^* vxy$. Com que A_j ha estat agafada del subcamí de longitud $n+1$, aquest subarbre no té cap camí de longitud superior a $n+1$, i per tant $|vxy| \leq 2^n = k$.

Així, podem repetir el fragment vAy tantes vegades com vulguem i per tant podem obtenir

$$S \Rightarrow_G^* uAz \Rightarrow_G^* uv^iAy^yz \Rightarrow_G^* uv^ixy^iz$$

i per tant $uv^ixy^iz \in L(G)\ \forall i \geq 0$. □

Exemple 11. *Suposem que el llenguatge $L = \{a^n \mid n$ és primer$\}$ és incontextual. Aleshores existeix un $n_L > 0$ tal que per a tota paraula de L amb longitud major o igual a n_L es compleix el teorema d'extracció. Prenem una paraula $w \in L$ tal que $w = a^p$ amb p primer i en considerem una representació $w = uvxyz$. Suposem que $uv = a^q$ i $xyz = a^r$, amb $q, r \in \mathbb{N}$ tal que $p = q + r$. Aleshores, pel teorema d'extracció tenim que $uv^ixy^i \in L\ \forall i \geq 0$, que implica que $r + qi$ és primer $\forall i \geq 0$, que no és cert (prenent, per exemple $i = 2q+r+2$ es té que $r+iq = (q+1)(2q+r)$). Per tant, L no és incontextual.*

Capítol 2

Màquines de Turing

Hem vist, doncs, que els models computacionals que ens donen els autòmats no són universals, ja que hi ha llenguatges que no poden ser reconeguts per aquests. Així, si es pretén obtenir un model de computació universal, es necessita algun model més potent, més general, que trobarem en les màquines de Turing.

En una màquina de Turing continuem tenint una cinta amb la paraula d'entrada i una unitat de control amb un conjunt d'estats. En aquest model, però, la cinta és infinita (encara que la paraula no ho sigui) i les cel·les de la cinta no només permeten la lectura, sinó que també s'hi poden escriure (o sobreescriure) caràcters. A més, el punter que defineix quina cel·la de la cinta es llegeix ara es podrà moure tant cap a l'esquerra com cap a la dreta. Així doncs, en certa manera, es pot utilitzar la cinta com a memòria sense les limitacions de les piles.

Definició 10. *Una màquina de Turing determinista és una estructura* $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ *on:*

- K *és un conjunt finit i no buit d'estats*
- Σ *és l'alfabet d'entrada, que no conté els símbols* $\$$ *i* $*$
- Γ *és l'alfabet de la cinta, complint que* $\Sigma \cup \{\$, *\} \subseteq \Gamma$ *(de fet, en la majoria de casos usuals tindrem que* $\Gamma = \Sigma \cup \{\$, *\}$*)*
- $q_0 \in K$ *és l'estat inicial*
- $q_F \in K$ *és l'únic estat acceptador*
- $\delta \colon (K \setminus \{q_F\}) \times \Gamma \nrightarrow K \times \Gamma \times \{L, R, S\}$ *és una funció parcial tal que,* $\forall q \in K \setminus \{q_F\}$
  - *Si* $\delta(q, \$) = (p, b, i) \implies b = \$ \wedge i \neq L$ *(i.e. si la màquina és a l'inici de la cinta, no es pot sobreescriure el símbol d'inici de cinta i no podem moure el punter més cap a l'esquerra)*
  - *Si* $a \neq \$,\ \delta(q, a) = (p, b, i) \implies b \neq \$$ *(i.e. no es pot escriure el símbol d'inici de la cinta en cap posició que no sigui l'inici de la cinta)*

Prendrem la cinta com a acotada per l'esquerra i amb $*$ el símbol de l'alfabet de la cinta utilitzat per denotar la cel·la buida. A la primera casella de la cinta hi haurà sempre el símbol inicial \$, i a continuació d'aquest símbol hi haurà escrita la paraula d'entrada. La resta de la cinta contindrà el caràcter de cel·la buida. En un pas de còmput, el punter podrà escriure un caràcter a la cel·la a la qual apunta i podrà moure's a la cel·la anterior o següent. Denotarem per

- *$L =$ moviment del punter cap a l'esquerra*
- *$R =$ moviment del punter cap a la dreta*
- *$S =$ sense moviment de punter*

Inicialment la màquina de Turing M es troba a l'estat inicial q_0 i el punter està situat a la primera cel·la de la paraula (i per tant a la segona cel·la de la cinta).

Els còmputs de la màquina es fan a partir de la funció δ que, de manera equivalent als autòmats, indica com passar d'un pas de còmput al següent en funció de l'estat actual i el símbol de la cinta al que apunta el punter. Així, si $a \in \Gamma$ és el símbol a què apunta el punter i $q \in K$ és l'estat actual, si $\delta(q, a) = (p, b, i)$, la màquina passarà a l'estat p, escriurà el símbol b a la cel·la que indica el punter (la que apuntava el punter i contenia a) i finalment,

$$\begin{cases} \text{es mou el punter una cel·la cap a la dreta} & \text{si } i{=}R \\ \text{es mou el punter una cel·la cap a l'esquerra} & \text{si } i{=}L \\ \text{no es mou el punter} & \text{si } i{=}S \end{cases}$$

Definició 11. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing determinista. Definim una **configuració** de M com (α, q, β) on $q \in K$ i $\alpha, \beta \in \Gamma^*$ tal que β no acaba en $*$.*

Direm que si, en un pas de còmput la configuració de M és (α, q, β), aleshores M està en l'estat q, el contingut de la cinta és $\alpha\beta$ seguit de cel·les buides i el punter senyala al primer caràcter de β (que possiblement és una cel·la buida).

*Si (α, q, β) i (α', q', β') són configuracions de M, direm que (α, q, β) **produeix** (α', q', β') **en un pas de còmput** (i ho escriurem $(\alpha, q, \beta) \vdash_M (\alpha', q', \beta')$) si podem passar de (α, q, β) a (α', q', β') aplicant una vegada la funció δ. Farem servir la notació exponencial sobre el símbol $\vdash$ per expressar que una configuració en produeix una altra amb n passos de còmput de la següent manera: $\vdash_M^n$. Finalment, direm que (α, q, β) **produeix** (α', q', β') i ho escriurem $(\alpha, q, \beta) \vdash_M^* (\alpha', q', \beta')$ si existeix $n \geq 0$ tal que $(\alpha, q, \beta) \vdash_M^n (\alpha', q', \beta')$.*

Observem que, de moment, no hem parlat en cap moment de la parada de la màquina. Observem, també, que en la definició de màquina de Turing, la funció de transició δ és parcial i que el domini exclou l'estat final q_F. Aquesta exclusió és ben intencional i, de fet, serà la que ens permetrà definir el concepte de parada:

Definició 12. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing determinista, (α, q, β) una configuració de M i sigui b el primer caràcter de β (el que apunta el*

punter). Direm que M ***para*** *sobre la configuració (α, q, β) si $\delta(q, b)$ no està definida. En particular, M para si $q = q_F$.*

Cal remarcar que, segons aquesta definició de parada, M sempre para en arribar a l'estat q_F, però que no sempre que para ho fa en una configuració amb l'estat q_F.

Definició 13. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing determinista i sigui $x \in \Sigma^*$.*

Direm que M ***para sobre la paraula*** *x si el còmput de M sobre l'entrada x para, i en aquest cas escriurem $M(x)\downarrow$. En cas que el còmput sobre x no pari, escriurem $M(x)\uparrow$. En cas que M pari sobre una entrada x, denotem per $M(x)$ la paraula de Σ^* que apareix en l'última configuració del còmput de M sobre x.*

Direm, també, que M és una màquina de ***parada segura*** *si $\forall x \in \Sigma^*$ tenim que $M(x)\downarrow$.*

Definició 14. *Si M és una màquina de Turing amb un alfabet d'entrada Σ, definim la* ***funció associada*** *a M com la funció parcial $f_M \colon \Sigma^* \nrightarrow \Sigma^*$ tal que, per a tot $x \in \Sigma^*$*

$$f_M(x) = \begin{cases} M(x) & \text{si } M(x)\downarrow \\ indefinit & \text{si } M(x)\uparrow \end{cases}$$

Aleshores, direm que una funció $f \colon \Sigma^* \nrightarrow \Sigma^*$ és ***computable*** *si existeix una màquina de Turing M tal que $f = f_M$.*

També es pot definir el concepte de funció computable de diverses variables de la següent manera:

Definició 15. *Sigui $n \geq 2$, sigui $f \colon (\Sigma^*)^n \longrightarrow \Sigma^*$ i sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing determinista tal que $\Sigma \cup \{\$, *, c\} \subseteq \Gamma$, on c és un caràcter de control tal que $c \notin \Sigma$. Diem que M* ***computa*** *f si, per a qualssevol $u_1, \ldots, u_n \in \Sigma^*$*

1. *Si $f(u_1, \ldots, u_n) = u$ aleshores el còmput de M sobre l'entrada $u_1 c u_2 c \ldots c u_n$ para i dona com a resultat u.*
2. *Si $f(u_1, \ldots, u_n)$ no està definida, aleshores el còmput sobre l'entrada $u_1 c u_2 c \ldots c u_n$ no para.*

Es pot demostrar que totes les funcions aritmètiques són computables per màquines de Turing.

2.1 Llenguatges decidibles

Definició 16. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing determinista. Diem que una paraula $x \in \Sigma^*$ és* ***reconeguda o acceptada*** *per M si M para sobre l'entrada x en l'estat acceptador q_F. Direm que és* ***rebutjada o no és reconeguda*** *si M para en qualsevol altre estat de K o bé no para.*

Definició 17. *Definim el llenguatge associat a una màquina de Turing M per $L(M) = \{x \in \Sigma^* \mid x \text{ és reconeguda per } M\}$. Direm que un llenguatge L és* ***acceptable*** *si existeix una màquina de Turing determinista M tal que $L(M) = L$.*

Definició 18. *Sigui L un llenguatge. Diem que L és* ***recursiu o decidible*** *si existeix una màquina de Turing M determinista i de parada segura tal que $L = L(M)$. En aquest cas, direm que M* ***decideix*** *el llenguatge L.*

Aleshores tot llenguatge recursiu és acceptable, però l'invers no és cert, ja que les màquines de Turing que accepten un llenguatge poden no parar en les paraules rebutjades.

Observem que aquests dos tipus de llenguatges estan formats per paraules que fan parar una màquina en l'estat acceptador. Un altre tipus de llenguatge interessant és el format per les paraules que fan parar una màquina, sense importar en quin estat:

Definició 19. *Definim el* ***conjunt de parada*** *d'una màquina de Turing M com $E(M) = \{x \in \Sigma^* \mid M(x) \downarrow\}$. Direm que un llenguatge L és* ***recursivament enumerable*** *si existeix una màquina de Turing determinista M tal que $L = E(M)$, i en aquest cas direm que M* ***semidecideix*** *L.*

Definició 20. *Diem que un subconjunt $\Gamma \subset \Sigma^*$ és* ***computable*** *si Γ i $\bar{\Gamma} := \Sigma^* \setminus \Gamma$ són recursivament enumerables.*

2.2 Màquines de Turing de múltiples cintes

A part de la definició que ja hem donat de la màquina de Turing, existeixen diverses definicions alternatives amb algunes variacions sobre l'estructura o comportament. A continuació es presenta una màquina de Turing amb múltiples cintes i a l'apèndix A.3 es tracta una màquina de Turing indeterminista. A priori pot semblar que aquestes modificacions augmenten, d'alguna manera, la capacitat de càlcul de les màquines, però veurem també que les "màquines ampliades" són equivalents a la definició original, reforçant la idea de màquina de Turing com a model computacional universal.

Una màquina de Turing amb múltiples cintes és una màquina de Turing estàndard a què se li afegeix un nombre finit de cintes i un punter associat a cada una d'aquestes cintes. Considerem la primera cinta com a cinta d'entrada i la resta com a cintes auxiliars. A l'inici de cada cinta hi haurà escrit el caràcter \$ d'inici de cinta seguit de la paraula d'entrada i espais en blanc a la primera cinta i d'espais en blanc a la resta de cintes. Inicialment els punters de cada cinta apunten a la segona posició (i.e. l'inici de la paraula a la primera cinta i el primer espai en blanc a la resta de cintes).

El funcionament d'aquesta màquina és idèntic al d'una màquina de Turing amb una sola cinta, amb la diferència que en un sol pas de còmput es pot escriure qualssevol de les posicions de les cintes indicades pels punters i es pot moure els punters de manera independent per a cada cinta. A més, en un pas de còmput es permet fer el canvi d'estat no només a partir de l'estat actual i el símbol de la cinta d'entrada, sinó també en funció dels símbols de les cintes auxiliars. Això, però, fa que la funció de transició sigui significativament més complicada. Vegem-ne la definició formal:

Definició 21. *Prenent $k \geq 2$, diem que una màquina de Turing determinista de k cintes és una estructura $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ on:*

- K, Σ, Γ, q_0 *i* q_F *es defineixen de la mateixa manera que en la màquina de Turing tradicional*
- $\delta\colon\ (K \setminus \{q_F\}) \times \Gamma^k \nrightarrow K \times (\Gamma \times \{L, S, R\})^k$

De manera que en un pas de còmput es determina el canvi d'estat, l'escriptura sobre les cintes i el moviment de cada punter en funció de l'estat actual i els símbols sota el punter de cada cinta.

La resta de conceptes associats a les màquines de Turing es defineixen de la mateixa manera que en les màquines tradicionals tenint en compte la nova definició de δ.

Com a tota extensió de model computacional que s'ha presentat durant el treball, sembla natural preguntar-nos si aquesta modificació aporta més capacitat de càlcul al model o bé és equivalent al model original.

Teorema 4. *Tota màquina de Turing determinista M amb múltiples cintes es pot simular per una màquina de Turing determinista M' d'una sola cinta.*

Idea de la demostració. Suposem que M és una màquina de Turing determinista de $k \geq 2$ cintes amb un alfabet d'entrada Σ i alfabet de cinta Γ. Creem una màquina de Turing M' amb

- Un alfabet d'entrada Σ i alfabet de cinta $\Gamma' = \Sigma \cup \{\bar{x} \mid x \in \Sigma\} \cup \{\$, *, \#\}$
- El contingut inicial de la cinta és el símbol \$ seguit de la concatenació de les k cintes de M, separades pel símbol #. Això genera una partició de la cinta en k fragments, el primer limitat per l'esquerra per \$ i la resta per #.
- Per simular el fet de tenir k punters, en cada fragment se substituirà el símbol x de la posició on senyalaria el punter pel símbol $\bar{x}$, com si es tractés d'un punter virtual.

Per simular un pas de còmput la màquina de k cintes, es faria el següent procediment:

1. Es mou el punter cap a l'esquerra fins a trobar el símbol \$ d'inici de cinta.
2. Es recorre la cinta d'esquerra a dreta des de \$ fins al k-èssim símbol # per tal de determinar els símbols sota els punters virtuals, codificant d'alguna manera aquestes posicions en l'estat de la màquina M'.
3. Aleshores es fa una altra passada a la cinta de dreta a esquerra, fent les modificacions de continguts de les cintes i punters virtuals que dictaria la funció de transició de M. Si en algun moment cal moure un punter virtual a una posició de la cinta amb un símbol # on comença la següent cinta, la màquina haurà d'escriure el símbol $*$ en aquella casella i moure tots els símbols posteriors una casella cap a la dreta.

Evidentment a aquesta idea hi falten molts detalls per esdevenir una demostració formal: caldria veure com definir els estats i l'enorme funció de transició de M, que en qualsevol moment podem moure els símbols a partir d'una posició cap a la dreta i després continuar amb l'operació, que si M para M' també i que aleshores $L(M) = L(M')$... Totes aquestes comprovacions, però, queden fora del treball i es poden trobar més desenvolupades a [4] i [13].

2.3 Màquines de Turing universals

Aparentment les màquines de Turing no són programables: una vegada en donem la definició mitjançant el conjunt d'estats, alfabets i funció de transició, el seu comportament queda, en essència, determinat, de manera que ens cal una màquina per a cada problema. A continuació, però, es mostra una manera de superar aquesta limitació mitjançant el que anomenarem una màquina de Turing universal. El que farem és donar una màquina que accepti com a entrada la descripció d'una altra màquina de Turing juntament amb l'entrada desitjada, i la màquina universal en durà a terme l'execució.

Si hem de passar una màquina de Turing $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ qualsevol com a entrada a una altra màquina de Turing, hem de donar una manera de codificar-les a partir d'un alfabet finit, que serà l'alfabet d'entrada de la màquina universal. Per codificar una màquina en particular, podríem utilitzar els símbols que estan continguts a K, Σ, Γ i δ, ja que tots aquests conjunts són finits. El problema és que això ens dona un alfabet que només seria vàlid pel subconjunt de màquines que estiguin limitades a aquests símbols, però sempre hi haurà màquines que no hi siguin representades, ja que els alfabets i conjunts d'estats d'una màquina poden ser arbitràriament grans. Ara bé, els alfabets són finits (o, com a molt, numerables) de manera que podem assignar un enter a cada símbol o estat i codificar-ho d'aquesta manera. Per obtenir l'alfabet més senzill possible per a la màquina universal, representarem l'enter en binari, de manera que només ens calen 2 símbols. Finalment, per separar els elements i poder distingir entre la representació binària d'un símbol i un estat, posarem el q abans de cada estat i a davant de cada símbol.

Així, si i és el mínim enter tal que $|K| \leq 2^i$, els estats seran paraules del llenguatge $q\{0,1\}^i$. Si suposem que $\Gamma = \Sigma \cup \{\$, *\}$ i prenem j el menor enter tal que $|\gamma|+3 \leq 2^j$, aleshores el conjunt de símbols necessaris per representar qualsevol màquina de Turing (notem que també necessitem codificar els símbols L, S, R per definir δ, d'aquí el $+3$) són del llenguatge $a\{0,1\}^j$.

L'assignació de cada element a l'enter corresponent pot ser arbitrària però, per conveni, prendrem les següents assignacions referents als símbols:

Pel que fa als estats, només tindrem en consideració que q_F sigui $q0^i$, q_0 sigui $q0^{i-1}1$ i la resta ja els podem agafar en ordre lexicogràfic.

Si M és una màquina de Turing i w és una paraula qualsevol, utilitzarem la notació $\rho(M)$ i $\rho(w)$ per referir-nos a la representació de la màquina i de la paraula en l'alfabet que acabem de descriure.

Símbol	Representació
\$	$a0^j$
$*$	$a0^{j-1}1$
L	$a0^{j-2}10$
S	$a0^{j-2}11$
R	$a0^{j-3}100$

Com que es pot codificar qualsevol alfabet donat amb l'alfabet que hem definit, podem representar qualsevol màquina de Turing mitjançant aquest nou alfabet. Com que els estats inicial i final ja queden determinats per la codificació, només cal donar la funció de transició δ per tal que una màquina de Turing quedi ben definida. Per convenció, escriurem les transicions com 5-tuples (q, a, p, b, c), amb q, p representacions d'estats i a, b, c representacions de símbols, amb totes les representacions separades per comes i cada 5-tupla envoltada de parèntesis. Aleshores passarem com a entrada a la màquina de Turing universal la paraula $\rho(M), \rho(w)$, on $\rho(M)$ correspondrà la concatenació de les 5-tuples que descriuen les transicions d'M i $\rho(w)$ la concatenació de les representacions dels símbols de la paraula d'entrada.

Exemple 12. *Considerem* $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ *tal que* $K = \{q_0, q_F\}$, $\Sigma = \{a, b, c\}$ *i* $\Gamma = \Sigma \cup \{\$, *\}$, *amb*

$$\begin{aligned}
\delta(q_0, \$) &= (q_0, \$, R) \\
\delta(q_0, a) &= (q_0, \$, R) \\
\delta(q_0, b) &= (q_0, \$, R) \\
\delta(q_0, c) &= (q_0, \$, R) \\
\delta(q_0, *) &= (q_F, *, S)
\end{aligned}$$

Aquesta és una màquina de Turing molt senzilla que recorre la cinta i substitueix cada caràcter de l'alfabet d'entrada pel caràcter de cel·la buida, efectivament esborrant la paraula de la cinta.

Com que només hi ha dos estats, un sol bit és suficient per codificar-los i aleshores escriurem q_0 *com a* $q0$ *i* q_F *com* $q1$. *Per altra banda, hi ha* $|\Gamma| + 3 = 5 + 3 = 8$ *símbols i aleshores necessitem 3 bits per representar-los tots. La representació de cada símbol serà la següent:*

Podem representar M *com*

$$\begin{aligned}
\rho(M) = &(q0, a000, q0, a000, a100), (q0, a001, q0, a001, a100), (q0, a101, q0, a001, a100), \\
&(q0, a110, q0, a001, a100), (q0, a111, q0, a001, a100), (q0, a001, q1, a000, a100)
\end{aligned}$$

i si tenim, per exemple, una paraula d'entrada $w = aacbc$ *la representarem per*

$$\rho(w) = a101a101a111a110a111$$

Símbol	Representació
\$	$a000$
$*$	$a001$
L	$a010$
S	$a011$
R	$a100$
a	$a101$
b	$a110$
c	$a111$

Observació. *Podem fer servir una simplificació d'aquest mateix procediment per obtenir representacions d'autòmats finits i autòmats amb pila, adaptant-lo a les funcions de transició de cada autòmat. Així, utilitzarem la mateixa notació ρ per denotar la representació d'un autòmat.*

Un cop podem donar una codificació d'una màquina qualsevol en un alfabet finit ja podem descriure una màquina de Turing universal U

Per simplificar les explicacions, no donarem directament la màquina de Turing U universal, sinó que donarem una màquina U' de tres cintes equivalent (de manera que U sigui la màquina de Turing que resulta de simular U' segons el procediment descrit al teorema 4). El contingut de cada cinta és el següent:

- La primera cinta simularà la cinta de la màquina M i contindrà la codificació del contingut de la cinta de M.
- La segona cinta conté $\rho(M)$ (i.e. la funció de transició de M codificada).
- La tercera cinta conté la codificació de l'estat actual de M.

Per tant, abans de poder simular la màquina M, s'ha de "configurar" U' de la següent manera:

1. Inicialment la primera cinta de U' conté la cadena "$\rho(M)\rho(w)$" i les altres dues cintes són buides.
2. U' mou $\rho(M)$ a la segona cinta i posa $\rho(w)$ a l'inici de la primera cinta, de manera que a la cinta de U' hi haurà "\$$\rho(w)$".
3. U' escriu $q0^{i-1}1$ (la codificació de l'estat inicial de M) a la tercera cinta.
4. U' recorre la segona cinta fins a trobar una 5-tupla tal que la codificació del primer element correspon a la codificació de l'estat de la tercera cinta i el segon element correspon a la codificació del símbol que el punter senyala de la primera cinta.
    - En cas que trobi tal 5-tupla, substitueix el contingut de la tercera cinta pel tercer component de la tupla, substitueix a la primera cinta la representació del símbol al qual senyala el punter pel quart component de la

tupla i finalment fa el moviment del punter de la primera cinta segons indica l'últim component de la tupla. Es torna al pas 4.

- En cas que no existeixi cap 5-tupla que compleixi la condició vol dir que la funció de transició de M no està definida per l'estat i símbol actual, de manera que M ha de parar i per tant també U'. Si l'estat de M era estat acceptador (a la tercera cinta hi havia $q0^i$), U' passarà a q_F i pararà, acceptant l'entrada. En cas contrari, U' pararà en algun altre estat, rebutjant l'entrada.

2.4 La tesi de Church-Turing

Tots els intents que s'han dut a terme per tal d'ampliar una màquina de Turing han acabat amb un model que pot ser simulat per una màquina de Turing convencional i que, per tant, és equivalent. Tot sembla indicar, doncs, que la màquina de Turing és un model de computació universal. Aquesta idea queda reflectida en la tesi de Church-Turing, formulada independentment per Alonzo Church i Alan Turing el 1936, que diu el següent:

> Tot algorisme és equivalent a una màquina de Turing.

Aquest enunciat no és demostrable, ja que el concepte d'algorisme no està definit formalment i és precisament aquesta tesi que en dona l'equivalència a un concepte ben definit com n'és la màquina de Turing. Això vol dir que és possible que en un futur es demostri que la tesi no és vàlida, però actualment aquest escenari es considera molt poc probable i es pren la tesi com a vàlida. Els principals arguments a favor de la tesi són els següents:

- Per a tots els models de computació proposats [1] s'ha demostrat que poden ser simulats per una màquina de Turing i per tant són equivalents. Ara com ara no s'ha pogut trobar cap model que superi les màquines de Turing en referència al fet que pot ser computat.
- Tota funció per la qual es pot donar un algorisme ha resultat ser computable per una màquina de Turing.
- Dona una base teòrica sòlida i relativament simple que unifica totes les nocions de computació i permet entendre què pot ser computat i què no.
- No s'ha trobat cap "teorema d'extracció" que es pugui aplicar a llenguatges acceptables.

[1] se'n pot trobar els originals a [3], que és un recull dels articles fundacionals de computabilitat

Capítol 3

Problemes decidibles

Ara que en principi ja tenim un model computacional universal en les Màquines de Turing i establerta la connexió d'aquestes amb el concepte d'algorisme mitjançant la tesi de Curch-Turing, podem començar a plantejar com sol·lucionar problemes. Malgrat tot, sembla que tant en els autòmats com en les màquines de Turing sempre s'ha treballat a partir de llenguatges: hem mostrat com generar màquines que comprovin si una paraula pertany a un tipus de llenguatge, però no s'ha anat més enllà. Ara veurem, però, que una gran quantitat de problemes es poden plantejar mitjançant el concepte de llenguatges i solucionar comprovant la pertinença d'una certa entrada a un llenguatge.

Considerem, per exemple el llenguatge

$$A_{DFA} = \{\rho(B)\rho(w) \mid B \text{ és un autòmat determinista que accepta } w\}$$

on cada paraula és una parella formada per un autòmat determinista i una paraula w. Aleshores el llenguatge A_{DFA} conté tots els autòmats deterministes cada un amb totes les paraules que accepta. Així, el problema de saber si un cert autòmat M reconeix una paraula x és equivalent a veure que $\rho(M)\rho(x) \in A_{DFA}$

Aquest és un exemple de com podem "codificar" problemes dins de llenguatges que sí que sabem com tractar. Aleshores, si demostrem que el llenguatge és decidible el problema que codifica també serà decidible.

A partir d'aquest moment i com ja ha començat a passar al final del capítol anterior, les màquines de Turing necessàries per fer les demostracions són, en la majoria dels casos, extremadament complicades, amb conjunts enormes d'estats i funcions de transició i no les donarem explícitament. Acollint-nos a la tesi de Church-Turing, però, podem donar un algorisme de més alt nivell que, segons la tesi, es pot materialitzar en una màquina de Turing. Així, les demostracions perdran, en certa manera, un grau de formalisme, però seguiran essent vàlides sempre que la tesi ho sigui.

Proposició 5. *A_{DFA} és decidible.*

Idea de la demostració. Per veure que un llenguatge és decidible hem de donar una màquina de Turing M de parada segura tal que $L(M) = A_{DFA}$. La idea principal de la demostració és construir una màquina de Turing que, rebent com a entrada un parell $\rho(B)\rho(w)$ amb un autòmat determinista i una paraula en el llenguatge d'entrada d'aquest autòmat, faci la simulació de B amb la paraula w. Si l'execució de B sobre w acaba en un estat d'acceptació, llavors fem que la màquina M pari en l'estat acceptador, reconeixent $\rho(B)\rho(w)$; si la simulació de B acaba en un estat de rebuig, llavors rebutgem el parell $\rho(B)\rho(w)$. Com que un autòmat determinista sempre arribarà a un estat de rebuig o d'acceptació després de processar una entrada finita, aquest procés sempre acabarà, obtenint així una màquina de parada segura.

Aquesta proposició la podem estendre fàcilment autòmats indeterministes:

Proposició 6. *Sigui*

$$A_{NFA} = \{\rho(B)\rho(w) \mid B \text{ és un autòmat indeterminista que accepta } w\}$$

Aleshores A_{NFA} és decidible.

Demostració. La manera més evident de demostrar la proposició seria donar una màquina de Turing N que fos capaç de simular un autòmat indeterminista. Això, però, no és una tasca senzilla, ja que s'hauria de simular totes les branques de l'arbre de còmput. Durant el capítol d'autòmats s'ha vist, mitjançant el teorema 1, l'equivalència entre els autòmats deterministes i indeterministes. Aquest teorema, a més, és constructiu, de manera que dona un algorisme per passar de l'un a l'altre. Podem aprofitar aquest fet i la proposició que acabem de demostrar per definir la màquina N. Així definim N com la màquina de Turing que, donada una entrada $\rho(B)\rho(w)$, on B codifica un autòmat indeterminista i w és una paraula en l'alfabet de B

1. Converteix l'autòmat indeterminista B en un autòmat determinista B' seguint el procediment descrit en el teorema 1.
2. Simula l'execució de B' executant la màquina de Turing M descrita en la proposició anterior amb entrada $\rho(B')\rho(w)$.
3. Si M accepta l'entrada, N també accepta; si M la rebutja, N també.

Com que tant el procés per obtenir un autòmat determinista d'un indeterminista és finit i la màquina M de la proposició anterior és de parada segura, N també és una màquina de parada segura. □

La demostració que tots aquests llenguatges són decidibles acaba recaient en el fet que una màquina de Turing pot simular un autòmat, però ni molt menys totes les màquines que decideixen llenguatges necessiten recórrer a tal simulació.

Prenem, per exemple, el problema de saber si una gramàtica incontextual G genera una certa paraula w. Aquest problema, de manera molt similar a com ho hem fet en els casos anteriors, es pot codificar mitjançant un llenguatge. Considerem

$$A_{CFG} = \{\rho(G)\rho(w) \mid G \text{ és una gramàtica incontextual que genera } w\}$$

el llenguatge de les parelles de gramàtiques incontextuals i paraules. Tal com abans, si determinem que aquest llenguatge és decidible, aleshores el problema que codifica també serà decidible.

Proposició 7. *A_{CFG} és decidible.*

Idea de la demostració. La idea més senzilla possible seria la de fer una cerca exhaustiva per totes les derivacions de G per veure si alguna resulta en w. Aquest procediment, però, té un problema, i és que el nombre de derivacions d'una gramàtica no necessàriament és finit. Això suposa un problema si G no genera w, ja que la màquina no pararia mai. Observem que amb aquest procediment obtenim una màquina que reconeix A_{CFG} però que no el decideix.

Així doncs, cal trobar una manera de no haver de comprovar infinites derivacions. Per aconseguir això, es pot passar la gramàtica a la forma normal de Chomsky on, per la proposició 3, qualsevol derivació en la qual s'obtingui w té llargada exactament $2|w|-1$. Aleshores, només caldrà comprovar les derivacions amb $2|w|-1$ passos, de les quals n'hi ha un nombre finit.

Demostració. Així, definim N com la màquina de Turing que, donada una entrada $\rho(G)\rho(w)$, amb G una gramàtica incontextual i w és una paraula,

1. Converteix G en una gramàtica equivalent amb forma normal de Chomsky.
2. Si $|w| \neq 0$
   (a) Genera la llista de totes les derivacions de longitud $2|w| - 1$.
   (b) Si alguna d'aquestes derivacions resulta en w, accepta, en cas contrari, rebutja.
3. Si $|w| = 0$
   (a) Genera la llista de totes les derivacions de longitud 1.
   (b) Si alguna d'aquestes derivacions acaba en λ, accepta, en cas contrari, rebutja.

Per tant, N és una màquina de Turing de parada segura que decideix A_{CGF}. □

Aquest últim problema està molt estretament relacionat amb la compilació de llenguatges de programació i ens diu que és possible determinar si un cert programa és vàlid en un llenguatge de programació (l'estructura del qual és, en general, modelitzable a partir de produccions d'una gramàtica incontextual). Evidentment el procediment que realment s'utilitza és molt més eficient que el descrit aquí, però aquest com a mínim assegura que és possible.

Un altre problema que es pot plantejar sobre gramàtiques és el de comprovar si una gramàtica incontextual G pot generar realment alguna paraula o si, en cas contrari, $L(G) = \emptyset$. Aquest problema es pot codificar en el llenguatge

$$E_{CFG} = \{\rho(G)|G \text{ és gramàtica incontextual } \wedge L(G) = \emptyset\}$$

que és el llenguatge de les gramàtiques incontextuals buides.

Proposició 8. *E_{CFG} és decidible.*

Idea de la demostració. Per la proposició 7 tenim una màquina de Turing que comprova si una paraula concreta és generada per una gramàtica. Semblaria que podríem utilitzar aquesta màquina i iterar sobre les paraules per comprovar si alguna paraula es pot generar, però el nombre de paraules sobre un alfabet és infinit, de manera que si la gramàtica fos realment buida aquest procediment no acabaria. En aquest cas, la màquina ni tan sols reconeixeria el llenguatge.

El que farem és, en certa manera, construir un hipotètic arbre de derivació des de sota, mirant les produccions que generen terminals a partir de variables i pujant cap amunt mirant quines produccions generen aquestes variables i així fins a arribar o bé al símbol inicial o haver comprovat totes les possibles produccions.

Necessitem, doncs, una màquina de Turing que implementi aquesta idea.

Demostració. Definim M com la màquina de Turing que, donada una entrada $\rho(G)$, amb $G = (V, \Sigma, P, S)$ una gramàtica incontextual

1. Marca tots els símbols terminals de G.
2. Repeteix mentre no es puguin marcar més variables:
   (a) Marca totes les variables A tals que hi hagi alguna producció a G de la forma $A \longrightarrow B_1B_2 \dots B_k$, amb $B_1B_2 \dots B_k \in \Sigma \cup V$ que ja hagin estat marcades.
3. Si en acabar S és marcada, accepta, en cas contrari, rebutja.

M és una màquina de parada segura, ja que el nombre de produccions a comprovar és un nombre finit. Per tant, M decideix E_{CFG}. □

Capítol 4

Problemes indecidibles

4.1 El problema de parada

Seguint el mateix patró que hem fet servir per demostrar la decidibilitat dels llenguatges generats per diversos models de computació i les paraules que accepten, sembla natural considerar el llenguatge equivalent per a les màquines de Turing:

$$A_{TM} = \{\rho(M)\rho(w) \mid M \text{ és una màquina de Turing que accepta } w\}$$

En aquest cas, però, de seguida veurem que no segueix l'exemple dels llenguatges anteriors.

Proposició 9. *A_{TM} no és decidible.*

Abans de demostrar que A_{TM} no és decidible, cal demostrar que el següent llenguatge K_0, definit per

$$K_0 = \{\rho(M) | M \text{ és una màquina de Turing determinista que para sobre } \rho(M)\}$$

tampoc és decidible. K_0 és el llenguatge format per les representacions de les màquines de Turing que paren en rebre la seva pròpia representació com a entrada. Això d'entrada pot semblar estrany, però de seguida veurem com ens pot ser útil.

Proposició 10. *K_0 és indecidible.*

Demostració. Suposem que K_0 és decidible i sigui U_0 una màquina de Turing que decideix K_0. Donada una entrada x suposarem que U_0 para en l'estat q_F si $x \in K_0$ i para en qualsevol altre estat si $x \notin K_0$. Considerem llavors la màquina de Turing U, que en rebre una entrada x actua de la següent manera:

- U determina si l'entrada correspon a la representació d'una màquina de Turing M, és a dir si $x = \rho(M)$ per a alguna M.
- En cas afirmatiu, U simula el còmput de U_0 sobre l'entrada $\rho(M)$. Si U_0 para en l'estat q_F acceptant l'entrada, aleshores farem que U entri en un bucle infinit, de manera que U no pararà mai. En cas contrari, si U_0 para rebutjant l'entrada, farem que U passi directament q_F, fent que U accepti $\rho(M)$.

Així tenim que

$$\begin{aligned} U \text{ para sobre } \rho(M) &\iff U_0 \text{ no accepta } \rho(M) \iff \rho(M) \notin K_0 \\ &\iff M \text{ no para sobre } \rho(M) \end{aligned}$$

Finalment, prenent $M = U$ tenim que

$$U \text{ para sobre } \rho(U) \iff U \text{ no para sobre } \rho(U)$$

que és una contradicció. Per tant, K_0 no pot ser decidible. □

A partir d'aquest resultat ja podem demostrar que A_{TM} tampoc és decidible:

Demostració. Demostració de la proposició 9 Suposem que A_{TM} sigui decidible i sigui U una màquina de Turing que decideix A_{TM}. Considerem la màquina de Turing U_0 que, donada una entrada $\rho(M)$ amb M una màquina de Turing, transforma el seu contingut de la cinta

$$\$\rho(M) * * * \ldots$$

en

$$\$\rho(M)\rho(\rho(M)) * * * \ldots$$

i a continuació passa el control a la màquina U. Aleshores tenim que la màquina U_0 decideix el llenguatge K_0, que acabem de demostrar que és indecidible. Per tant, necessàriament A_{TM} és també indecidible. □

De fet, ni tan sols és decidible el problema de determinar si una màquina de Turing para donada una certa entrada:

Proposició 11.

$HALT_{TM} = \{\rho(M)\rho(w) \mid M$ *és una màquina de Turing que para sobre l'entrada* $w\}$

és indecidible.

Demostració. Evidentment no podem donar una màquina universal que directament simuli M amb l'entrada w, ja que és possible que M no pari, i aleshores la màquina universal tampoc pararia (la màquina universal, però, sí que semidecidiria $HALT_{TM}$). En comptes d'això, suposem que hi ha una màquina H que decideix $HALT_{TM}$, de manera que donat $\rho(M)\rho(w)$ podem comprovar si M para sobre w. Aleshores podem donar una màquina de Turing universal S tal que:

1. S rep com a entrada $\rho(M)\rho(w)$
2. Executa la màquina H amb l'entrada $\rho(M)\rho(w)$
   (a) Si H rebutja l'entrada, S també la rebutja.
   (b) Si H accepta l'entrada vol dir que M para sobre w, de manera que podem executar la màquina M amb l'entrada w. Si en acabar l'execució de M s'accepta w, S accepta l'entrada; si M rebutja, S també.

Aleshores S és una màquina de parada segura que decideix A_{TM}, però per la proposició 9 sabem que això no és possible. Aquesta contradicció prové de suposar que H decideix $HALT_{TM}$, de manera que $HALT_{TM}$ és indecidible. □

Aquesta proposició té una gran importància en el disseny de compiladors, ja que ens nega la possibilitat de determinar si un programa arbitrari acaba o bé entra en un bucle infinit, un error comú en programació. Aquests errors poden ser crítics en sistemes on una component necessita l'entrada d'una altra i no sempre són fàcils de detectar. Hi ha compiladors que implementen mètodes de detecció d'aquest tipus de bucles en els casos trivials, però aquesta proposició ens diu que no és possible fer-ho en el cas general.

4.2 Sistemes de Thue

Definició 22. *Un **sistema semi-Thue** T és un parell (Σ, P) amb*

- *Σ l'alfabet finit dels símbols amb els quals treballarà el sistema*
- *$P \subseteq \Sigma^* \times \Sigma^*$ un conjunt de parells ordenats anomenats **produccions***

Donada una paraula $w \in \Sigma^*$, una producció $(x, y) \in P$ permet substituir en w una instància de la paraula x per y. Les produccions de P comunament s'escriuen com si es tractessin de produccions d'una gramàtica, posant $x \longrightarrow y$ si $(x, y) \in P$.

Seguint la nomenclatura que hem definit per les gramàtiques, escriurem $x \Rightarrow y$ si podem passar de la paraula x a la paraula y utilitzant només una producció de R i $x \Rightarrow^* y$ si es pot passar de x a y utilitzant zero o més produccions.

Observació. *$\Rightarrow^*$ és la relació reflexiva i transitiva sobre Σ^* induïda per les produccions de P.*

Exemple 13. *Sigui $T = (\Sigma, P)$ un sistema semi-Thue amb $\Sigma = \{a, b, c, d, e\}$ i $P = \{(ae, cd), (ad, ed), (db, ea), (a, d)\}$. Aleshores $aabe \Rightarrow^* cdcd$ de la següent manera:*

- *Aplicant la producció (a, d) a la segona a passem de $aabe$ a $adbe$*
- *Aplicant la producció (db, ea) passem de $adbe$ a $aeae$*
- *Aplicant la producció (ae, cd) a les dues primeres parelles de símbols obtenim $cdcd$.*

El problema que immediatament es planteja donat un sistema semi-Thue és el de determinar si dues paraules $u, v \in \Sigma^*$ són equivalents o, en altres paraules, si $u \Rightarrow^* v$. Aquest problema, per innocent que pugui semblar, és indecidible.

Proposició 12. *Hi ha un sistema semi-Thue $T = (\Sigma, P)$ i una paraula $w \in \Sigma^*$ tal que no existeix un algorisme on, donada una paraula v, determini si $w \Rightarrow^* v$.*

Demostració. La idea de la demostració és de reduir el problema de la parada a aquest problema. El fet és que els sistemes semi-Thue són equivalents a les màquines de Turing (com veurem a continuació) i veurem que la decidibilitat del problema de

la paraula en sistemes semi-Thue implicaria una solució del problema de la parada en màquines de Turing.

Considerem una màquina de Turing $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ amb els elements de K numerats com $K = \{q_0, q_1, \ldots, q_n = q_F\}$. Suposem la configuració de M (u, q_i, v) amb $q_i \in K$ i $u, v \in \Gamma^*$.

Si prenem un sistema semi-Thue $T = (K \cup \Gamma \cup \{|, *, \omega_0, \omega_1\}, R)$, podem representar la configuració (u, q_i, v) de la màquina M en la paraula $|uq_iv|$. Aleshores, si $y \in \Gamma$ és el primer símbol de v i $x \in \Gamma$ és l'últim símbol de u, en un únic pas de còmput de M, es pot alterar la paraula xq_iy de les següents maneres:

$$xq_iy \longrightarrow \begin{cases} q_jxz & \text{si } \delta(q_i, y) = (q_j, z, L) \\ xq_jz & \text{si } \delta(q_i, y) = (q_j, z, S) \\ xzq_j & \text{si } \delta(q_i, y) = (q_j, z, R) \end{cases}$$

L'evolució d'una màquina de Turing, doncs, correspon a un conjunt finit de produccions d'un sistema semi-Thue, de manera que obtenim una correspondència entre els dos models.

Amb aquestes produccions, però, no és suficient, ja que no s'ha considerat que la cinta de la màquina de Turing és infinita cap a la dreta. Això només suposarà un problema quan el punter assenyali l'última casella representada per la paraula de T. En aquest cas és necessària una substitució que afegeixi el símbol de cel·la buida $*$ al final de la paraula:

$$q| \longrightarrow q * | \qquad\qquad \forall q \in Q \setminus \{q_F\}$$

També ens cal tractar els casos de parada per separat. Si la màquina M para és perquè δ no està definida (recordem que δ mai està definida per l'estat q_F) i en aquest cas pot ser que l'estat sigui $q_n = q_F$, que acceptarà l'entrada o q_i amb $i \neq n$, que la rebutjarà. En ambdós casos, farem que hi hagi una única seqüència de substitucions que col·lapsin la paraula a un únic símbol corresponent a l'estat:

$$\begin{array}{ll} q_Fz \longrightarrow q_F & \forall z \in \Gamma \\ q_F| \longrightarrow \omega_1| & \\ z\omega_1 \longrightarrow \omega_1 & \forall z \in \Gamma \\ q_iz \longrightarrow \omega_0 & \text{amb } z \in \Gamma \text{ si } \delta(q_i, z) \text{ no està definit i } i \neq n \\ \omega_0z \longrightarrow \omega_0 & \forall z \in \Gamma \\ z\omega_0| \longrightarrow \omega_0| & \forall z \in \Gamma \end{array}$$

La primera producció es podrà dur a terme només si la màquina M arriba a l'estat acceptador. Com que en el moment que es pugui utilitzar aquesta producció l'execució de M ja haurà acabat, afegim una producció que elimini els caràcters a la dreta del punter. Donem també una producció que, un cop eliminats els símbols de la dreta del punter la paraula, passi al símbol ω_1 que indicarà que s'accepta l'entrada obtenint la paraula final $|\omega_1|$. Per tractar el cas on M para però no accepta

l'entrada, també cal afegir un símbol ω_0 amb les produccions que permetin netejar la paraula i obtenir finalment $|\omega_0|$.

Amb aquesta construcció, tenim que la màquina de Turing M accepta una paraula w si i només si $|q_0 w| \Rightarrow^* |\omega_1|$ en el sistema semi-Thue T.

Aplicant aquest resultat a la màquina de Turing universal utilitzada en el problema de la parada veiem que una solució del problema de la paraula pel sistema semi-Thue T implicaria una solució del problema de la parada. Com que no hi ha un algoritme que resolgui el problema de la parada en màquines de Turing, tampoc n'hi pot haver cap que resolgui el problema de la paraula en sistemes semi-Thue. □

Podem considerar també què passaria si imposéssim que les produccions d'un sistema semi-Thue fossin bidireccionals. En aquest cas obtenim un sistema de Thue complet:

Definició 23. *Un sistema de Thue T és un parell (Σ, P) amb*

- *Σ l'alfabet dels símbols amb els quals treballarà el sistema*
- *$P \subseteq \Sigma^* \times \Sigma^*$ un conjunt de parells **no ordenats** anomenats produccions.*

En altres paraules, un sistema de Thue és un sistema semi-Thue on $(x, y) \in P \implies (y, x) \in P$. Per simplificar la notació, sovint escriurem les dues produccions $x \longrightarrow y, y \longrightarrow x$ com $\{x, y\}$, on $\{\}$ denota parell no ordenat.

Exemple 14. *Sigui $T = (\Sigma, P)$ un sistema de Thue amb $\Sigma = \{0, 1\}$ i $P = \{\{01, 10\}, \{0, 00\}, \{11, 1\}\}$. Aleshores, donada la paraula* 01011*, la podem convertir a la paraula* 01010101 *aplicant les següents produccions:*

1. *Aplicant la producció $\{01, 10\}$ podem convertir les dues instàncies de* 01 *en* 10*, obtenint* 101011
2. *Aplicant la producció $\{11, 1\}$ podem convertir el segon* 1 *en* 11 *i els darrers* 11 *en un únic* 1*, obtenint* 101101
3. *Aplicant la producció $\{0, 00\}$ en tots els* 0 *obtenim* 10011001
4. *Finalment, aplicant $\{01, 10\}$ en el primer i tercer parell de símbols, s'obté* 01010101*.*

En aquest cas, la relació $u \Rightarrow^* v$ esdevé una relació d'equivalència (es pot comprovar fàcilment que és reflexiva, simètrica i transitiva) anomenada **congruència de Thue** i l'escriurem com $u \sim v$. Vegem a continuació si el problema de determinar si $u \sim v$ és decidible:

Proposició 13. *Hi ha un sistema de Thue $T = (\Sigma, P)$ i una paraula $u \in \Sigma^*$ tal que no existeix un algorisme on, donada una paraula v, determini si $u \sim v$.*

Demostració. Considerem un sistema semi-Thue $T' = (\Sigma, P')$ construït a partir d'una màquina de Turing i construïm, a partir de T' un sistema de Thue $T = (\Sigma, P)$ tal que $P = \{\{x, y\} | (x, y) \in P' \vee (y, x) \in P'\}$ és a dir, agafant les mínimes

produccions a P' tal que $T = (\Sigma, P)$ sigui sistema de Thue. Suposem que es compleix que $u \sim v$. Aleshores, per a algun enter n es compleix que

$$u = u_1 \sim u_2 \sim \ldots \sim u_n = v$$

on $u_k \Rightarrow_T u_{k+1}$, de manera que cada pas involucra només una producció de P.

Com que $P' \subseteq P$, algunes d'aquestes produccions també es poden realitzar sobre el sistema semi-Thue T'. Considerem, en cas que existeixi, u_i l'última paraula d'aquesta cadena tal que no es pot realitzar la producció $u_i \Rightarrow_{T'} u_{i+1}$ sobre T'.

Observem que, com que u_n és la paraula final provinent d'executar una màquina de Turing, o bé tenim que $u_n = |\omega_0|$ o bé $u_n = |\omega_1|$. Tal com hem construït el sistema semi-Thue, no hi ha cap producció que tingui únicament el símbol ω_1 o ω_0 al costat esquerre, de manera que necessàriament es pot realitzar la producció $u_{n-1} \Rightarrow u_n$ sobre T'.

Així, necessàriament $i \leq n-2$. Si tenim que no es pot realitzar $u_i \Rightarrow u_{i+1}$ sobre T', necessàriament es pot realitzar la producció en sentit invers, obtenint que $u_{i+1} \Rightarrow u_i$ sobre T'. A més, com que u_i és l'última paraula que no permet realitzar la producció sobre T', podem assegurar que $u_{i+1} \Rightarrow u_{i+2}$ es pot realitzar sobre T'. Si observem com hem definit les produccions del sistema semi-Thue T provinent d'una màquina de Turing, veurem que T és totalment determinista: donada una paraula amb un sol símbol del conjunt $K \cup \{\omega_0, \omega_1\}$ hi ha una única producció que es pugui aplicar. Però acabem de veure que tenim tant $u_{i+1} \Rightarrow u_i$ com $u_{i+1} \Rightarrow u_{i+2}$ en T', de manera que necessàriament $u_i = u_{i+2}$. Aleshores podem eliminar $u_i \Rightarrow u_{i+1}$ i $u_{i+1} \Rightarrow u_{i+2}$ de la cadena de produccions que hem considerat inicialment. Si anem repetint aquest procés, podem obtenir una cadena de produccions tals que sempre es pugui realitzar la producció $u_k \Rightarrow u_{k+1}$ sobre el sistema semi-Thue T', obtenint finalment que

$$u \sim v \iff u \Rightarrow^*_{T'} v$$

Així, el problema de la paraula en sistemes de Thue es pot reduir al problema de la paraula en sistemes semi-Thue i, per tant, no és decidible.

□

4.3 El problema de la correspondència de Post

Definició 24. *Un **sistema de correspondència** sobre un alfabet Σ és un conjunt finit P de parells ordenats de paraules no buides sobre Σ. Un **emparellament** a P és una paraula $w \in \Sigma^*$ tal que per a alguna $n > 0$ i per a parelles no necessàriament diferents $(u_1, v_1), (u_2, v_2) \ldots, (u_n, v_n) \in P$ es té que*

$$w = u_1 u_2 \ldots u_n = v_1 v_2 \ldots v_n$$

El problema de la correspondència de Post consisteix a determinar si un determinat sistema de correspondència P té algun emparellament.

Comunament les parelles dels sistemes de correspondència es representen per $\left[\begin{smallmatrix} u_i \\ v_i \end{smallmatrix}\right]$, com si es tractessin de fitxes de dòmino. Aleshores, donat un nombre infinit de peces de cada tipus, el problema de la correspondència de Post consisteix a determinar si existeix una seqüència de peces tals que, posades de costat, la paraula que es llegeixi a la primera fila sigui la mateixa que es llegeixi a la segona:

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} \dots \begin{bmatrix} u_n \\ v_n \end{bmatrix}$$

tal que $u_1 u_2 \dots u_n = v_1 v_2 \dots v_n$.

Teorema 5. *El problema de la correspondència de Post és indecidible.*

Demostració. Sigui $T = (\Sigma, P_T)$ un sistema semi-Thue. Per a cada $u, v \in \Sigma^*$ construirem un sistema de correspondència $C_{u,v}$ tal que el problema de correspondència tingui solució si i només si $u \Rightarrow_T^* v$. Com que sabem que aquest últim problema és indecidible, el problema de la correspondència de Post també serà indecidible.

Comencem afegint les produccions $a \to a$ $\forall a \in \Sigma$ a P_T. Observem que amb aquestes produccions no canviem el sistema semi-Thue, ja que totes les derivacions possibles són idèntiques excepte per la possible aplicació d'una producció que no canvia la paraula. Afegim aquestes produccions per poder suposar que tota derivació d'una paraula a una altra es pot fer en un nombre parell de produccions. Per abús de notació, anomenarem també T al sistema semi-Thue amb aquestes produccions afegides.

Prenent $u, v \in \Sigma^*$ qualssevol construirem el sistema de correspondència $C_{u,v}$, sobre l'alfabet Γ, que contindrà l'alfabet Σ de T i hi afegirem els símbols de Σ amb una cometa i el conjunt de símbols $\{[,], *, *'\}$. Així, $\Gamma = \Sigma \cup \Sigma' \cup \{[,], *, *'\}$, on $\Sigma' = \{a' \mid a \in \Sigma\}$ i $[,], *, *' \notin \Sigma$. Per simplificar la notació, si $w = a_1 a_2 \dots a_k \in \Sigma^*$, escriurem w' per indicar la paraula $a'_1 a'_2 \dots a'_k \in (\Sigma')^*$. Aleshores, per a cada producció de $x \longrightarrow y$ de P_T (incloses les del tipus $x \longrightarrow x$) definirem les parelles

$$\begin{bmatrix} y' \\ x \end{bmatrix}, \begin{bmatrix} y \\ x' \end{bmatrix}$$

A aquestes afegim les parelles

$$\begin{bmatrix} [u* \\ [\end{bmatrix}, \begin{bmatrix} * \\ *' \end{bmatrix}, \begin{bmatrix} *' \\ * \end{bmatrix}, \begin{bmatrix}] \\ *'v] \end{bmatrix}$$

També, per simplificar la notació, quan tinguem seqüències de peces de la forma

$$\begin{bmatrix} a_1 \\ a'_1 \end{bmatrix} \begin{bmatrix} a_2 \\ a'_2 \end{bmatrix} \dots \begin{bmatrix} a_k \\ a'_k \end{bmatrix} \text{ o } \begin{bmatrix} a'_1 \\ a_1 \end{bmatrix} \begin{bmatrix} a'_2 \\ a_2 \end{bmatrix} \dots \begin{bmatrix} a'_k \\ a_k \end{bmatrix}$$

amb $a_i \in \Sigma$, les podrem simplificar per les peces compostes

$$\begin{bmatrix} w \\ w' \end{bmatrix} \text{ i } \begin{bmatrix} w' \\ w \end{bmatrix}$$

respectivament, on $w = a_1 \dots a_k \in \Sigma^*$.

Abans de res, anem a veure que si trobem una solució $u \Rightarrow_T^* v$ del sistema semi-Thue T, aleshores el problema de correspondència de Post $C_{u,v}$ també en té una. Suposem que

$$u = u_0 \Rightarrow u_1 \Rightarrow \dots u_n = v$$

és una derivació en T de longitud parella (si $u \Rightarrow^* v$ és senar podem afegir una producció del tipus $a \longrightarrow a$ i obtenir un nombre parell de produccions). Observem que si l'i-éssim pas de la derivació és $u_{i-1} \Rightarrow u_i$ i aquesta derivació s'obté d'aplicar la producció $x_j \longrightarrow y_j$ de P_T, el podem escriure com

$$u_{i-1} = p_{i-1}x_jq_{i-1} \Rightarrow p_{i-1}y_jq_{i-1} = u_i$$

on $p_{i-1}, q_{i-1} \in \Sigma^*$. Aleshores, la paraula $[u_0 * u_1' *' u_2 * \dots u_{n-1}' *' u_n]$ és una solució del sistema de correspondència de Post, i es pot construir de la següent manera:

1. Inicialment s'utilitza el parell
$$\begin{bmatrix} [u* \\ [\end{bmatrix}$$

2. Aleshores cal escriure la paraula u seguida de $*$ a la segona component per compensar la que hi ha a la primera component. Com que $u = u_0$ i per l'observació que hem fet abans tenim que $u_0 = p_0x_{j_0}q_0 \Rightarrow p_0y_{j_0}q_0 = u_1$, podem completar u fent
$$\begin{bmatrix} [u* \\ [\end{bmatrix} \begin{bmatrix} p_0' \\ p_0 \end{bmatrix} \begin{bmatrix} y_{j_0}' \\ x_{j_0} \end{bmatrix} \begin{bmatrix} q_0' \\ q_0 \end{bmatrix} \begin{bmatrix} *' \\ * \end{bmatrix}$$
on les peces $\left[\begin{smallmatrix} p_0' \\ p_0 \end{smallmatrix}\right]$ i $\left[\begin{smallmatrix} q_0' \\ q_0 \end{smallmatrix}\right]$ són peces compostes.

3. Com que $p_0y_{j_0}q_0 = u_1$, la cadena resultant de concatenar la primera component es pot escriure com $[u_0 * u_1'*'$ i la de la segona component com $[u_0*$. Seguint la mateixa estratègia que en el pas anterior podem obtenir u_1' a la segona component:
$$\begin{bmatrix} [u* \\ [\end{bmatrix} \begin{bmatrix} p_0' \\ p_0 \end{bmatrix} \begin{bmatrix} y_{j_0}' \\ x_{j_0} \end{bmatrix} \begin{bmatrix} q_0' \\ q_0 \end{bmatrix} \begin{bmatrix} *' \\ * \end{bmatrix} \begin{bmatrix} p_1 \\ p_1' \end{bmatrix} \begin{bmatrix} y_{j_1} \\ x_{j_1}' \end{bmatrix} \begin{bmatrix} q_1 \\ q_1' \end{bmatrix} \begin{bmatrix} * \\ *' \end{bmatrix}$$
obtenint la paraula $[u_0 * u_1' *' u_2*$ a la primera component.

4. Aquest mateix procés es va repetint pels passos $2, 3, \dots, n-1$ de la derivació, obtenint
$$\begin{bmatrix} [u* \\ [\end{bmatrix} \begin{bmatrix} p_0' \\ p_0 \end{bmatrix} \begin{bmatrix} y_{j_0}' \\ x_{j_0} \end{bmatrix} \begin{bmatrix} q_0' \\ q_0 \end{bmatrix} \begin{bmatrix} *' \\ * \end{bmatrix} \begin{bmatrix} p_1 \\ p_1' \end{bmatrix} \begin{bmatrix} y_{j_1} \\ x_{j_1}' \end{bmatrix} \begin{bmatrix} q_1 \\ q_1' \end{bmatrix} \begin{bmatrix} * \\ *' \end{bmatrix} \dots \begin{bmatrix} p_{n-1} \\ p_{n-1}' \end{bmatrix} \begin{bmatrix} y_{j_{n-1}} \\ x_{j'_{n-1}} \end{bmatrix} \begin{bmatrix} q_{n-1} \\ q_{n-1}' \end{bmatrix}$$

5. Finalment, es completa la seqüència afegint
$$\begin{bmatrix}] \\ *'v] \end{bmatrix}$$
amb la qual s'obté la paraula $[u_0*u_1'*'u_2*\dots u_{k-1}'*'u_n]$ a les dues components, resolent el problema de la correspondència de Post $C_{u,v}$.

Per tant, tenim que $u \Rightarrow_T^* v \implies C_{u,v}$ té una solució.

Ara ens cal veure que si $w \in \Sigma$ tal que $u \Rightarrow_T^* w$, aquesta derivació es pot obtenir mitjançant una solució del sistema de correspondència $C_{u,v}$. Tal com s'ha construït les parelles de $C_{u,v}$, necessàriament tota solució ha de començar amb

$$\left[\begin{matrix}[u* \\ [\end{matrix}\right]$$

ja que és l'única parella que té el mateix símbol inicial a les dues components. Pel mateix argument, una solució ha d'acabar amb

$$\left[\begin{matrix}] \\ *'v]\end{matrix}\right]$$

Així, tota paraula que solucioni el problema de correspondència és de la forma $[u * r *' v]$, per a alguna $r \in \Gamma^*$. Si r conté], aleshores la solució es pot escriure com $[u * s *' v]t *' v]$ on $r = s *' v]t$, ja que el símbol] només apareix a la parella $\left[\begin{smallmatrix}] \\ *'v]\end{smallmatrix}\right]$. En aquest cas, tindríem que $[u * s *' v]$ també és solució de $C_{u,v}$. Així, sense pèrdua de generalitat podem suposar que $[u * \ldots *' v]$ és una solució de $C_{u,v}$ on el símbol] apareix una sola vegada.

Les parelles dels extrems ja determinen l'estructura que ha de tenir tota la solució: com que ha de començar amb $\left[\begin{smallmatrix}[u* \\ [\end{smallmatrix}\right]$, cal afegir peces que de manera que a la segona component aparegui u. Suposem que $u = x_{i_0}x_{i_1}\ldots x_{i_k}$ amb $x_{i_j} \in \Sigma^*$. Aleshores, per completar u a la segona component hem d'afegir peces tals que la concatenació de les segones components sigui $x_{i_0}x_{i_1}\ldots x_{i_k}$:

$$\left[\begin{matrix}[u* \\ [\end{matrix}\right] \left[\begin{matrix}y'_{i_0} \\ x_{i_0}\end{matrix}\right] \left[\begin{matrix}y'_{i_1} \\ x_{i_1}\end{matrix}\right] \cdots \left[\begin{matrix}y'_{i_k} \\ x_{i_k}\end{matrix}\right] \left[\begin{matrix}* \\ *'\end{matrix}\right]$$

Com que les úniques peces que no contenen $*$ són peces que provenen de produccions (ja siguin de les originals de P_T o de les que hem afegit del tipus $a \longrightarrow a$), tenim que cada peça representa una producció $x_{i_j} \longrightarrow y_{i_j}$ i, per tant, si definim $u_1 = y_{i_0}y_{i_1}\ldots y_{i_k}$ tenim que $u \Rightarrow_T^* u_1$ (aplicant precisament la seqüència de produccions que representen les peces que hem afegit).

Ara la paraula de la primera component és $[u * u'_1 *'$ i a la segona component hi ha $[u*$. Seguint aquest mateix procés, obtenim la seqüència de paraules següent:

$$\begin{aligned}
&[u * u'_1 *' u_2* \\
&[u * u'_1 *' u_2 * u'_3 *' \\
&[u * u'_1 *' u_2 * u'_3 *' u_4* \\
&\vdots \\
&[u * u'_1 *' u_2 * u'_3 *' \ldots u'_{n-1} *' v]
\end{aligned}$$

On $u_k \Rightarrow_T^* u_{k+1}$, amb $u_0 = u$ i $u_n = v$. Efectivament l'última paraula d'aquesta seqüència és la solució del problema de correspondència de Post i seguint les produccions que defineixen la solució podem obtenir que $u \Rightarrow_T^* v$, tal com volíem veure.

Així, hem demostrat que

$$u \Rightarrow_T^* v \iff C_{u,v} \text{ té solució}$$

i per tant hem obtingut una reducció del problema de la paraula per sistemes semi-Thue al problema de la correspondència de Post. Com que el primer és indecidible, necessàriament el segon també ho és. □

4.4 Ambigüitat de les gramàtiques incontextuals

Recordem que una gramàtica incontextual G és ambigua si per a alguna paraula $w \in L(G)$ hi ha dos arbres de derivació diferents.

Proposició 14. *No hi ha cap algorisme que permeti determinar si una gramàtica incontextual G és ambigua o no.*

Observació. *La classe dels llenguatges incontextuals és tancada respecte a la unió.*

Demostració. Prenem $G_1 = (V_1, \Delta, R_1, S_1)$ i $G_2 = (V_2, \Delta, R_2, S_2)$ i sigui

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \longrightarrow S_1, S \longrightarrow S_2\}, S)$$

on S és un nou símbol. Aleshores volem veure que $L(G_1) \cup L(G_2) = L(G)$. Com que les úniques produccions que involucren S són $S \longrightarrow S_1$ i $S \longrightarrow S_2$, tenim que $S \Rightarrow_G^* w$, amb $w \in (\Sigma_1 \cup \Sigma_2)^*$, si i només si o bé $S_1 \Rightarrow_G^* w$ o bé $S_2 \Rightarrow_G^* w$. D'altra banda, com que el conjunt de variables de G_1 és disjunt del de G_2 (ho podem suposar sense pèrdua de generalitat, en cas que no ho fossin només caldria reanomenar les variables d'una de les gramàtiques), tenim que $S_1 \Rightarrow_G^* w$ si i només si $S_1 \Rightarrow_{G_1}^* w$ i podem fer el raonament simètric per a G_2. □

Demostració (Proposició 14). Prenem un sistema de correspondència $P \subseteq \Sigma^* \times \Sigma^*$ qualsevol sobre un alfabet Σ. Suposem que $P = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ conté exactament n parelles i considerem n símbols $a_1, \ldots, a_n$ tal que $a_i \notin \Sigma \; \forall i \in \{1, \ldots, n\}$ juntament amb dos símbols S_1 i S_2 que tampoc són de Σ. Sigui $\Delta = \Sigma \cup \{a_1, \ldots, a_n\}$ i considerem $V_1 = \Delta \cup \{S_1\}$ i $V_2 = \Delta \cup \{S_2\}$. Sobre aquests alfabets construïm les gramàtiques $G_1 = (V_1, \Delta, R_1, S_1)$ i $G_2 = (V_2, \Delta, R_2, S_2)$ on

$$R_1 = \{S_1 \longrightarrow a_i S_1 u_i \mid i = 1, \ldots, n\} \cup \{S_1 \longrightarrow a_i u_i \mid i = 1, \ldots, n\}$$

$$R_2 = \{S_2 \longrightarrow a_i S_2 v_i \mid i = 1, \ldots, n\} \cup \{S_2 \longrightarrow a_i v_i \mid i = 1, \ldots, n\}$$

A partir d'aquestes produccions és senzill veure que

$$L(G_1) = \{a_{i_k} \ldots a_{i_1} u_{i_1} \ldots u_{i_k} \mid k \geq 1 \wedge 1 \leq i_1, \ldots i_k \leq n\}$$

$$L(G_2) = \{a_{i_k} \dots a_{i_1} v_{i_1} \dots v_{i_k} \mid k \geq 1 \wedge 1 \leq i_1, \dots i_k \leq n\}$$

Prenem, doncs, la gramàtica provinent de la unió de G_1 i G_2

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \longrightarrow S_1, S \longrightarrow S_2\}, S)$$

on S és una nova variable. Com ja hem vist, $L(G) = L(G_1) \cup L(G_2)$. A continuació demostrarem que la gramàtica G és ambigua si i només si el problema de la correspondència de Post sobre P té solució.

Suposem que P té una solució donada per les parelles $(u_{i_1}, v_{i_1}), \dots, (u_{i_k}, v_{i_k})$, de manera que $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$. En aquest cas, la paraula $a_{i_k} \dots a_{i_1} u_{i_1} \dots u_{i_k} = a_{i_k} \dots a_{i_1} v_{i_1} \dots v_{i_k}$ té dues derivacions possibles: una provinent de G_1 i una altra provinent de G_2 i per tant, G és ambigua.

Per altra banda, si $w \in L(G)$ té dos arbres de derivació diferents, aleshores necessàriament $w \in L(G_1)$ i $w \in L(G_2)$, ja que ni G_1 ni G_2 són ambigües de manera separada. Però llavors w es pot escriure com $w = a_{i_k} \dots a_{i_1} u_{i_1} \dots u_{i_k} = a_{i_k} \dots a_{i_1} v_{i_1} \dots v_{i_k}$ per a alguns $i_1, \dots, i_k$ amb $k \geq 1$, i per tant $u_{i_1} \dots u_{i_k}$ és solució del sistema de correspondència de Post. □

4.5 El problema de la paraula per a semigrups

Definició 25 (Definicions bàsiques d'àlgebra)**.**

1. *Un **semigrup** $(S, \cdot)$ és una estructura algebraica formada per un conjunt S i una operació interna, binària i associativa $\cdot\colon S \times S \longrightarrow S$, en el context d'àlgebra sovint anomenada **producte**, tot i que en el context de lògica l'anomenarem **concatenació**. Fent abús de notació, denotarem el semigrup per S sense fer referència a l'operació i, si $a, b \in S$, escriurem ab per denotar $a \cdot b$.*

2. *Si S conté també un element neutre respecte l'operació (i.e. un element $e \in S$ tal que $ex = xe = x \ \forall x \in S$), aleshores diem que S és un **monoide**. Mentre que en àlgebra aquest element se sol representar mitjançant 1, que és l'element neutre del producte, des del punt de vista dels llenguatges, aquest element seria la paraula buida λ.*

3. *Si S és un semigrup, denotem per S^1 el monoide idèntic a S en cas que S tingui element neutre o $S \cup \{\lambda\}$ en cas contrari. En el segon cas, el producte original del semigrup s'estén de manera que $\lambda x = x\lambda = x \ \forall x \in S^1$.*

4. *Si $(S, \cdot)$ és un semigrup (respectivament monoide), diem que $T \subseteq S$ és un **subsemigrup** (resp. **submonoide**) si $(T, \cdot)$ forma un semigrup (resp. monoide). De manera potser més formal, $(T, \cdot)$ és un subsemigrup de $(S, \cdot)$ si la restricció de $\cdot$ a $T \times T$ és també interna, binària i associativa.*

5. *Un **monoide lliure sobre un conjunt** Σ és el monoide els elements del qual són totes les seqüències de 0 o més elements de Σ (prenent la concatenació com a operació del monoide i l'element neutre com a seqüència de longitud 0). El monoide lliure sobre un conjunt Σ es denota per Σ^*. El **semigrup lliure***

sobre Σ és el subsemigrup de Σ^* que conté tots els elements de Σ^* excepte la paraula buida i es denota per Σ^+.

La tria de nomenclatura per a aquesta última definició no és pas casual: si Σ és un alfabet, el monoide Σ^* no és res més que el conjunt de paraules sobre Σ, que sempre hem escrit com Σ^*. Amb aquesta última definició, doncs, ja es comença a entreveure el lligam entre l'àlgebra i els llenguatges formals.

Proposició 15. *Donat un sistema de Thue (Σ, P), tenim que Σ^* és el monoide lliure sobre Σ amb l'operació de concatenació i que la congruència de Thue $\sim_P$ és una relació d'equivalència sobre Σ^*. Aleshores el quocient $M := {}^{\Sigma^*}\!/_{\sim}$ és també un monoide amb la concatenació.*

Demostració. Abans de res, cal observar que la congruència de Thue és compatible amb l'operació de concatenació, és a dir, si $u_0 \sim v_0$ i $u_1 \sim v_1$, aleshores $u_0u_1 \sim v_0v_1$, $\forall u_0, u_1, v_0, v_1 \in \Sigma^*$. Aleshores, tenim que $[u]_\sim = \{x \in \Sigma^* \mid x \sim u\} \in M$ i, com que $\sim$ és compatible amb la concatenació, $[u]_\sim[v]_\sim = [uv]_\sim$. També tenim que $\lambda u \sim u$, de manera que $[\lambda]_\sim[u]_\sim = [\lambda u]_\sim = [u]_\sim$, i per tant $[\lambda]_\sim$ actua com a element neutre de M.

Així, M compleix totes les condicions per ser un monoide, i diem que Σ és el conjunt de **generadors** de M i P és el conjunt de **relacions** de M. Diem també que el sistema (Σ, P) forma una **presentació** de M i escriurem $M = \langle \Sigma | P \rangle$. □

Un altre resultat diu que l'invers d'aquesta posposició també és cert, és a dir, es pot demostrar que tot monoide accepta a un sistema de Thue (amb alfabet possiblement infinit) com a presentació. Aquest resultat no l'utilitzarem, però essencialment ens dona una manera de passar d'un sistema de Thue (i per tant també d'una màquina de Turing!) a un monoide i al revés. Es pot trobar una demostració d'aquest resultat a [9], pàg. 149.

Finalment, podem enunciar el problema de la paraula per a monoides: donada una presentació (Σ, P) d'un monoide M i un parell de paraules $u, v \in M$, determinar si $u = v$.

Teorema 6. *Hi ha una presentació (Σ, P) d'un monoide M i una paraula $u \in M$ tal que no existeix un algorisme on, donada una paraula $v \in M$, determini si $u = v$.*

Demostració. Se segueix immediatament del problema de la paraula per a sistemes de Thue, només cal prendre la presentació donada pel sistema de Thue (Σ, P) de la proposició 12. □

Per tant, el problema de la paraula per a monoides és, en general, indecidible.

4.6 El problema de la paraula per a grups

Definició 26. *Diem que un monoide M és un **grup**, si $\forall u \in M\ \exists v \in M$ tal que $uv = \lambda$. És a dir, un grup és un monoide on cada element té un invers respecte*

l'operació.

Aquesta condició sembla curiosa des del punt de vista de llenguatges formals, ja que la concatenació de dues paraules no necessàriament buides en general no és la paraula buida. Ara bé, donat un sistema de Thue $T = (\Sigma, P)$, podem generar un altre sistema de Thue $T' = (\Sigma', P')$ tal que el monoide amb presentació (Σ', P') sigui un grup:

Proposició 16. *Donat $T = (\Sigma, P)$, definim l'alfabet $\Sigma^{-1} := \{x^{-1} \mid x \in \Sigma\}$ i prenem el sistema de Thue T' amb alfabet $\Sigma' := \Sigma \cup \Sigma^{-1}$ i $P' = P \cup \{\{xx^{-1}, \lambda\}\ \forall x \in \Sigma\} \cup \{\{x^{-1}x, \lambda\}\ \forall x \in \Sigma\}$. Aleshores el monoide presentat per $T' = (\Sigma', P')$ és un grup.*

Demostració. Partint del fet que $T' = (\Sigma', P')$ és un sistema de Thue, cal veure que el monoide $M_{T'}$ presentat per T' és un grup. Els elements de $M_{T'}$ són, per definició, els elements del quocient del monoide lliure $(\Sigma')^*$ i la congruència de Thue $\sim_{P'}$. Així, la paraula buida λ és a $M_{T'}$ i la resta d'elements són paraules $a_1^{e_1}a_2^{e_2}\dots a_k^{e_k}$, amb $a_i \in \Sigma$ i $e_i \pm 1$, on fem la identificació de a_i^{+1} amb a_i. Per tant, donada una paraula $w = w_1^{e_1}w_2^{e_2}\dots w_{k-1}^{e_{k-1}}w_k^{e_k} \in (\Sigma')^*$, el seu invers $w_k^{-e_k}w_{k-1}^{-e_{k-1}}\dots w_2^{e_2}w_1^{e_1}$, que denotem w^{-1}, també és a $M_{T'}$ i $ww^{-1} = w^{-1}w = \lambda$. □

Definició 27. *Diem que un monoide M amb presentació (Σ, P) és **finitament generat** si Σ és finit; és **finitament presentat** si tant Σ com P són finits; i és **recursivament presentat** si Σ és enumerable i P és recursivament enumerable.*

Un cop hem vist el problema de la paraula per a monoides, el més natural és considerar un problema similar per a grups. L'enunciat del problema és el següent: donat un grup finitament presentat G i una paraula $w \in G$, determinar si $w = \lambda$.

Teorema 7 (Novikov-Boone)**.** *El problema de la paraula per a grups finitament presentats és indecidible.*

Aquest problema el va plantejar per primera vegada M. Dehn el 1911, i tant P.S. Novikov com W. Boone van demostrar que era indecidible el 1955 i 1959 respectivament. Les demostracions feien una reducció al problema de la parada de màquines de Turing, generant un grup finitament presentat a partir d'una màquina de Turing de manera que un element del grup era la identitat si i només si la màquina de Turing parava donat l'element com a entrada. Aquesta demostració és complexa i molt llarga, i queda fora de l'àmbit del treball.

El 1991, però, G. Baumslag va donar una altra demostració mitjançant el teorema de la immersió de Higman, que utilitza eines que sí que s'ha desenvolupat al llarg del treball:

Teorema 8 (Immersió de Higman)**.** *Un grup R finitament generat és recursivament presentable si i només si és un subgrup d'algun grup finitament presentat G.*

La demostració d'aquest teorema es pot trobar a [10], pàgs. 451-464, i també queda fora de l'àmbit del treball.

Ara bé, a partir del teorema de la immersió de Higman es pot demostrar Novikov-Boone de la següent manera:

Idea de la demostració (Novikov-Boone). Sigui S un conjunt recursivament enumerable però no computable d'enters. Aleshores, en el grup recursivament presentat

$$G_S = \langle a, b, c, d \mid a^n b a^{-n} = c^n d c^{-n} \rangle$$

tenim que $a^n b a^{-n} c^{-n} d^{-1} c^n = 1 \iff n \in S$, però com que S no és computable, el problema de la paraula en G_S és indecidible. Ara, pel teorema de la immersió de Higman, G_S és subrup d'algun grup finitament presentat G, de manera que G també té un problema de la paraula indecidible.

A part del teorema de la immersió de Higman, l'altre element clau de la demostració és el conjunt recursivament enumerable però no computable d'enters. Amb les eines que hem desenvolupat, però, es pot construir un conjunt amb aquestes característiques:

Proposició 17. *El conjunt $S_H = \{\rho(M)\rho(w) \mid M(w)\downarrow\}$ és recursivament enumerable però no computable.*

Demostració. Una màquina de Turing universal semidecideix S_H, ja que fa una simulació de la màquina M amb entrada w i per tant només para si $M(w)\downarrow$. Ara bé, el seu complement, $\bar{S_H} = \{\rho(M)\rho(w) \mid M(w)\uparrow\}$ no pot ser recursivament enumerable pel problema de la parada en màquines de Turing. □

Com que la codificació $\rho(M)\rho(w)$ queda determinada per seqüències binàries, podem assignar a cada element de S_H el nombre natural representat per la seqüència binària, obtenint així un conjunt de nombres naturals recursivament enumerable però no computable.

4.7 El problema de l'anul·lació de matrius

Considerem un conjunt S de matrius quadrades $n \times n$, amb $n \in \mathbb{N}$ amb elements enters i considerem el semigrup multiplicatiu lliure M generat per S. Els elements de M són, doncs, productes qualssevol de matrius de S. El problema de l'anul·lació de matrius consisteix a determinar si $0 \in M$.

Exemple 15. *El semigrup multiplicatiu lliure generat per les matrius $\left(\begin{smallmatrix} 3 & 2 \\ -1 & 0 \end{smallmatrix}\right)$ i $\left(\begin{smallmatrix} -4 & 1 \\ 0 & -5 \end{smallmatrix}\right)$ no es pot anul·lar, ja que les dues matrius tenen determinant no nul i per tat tot producte d'aquestes matrius també té determinant no nul. En canvi, el semigrup lliure generat per $\left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$ i $\left(\begin{smallmatrix} 0 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ sí que es pot anul·lar, ja que el producte dels dos generadors ja dona una matriu nilpotent.*

La decidibilitat del problema de l'anul·lació depèn de la dimensió de les matrius i, per exemple, es pot demostrar que és indecidible per conjunts de 15 matrius 3×3 o conjunts de dues matrius 24×24 però, en canvi, en data d'avui no se sap si és indecidible per tres matrius 2×2.

Proposició 18. *El problema de l'anul·lació de matrius és indecidible.*

Demostració. Demostrarem el resultat per a matrius 3×3 fent una reducció al problema de la correspondència de Post. Per fer-ho, necessitem una manera de codificar paraules dins una matriu.

Considerem l'alfabet $\Sigma = \{1, 2, 3\}$ i per a cada paraula $w = a_1 \dots a_n \in \Sigma^*$ definim l'aplicació injectiva $W \colon \Sigma^* \longrightarrow \mathbb{N}$ tal que

$$W(w) = W(a_1 \dots a_n) = \sum_{i=1}^{n} 4^{n-i} a_i$$

prenent en aquest cas el valor numèric d'a_i. Definim també una aplicació $M \colon \Sigma^* \times \Sigma^* \mapsto \mathbb{N}^{3\times 3}$ tal que

$$M(u, v) = \begin{pmatrix} 4^{|u|} & 0 & 0 \\ 0 & 4^{|v|} & 0 \\ W(u) & W(v) & 1 \end{pmatrix}$$

Observem que si $u_1, v_1, u_2, v_2 \in \Sigma^*$, aleshores $M(u_1, v_1)M(u_2, v_2) = M(u_1u_2, v_1v_2)$, prenent el producte de matrius al domini i l'operació de concatenació al codomini (de fet M és un homomorfisme de semigrups).

Prenem ara la matriu B definida com

$$\begin{pmatrix} 1 & 0 & 1 \\ -1 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix}$$

Aquesta matriu compleix que $B = B^2$ i també compleix que $BM(u, v)B = (4^{|u|} + W(u) - W(v))B$. Aquesta igualtat implica que

$$BM(u, v)B = 0 \iff 4^{|u|} + W(u) - W(v) = 0$$

Per la construcció de W, tenim que $W(1u) = 4^{|u|} + W(u)$, de manera que podem escriure la implicació anterior com

$$BM(u, v)B = 0 \iff W(1u) = W(v) \iff 1u = v$$

Utilitzarem aquestes relacions per establir la reducció del problema de l'anul·lació al problema de la correspondència de Post.

Agafem un sistema de Post amb k peces sobre un alfabet Γ. Podem suposar sense pèrdua de generalitat que l'alfabet Γ és binari (mitjançant una construcció similar a la utilitzada en l'apartat 2.3 per representar màquines de Turing), però en comptes d'utilitzar els símbols 0 i 1, prendrem els símbols 2 i 3 (simplement prenent $0 \mapsto 2$ i $1 \mapsto 3$). Siguin (x_i, y_i) amb $i = 1, \dots, k$ les peces que conformen el sistema de Post. Aleshores, per a cada peça definim dues matrius: $M_i = M(x_i, y_i)$ i $M'_i = M(x_i, 1y_i)$. Sigui G el semigrup multiplicatiu lliure generat pel conjunt de $2k + 1$ matrius $\{M_i \mid i = 1, \dots, k\} \cup \{M'_i \mid i = 1, \dots, k\} \cup \{B\}$.

Suposem que el problema de correspondència de Post té una solució que s'obté mitjançant la seqüència de peces $(x_{w_1}, y_{w_1}), (x_{w_2}, y_{w_2}), \dots (x_{w_n}, y_{w_n})$. Aleshores afirmem que

$$BM'_{w_1} M_{w_2} \dots M_{w_n} B = 0$$

Anem a veure que és cert: tenim, en primer lloc, que

$$x_{w_1} x_{w_2} \dots x_{w_n} = y_{w_1} y_{w_2} \dots y_{w_n}$$

per ser solució del problema de correspondència de Post. D'altra banda, podem escriure el producte de matrius anterior com

$$\begin{aligned} M'_{w_1} M_{w_2} \dots M_{w_n} &= M(x_{w_1}, 1y_{w_1}) M(x_{w_2}, y_{w_2}), \dots M(x_{w_n}, y_{w_n}) = \\ &= M(x_{w_1} x_{w_2} \dots x_{w_n}, 1y_{w_1} y_{w_2} \dots y_{w_n}) \end{aligned}$$

Si ara prenem $u = x_{w_1} x_{w_2} \dots x_{w_n}$ i $v = 1y_{w_1} y_{w_2} \dots y_{w_n}$, es compleix que $1u = v$, de manera que

$$BM(u, v)B = 0$$

tal com volíem veure.

Ara només ens cal veure que si $0 \in G$, el sistema de correspondència de Post associat admet una solució. Suposem, doncs, que $0 \in G$. Aleshores hi ha una seqüència de matrius tals que el seu producte és 0. Com que $B^2 = B$ i M és homomorfisme, podem escriure aquest producte sense pèrdua de generalitat com

$$BA(u_{w_1}, v_{w_1}) BA(u_{w_2}, v_{w_2}) B \dots A(u_{w_n}, v_{w_n}) B = 0$$

per a alguna seqüència $(w_k)_{k=1}^n$ i on $A(u_{w_t}, v_{w_t})$ és o bé $M_{w_t} = M(u_{w_t}, v_{w_t})$ o bé $M'_{w_t} = M(u_{w_t}, 1v_{w_t})$. Tot aquest producte només es pot anul·lar si hi ha algun $i \in \{1, \dots, k\}$ tal que $BA(u_{w_i}, v_{w_i})B = 0$ i, per tant si $1u_{w_i} = v_{w_i}$. Notem que per com hem definit els elements del semigrup necessàriament $u_{w_i} \in \{2, 3\}^*$, però $v_{w_i} \in \{1, 2, 3\}^*$, ja que les matrius M' afegeixen el símbol 1. Si tenim que $1u_{w_i} = v_{w_i}$, però, l'únic símbol de v_{w_i} que és 1 és el primer, i aleshores, eliminant el primer símbol de $1u_{w_i} = v_{w_i}$ obtenim una solució del problema de la correspondència de Post. □

Capítol 5

Conclusions

Durant aquest treball hem anat seguint un camí que ens ha portat des dels inicis de la formalització dels llenguatges fins a les demostracions de diversos problemes indecidibles, utilitzant eines per reconèixer i generar llenguatges. Hem demostrat les limitacions de diversos autòmats i gramàtiques per tractar llenguatges arbitraris però també la seva importància, com per exemple en el disseny de compiladors. Hem vist com les màquines de Turing superaven aquestes limitacions i la tesi de Church-Turing, que en última instància les equipara a la noció d'algorisme. Hem demostrat que el problema de la parada per a màquines de Turing és indecidible i com podem reduir nombrosos problemes indecidibles a aquest. Finalment hem donat encara més sistemes que tenen problemes indecidibles, veient que aquests problemes no estan ni molt menys restringits a la teoria d'autòmats i llenguatges i que, problemes aparentment innocents, poden acabar resultant indecidibles.

Al llarg d'aquest camí, però, hem hagut d'abandonar molts trencants que proposaven qüestions interessants però que han hagut de quedar fora del treball. Hi ha, per exemple, tota una teoria de llenguatges formals que permet obtenir el que s'anomena la jerarquia de Chomsky, una classificació de tots els possibles tipus de llenguatges; hi ha tots els models de computació proposats paral·lelament a la màquina de Turing, cadascun amb les seves particularitats; hi ha tota la teoria que estudia les funcions recursives, esmentades durant el treball però que no hem desenvolupat; els lligams de la decidibilitat amb les nocions de completesa i consistència que plantejava Hilbert; la formalització de la teoria d'autòmats a partir de l'àlgebra...

Així, el treball ha servit com a presa de contacte amb la teoria de computabilitat, però també com a punt de partida per a la descoberta de totes aquestes meravelloses idees, que esperen ser explorades.

I és que, com Alan Turing digué,

> "Només en podem veure una mica, del futur, però en veiem prou com per saber que hi ha molt per fer."

Apèndix A

Extensió de teoria d'autòmats, gramàtiques i màquines de Turing

A.1 Expressions regulars

Definició 28. *Podem definir les expressions regulars sobre un alfabet Σ recursivament de la següent manera:*

- $\emptyset$*, denotant el conjunt buit, és una expressió regular*
- λ*, la paraula buida, és una expressió regular*
- $\forall a \in \Sigma$*, a és una expressió regular*

Si α i β són expressions regulars, $(\alpha \cup \beta)$, $(\alpha \circ \beta)$ i (α^*) també són expressions regulars.

Aleshores, donada una expressió regular α, denotem per $L(\alpha)$ el llenguatge generat mitjançant les següents regles:

- $L(\emptyset) = \emptyset$ i $L(\lambda) = \{\lambda\}$
- Si $a \in \Sigma$, aleshores $L(a) = \{a\}$
- $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$
- $L(\alpha \circ \beta) = L(\alpha) \circ L(\beta)$
- $L(\alpha^*) = L(\alpha)^*$

Direm que un llenguatge L és **regular** si existeix una expressió regular α tal que $L = L(\alpha)$.

Teorema 9. *Un llenguatge és regular si i només si és acceptat per un autòmat finit.*

Demostració. Sigui α una expressió regular. Volem generar un autòmat indeterminista M tal que $L(M) = L(\alpha)$. Separem la demostració en diferents casos base:

- $\boxed{\alpha = \emptyset}$: Aleshores $L(\alpha) = \emptyset$ i l'autòmat indeterminista $M = (\{q\}, \Sigma, \Delta, q, \emptyset)$ amb $\Delta(p, a) = \emptyset \ \forall p, a$ reconeix $L(\alpha)$.
- $\boxed{\alpha = \lambda}$: En aquest cas $L(\alpha) = \{\lambda\}$ i l'autòmat indeterminista $M = (\{q\}, \Sigma, \Delta, q, \{q\})$ amb $\Delta(p, a) = \emptyset \ \forall p, a$ reconeix $L(\alpha)$.
- $\boxed{\alpha = x}$ amb $x \in \Sigma$: En aquest cas $L(\alpha) = \{x\}$ i l'autòmat indeterminista $M = (\{q_1, q_2\}, \Sigma, \Delta, q_1, \{q_2\})$ amb $\Delta(q_1, x) = \{q_2\}$ i $\Delta(p, a) = \emptyset \ \forall p \neq q_1, \forall a \neq x$ reconeix $L(\alpha)$

Aleshores, com que hem demostrat que la classe dels llenguatges reconeguts per autòmats és tancada per unió, concatenació i clausura de Kleene i es pot generar qualsevol expressió regular a partir de les tres que hem definit i aquestes tres operacions, podem construir qualsevol autòmat indeterminista a partir d'una expressió regular fent servir els mateixos procediments per construir autòmats que hem fet servir abans.

Sigui ara $M = (K, \Sigma, \Delta, q_0, F)$ un autòmat indeterminista i volem construir una expressió regular α tal que $L(\alpha) = L(M)$. Assignem als estats de K un enter de manera creixent quedant $K = \{q_1, \ldots, q_n\}$, amb $q_1 = s$ i per a cada q_k anomenem a $k \in \mathbb{N}$ com l'**índex** de l'estat. Definim per $R(i, j, k)$, amb $i, j = 1 \cdots n$ i $k = 0 \cdots n$ el conjunt de totes les paraules de Σ^* que poden portar a M des de l'estat q_i fins a l'estat q_j passant només pels estats amb índex no superior a k (és a dir, $q_1, \ldots, q_k$). Observem que si $k = n$, aleshores tenim

$$R(i, j, n) = \{w \in \Sigma^* \mid (q_i, w) \vdash_M^* (q_j, \lambda)\}$$

Així, tenim que

$$L(M) = \bigcup_{j=1\ldots n} R(1, j, n)$$

Per tant, si veiem que els $R(i, j, k)$ són expressions regulars, ja haurem demostrat el que volíem. Provem-ho per inducció:

- $\boxed{k = 0}$:

  $$R(i, j, 0) = \{a \in \Sigma \cup \{\lambda\} \mid q_j \in \Lambda(q_i, a)\} \text{ si } i \neq j$$
  $$R(i, i, 0) = \{\lambda\} \cup \{a \in \Sigma \cup \{\lambda\} \mid q_i \in \Lambda(q_i, a)\}$$

  és a dir, l'expressió regular formada per la unió de símbols de les arestes que van des de l'estat q_i a q_j, prenent λ si no hi ha cap aresta.

- $\boxed{k - 1 \implies k}$: Suposem ara que $R(i, j, k - 1)$ és una expressió regular per a tot i, j, i anem a veure que $R(i, j, k)$ també ho és. Podem escriure

  $$R(i, j, k) = R(i, j, k - 1) \cup R(i, k, k - 1)R(k, k, k - 1)^* R(k, j, k - 1)$$

  La interpretació de la fórmula anterior és que, per passar de l'estat q_i a l'estat q_j sense passar per cap estat amb índex major que k, es pot

  - ⋆ O bé anar des de q_i a q_j sense passar per cap estat amb índex major que $k - 1$

⋆ O bé anar des de q_i a q_k, seguidament anar de q_k a q_k zero o més vegades i després anar de q_k a q_j, en els tres casos sense passar per cap estat amb índex superior a $k-1$.

Per hipòtesi d'inducció, $R(i,j,k-1)$, $R(i,k,k-1)$, $R(k,k,k-1)$ i $R(k,j,k-1)$ són regulars, de manera que la seva clausura de Kleene, concatenació i unió també ho són. Amb això obtenim que $R(i,j,k)$ és també regular.

□

Amb aquesta demostració hem vist que els autòmats finits (tant deterministes com indeterministes) són equivalents a les expressions regulars.

Corol·lari. *Per a tot llenguatge L existeix una expressió regular α tal que $L = L(\alpha)$ si i només si existeix un autòmat determinista M tal que $L(M) = L$.*

A.2 Autòmats amb pila

Els autòmats amb pila són una extensió dels autòmats finits que introdueixen una nova dimensió en la capacitat de càlcul. A diferència dels autòmats finits, aquest nou tipus d'autòmats tenen associada una pila, una estructura de dades seqüencial que permet afegir, treure o llegir elements del cim de la pila[1]. Aquesta nova capacitat ens permetrà reconèixer llenguatges que no podíem abordar amb els autòmats finits, ja que podem utilitzar la pila per recordar informació i realitzar càlculs més complexos.

Definició 29. *Un **autòmat amb pila** és una estructura $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ on:*

- *K és un conjunt finit i no buit d'estats*
- *Σ és un alfabet finit, que anomenarem **alfabet d'entrada***
- *Γ és un altre alfabet finit, que anomenarem **alfabet de la pila***
- *$q_0 \in K$ és l'estat inicial*
- *$F \subseteq K$ és el conjunt d'estats acceptadors o finals*
- *Δ és un subconjunt de $(K \times (\Sigma \cup \{\lambda\})) \times (\Gamma \cup \{\lambda\}) \times (K \times \Gamma^*)$, els elements del qual anomenarem **transicions***

En començar un còmput en l'autòmat M, estarem a l'estat inicial q_0, tindrem una paraula $w \in \Sigma^*$ i tindrem la pila buida (o, de manera equivalent, hi haurà λ al cim de la pila). En un pas de còmput es pot aplicar la transició $((p,a,b),(q,x))$ si $p \in K$ és l'estat actual, $a \in \Sigma$ és el símbol de la cinta que toca llegir i $b \in \Gamma \cup \{\lambda\}$ és al cim de la pila. Aleshores, l'autòmat passa a l'estat q i se substitueix el símbol b per x al cim de la pila.

[1]Es pot pensar literalment com una torre de dades: es pot accedir o treure únicament l'element de dalt de tot de la pila i només es pot afegir un element a dalt de tot. Així, l'últim element que es posa a la pila ha de ser el primer a sortir-ne

Definició 30. *Si $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ és un autòmat amb pila, definim una* ***configuració*** *de M com una paraula $\alpha p x \in \Gamma^* K \Sigma^*$ (per facilitar la lectura, a vegades escriurem una configuració $\alpha p x$ com a (α, p, x)). Si en un pas de còmput la configuració d'M és $\alpha p x$, l'autòmat es troba a l'estat p, α és el contingut de la pila i x és la part de la paraula d'entrada que encara no s'ha llegit.*

De manera molt similar als autòmats finits, definim també els següents termes:

- *Si $\alpha p x$ i $\beta q y$ són configuracions d'un autòmat M, diem que $\alpha p x$* ***produeix*** *$\beta q y$* ***en un pas de còmput*** *si es pot passar de la primera configuració a la segona aplicant una transició de Δ. Seguint la nomenclatura d'autòmats finits, ho representarem amb $\alpha p x \vdash_M \beta q y$.*
- *De manera més general, direm que $\alpha p x$* ***produeix*** *$\beta q y$ si es pot passar de la primera configuració a la segona mitjançant un nombre finit de transicions de Δ. Ho representarem per $\alpha p x \vdash_M^* \beta q y$.*
- *Una paraula $x \in \Sigma^*$* ***és reconeguda o acceptada*** *per l'autòmat M si $\exists q \in F$ tal que $\lambda p x \vdash_M^* \lambda q \lambda$. És a dir, una paraula es reconeix només si en llegir tota la paraula, s'arriba a un estat acceptador amb la pila buida.*
- *Definim el* ***llenguatge associat a*** *M com*

$$L(M) = \{x \in \Sigma^* \mid x \text{ és reconeguda per } M\}$$

Tal com en els autòmats finits, amb els autòmats amb pila també podem fer la distinció entre autòmats amb pila deterministes i indeterministes. Per definir-los correctament, però, necessitem el concepte de transicions compatibles:

Definició 31. *Sigui $M = (K, \Sigma, \Gamma, \Delta, q_0, F)$ un autòmat amb pila. Diem que dues transicions $((p_1, a_1, b_1), (q_1, \alpha_1))$ i $((p_2, a_2, b_2), (q_2, \alpha_2))$ de Δ són* ***compatibles*** *si es compleixen les següents condicions:*

1. $p_1 = p_2$
2. $a_1 = a_2$ *o bé* $a_1 = \lambda$ *o bé* $a_2 = \lambda$
3. $b_1 = b_2$ *o bé* $b_1 = \lambda$ *o bé* $b_2 = \lambda$

En altres paraules, diem que dues transicions són compatibles si es poden aplicar en un mateix pas de còmput.

Definició 32. *Direm que un autòmat amb pila és* ***determinista*** *si no té dues transicions diferents que siguin compatibles.*

A.2.1 Equivalència entre els autòmats amb pila i les gramàtiques incontextuals

Anem a veure, finalment, l'equivalència entre els autòmats amb pila i les gramàtiques incontextuals:

Teorema 10. *Per a qualsevol llenguatge L, existeix un autòmat amb pila M tal que $L(M) = L$ si i només si existeix una gramàtica incontextual G tal que $L(G) = L$*

Demostració. Demostrarem primer que tot llenguatge incontextual és reconegut per un autòmat amb pila. Per fer-ho, prenem una gramàtica incontextual $G = (V, \Sigma, P, S)$ i construirem un autòmat amb pila M tal que $L(G) = L(M)$.

Sigui $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, on Δ conté les següents transicions:

1. $((p, \lambda, \lambda), (q, S))$
2. $((q, \lambda, A), (q, x))$ per a cada producció $A \longrightarrow x$ de P
3. $((q, a, a), (q, \lambda))$ per a cada $a \in \Sigma$.

Així, M utilitza l'alfabet de variables V de la gramàtica incontextual G com a alfabet de la pila i l'alfabet de terminals de G com a alfabet d'entrada d'M. Observem també que a l'autòmat M només hi ha dos estats: p, que és l'estat inicial, i q l'únic estat acceptador, al que forçosament haurem de passar en el primer pas de còmput (ja que és l'única transició possible amb la pila buida). A cada pas posterior hi ha dues opcions: o bé hi ha una variable al cim de la pila a la qual s'aplica una producció de P, o bé treu del cim de la pila un terminal, suposant que coincideixi amb el símbol que toca llegir de la cinta. Ara cal provar, doncs, que $L(M) = L(G)$, que ho farem a partir de la següent proposició:

Sigui $w \in \Sigma^*$ i $\alpha \in (V \setminus \Sigma)V^* \cup \{\lambda\}$. Aleshores $S \overset{L}{\Rightarrow}^* w\alpha$ si i només si $(q, w, S) \vdash_M^* (q, \lambda, \alpha)$.

Provant aquesta proposició haurem demostrat que qualsevol llenguatge incontextual és acceptat per un autòmat amb pila, ja que, prenent $\alpha = \lambda$, tindrem que $S \overset{L}{\Rightarrow}^* w \iff (q, w, S) \vdash_M^* (q, \lambda, \lambda)$. Utilitzant la primera transició de Δ descrita a dalt, tindrem que $(p, w, \lambda) \vdash_M (q, w, S) \vdash_M^* (q, \lambda, \lambda)$, és a dir que $w \in L(G) \iff w \in L(M)$.

Així doncs, procedim a demostrar les dues implicacions per inducció sobre el nombre de derivacions:

- $\boxed{\Longleftarrow}$: Suposem que $S \overset{L}{\Rightarrow}^* w\alpha$ per un cert nombre k de derivacions i volem veure que $(q, w, S) \vdash_M^* (q, \lambda, \alpha)$
  - ⋆ $\boxed{k = 0}$: Si la derivació té longitud $k = 0$ necessàriament $w = \lambda$ i $\alpha = S$, i aleshores $(q, w, S) \vdash_M^* (q, \lambda, \alpha)$
  - ⋆ $\boxed{k \implies k + 1}$: Suposem que si $S \overset{L}{\Rightarrow}^* w\alpha$ amb una derivació de longitud $n \leq k$ aleshores $(q, w, S) \vdash_M^* (q, \lambda, \alpha)$. Sigui ara una derivació per l'esquerra amb longitud $k + 1$ de la següent manera:

    $$S = u_0 \overset{L}{\Rightarrow} u_1 \overset{L}{\Rightarrow} \ldots \overset{L}{\Rightarrow} u_k \overset{L}{\Rightarrow} u_{k+1} = w\alpha$$

    Suposem que A és la primera variable de u_k. Aleshores, com que $u_k \overset{L}{\Rightarrow} u_{k+1}$, tenim que $u_k = xAy$ i $u_{k+1} = xzy$, per a alguns $x \in \Sigma^*,\ y, z \in V^*$

i que $A \longrightarrow z$ és una producció de P. Aplicant la hipòtesi d'inducció sobre $u_k = xAy$ i prenent $\alpha = Ay$ obtenim

$$(q, x, S) \vdash_M^* (q, \lambda, Ay)$$

Com que $A \longrightarrow z$, aplicant una transició del tipus (2) tenim que $(q, \lambda, Ay) \vdash_M (q, \lambda, zy)$. Observem ara que $u_{k+1} = w\alpha$ però també $u_{k+1} = xzy$, de manera que existeix una paraula $\beta \in \Sigma^*$ tal que $w = x\beta$ i per tant $\beta\alpha = zy$. Aleshores

$$(q, \beta, zy) = (q, \beta, \beta\alpha) \vdash_M^* (q, \lambda, \alpha)$$

realitzant una seqüència de $|\beta|$ transicions de tipus (3). Amb tot això obtenim

$$(q, w, S) = (q, x\beta, S) \vdash_M^* (q, \beta, Ay) \vdash_M (q, \beta, zy) = (q, \beta, \beta\alpha) \vdash_M^* (q, \lambda, \alpha)$$

com volíem veure.

- $\boxed{\Longrightarrow}$: Suposem ara que $(q, w, S) \vdash_M^* (q, \lambda, \alpha)$ i volem veure que $S \overset{L}{\Rightarrow}^* w\alpha$. Tornarem a fer la demostració per inducció, ara sobre el nombre de transicions de tipus (2) en el còmput:

  - $\boxed{k = 0}$: Observem, abans de res que immediatament després de fer la transició (1) inicial, com que la pila és buida, només es pot fer una transició de tipus (2). Si no hi ha cap transició de tipus (2), aleshores $(q, w, S) \vdash_M^* (q, \lambda, \alpha) \implies w = \lambda \wedge \alpha = S$, i per tant $S = w\alpha$.

  - $\boxed{k \implies k+1}$: Suposem que si $(q, w, S) \vdash_M^* (q, \lambda, \alpha)$ amb $n \leq k$ transicions de tipus (2) (possiblement amb transicions de tipus (3) abans i després) aleshores $S \overset{L}{\Rightarrow}^* w\alpha$. Suposem ara que $(q, w, S) \vdash_M^* (q, \lambda, \alpha)$ amb $k+1$ transicions de tipus (2) i separem les transicions a la penúltima transició de tipus (2):

    $$(q, w, S) \vdash_M^* (q, \beta, Ay) \vdash_M (q, \beta, zy) \vdash_M^* (q, \lambda, \alpha)$$

    amb $w = x\beta$ per a algun $x, \beta \in \Sigma^*$ i $A \longrightarrow z$ és una producció de P. Com que $(q, x\beta, S) \vdash_M^* (q, \beta, Ay)$ deduïm que $(q, x, S) \vdash_M^* (q, \lambda, Ay)$. Ara, per la hipòtesi d'inducció, tenim que $S \overset{L}{\Rightarrow}^* xAy$ i, aplicant la producció $A \longrightarrow z$, $S \overset{L}{\Rightarrow}^* xzy$. Com que també $(q, \beta, zy) \vdash_M^* (q, \lambda, \alpha)$ utilitzant només transicions de tipus (3), obtenim que $\beta\alpha = zy$ i per tant $S \overset{L}{\Rightarrow}^* xzy = x\beta\alpha = w\alpha$.

□

A.3 Màquines de Turing indeterministes

Una altra possible modificació sobre les màquines de Turing prové d'agafar el concepte d'indeterminisme desenvolupat en l'apartat d'autòmats indeterministes i adaptar-lo a màquines de Turing.

Definició 33. *Una* ***màquina de Turing indeterminista*** *és una estructura* $M = (K, \Sigma, \Gamma, \delta, q_o, q_F)$ *on:*

- K, Σ, Γ, q_0 *i* q_F *es defineixen de la mateixa manera que en la màquina de Turing tradicional*
- $\delta\colon ((K \setminus \{q_F\}) \times \Gamma) \nrightarrow \mathcal{P}((K \times \Gamma \times \{L, S, R\}))$. *És a dir, que en lloc de ser un element de* $(K \times \Gamma \times \{L, S, R\})$ *tal com ho era abans,* $\delta(p, a)$ *n'és un subconjunt finit.*

En aquestes màquines els còmputs no estan unívocament determinats, sinó que per un estat i un símbol d'entrada hi pot haver múltiples transicions possibles, generant un arbre de còmputs.

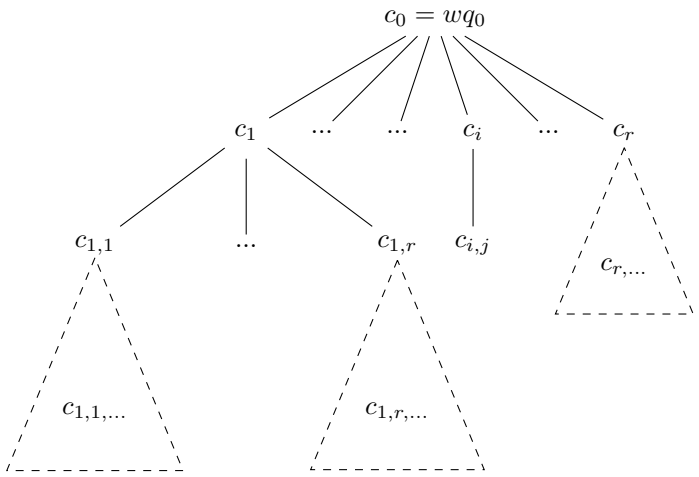


Figura A.1: Esquema d'un arbre de còmputs

Si $r = max_{q,x}|\delta(q, x)|$, aleshores es pot pensar l'arbre de còmputs d'M_I com un arbre r-ari (encara que no necessàriament n'existeixin totes les branques) on $c_{i_1,i_2,...i_n}$ és una possible configuració de M_I després d'n còmputs en la qual s'ha triat la branca i_k (i.e. s'ha seleccionat l'i_k-èssim element d'entre els possibles donats per la funció de transició) en el pas k.

En general podem definir tota la resta de conceptes associats a una màquina de Turing indeterminista de la mateixa manera que ho hem fet a les deterministes, però cal redefinir-ne les següents:

- Diem que una màquina de Turing indeterminista M **accepta o reconeix** una entrada $x \in \Sigma^+$ si en l'arbre de còmputs associats a x hi ha alguna fulla que contingui l'estat q_F.
- Diem que M és de parada segura si per a tota entrada $x \in \Sigma^*$ totes les branques de l'arbre de còmputs associat a x té totes les branques finites.

Altre cop, cal veure si aquesta nova màquina és equivalent a l'anterior. Tal com ha passat amb les ampliacions anteriors, veurem que efectivament els models són

equivalents:

Teorema 11. *Per a tota màquina de Turing indeterminista M_I existeix una màquina de Turing determinista M_D tal que $L(M_I) = L(M_D)$.*

Idea de la demostració. Donada una màquina de Turing indeterminista M_I volem generar una màquina de Turing determinista M_D tal que $L(M_I) = L(M_D)$. Donada una certa entrada $x \in L(M_I)$, sabem que M_I generarà un arbre de còmputs on, com a mínim una branca, té una fulla que conté q_F. La idea és que M_D busqui, en l'arbre de còmputs, una configuració que accepti x. Per fer-ho el que voldríem és que M_D fes una cerca per amplada[2]. Observem que l'ordre en què aquest algorisme visita els nodes correspon a l'ordenació lexicogràfica dels seus subíndexs.

Es pot fer aquesta simulació en una màquina determinista de 3 cintes (que ja hem vist que és equivalent a una màquina de Turing convencional) on:

- La primera cinta, que anomenarem d'*entrada*, sempre contindrà la paraula d'entrada.
- La segona cinta serà la que s'utilitzarà per simular una part de la seqüència de còmputs d'M_I d'una branca de l'arbre. L'anomenarem cinta de *simulació*.
- La tercera cinta guardarà la seqüència de còmputs que representa la branca de l'arbre. L'anomenarem cinta de *seqüència*.

La màquina operarà de la següent manera:

1. Inicialment la cinta d'entrada conté la paraula d'entrada i les cintes de simulació i seqüència són buides.
2. Es copia el contingut de la cinta d'entrada a la cinta de simulació.
3. Se simula M_I utilitzant només la cinta de simulació de la següent manera:
   (a) En el següent pas de M_I, si l'estat és q, a la cinta d'entrada es llegeix el símbol x, i a la cinta de seqüència es llegeix el símbol i, aleshores se simula el còmput mitjançant l'i-èssim element de $\delta(q, x)$. En cas que no existeixi tal element, es va al pas (4).
   (b) Es fa el canvi d'estat, escriptura i moviment de punter de la cinta de simulació i a continuació es mou el capçal de la cinta de seqüència cap a la dreta. En cas que el punter de la cinta de seqüència llegeixi λ, es va al pas (4).
   (c) Si M_I para en l'estat acceptador, aleshores M_D també farà una transició cap a l'estat acceptador i pararà, en cas contrari, es va al pas (3.a).
4. Si el següent node de l'arbre segons l'algorisme BFS és $c_{i_1,i_2,\ldots,i_k}$, s'escriu la seqüència $i_1, i_2, \ldots, i_k$ a l'inici de la cinta de seqüència. S'esborra tot el contingut de la cinta de simulació i es va al pas (2).

[2]algorisme de cerca per amplada sobre grafs, comunament conegut com a BFS (Breadth-first search)

Bibliografia

[1] John L Britton. "The word problem". A: *Annals of Mathematics* (1963), pàg. 16 - 32.

[2] Martin Davis. *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions*. Dover Ed. Dover Publications, 2004. ISBN: 0486432289,9780486432281.

[3] Weyuker E. Davis M. Sigal R. *Computability, complexity, and languages: Fundamentals of theoretical computer science*. 2ed. Computer Science and Scientific Computing. AP, 1994. ISBN: 0122063821; 9780122063824.

[4] Adrian Francalanza. *Variations on Turing Machines*. `http://staff.um.edu.mt/afra1/teaching/coco4.pdf`. Consultat: 2023-9-21. 2009.

[5] Vesa Halava i Tero Harju. "Mortality in matrix semigroups". A: *The American Mathematical Monthly* 108.7 (2001), pàg. 649 - 653.

[6] Harry Lewis i Christos H. Papadimitriou. *Elements of the Theory of Computation*. First Edition. Upper Saddle River, NJ: Prentice-Hall, 1998. ISBN: 0132624788.

[7] Michael S Paterson. "Unsolvability in 3× 3 matrices". A: *Studies in Applied Mathematics* 49.1 (1970), pàg. 105 - 107.

[8] Bjorn Poonen. "Undecidable problems: a sampler". A: *Interpreting Gödel: Critical Essays* (2014), pàg. 211 - 241.

[9] Friedrich Otto Ronald V. Book. *String-Rewriting Systems*. 1; online version 2011. Texts and Monographs in Computer Science. Springer, 1993. ISBN: 978-1-4613-9773-1.

[10] Joseph J Rotman. "The Word Problem". A: *An Introduction to the Theory of Groups* (1995), pàg. 418 - 470.

[11] Michael Sipser. *Introduction to the Theory of Computation*. Third Edition. Boston, MA: Course Technology, 2013. ISBN: 113318779X.

[12] Thomas A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. 2ed. Addison Wesley, 1996. ISBN: 0201821362; 9780201821369.

[13] Mahesh Viswanathan. *Variants of Turing Machines*. `https://courses.engr.illinois.edu/cs373/sp2013/Lectures/lec20.pdf`. Consultat: 2023-9-22. 2013.

[14] Michael M. Wolf. *Lecture on undecidability*. `https://problem24.wordpress.com/lecture-on-undecidability/`. Consultat: 2023-11-20. 2011.