UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

TESIS DE CHURCH - TURING

Autor: Pau Vendrell Titó

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

Barcelona, 18 de juny de 2021

Abstract

Church-Turing thesis states that the intuitive notion of an algorithm corresponds to the notion of a Turing machine. In this work we will show the main arguments that support this claim: the implementation of the basic algorithmic structures by means of Turing machines, and specially the equivalence between the notion of a Turing machine and other mathematical formulations of the notion of an algorithm.

Resum

La tesis de Church-Turing afirma que la noció intuïtiva d'algorisme correspon amb el concepte de màquina de Turing. En aquest treball donarem raonaments que avalen la certesa d'aquest enunciat tals com la implementació per part de les màquines de Turing, de les estructures algorísmiques bàsiques i l'equivalència entre aquest model i altres nocions associades al concepte d'algorisme.

2020 Mathematics Subject Classification. 68V20, 68V15

Agraïments

Vull agrair al meu tutor Juan Carlos Martinez Alonso la paciència, la constància, l'ajuda, el temps dedicat i els coneixements que m'ha proporcionat per poder elaborar aquest treball.
També vull agrair als meus pares i al meu germà el seu suport, els ànims i la confiança que han dipositat en mi durant aquests 22 anys.

Índex

1 Introducció

El projecte

La tesis de Church-Turing afirma que el model computacional de màquina de Turing és equivalent a la noció intuïtiva d'algorisme. Aquesta tesis s'ha utilitzat per poder demostrar que certs problemes computacionals són irresolubles, en el sentit que no existeix cap algorisme que els resolgui. Un dels problemes computacionals irresolubles més cèlebres és el Desè Problema de Hilbert, el qual pregunta si existeix un algorisme que, donada una equació diofàntica, determina si aquesta equació té solució en els nombres enters. Aquest problema va ser proposat per David Hilbert a l'any 1900, i va ser resolt en sentit negatiu a l'any 1976 pels matemàtics Yuri Matiyasévich, Martin Davis, Julia Robinson i Hilary Putnam en l'anomenat Teorema MRDP, en el que utilitzant el concepte de màquina de Turing i la tesis de Church-Turing, van demostrar que tal algorisme no existeix.

En aquest treball veurem com el concepte de màquina de Turing és un model robust, és a dir que aparents millores d'aquest model no comporten cap modificació de potència de càlcul. Aquesta noció ens permet computar les operacions aritmètiques i ens permet implementar les estructures algorísmiques bàsiques.

Igualment es veuran altres formulacions matemàtiques associades al concepte intuïtiu d'algorisme i demostrarem l'equivalència entre aquestes nocions i el model de màquina de Turing.

Aquestes consideracions, que seran vistes al llarg d'aquest treball, són la justificació de l'us de la màquina de Turing com a model formal d'algorisme. La tesis de Church-Turing no és objecte de demostració, ja que el concepte d'algorisme no és una noció matemàtica. Tot i això, en l'actualitat s'assumeix com a certa, pel fet de que tot intent de formalitzar el concepte d'algorisme resulta equivalent al model de màquina de Turing, tal i com es veurà en posteriors capítols.

Estructura de la Memòria

Primerament definirem el concepte de màquina de Turing i demostrarem que és un model robust. Seguidament, veurem com aquesta noció permet la implementació de les estructures algorísmiques bàsiques. A continuació passarem a definir la noció de gramàtica i veurem com tota funció computable per una màquina de Turing és gramaticalment computable. Tot seguit, definirem el concepte de funció μ-recursiva i demostrarem com tota funció gramaticalment computable és també μ-recursiva. I finalment, definirem la noció de programa de registres i demostrarem que tota funció μ-recursiva és una funció computable per un programa de registres i com aquests, al mateix temps, també són computables per màquines de Turing.

2 Màquines de Turing

2.1 Conceptes Previs

Avans de passa a definir l'estructura de màquina de Turing i de les seves propietats, és necessari definir prèviament uns conceptes per tal de poder entendre les definicions que es donaran en els apartats següents.

Definició 2.1. *Un alfabet és un conjunt finit i no buit de símbols.*

Definició 2.2. *Una paraula sobre un alfabet Σ és una seqüència finita de símbols de Σ.*

Denotarem per λ a la paraula buida, és a dir, a la paraula que no té cap símbol.

Definirem la longitud d'una paraula x com el número de símbols que la componen, contant possibles repeticions de símbols. Denotarem la longitud de la paraula x per $|x|$.

Definició 2.3. *Donat un alfabet Σ, direm llenguatge a un conjunt de paraules sobre aquest alfabet.*

Definició 2.4. *Si Σ és un alfabet, denotem per Σ^* al conjunt de totes les paraules sobre Σ.*

En Σ^* es considera l'anomenat *ordre lexicogràfic*, que consisteix en ordenar les paraules de Σ^* per ordre de longitud creixent on les paraules de igual longitud s'ordenaran seguint l'ordre d'un diccionari.
D'aquesta manera, si $\Sigma = \{a_1, ..., a_n\}$, on $a_1 < ... < a_n$, l'ordre lexicogràfic de Σ^* serà:

$\lambda, a_1, ..., a_n, a_1a_1, a_1a_2, ..., a_1a_n, ..., a_na_1, ..., a_na_n, a_1a_1a_1, ...$

Donat un alfabet Σ, tindrem en Σ^* una operació que se l'anomena la *concatenació* que consisteix en la juxtaposició d'una paraula x amb una paraula y, que és representa per $x \cdot y$ o, més abreviadament, per xy. Per tant, la paraula $x \cdot y$ consisteix en situar primerament els símbols que conformen la paraula x i, a continuació, els que conformen la paraula y.

2.2 Màquines de Turing Deterministes

En aquesta secció, definirem el concepte de màquina de Turing determinista, el concepte de còmput d'una màquina de Turing i el concepte de funció computable per una màquina de Turing.

Una màquina de Turing està constituïda per una cinta semiinfinita, dividida en cel·les i la màquina accedeix a la informació emmagatzemada a la cinta a través d'un punter el qual pot llegir o bé escriure en la casella a la que apunta. A més, aquest punter es pot desplaçar cap a la dreta o a l'esquerra de la cinta, però únicament una cel·la per cada moviment.

Inicialment, la cel·la situada més a l'esquerra de la cinta conté un caràcter especial \$, i a continuació, en les caselles adjacents es troba la paraula d'entrada. Cada símbol que conforma aquest mot es situarà en una cel·la diferent i en l'ordre en que apareixen en la paraula. La resta de cel·les de la cinta contindran el caracter blanc, que representarem amb el símbol $*$, i el punter senyalarà al primer caràcter blanc.

En un pas de còmput de la màquina de Turing, el punter pot escriure un caràcter a la cel·la a la que apunta, podent-se, posteriorment, moure cap a la casella situada a la seva

dreta o bé la situada cap a la seva esquerra, exceptuant en aquest últim cas que estigui apuntant a la cel·la inicial.

Denotarem per:
L = moviment del punter cap a l'esquerra de la cinta
S = no hi ha moviment del punter
R = moviment del punter cap a la dreta de la cinta

Tot seguit, definirem el concepte estàndar de màquina de Turing, més conegut com a màquina de Turing determinista.

Definició 2.5. *Una màquina de Turning determinista és una estructura de la forma* $(K, \Sigma, \Gamma, \delta, q_0, q_F)$ *on*

K *és un conjunt finit i no buit d'estats*

Σ *és un alfabet, que anomenarem alfabet d'entrada*

Γ *és un alfabet tal que* $\Sigma \cup \{\$, *\} \subseteq \Gamma$ *i* $\$, * \notin \Sigma$*, i l'anomenarem alfabet de la cinta*

$q_0 \in K$ *és l'estat inicial*

$q_F \in K$ *és l'únic estat final o acceptador*

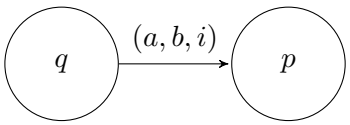
$\delta : (K \setminus \{q_F\}) \times \Gamma \to K \times \Gamma \times \{L, R, S\}$ *és la funció de transició tal que, per a tot* $q \in K \setminus \{q_F\}$*:*

a) Si $\delta(q, \$) = (p, b, j)$*, aleshores* $b = \$$ *i* $j \neq L$

b) Si $a \neq \$$ *i* $\delta(q, a) = (p, b, j)$*, aleshores* $b \neq \$$

Si $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ és una màquina de Turing determinista, inicialmet es troba en un estat inicial q_0 amb entrada una paraula $x \in \Sigma^*$ escrita al començament de la cinta i a continuació del símbol \$. El punter senyala al primer símbol blanc $*$ que es troba després de l'últim caràcter de la paraula x. Els còmputs de la màquina es durant a terme a partir de l'aplicació δ, la qual s'aplica al parell (q, a) on q representa l'estat actual de la màquina i a és el símbol al que senyala el punter. Aquesta funció dóna lloc a una tríada (p, b, i) on p denota el nou estat de la màquina, b serà el símbol que reemplaça al símbol a en la cinta i $i \in \{R, S, L\}$ de forma que:
Si i = R, la màquina mou el punter a la següent cel·la cap a la dreta.
Si i = L, la màquina mou el punter a la següent cel·la cap a l'esquerra.
Si i = S, la màquina no mou el punter.

Les màquines de Turing deterministes és poden representar de forma gràfica tal i com es mostra tot seguit. Si $\delta(q, a) = (p, b, i)$, representarem aquesta transició en el gràfic de la màquina posant:



I si $q = q_F$ (és a dir, q és l'estat acceptador) el representarem posant:

A continuació, passarem a explicar la noció de configuració en una màquina de Turing que ens servirà per definir el concepte de còmput.

Definició 2.6. *Una configuració és una paraula de la forma $\alpha qa\beta$ on $q \in K$, $a \in \Gamma$, $\alpha, \beta \in \Gamma^*$ i β no acaba en $*$.*

Donada una configuració $\alpha qa\beta$ diem que és de parada si $q = q_F$

Explicat de manera informal, una configuració és una descripció de la situació en què és troba la màquina de Turing en un instant donat i consta de la informació relativa a l'estat (representat per la lletra q en la definició), la posició del punter (el punter senyala a a si aquesta és diferent de λ o bé a un blanc si $a = \lambda$) i del contingut de la cinta (que està format per la paraula $\alpha a\beta$).

Definició 2.7. *Donada una màquina de Turnig determinista $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ i dues configuracions, $\alpha qa\beta$ i $\alpha' q' a' \beta'$, diem que $\alpha qa\beta$ produeix $\alpha' q' a' \beta'$ en un pas de M, i ho expressem en la forma*

$$\alpha qa\beta \vdash_M \alpha' q' a' \beta'$$

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

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

$$\alpha qa\beta \vdash_M^t \alpha' q' a' \beta'$$

si a partir de $\alpha qa\beta$, arribem a la configuració $\alpha' q' a' \beta'$ aplicant t vegades la funció de transició.

Diem que $\alpha qa\beta$ produeix $\alpha' q' a' \beta'$ en un nombre finit de passos de M, i ho expressem en la forma

$$\alpha qa\beta \vdash_M^* \alpha' q' a' \beta'$$

si existeix $t \geq 0$ tal que $\alpha qa\beta \vdash_M^t \alpha' q' a' \beta'$.

Definició 2.8. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turning determinista. Sigui $x \in \Sigma^*$.*

Direm que M para sobre x si el còmput de M sobre l'entrada x acaba. Llavors escriurem $M(x) \downarrow$

Si el còmput de M sobre l'entrada x no acaba, escriurem $M(x) \uparrow$

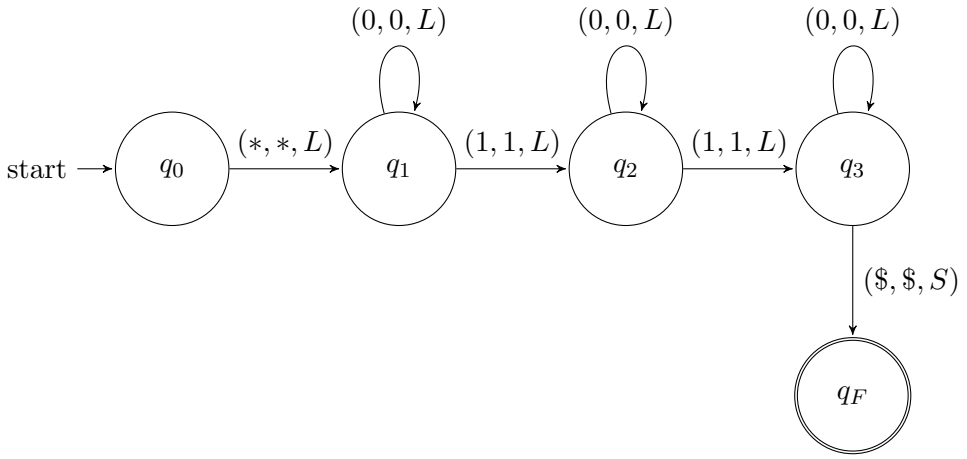
Definició 2.9. *Sigui M una màquina de Turing determinista amb alfabet d'entrada Σ. Direm que una paraula $x \in \Sigma^*$ és acceptada (o reconeguda) per M, si M amb entrada x s'atura i ho fa en l'estat acceptador.*

El llenguatge acceptat o reconegut per una màquina de Turing determinista M, representat per L(M), està format pel conjunt de paraules acceptades per M.

Donat un llenguatge L, direm que L és acceptable si existeix una màquina de Turnig determinista M tal que L = L(M).

Exemple 2.10. En aquest exemple descriurem una màquina de Turing M que accepta el llenguatge $A = \{0^n 10^m 10^k : n, m, k \geq 0\}$, és a dir el llenguatge de les paraules en l'alfabet $\{0, 1\}$ i que tenen exactament dos aparicions del símbol 1.

$M = (K, \{0, 1\}, \{0, 1, *, \$\}, \delta, q_0, q_F)$ ve descrita pel graf següent:



Definició 2.11. *Siguin Σ_0 i Σ_1 dos alfabets que no contenen els símbols blanc $*$ i \$.*

a) Sigui $f : \Sigma_0^* \to \Sigma_1^*$. Es diu que una màquina de Turing determinista $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ computa f si $\Sigma_0 \cup \Sigma_1 \subseteq \Sigma$ i per a qualsevol paraula $x \in \Sigma_0^*$, si $f(x) = y$ aleshores

$$\$xq_0* \vdash_M^* \$yq_F*$$

Si existeix una màquina de Turing determinista que compleixi amb la definició anterior, llavors f es diu que és una funció T-computable.

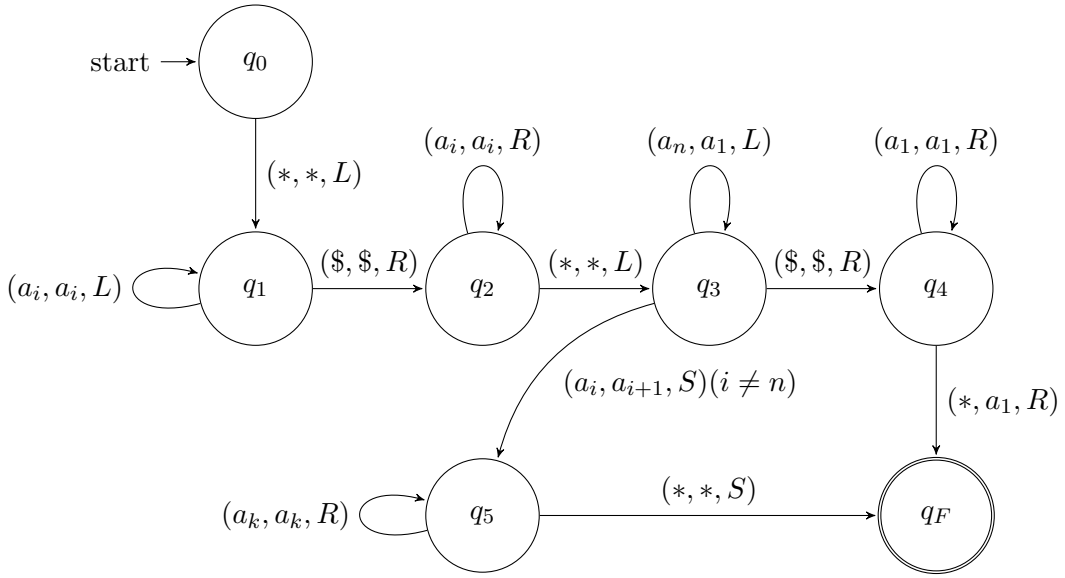
b) Sigui $n \geq 2$. Sigui $f : (\Sigma_0^*)^n \to \Sigma_1^*$. Es diu que una màquina de Turing determinista $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ computa f si $\Sigma_0 \cup \Sigma_1 \subseteq \Sigma$ i per a qualsevols paraules $x_1, ..., x_n \in \Sigma_0^*$, si $f(x_1, ..., x_n) = y$ aleshores

$$\$x_1 * ... * x_n q_0* \vdash_M^* \$yq_F*$$

Anàlogament a l'apartat anterior, si existeix una màquina de Turing determinista que compleix amb aquesta definició, llavors f es diu que és una funció T-computable.

Exemple 2.12. Donat un alfabet $\Sigma = \{a_1, ..., a_n\}$ tal que $a_1 < ... < a_n$, definim una màquina de Turing M que computa la funció que donada una paraula $x \in \Sigma$ li assigni la següent paraula segons l'ordre lexicogràfic:

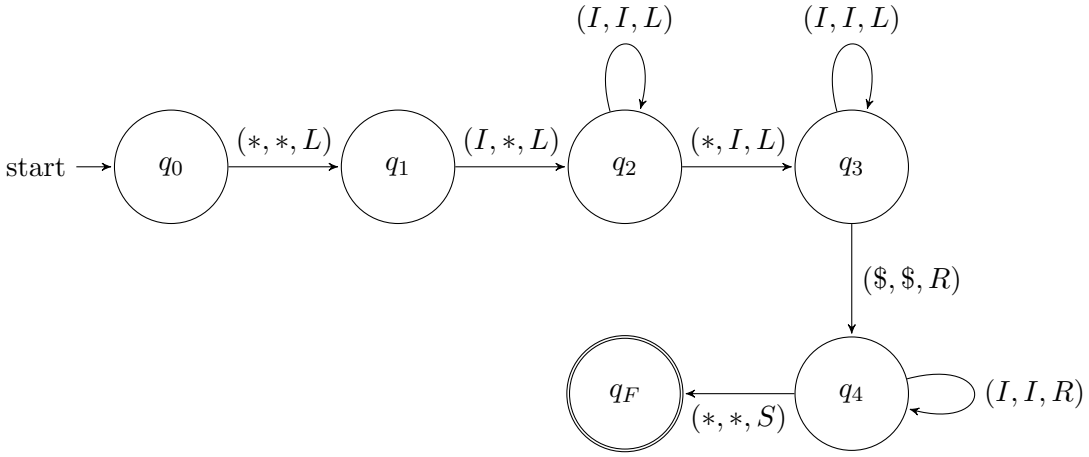
$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_F\}, \Sigma, \Sigma \cup \{*, \$\}, \delta, q_0, q_F)$ ve descrita pel graf següent:



Exemple 2.13. Donat l'alfabet unari $\Sigma = \{I\}$, anem a definir mitjançant un graf una màquina de Turing N la qual computi la funció suma de dos nombres. Així, si $n, m \in \mathbb{N}$, a l'inici del còmput la configuració de la cinta és

$$\$I^n * I^m q_0 *$$

és a dir, inicialment el punter senyala al primer blanc que es troba a continuació del nombre m. Aleshores N ve donada pel graf següent:



Definició 2.14. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing determinista. Direm que M és una màquina de parada segura si, per a tota paraula $x \in \Sigma^*$, $M(x) \downarrow$.*

Direm llavors que un llengutage L és recursiu (o decidible), si existeix una màquina de Turing determinista M de parada segura tal que $L = L(M)$. Llavors direm que M decideix el llenguatge L.

Definició 2.15. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing determinista. Direm que M és una màquina condicional si compleix les condicions següents:*

a) M és de parada segura

b) M té dos estats diferenciats q_s i q_n tals que $q_s \neq q_n$ i $q_s = q_F$, de manera que, per a tota paraula $x \in \Sigma^*$, es té que M sobre l'entrada x para o bé a q_s o bé a q_n.

2.3 Extensions del model estàndar de màquina de Turing

En aquest apartat veurem dues extencions del model de màquina de Turing determinista, els quals seran la màquina de Turing determinista multicinta i la màquina de Turing indeterminista i també demostrarem l'equivalència entre aquests dos conceptes i el model estàndar. D'aquesta manera, podrem dir que la noció de màquina de Turing és un model sòlid.

Una màquina de Turing determinista multicinta és una màquina de Turing que disposa d'un nombre finit de cintes semiinfinites les quals disposen d'un punter amb moviment independent per a cada una. La primera cinta és la cinta d'entrada, és a dir, on s'emmagatzema la paraula d'entrada i, les cintes restants, són utilitzades per la màquina com a auxiliars. Inicialment, cada cinta tindrà com a caràcter inicial el símbol \$, però mentre que en la primera cinta a continuació d'aquest caràcter hi trobarem la paraula d'entrada, en les altres cintes trobarem en les seves cel·les restants el caràcter blanc $*$. En l'inici del càlcul, el punter de la primera cinta senyala el primer caràcter blanc que es situa a continuació de l'últim símbol de la paraula d'entrada i, en les altres cintes, el punter senyala la segona cel·la.
En un pas de còmput la màquina realitzarà una de les tres accions següents:

1) Un possible canvi d'estat

2) Una possible escriptura en les cel·les senyalades pels punters

3) Un possible moviment dels punters

Seguidament, definirem de manera formal el concepte de màquina de Turing determinista multicinta.

Definició 2.16. *Una màquina de Turing determinista amb k cintes, $k \geq 1$, és una estructura de la forma $(K, \Sigma, \Gamma, \delta, q_0, q_F)$ on*

K és un conjunt finit i no buit d'estats

Σ és un alfabet, que anomenarem alfabet d'entrada

*Γ és un alfabet tal que $\Sigma \cup \{\$, *\} \subseteq \Gamma$ i $\$, * \notin \Sigma$, i l'anomenarem alfabet de la cinta*

$q_0 \in K$ és l'estat inicial

$q_F \in K$ és l'únic estat final o acceptador

$\delta : (K \setminus \{q_F\}) \times \Gamma^k \to K \times (\Gamma \times \{L, R, S\})^k$ és la funció de transició on, si $\delta(q, (a_1, ..., a_k))$ $= (p, (b_1, ..., b_k), (j_1, ..., j_k))$ i $a_i = \$$ per algun $1 \leq i \leq k$, aleshores $b_i = \$$ i $j_i \neq L$. I si $a_i \neq \$$ per algun $1 \leq i \leq k$, aleshores $b_i \neq \$$.

A continuació donarem l'extenció de la definició de funció T-computable per a màquines de Turing multicintes.

Definició 2.17. *Siguin Σ_0 i Σ_1 dos alfabets que no contenen els símbols blanc $*$ i \$. Sigui $n \geq 1$. Sigui $f : (\Sigma_0^*)^n \to \Sigma_1^*$. Es diu que una màquina de Turing de n cintes $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ computa f si $\Sigma_0 \cup \Sigma_1 \subseteq \Sigma$ i per a qualsevols paraules $u_1, ..., u_n \in \Sigma_0^*$, si $f(u_1, ..., u_n) = v$, aleshores els continguts inicials de les cintes són els següents:*

$$\begin{array}{ll} \$u_1 * ... * u_n * *... & ...(n-1 vegades) \\ \$ * *... & \$ * *... \end{array}$$

on el punter de la primera cinta senyala al primer símbol blanc que es troba a continuació

de la paraula u_n i en la resta de cintes els punters senyalen al símbol $*$ que va a continuació del símbol \$. I els continguts finals de les cintes són els següents:

$$\begin{array}{ll} \$v**... & ...(n-1 vegades) \\ \$z_2**... & \$z_n**... \end{array}$$

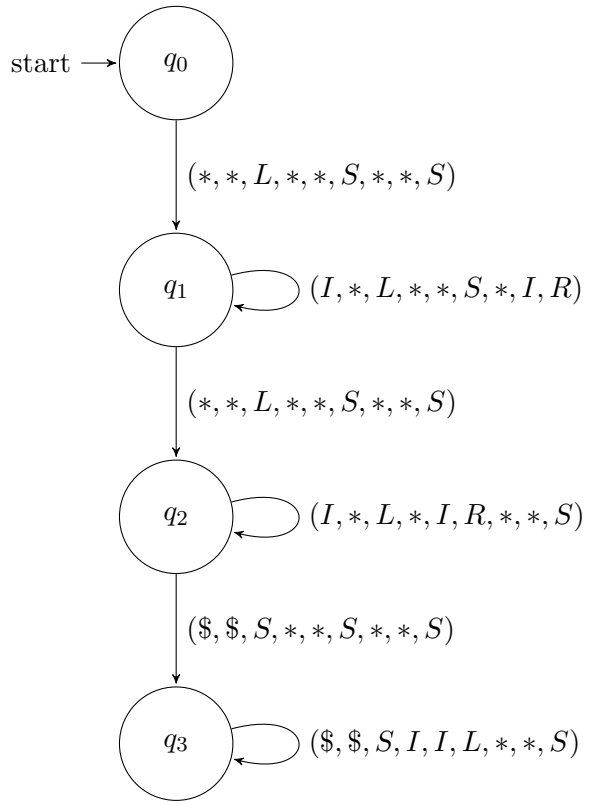
on $z_2, ..., z_n \in \Sigma_1^*$ són paraules qualsevol que resulten del càlcul i el punter de la primera cinta senyala al primer símbol blanc que es troba a continuació de la paraula v i en la resta de cintes no es rellevant la cel·la a on apuntin els punters.

Tot seguit anem a posar un exemple el qual il·lustra com una màquina de Turing de tres cintes es capaç de calcular el producte entre dos nombres.

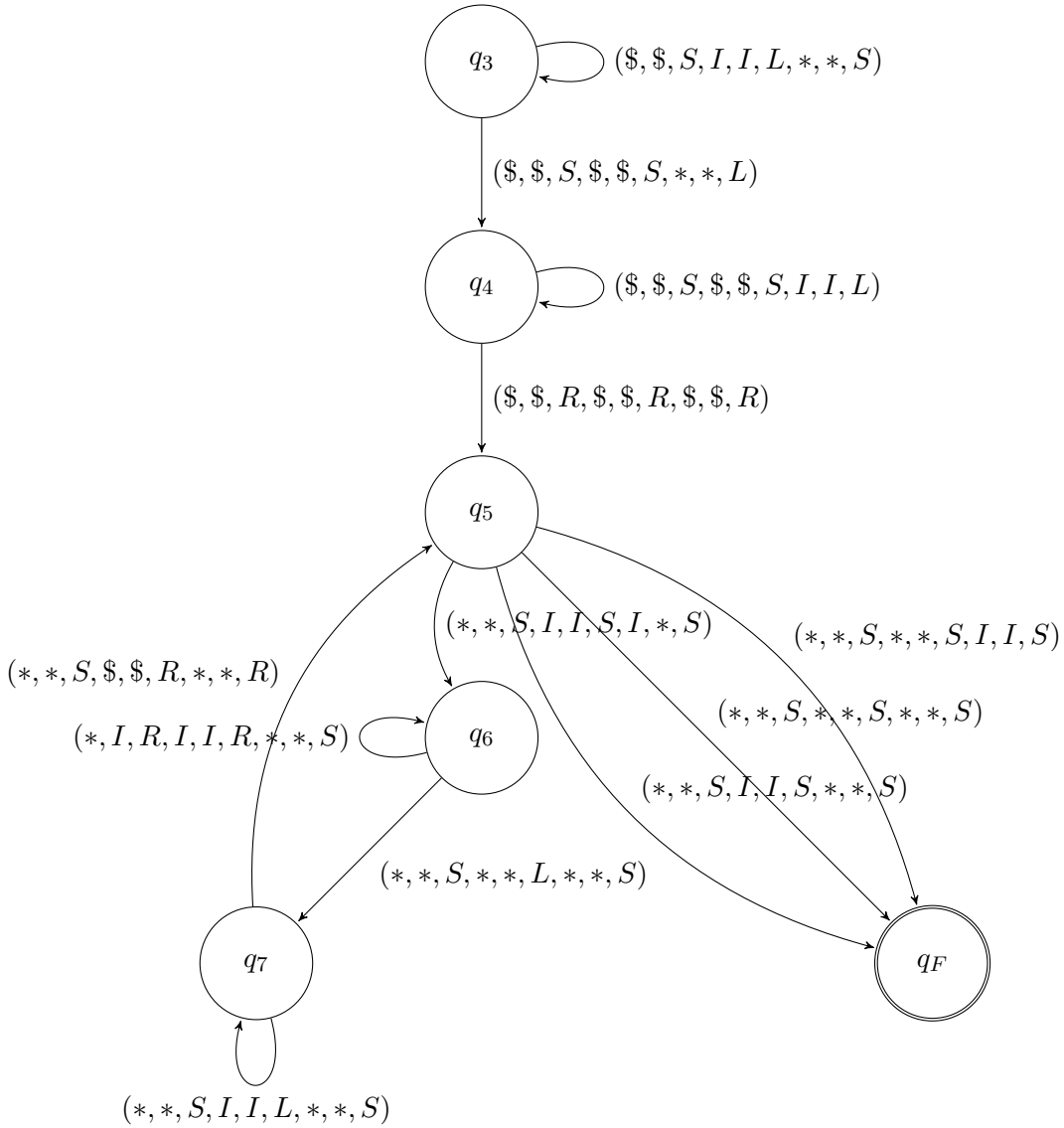
Exemple 2.18. Donat l'alfabet unari $\Sigma = \{I\}$, definirem, tot utilitzant la representació per un graf, una màquina de Turing M de tres cintes la qual computi la funció producte de dos nombres. D'aquesta manera, si $n, m \in \mathbb{N}$, a l'inici del còmput el contingut de la primera, la segona i la tercera cinta, respectivament, és

$$\$I^n * I^m **...$$
$$\$ **...$$
$$\$ **...$$

On el punter de la primera cinta senyala al primer símbol blanc que va a continuació del nombre m i els punters de la segona i tercera cinta senyalen al símbol blanc que va després del caràcter \$. Aleshores, M ve donada pel graf següent:



(El graf que representa a la màquina de Turing M segueix a la següent pàgina)



Explicant la màquina M de manera informal, aquesta comença el còmput amb els nombres n i m en la primera cinta, separant els dos nombres per un símbol blanc $*$ i les altres dues cintes estan buides, és a dir que després del símbol \$ només hi ha símbols blancs. Inicialment el punter senyala al primer símbol $*$ que es troba a continuació del nombre m. Seguidament, buida la primera cinta passant el nombre m a la tercera cinta i el nombre n a la segona cinta. I finalment, suma a la primera cinta el nombre n tantes vegades com símbols I hi ha en la segona cinta, és a dir m vegades.

Observem que tota màquina de Turing determinista unicinta també és multicinta, per tant, per veure l'equivalència entre aquests dos models només és necessari demostrar que podem obtenir una màquina de Turing unicinta a partir d'una multicinta. Aquest fet es el que veurem en el teorema següent.

Teorema 2.19. *Per a tota màquina de Turing determinista multicinta M, existeix una màquina de Turing determinista unicinta M' que simula M.*

Demostració: Sigui M una màquina de Turing determinista de k cintes, amb $k \geq 2$. Anem a construir una màquina de Turing determinista unicinta M' que simuli el còmput de M.

Els símbols de l'alfabet de la cinta de M pertanyeran també a l'alfabet de la cinta de M'. A més, per a cada símbol s de l'alfabet de la cinta de M, afegirem a l'alfabet de la cinta de M' un símbol nou s'. Llavors, M' simularà els còmputs de la màquina M de k cintes, guardant la informació de les k cintes en la seva únic cinta de la manera següent:

M' utilitzarà un nou símbol de cinta, #, per separar els continguts de les diferentes cintes i, per controlar la ubicació dels punters de les cintes de M, M' utilitzarà els nou símbols s', els quals seran utilitzats per representar els símbols als que el punter de cada cinta de M apunta.

Aleshores, per simular un pas de còmput de M, la màquina M' explorarà la seva cinta des del primer símbol # fins a trobar el k+1-símbol # per a determinar els caràcters als que senyalen els punters de M i, llavors, actualitzarà la seva cinta seguint amb el que defineixi la funció de transició de M. Si el punter d'alguna cinta de M es mou cap a un blanc que fins aquell moment no s'havia llegit, la màquina M' desplaçarà una casella a la dreta el contingut de la cinta comprès entre la cel·la a la que el punter actualment apunta i l'últim símbol # i escriurà un blanc en la casella que li correspon seguint amb la regla descrita amb anterioritat. ■

A continuació anem a posar un exemple de simulació d'una màquina de Turing multi-cinta per una màquina de Turing unicinta per ajudar en la comprenció de la demostració del teorema anterior.

Exemple 2.20. Amb aquest exemple es preten il·lustrar la idea que hi ha radere de la demostració del teorema anterior.

Sigui M una màquina de Turing de tres cintes tal que el contingut de les cintes és el següent:

$$\$|0|1|0|1|0| * ...$$
$$\$|a|a|a| * ...$$
$$\$|b|a| * ...$$

on la primera fila correspon al contingut de la primera cinta, la segona fila correspon al contingut de la segona cinta i la tercera fila correspon al contingut de la tercera cinta. Suposem que en la primera cinta el punter apunta a la tercera casella, en la segona cinta el punter apunta a la quarta cel·la i en la tercera cinta el punter apunta a la segona casella. Aleshores, el contingut de la màquina unicita M' que simula M és

$$\#|\$|0|1'|0|1|0|\#|\$|a|a|a'|\#|\$|b'|a|\#| * ...$$

Llavors, la màquina M' realitza els seus còmputs seguint amb el procediment descrit en la demostració del teorema anterior.

Una de les avantanges del model de màquina de Turing és la possibilitat de definir-ne un model indeterminista, el qual definirem tot seguit. Aquest fet no es dóna amb altres formalitzacions del concepte d'algorisme, alguns dels quals veurem en posteriors capítols.

Una màquina de Turing indeterminista és una estructura semblant al concepte de màquina de Turing determinista, amb la diferència de que no té perquè existir una forma única de fer el següent pas de còmput.

Definició 2.21. *Una màquina de Turing indeterminista és una estructura de la forma* $(K, \Sigma, \Gamma, \Delta, q_0, q_F)$ *on*

K *és un conjunt finit i no buit d'estats*

Σ *és un alfabet, que anomenarem alfabet d'entrada*

Γ *és un alfabet tal que* $\Sigma \cup \{\$, *\} \subseteq \Gamma$ *i* $\$, * \notin \Sigma$*, i l'anomenarem alfabet de la cinta*

$q_0 \in K$ *és l'estat inicial*

$q_F \in K$ *és l'únic estat final o acceptador*

$\Delta \subseteq ((K \setminus \{q_F\}) \times \Gamma) \times (K \times \Gamma \times \{L, R, S\})$ *tal que:*

a) Si $((q, a), (p, b, m)) \in \Delta$ *i* $a = \$$*, aleshores* $b = \$$ *i* $m \neq L$

b) Si $((q, a), (p, b, m)) \in \Delta$ *i* $a \neq \$$*, aleshores* $b \neq \$$

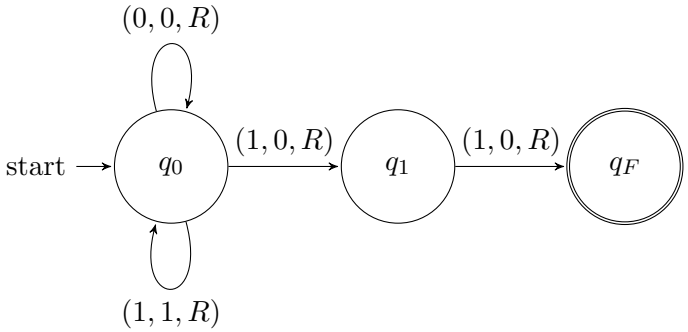
A Δ *se l'anomena el conjunt de transicions de M i a cada element de* Δ *se l'anomena una transició.*

Cal observar que el model de màquina de Turing indeterminista es pot extendre al d'una màquina de Turing indeterminista multicinta de la mateixa forma que en el cas del model estàndar i de la mateixa forma, també és possible demostrar-ne la seva equivalència.

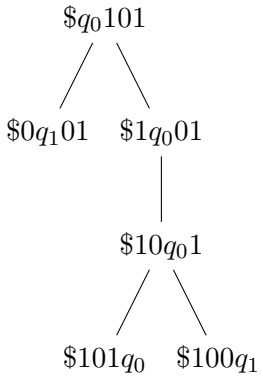
Una configuració d'una màquina de Turing indeterminista es defineix de manera anàloga al cas d'una màquina de Turing determinista.

Els còmputs d'una màquina de Turing indeterminista M es realitzen utilitzant el conjunt Δ. Per altra banda, en aquest cas els còmputs de M no estan unívocament determinats, i és per això que els possibles càlculs de M amb una entrada donada poden ser representats utilitzant un *arbre de computació*. Cada nus de l'arbre té associada una configuració de la màquina. L'arrel té associada la configuració inicial i un nus té tants fills com configuracions diferents a les què és pot arribar en un pas de M.

Exemple 2.22. Amb aquest exemple pretenem il·lustrar la idea de màquina de Turing indeterminista i la de arbre de computació d'una paraula d'entrada determinada.



Considerem la paraula d'entrada $x = 101$. Llavors l'arbre de computació per aquesta paraula és el següent;



Observem que tota màquina de Turing determinista també és una màquina de Turing indeterminista. Per tant, per veure l'equivalència entre models, només serà necessaria la demostració de que es possible obtenir una màquina de Turing determinista a partir d'una d'indeterminista. Aquest fet és el que veurem amb el teorema que ve a continuació.

Teorema 2.23. *Tota màquina de Turing indeterminista M es pot simular per una màquina de Turing determinista M'*

Demostració: Sigui $M = (K, \Sigma, \Gamma, \Delta, q_0, q_F)$ una màquina de Turing indeterminista. Sigui x una entrada per la màquina M. Construim una màquina de Turing M' determinista que utilitzi dues cintes, on en la primera, emmagatzema configuracions de M i, la segona, la utilitza per simular un recorregut en amplada de l'arbre de configuració de l'entrada x. Seguidament veurem com fer la simulacio del comput de M per l'entrada donada a partir de la màquina M'.

Per fer el recorregut en amplada, podem fer servir una seqüència de valors per determinar el nus en que ens trobem, de manera que resulti senzill determinar quin és el nus següent. Si per a un parell (estat, símbol) la relació de transició té, com a màxim, k possibilitats diferents, és possible utilitzar un recorregut en ordre lexicogràfic per longituds dels mots de l'alfabet {1,...,k}. Partint de l'arrel, cada paraula identifica un camí.

Posant un exemple, si tenim 1213, haurem de començar a l'arrel, passar al primer fill, seguidament passar al segon fill del nus on ens trobem actualment. A continuació passar al primer fill i així successivament.

La longitud de la paraula que descriu el camí que cal seguir ens indicarà la profunditat a la què és troba el nus.

D'aquesta manera, combinant la màquina de Turing que calcula la paraula següent en ordre lexicogràfic, amb una que utilitzi la paraula per seleccionar a cada pas la transició marcada per la paraula, obtenim una màquina de Turing determinista que simula M.

Si arribem a una configuració de parada, la màquina M' s'atura i ho fa acceptant. Altrament, M' procedeix assajant noves possibilitats. ■

A continuació enunciarem la tesis de Church-Turing.

Tesis de Church-Turing

Tot algorisme es pot simular per una màquina de Turing

En la següent secció d'aquest capítol veurem com les estructures algorísmiques bàsiques, tals com són la seqüenciació, els condicionals i les iteracions és poden implementar mit-

jançant l'us de màquines de Turing, fet que ens porta a admetre aquest concepte com a model formal d'algorisme. Aquest és un dels fets que fa pensar amb la veracitat de la tesis de Church-Turing.

2.4 Màquines de Turing i algorismes

Composició seqüencial: Donades dues màquines de Turing, M i N, volem construir una màquina de Turing que implementi la seva composició seqüencial, és a dir, l'esquema

M ; N

La nova màquina de Turing ha de connectar la màquina M amb la màquina N, preservant el contingut de la cinta. Observem que, perquè N comenci, M s'haurà hagut d'aturar i ,a més, si M s'atura acceptant, la nova màquina de Turing que implementi la composició haurà d'acceptar.

Recordem que la condició de que M pari es dóna quan s'arriba a una configuració per a la qual no hi ha cap transició definida. Aquests casos es poden detectar mirant els parells (estat, caràcter), en la definició de la funció de transició de M.

Procedim a definir de manera formal la composició seqüencial de dues màquines de Turing. Siguin $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ i $N = (K', \Sigma, \Gamma, \delta', q_0', q_F')$ dues màquines de Turing. Podem suposar que $K \cap K' = \emptyset$, ja que si no fos així, podem redefinir els estats d'una de les dues màquines. De la mateixa manera, podem suposar que els alfabets de la cinta de les dues màquines son els mateixos, ja que en cas contrari agafaríem la unió dels respectius alfabets com a alfabet de la nova màquina. Una màquina que correspon a la composició seqüencial és la que ve descrita a continuació:

$$cs(M, N) = (K'', \Sigma, \Gamma, \delta'', q_0, q_F')$$

on

a) $K'' = K \cup K'$

b) Les transicions de δ i δ' estan contingues en δ''. Amb l'afegit de que, per a tot $q \in K$ i $a \in \Gamma$, si $q \neq q_F$ i $(q, a) \notin$ Dom δ, llavor afegim $\delta''(q, a) = (q_0', a, S)$. Per finalitzar, per a qualsevol $a \in \Gamma$ afegim $\delta''(q_F, a) = (q_F', a, S)$.

Observem que quan M i N son màquines de Turing de parada segura, la seva composició seqüencial també és una màquina de Turing de parada segura.

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

si D llavors M si no N fi si

La màquina D avaluarà la condició afirmativament quan s'aturi en l'estat q_s i negativament quan s'aturi en l'estat q_n. Si $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$, $N = (K', \Sigma, \Gamma, \delta', q_0', q_F')$ i $D = (K'', \Sigma, \Gamma, \delta'', q_0'', q_s, q_n)$, de manera formal, una màquina que correspon a l'esquema de composició condicional presentat és

$$cc(D, M, N) = (K''', \Sigma, \Gamma, \delta''', q_0'', q_F''')$$

on

a) $K''' = K \cup K' \cup K'' \cup \{q_F'''\}$

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

$\delta'''(q_s, a) = (q_0, a, S)$

$\delta'''(q_n, a) = (q_0', a, S)$

$\delta'''(q_F, a) = (q_F''', a, S)$

$\delta'''(q_F', a) = (q_F''', a, S)$

Quan D, M i N son màquines de Turing de parada segura, aleshores la seva composició condicional també és una màquina de Turing de parada segura.

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

mentre D fer M fi mentre

Si $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ i $D = (K', \Sigma, \Gamma, \delta', q_0', q_s, q_n)$, una màquina que implementi la composició iterativa de M i D és

$$ci(D, M) = (K'', \Sigma, \Gamma, \delta'', q_0', q_n)$$

on

a) $K'' = K \cup K'$

b) δ'' conté totes les transicions de δ i de δ' i, a més, per a tor $a \in \Gamma$, les transicions:

$\delta''(q_s, a) = (q_0, a, S)$

$\delta''(q_F, a) = (q_0', a, S)$

Explicat de manera informal, la màquina de Turing $ci(D, M)$ computa de la forma següent: si donada una paraula d'entrada x l'estat de la configuració de parada de la màquina condicional D és q_s, aleshores es procedirà a realitzar els còmputs de la màquina M i, en cas contrari, el procediment s'aturarà. I si donada una paraula d'entrada y, l'estat de la configuració de parada de M és q_F, es procedirà a realitzar el còmputs de la màquina D.

Observem que, a diferència de la composició seqüencial i de la composició condicional, si M i D són màquines de Turing de parada segura, la seva composició iterativa pot no ser una màquina de Turing de parada segura. Per garantir que aquest fet es compleixi, hem de poder demostrar que, per a qualsevol entrada, el nombre d'iteracions és finit.

En el capítol 5 veurem com les funcions de l'aritmètica són computables per màquines de Turing.

3 Gramàtiques

En aquesta secció, veurem una altra formalització del concepte d'algorisme com són les gramàtiques i demostrarem que tota funció computable per una màquines de Turing és gramaticalment computable.

Ara passarem a definir el concepte de gramàtica i el concepte de derivació en una gramàtica.

Definició 3.1. *Una gramàtica és una quàdrupla $G = (V, \Sigma, P, S)$ on*

V i Σ són dos alfabets tals que $\Sigma \subseteq V$ l'anomenem l'alfabet dels símbols terminals i anomenarem als elements del conjunt $V \setminus \Sigma$ símbols no terminals o variables.

P és un subconjunt finit de $((V^*(V \setminus \Sigma)V^*) \times V^*)$ i als seus elements se'ls anomena regles o produccions.

$S \in V$ és la variable inicial.

Usarem la notació $u \to_G v$ per designar que $(u, v) \in P$. Si $w \in V^*$, l'efecte de l'aplicació d'una producció $u \to_G v$ a la paraula w és la substitució en aquest mot d'una aparició de u per v.

A continuació definirem el concepte de derivació en una gramàtica. Explicat de manera informal una derivació en una gramàtic procedeix de la manera següent. S'inicia el còmput amb la variable inicial S i, a continuació, s'aplica una producció a S la qual canvia aquesta variable per una paraula de V^*. La gramàtica segueix amb el procés d'aplicar produccions a la paraula fins que s'arriba a un mot en el qual no hi ha cap aparició d'alguna variable.

Definició 3.2. *Sigui $G = (V, \Sigma, P, S)$ una gramàtica.*

1) Siguin $u, v \in V^*$. Escriurem $u \Rightarrow_G v$ si existeixen $w_1, w_2 \in V^*$ i una regla $u' \to_G v'$ tals que $u = w_1 u' w_2$ i $v = w_1 v' w_2$.

2) Siguin $u, v \in V^*$, direm que v es deriva de u, en símbols $u \Rightarrow_G^* v$, si obtenim v a partir de u aplicant un nombre finit de produccions, és a dir si existeixen $u_1, ..., u_n \in V^*$ tals que

$$u \Rightarrow_G u_1 \Rightarrow_G ... \Rightarrow_G u_n \Rightarrow_G v$$

3) Sigui $x \in \Sigma^*$ una paraula. Direm que x està generada per G si $S \Rightarrow_G^* x$

Definició 3.3. *Sigui G una gramàtica. Direm $L(G)$ al llenguatge de totes les paraules de Σ^* generades per G.*

Exemple 3.4. Anem a definir una gramàtica G que generi $\{x \in \{a, b, c\}^* : x$ té el mateix nombre de a's, de b's i de c's$\}$. G té com a símbols no terminals S (el símbol d'inici) i A, B i C i com a símbols terminals a, b i c. I les produccions de G són les següents:

$$S \to \lambda,\ S \to ABCS,\ AB \to BA,\ AC \to CA,\ BC \to CB,\ BA \to AB,\ CA \to AC,$$
$$CB \to BC,\ A \to a,\ B \to b \text{ i } C \to c.$$

Anem a posar com a exemple la generació de la paraula *ccbaba* a partir de la gramàtica G.

$$S \Rightarrow ABCS \Rightarrow ABCABCS \Rightarrow ABCABC \Rightarrow ABCACB \Rightarrow ABCCAB \Rightarrow ACBCAB \Rightarrow$$
$$CABCAB \Rightarrow CACBAB \Rightarrow CCABAB \Rightarrow CCBAAB \Rightarrow CCBABA \Rightarrow cCBAB \Rightarrow$$
$$ccBABA \Rightarrow ccbABA \Rightarrow ccbaBA \Rightarrow ccbabA \Rightarrow ccbaba$$

A continuació, introduïrem un lema el qual serà usat posteriorment per demostrar que tota funció computable per una màquina de Turing és gramaticalment computable.

Lema 3.5. *Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing. Aleshores, existeix una gramàtica G tal que per a tot parell de configuracions $uqav$ i $u'q'a'v'$ de M,*

$$uqav \vdash_M^* u'q'a'v'$$

si i només si

$$[uqav] \Rightarrow_G^* [u'q'a'v']$$

On el símbol [i el símbol] són nous i no pertanyen ni a Σ ni a K. A més, assumirem que $\Sigma \cap K = \emptyset$.

Demostració. Observem que la idea general de la demostració és representar la configuració $uqav$ per la paraula $[uqav]$. Cal fixar-se que la posició de l'estat ens indica la posició del punter de la màquina de Turing.

El símbol [i el símbol] apareixen únicament al final de la representació d'una configuració, però mentres que el símbol [està inclòs aquí per facilitar la lectura, el símbol] ens indica l'inici de la cadena infinita de símbols blancs que hi ha en tota cinta d'una màquina de Turing a continuació de la paraula de sortida.

De manera formal, sigui $G = (V, \Gamma, P, S)$ una gramàtica tal que

$$V = K \cup \Gamma \cup \{[,], S\}$$

i P està constituïda per les produccions següents:

a) Per a tot $q \in K$ i $a \in \Gamma$, si $\delta(q, a) = (p, b, S)$, on $p \in K$ i $b \in \Gamma$, llavors P conté

$$qa \to pb$$

b) Per a tot $q \in K$ i $a \in \Gamma$, si $\delta(q, a) = (p, b, R)$, on $p \in K$ i $b \in \Gamma$, llavors P conté

$$qac \to bpc$$

per a tot $c \in \Gamma$, i

$$qa] \to bp*]$$

c) Per a tot $q \in K$ i $a \in \Gamma$, si $\delta(q, a) = (p, b, L)$, on $p \in K$ i $b \in \Gamma$, llavors P conté

$$dqac \to pdbc$$

per a tot $c \in \Gamma$, i

$$dq*] \to pd]$$

Per tant, em vist que $uqav \vdash_M u'q'a'v'$ si i només si $[uqav] \Rightarrow_G [u'q'a'v']$. De forma anàloga s'aplicara per $\vdash_M^*$ i per $\Rightarrow_G^*$, ja que aquestes operacions es deriven de les primeres a partir de la seva definició. ■

Tot seguit, definirem el concepte de funció gramaticalment computable per, a continuació, enunciar i demostrar el teorema que ens explica com tota funció computable per màquines de Turing és gramaticalment computable.

Definició 3.6. *Siguin Σ_0 i Σ_1 dos alfabets que no contenen a $*$ ni a \$ i sigui $f : \Sigma_0^* \to \Sigma_1^*$ una funció. Direm que f és gramaticalment computable si existeix una gramàtica $G = (V, \Sigma, P, S)$, on $\Sigma_0, \Sigma_1 \subseteq \Sigma$, i hi ha paraules $x, x', y, y' \in V^*$ tals que per a tota paraula $u \in \Sigma_0^*$ i $v \in \Sigma_1^*$,*

$$f(u) = v \text{ si, i només si, } xuy \Rightarrow_G^* x'vy'$$

En aquest cas direm que G computa f.

Prèviament a anunciar i demostrar el teorema esmentat anteriorment, donarem un exemple que il·lustri el concepte de funció gramaticalment computable.

Exemple 3.7. Considerem l'alfabet $\Sigma_0 = \{a, b\}$ i sigui $f : \Sigma_0^* \to \Sigma_0^*$ definida per $f(x) = x^I$, és a dir la funció que donada una paraula $x = a_1...a_n$ (on $a_i \in \Sigma_0$, per a qualsevol $i \in \{1, ..., n\}$) ens retornarà la paraula inversa, és a dir ens donarà com a sortida la paraula $x^I = a_n...a_1$. Aleshores f és computable per la gramàtica $G = (V, \Sigma_0, P, S)$, on $V = \{a, b, A, B, *, S, [,]\}$, i P està formada per les produccions següents:

$$1.\ [a \to [A,\ [b \to [B$$
$$2.\ Aa \to aA,\ Ab \to bA,\ Ba \to aB,\ Bb \to bB$$
$$3.\ A* \to *a,\ B* \to *b$$

Siguin $x = [$, $y = *]$, $x' = [*$ i $y' =]$ les paraules de V^* que ens demana la definició de funció gramaticalment computable. G computa f de la forma següent: Comença amb la paraula $[x*]$, on $x \in \Sigma_0^*$. Les produccions del tipus 1 ens canvien el primer símbol de la paraula x, sigui a o b, pels corresponents símbols no terminals, A o B respectivament. Seguidament amb l'us de les produccions del tipus 2 fem passar el primer símbol a l'últim de la paraula, sense contar com a últim símbol el caràcter $*$. Finalment, aplicarem les transicions de tipus 3.

$$[abb*] \Rightarrow [Abb*] \Rightarrow [bAb*] \Rightarrow [bbA*] \Rightarrow [bb*a] \Rightarrow [Bb*a] \Rightarrow [bB*a] \Rightarrow [b*ba] \Rightarrow [B*ba] \Rightarrow [*bba]$$

Teorema 3.8. *Tota funció computable per una màquina de Turing és gramaticalment computable.*

Demostració: Sigui $f : \Sigma_0^* \to \Sigma_1^*$ una funció T-computable. Per tant, per definició de funció computable per màquines de Turing, existeix una màquina $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ tal que per a tota paraula $u \in \Sigma_0^*$, f(u) = v si i només si,

$$\$uq_0* \vdash_M^* \$vq_F*$$

Siguin x, y, x' i y' definits de la manera següent.

$$x = [\$$$
$$y = q_0*]$$

$$x' = [\$$$$y' = q_F*]$$

Llavors la gramàtica G construïda a partir del Lema 3.5 computa M mitjançant x, x', y i y', ja que

$$\$uq_0* \vdash_M^* \$vq_F* \iff [\$uq_0*] \Rightarrow_G^* [\$vq_F*]$$

■

Observem que en la demostració del teorema hem vist que les funcions computables per màquines de Turing que tenen per entrada i sortida paraules sòn gramaticalment computables. Un cas particular és veure que tota funció computable per una màquina de Turing la qual tingui com a domini i recorregut els nombres naturals és gramaticalment computable. Aquest fet és possible considerant la igualtat $\Sigma_0 = \Sigma_1 = \{I\}$, on I és el codi unitari i per representar els nombres naturals ho farem de la manera següent: si n és un nombre natural qualsevol, aleshores en el llenguatge $\{I\}^*$, n serà representat com I...I n vegades.

4 Funcions Recursives

4.1 Conceptes bàsics

En aquesta secció veurem la definició de funció recursiva i la demostració de com tota funció gramaticalment computable és també una funció recursiva. Previament, ens serà necessaria la introducció de diferentes nocions requerides per a la definició d'aquest concepte.

Ara passarem a definir les anomenades funcions inicials, de les quals n'hi ha tres tipus, i dos regles de combinació.

Definició 4.1. *a) La funció zero ζ és la funció de 0 arguments tal que*

$$\zeta() = 0$$

b) Sigui $k \geq 1$ i sigui $1 \leq i \leq k$. Llavors, la funció i-èsima de k arguments projecció π_i^k és la funció de $\mathbb{N}^k$ en $\mathbb{N}$ tal que

$$\pi_i^k(n_1, ..., n_k) = n_i \text{ per a qualsevols } n_1, ..., n_k \in \mathbb{N}$$

c) La funció successor σ és la funció de $\mathbb{N}$ en $\mathbb{N}$ tal que

$$\sigma(n) = n + 1 \text{ per a qualsevol } n \in \mathbb{N}$$

d) Anomenarem funció inicial a una de les funcions seguents: ζ, σ o bé π_i^k per a $i \in \{1, ..., k\}$.

Definició 4.2. *a) Siguin $l > 0$ i $k \geq 0$, g una funció de l arguments i $h_1, ..., h_l$ funcions de k arguments. Sigui f la funció de k arguments tal que, per a tot $\overline{n} \in \mathbb{N}^k$,*

$$f(\overline{n}) = g(h_1(\overline{n}), ..., h_l(\overline{n}))$$

Llavors, f es diu que ha estat obtinguda a partir de $g, h_1, ..., h_l$ per composició.

b) Sigui $k \geq 0$, g una funció de k arguments i h una funció de k+2 arguments. Sigui f la funció de k+2 arguments tal que per a tot $\overline{n} \in \mathbb{N}^k$,

$$f(\overline{n}, 0) = g(\overline{n})$$

i que per a tot $\overline{n} \in \mathbb{N}^k$ i $m \in \mathbb{N}$

$$f(\overline{n}, m + 1) = h(\overline{n}, m, f(\overline{n}, m))$$

Llavors, f es diu que ha estat obtinguda a partir de g i h per recursió primitiva.

A continuació, donarem la definició de funció recursiva primitiva el qual és un concepte necessari per a la definició de la noció de funció recursiva.

Definició 4.3. *Una funció es diu que és recursiva primitiva si és una funció inicial o pot ser generada a partir de les funcions inicials mitjançant l'aplicació d'alguna seqüència de les operacions de composició i de recursió primitiva. És a dir, les funcions recursives primitives són la classe més petita de funcions que contenen les funcions inicilas i tancada sota les operacions de composició i de recursió primitiva.*

Per exemple, sigui $\sigma^2 : \mathbb{N} \to \mathbb{N}$ la funció tal que $\sigma^2(n) = n + 2$ per a qualsevol $n \in \mathbb{N}$. Aleshores σ^2 és recursiva primitiva, ja que s'obté a partir de la composició de la funció σ (la funció successor la qual és una funció inicial) amb ella mateixa, és a dir $\sigma^2(n) = \sigma(\sigma(n))$.

Anem ara a estudiar les funcions recursives primitives des d'un punt de vista més sistemàtic.

Definició 4.4. *Per a qualsevols $k \geq 0$ i $j \geq 0$, definim la funció j-èsima de k arguments constant $K_j^k : \mathbb{N}^k \to \mathbb{N}$ per $K_j^k(\overline{n}) = j$ per a tot $\overline{n} \in \mathbb{N}^k$.*

Lema 4.5. *Totes les funcions constants són funcions recursives primitives.*

Demostració. La demostració es fa per inducció sobre k.

Cas k = 0: K_0^0 és ζ, que és una de les funcions inicials;

$$K_{j+1}^0() = \sigma(K_j^0()) \text{ per a tot } j \geq 0,$$

per tant, K_{j+1}^0 s'obté a partir de σ i K_j^0 per composició.

Cas inductiu: Suposem que per algun $k \geq 0$ la funció K_j^k és recursiva primitiva per a tot $j \geq 0$, aleshores anem a mostrar que K_j^{k+1} és una funció recursiva primitiva per a tot $j \geq 0$.
Tenim que

$$K_j^{k+1}(\overline{n}, 0) = K_j^k(\overline{n})$$
$$K_j^{k+1}(\overline{n}, m+1) = \pi_{k+2}^{k+2}(\overline{n}, m, K_j^{k+1}(\overline{n}, m))$$

Per tant, K_j^{k+1} es obtinguda a partir de K_j^k i de π_{k+2}^{k+2} per recursió primitiva, finalitzant així amb la demostració. ■

Definició 4.6. *Sigui g una funció de k arguments i f una funció de l arguments, on $k, l \geq 0$. Suposem que existeixen funcions $h_1, ..., h_k : \mathbb{N}^l \to \mathbb{N}$, on cada una d'elles és o bé una projecció o bé una funció constant de l arguments, tals que per a tot $\overline{n} \in \mathbb{N}^l$,*

$$f(\overline{n}) = g(h_1(\overline{n}), ..., h_k(\overline{n}))$$

Llavors f es diu que ha estat obtinguda a partir de g per transformació explícita.

Com que la funció projecció i les funcions constants són recursives primitives, obtenim de forma immediata el lema següent.

Lema 4.7. *Si g és una funció recursiva primitiva, aleshores qualsevol funció obtinguda de g a partir de transformació explícita també és recursiva primitiva.*

Exemple 4.8. Sigui $m \in \mathbb{N}$. La funció predecessor V ve descrita de la manera següent:

$$V(m) = \begin{cases} 0 & \text{si } m = 0 \\ m - 1 & \text{si } m > 0 \end{cases}$$

Siguin $n, m \in \mathbb{N}$. La funció $minus(n, m) = n \dot{-} m$ ve definida de la manera següent: $minus(n, m)$ és 0 si $n < m$ i val $n - m$ en cas que $n \geq m$.

Sigui $m \in \mathbb{N}$. La funció signe, que denotarem per sg, ve descrita de la forma següent: $sg(m)$ val 0 si $m = 0$ i val 1 si $m > 0$.

Observem que les funcions V, minus i signe són funcions recursives primitives, ja que V es pot descriure per

$$V(0) = K_0^0()$$
$$V(m+1) = \pi_1^2(m, V(m))$$

minus es pot descriure per

$$n \dot{-} 0 = n = \pi_1^1(n)$$
$$n \dot{-} m + 1 = V(n \dot{-} m) = h(n, m, n \dot{-} m)$$

on $h(n_1, n_2, n_3) = V(\pi_3^3(n_1, n_2, n_3))$ és una funció recursiva primitiva.
I signe es pot descriure per

$$sg(0) = 0$$
$$sg(m+1) = 1$$

Exemple 4.9. Anem a veure com les funcions suma, producte i potència són recursives primitives.

a) Siguin $n_1, n_2 \in \mathbb{N}$. Sigui $suma(n_1, n_2) = n_1 + n_2$. Aleshores la funció $suma$ s'obté a partir de π_1^1 i σ per recursió primitiva com es pot veure a continuació.

$$suma(n, 0) = n + 0 = \pi_1^1(n) = n$$
$$\text{i}$$
$$suma(n, m+1) = \sigma(n + m)$$

per a qualsevols $n, m \in \mathbb{N}$.

b) Siguin $n, m \in \mathbb{N}$. La funció producte $n \cdot m$ es pot definir a partir de recursió primitiva de la manera següent:

$$n \cdot 0 = n$$
$$\text{i}$$
$$n \cdot (m+1) = n \cdot m + n$$

demostrant així que la funció producte és recursiva primitiva.

c) Siguin $n, m \in \mathbb{N}$. La funció potència n^m es possible definir-la a partir de recursió primitiva com es veu a continuació:

$$n^0 = 1$$
$$\text{i}$$
$$n^{m+1} = n^m \cdot n$$

provant d'aquesta manera que la funció potència és recursiva primitiva.

Lema 4.10. *Sigui $k \geq 0$. Si g és una funció recursiva primitiva de k+1 arguments, alsehores la funció f següent també és recursiva primitiva*

$$f(\overline{n}, m) = \Pi_{i=0}^{m} g(\overline{n}, i) = g(\overline{n}, 0) \cdot \cdots \cdot g(\overline{n}, m)$$

En aquest cas direm que f s'obté a partir de g per producte acotat.

Demostració. Observem que

$$f(\overline{n}, 0) = g(\overline{n}, 0), \text{ per a tot } \overline{n} \in \mathbb{N}^k$$

i que

$$f(\overline{n}, m+1) = f(\overline{n}, m) \cdot g(\overline{n}, m+1), \text{ per a tot } \overline{n} \in \mathbb{N}^k \text{ i per a tot } m \in \mathbb{N}$$

Per tant, podem concloure que f és una funció recursiva primitiva ■

Exemple 4.11. La funció factorial, $f(n) = n!$, és recursiva primitiva, ja que

$$n! = \Pi_{i=1}^{n} i$$

i, pel lema anterior tenim que f és recursiva primitiva.

Tot seguit definirem el concepte de predicat recursiu primitiu, el qual és necessari per la comprensió del significat de funció recursiva.

Definició 4.12. *a) Sigui $k \geq 0$. Per un predicat de k arguments entendrem un subconjunt de $\mathbb{N}^k$.*
En general utilitzarem lletres majúscules pels predicats i, si P és un predicat, escrivim $P\overline{n}$ per designar que $\overline{n} \in P$.

b) La funció característica d'un predicat P de k arguments és la funció de k arguments $f : \mathbb{N}^k \to \{0, 1\}$ tal que, per a qualsevol $\overline{n} \in \mathbb{N}^k$, $f(\overline{n})$ és 1 si $P\overline{n}$ i $f(\overline{n})$ és 0 en cas contrari.

c) Es diu que un predicat és recursiu primitiu si la seva funció característica és recursiva primitiva.

Exemple 4.13. La relació d'igualtat $=$ és primitiva recursiva, ja que la seva funció característica és

$$\epsilon(n, m) = 1 \dot{-} sg((n \dot{-} m) + (m \dot{-} n))$$

D'igual forma, la relació $<$ és primitiva recursiva, ja que la seva funció característica és

$$\lambda(m, n) = sg(n \dot{-} m)$$

Definició 4.14. *Siguin P i Q dos predicats. Si P s'obté a partir de Q a partir de duplicar, permutar o substituir valors constants pels arguments de Q, aleshores es diu que P és obtingut a partir de Q per transformació explícita.*

Lema 4.15. *Si Q és un predicat recursiu primitiu, alsehores qualsevol predicat obtingut a partir de Q per transformació explícita també és recursiu primitiu.*

Demostració. Sigui P un predicat obtingut a partir de Q per transformació explícita. Sigui h la funció característica de P i sigui g la funció característica de Q. Aleshores h s'obtindrá a partir de g per transformació explícita que, com ja hem demostrat amb anterioritat, com que g és recursiva primitiva, llavors h també ho serà. ■

Passem ara a considerar quatre operacions que es duen a terme entre predicats, les quals són la conjunció, la disjunció, la negació i la implicació.

Lema 4.16. *Si Q i R són predicats recursius primitius, aleshores també ho són la seva conjunció, la seva disjunció, la seva implicació i la negació del predicat Q.*

Demostració. Siguin g i h les funcions característiques recursives primitives dels predicats Q i R, respectivament. Aleshores les funcions característiques de la conjunció de Q i R, de la disjunció de Q i R, de Q implica R i de la negació de Q són f_1, f_2 i f_3, respectivament, on
$f_1(\overline{n}, \overline{m}) = g(\overline{n}) \cdot h(\overline{m})$
$f_2(\overline{n}, \overline{m}) = sg(g(\overline{n}) + h(\overline{m}))$
$f_3(\overline{n}, \overline{m}) = sg((1 - g(\overline{n})) + h(\overline{m}))$
$f_4(\overline{n}) = 1\dot{-}g(\overline{n})$ ■

Exemple 4.17. Pel fet de que les relacions d'igualtat $=$ i de menor estricte $<$ són primitives recursives, aleshores també ho són les relacions $>$, $\leq$, $\geq$ i $\neq$ aplicant el lema anterior.

A continuació, definirem les nocions de quantificador existencial acotat i la de quantificador universal acotat.

Definició 4.18. *Sigui $k \geq 0$. Si Q és un predicat de k+1 arguments.*

a) Diem quantificador existencial acotat de Q al predicat R de k+1 arguments tal que per a tot $\overline{n} \in \mathbb{N}^k$ i $m \in \mathbb{N}$, $R\overline{n}m$ si i només si existeix una i, $0 \leq i \leq m$, tal que $Q\overline{n}i$.
De forma més abreviada, escrivim

$$R\overline{n}m \text{ si i només si } \exists i \leq m[Q\overline{n}i]$$

b) Diem quantificador universal acotat de Q al predicat S de k+1 arguments tal que $S\overline{n}m$ si i només si per a qualsevol i, $0 \leq i \leq m$, $Q\overline{n}i$.
De forma més abreviada, escrivim

$$S\overline{n}m \text{ si i només si } \forall i \leq m[Q\overline{n}i]$$

Lema 4.19. *Si Q és un predicat recursiu primitiu, aleshores també ho són els seus quantificador existencial acotat i quantificador universal acotat.*

Demostració. Sigui g la funció característica recursiva primitiva del predicat Q. Aleshores les funcions característiques f_1 i f_2 del quantificador universal acotat i del quantificador existencial acotat de Q, respectivament, vindran donades per

$f_1(\overline{n}, m) = \Pi_{i=0}^{m} g(\overline{n}, i)$

i

$f_2(\overline{n}, m) = 1\dot{-}\Pi_{i=0}^{m}(1\dot{-}g(\overline{n}, i))$

Observem que $f_1(\overline{n}, m) = 1$ si i només si $g(\overline{n}, i) = 1$, $\forall 0 \leq i \leq n$, és a dir que $Q\overline{n}i$ per a qualsevol $0 \leq i \leq m$, i, per tant és clar que f_1 ens defineix la funció característica del quantificador universal acotat. Per altra banda, observem també que $f_2(\overline{n}, m) = 1$ si i només si existeix algun $i \in \{0, ..., m\}$ tal que $g(\overline{n}, i) = 1$, és a dir que $Q\overline{n}i$ i que per tant faci que $\Pi_{i=0}^{m}(1\dot{-}g(\overline{n}, i)) = 0$, i d'aquesta manera podem concloure que f_2 ens defineix la funció característica del quantificador existencial acotat. ■

Lema 4.20. *Siguin $k \geq 0$ i $l \geq 2$. Siguin $g_1, ..., g_l$ funcions recursives primitives de k arguments i siguin $P_1, ..., P_l$ predicats primitius recursius de k arguments, tals que per a tot $\overline{n} \in \mathbb{N}^k$, existeixi exactament una i, $0 \leq i \leq l$, tal que $P_i\overline{n}$. Aleshores, la funció de k arguments f definida com*

$$f(\overline{n}) = \begin{cases} g_1(\overline{n}) & \text{si } P_1\overline{n} \\ \vdots & \\ g_l(\overline{n}) & \text{si } P_l\overline{n} \end{cases}$$

és també primitiva recursiva. (La funció f es diu que està definida per casos a partir de $g_1, ..., g_l$ i de $P_1, ..., P_l$).

Demostració. Si els predicats $P_1, ..., P_l$ tenen com a funcions característiques recursives primitives $h_1, ..., h_l$, aleshores

$$f(\overline{n}) = g_1(\overline{n}) \cdot h_1(\overline{n}) + ... + g_l(\overline{n}) \cdot h_l(\overline{n}), \text{ per a tot } \overline{n} \in \mathbb{N}^k$$

Observem que per a cada $\overline{n}$, un i solsament un dels sumants és diferent de zero. ■

Tot seguit, anem a introduir el concepte de minimalització acotada el qual ens defineix una funció a partir d'un predicat.

Definició 4.21. *Sigui $k \geq 0$ i sigui Q un predicat de k+1 arguments. Aleshores la minimalització acotada de Q és la funció de k+1 arguments f tal que $f(\overline{n}, m)$ és el nombre p més petit, $0 \leq p \leq m$, tal que $Q\overline{n}p$ si existeix tal $p \in \{0, .., m\}$, i val 0 altrament.*

Escrivim $f(\overline{n}, m)$ com $\mu p \leq m[Q\overline{n}p]$.

Lema 4.22. *Si Q és un predicat primitiu recursiu, aleshores també ho és la seva minimalització acotada.*

Demostració. Sigui $f(\overline{n}, m) = \mu p \leq m[Q\overline{n}p]$. Sigui h la funció de k+2 arguments definida de la forma següent:

$$h(\overline{n}, m, p) = \begin{cases} p & \text{si } \exists i \leq m[Q\overline{n}i] \\ m+1 & \text{si } Q\overline{n}m+1 \text{ i no } \exists i \leq m[Q\overline{n}i] \\ 0 & \text{si no } \exists i \leq m+1[Q\overline{n}i] \end{cases}$$

Aleshores h és primitiva recursiva, ja que ha estat construïda per casos a partir de funcions primitives recursives i predicats. Llavors tenim que

$$f(\overline{n}, 0) = 0$$
$$f(\overline{n}, m+1) = h(\overline{n}, m, f(\overline{n}, m))$$

Per tant, f és recursiva primitiva. ■

4.2 Nombres de Gödel

En aquest apartat introduïrem els nombres de Gödel, els quals seran posteriorment usats per codificar de forma adequada el concepte de derivació en una gramàtica i, d'aquesta manera, poder demostrar que tota funció gramaticalment computable és recursiva.

Per veure la formalització d'aquesta idea, sigui Σ un alfabet i sigui $\beta = |\Sigma| + 1$. Fixem en Σ un ordre en els seus elements, és a dir si $\Sigma = \{d_1, ..., d_{\beta-1}\}$, llavors fixem l'ordre $d_1 < ... < d_{\beta-1}$.
Com veurem, cada paraula de Σ^* pot ser vista com un enter en base β. Més especificament, definirem la funció $gn : \Sigma^* \to \mathbb{N}$ de la forma següent: si $x = d_{i_1}...d_{i_k}$, on $k \geq 0$ i $1 \leq i_j \leq \beta - 1$ per a tot $j = 1, ..., k$, aleshores

$$gn(x) = \beta^{k-1} \cdot i_1 + \beta^{k-2} \cdot i_2 + ... + \beta \cdot i_{k-1} + i_k$$

I en particular, $gn(\lambda) = 0$. Llavors direm que $gn(x)$ és el *nombre de Gödel* de la paraula x.

Observem que, la funció gn codifica a les paraules que pertanyen a Σ^*, és a dir que per a cada paraula li correspon exactament un nombre enter i que cap nombre correspon a més d'una paraula. Però la funció gn no correspont, tal i com l'hem definit, a una bijecció, ja que certs nombres, com per exemple el nombre β, no corresponen a cap paraula. Per remediar aquesta situació, introduirem un nou símbol, d_0, el qual correspondrà al dígit 0. D'aquesta manera, la funció gn és pot extendre de forma natural a una bijecció entre els conjunts $\{\lambda\} \cup \Sigma(\Sigma \cup \{d_0\})^*$ i $\mathbb{N}$. Sigui $D = \{\lambda\} \cup \Sigma(\Sigma \cup \{d_0\})^*$, és a dir el conjunt de totes les paraules a $(\Sigma \cup \{d_0\})^*$ que no comencen per d_0. Aleshores, gn^{-1} està ben definida entre els conjunts $\mathbb{N}$ i D.

Exemple 4.23. Sigui $\Sigma = \{a, b, c\}$. Llavors $\beta = 4$. Si es fixa la correspondència

$$d_1 = a,\ d_2 = b \text{ i } d_3 = c$$

aleshores, per exemple tenim que

$$gn(\lambda) = 0$$
$$gn(ba) = gn(d_2d_1) = 9$$
$$gn(ccc) = gn(d_3d_3d_3) = 63$$

Observem que el conjunt D conté, entre altres, les paraules λ, ba, ccc, bd_0a i cd_0d_0. Però les paraules d_0c, d_0 i d_0ad_0 no es troben dins de D.

Donats els nombres 1, 4 i 7, aleshores tenim que

$$gn^{-1}(1) = a$$
$$gn^{-1}(4) = ad_0$$
$$gn^{-1}(7) = ac$$

Anem ara a descriure algunes propietats dels nombres de Gödel amb el següent lema.

Lema 4.24. *a) La llargada de la paraula $x \in D$ és el nombre $l \in \mathbb{N}$ més petit tal que $\beta^l > gn(x)$.*

b) Si $x, y \in \Sigma^*$ i x és una subparaula de y, aleshores $gn(x) \leq gn(y)$.

Demostració. a) Si $x \in D$, aleshores x és de la forma $x = d_{i_1}...d_{i_l}$, on $0 \leq i_1, ..., i_l \leq \beta - 1$ i $i_1 > 0$. Aleshores

$$gn(x) = \beta^{l-1} \cdot i_1 + \beta^{l-2} \cdot i_2 + ... + \beta \cdot i_{l-1} + i_l$$

Ara només cal observar que aquest nombre és menor que β^l, ja que $i_1, ..., i_l < \beta$, i que és més gran o igual que β^{l-1}, ja que $i_1 \geq 1$.

b) Si $x = y$, clarament obtenim que $gn(x) = gn(y)$. I en el cas en que $x \neq y$, aleshores $|x| \leq |y| - 1$, però llavors, per l'apartat a),

$$gn(x) < \beta^{|x|} \leq \beta^{|y|-1} \leq gn(y)$$

arribant així al resultat que voliem demostrar. ■

A partir d'ara, fixem una gramàtica $G = (V, \Delta, P, S)$. També introduïm un nou símbol \$, el qual no pertany a V i definim $\Sigma = V \cup \{\$\}$. El símbol \$ serà utilitzat per codificar derivacions de G.

Ara introduirem una sèrie de funcions i predicats els quals demostrarem que són primitius recursius.

a) *Llargada*: La funció de 1 argument llargada és defineix com segueix. Per a tot $n \in \mathbb{N}$, $llargada(n) = |gn^{-1}(n)|$, és a dir que la funció dóna com a resultat la llargada de la paraula que tingui com a nombre de Gödel n. Pel lema 4.18, la funció llargada pot ser expresada per

$$llargada(n) = \mu i \leq n[n < \beta^i]$$

i, per tant, és recursiva primitiva. L'us de la minimalització acotada és apropiat en aquest cas ja que si $x \in$ D, aleshores $|x|$ mai serà superior a $gn(x)$.

b) *Concatenació*: Volem definir una funció de 2 arguments, *concat*, tal que

$$concat(m, n) = gn(gn^{-1}(m)gn^{-1}(n))$$

que expressa el nombre de Gödel de la concatenació de dues paraules amb nombres de Gödel m i n associats, respectivament. Aquest fet es pot aconseguir definint

$$concat(m, n) = n + m \cdot \beta^{llargada(n)}$$

i, per tant, la funció concat és recursiva primitiva. A més, podem generalitzar aquest concepte i definir, per a $k \geq 2$, la funció de k+1 arguments, $concat^{k+1}$, tal que, per a $\overline{n} \in \mathbb{N}^k$ i per a tot $m \in \mathbb{N}$,

$$concat^{k+1}(\overline{n}, m) = concat(concat^k(\overline{n}), m)$$

la qual és una funció primitiva recursiva.

Exemple 4.25. Sigui $\Delta = \{a, b, c\}$. Aleshores $\beta = 4$. Si fixem la correspondència

$$d_1 = a$$
$$d_2 = b$$
$$d_3 = c$$

llavors podem veure que, per exemple

$$concat(gn(b), gn(a)) = concat(2, 1) = 1 + 2 \cdot 4^{llargada(1)} = 1 + 2 \cdot 4^1 = 9 = gn(ba)$$

c) *Membre de* Σ^*: El predicat de 1 argument Σ^* és veritat si el seu argument és el nombre de Gödel d'una paraula de Σ^*, és a dir que no contingui el símbol especial d_0. És a dir,

$$\Sigma^* n \text{ si i només si } gn^{-1}(n) \in \Sigma^*$$

El predicat Σ^* és recursiu primitiu, ja que $\Sigma^* n$ si i només si

$$\neg\exists n_1 \geq n \exists n_2 \leq n[n = concat(n_1, n_2) \wedge n_1 \neq 0 \wedge \exists n_3 \leq n[n_1 = n_3 \cdot \beta]]$$

És a dir, $gn^{-1}(n)$ no pot ser expressat de la forma uv, on $u, v \in D$ i u acabi en d_0. Si u acaba en d_0, aleshores $gn(u)$ és diferent de zero i divisible per β.

d) *Membre de* V^*: El predicat de 1 argument V^* és verdader si el seu argument és el nombre de Gödel d'una paraula de V^*, és a dir que no contingui \$ ni el símbol especial d_0. Això és que,

$$V^* n \text{ si i només si } gn^{-1}(n) \in V^*$$

El predicat V^* és recursiu primitiu, ja que $V^* n$ si i només si

$$\Sigma^* n \wedge \forall n_1 \leq n[\forall n_2 \leq n[n \neq concat^3(n_1, gn(\$), n_2)]]$$

e) *Substitució*: Definim el predicat de 4 arguments SB de la següent manera.

$$SBn_1n_2n_3n_4 \text{ si i només si } w_1 = gn^{-1}(n_1),\ w_2 = gn^{-1}(n_2),\ w_3 = gn^{-1}(n_3) \text{ i}$$
$$w_4 = gn^{-1}(n_4) \text{ pertanyen a } \Sigma^* \text{ i existeixen paraules } u \text{ i } v, \text{ tenim que } w_1 = uw_3v \text{ i}$$
$$w_2 = uw_4v.$$

És a dir, $SBn_1n_2n_3n_4$ és verdader si i només si w_2 s'obté a partir de w_1 substituint w_4 per una ocurrencia de w_3. Aleshores, SB és un predicat recursiu primitiu, ja que $SBn_1n_2n_3n_4$ si i només si

$$\Sigma^* n_1 \wedge \Sigma^* n_2 \wedge \Sigma^* n_3 \wedge \Sigma^* n_4 \wedge (\exists m \leq n_1[\exists p \leq n_1[n_1 = concat^3(m, n_3, p) \text{ i}$$
$$n_2 = concat^3(m, n_4, p)]])$$

f) *Derivació en un pas*: El predicat de 2 arguments Y és defineix de la manera següent.

$$Ynm \text{ si i només si}$$
$$gn^{-1}(n), gn^{-1}(m) \in V^* \wedge gn^{-1}(n) \Rightarrow_G gn^{-1}(m)$$

És a dir, Ynm és verdader si la paraula de nombre de Gödel n es deriva en un pas a la paraula de nombre de Gödel m. Aleshores Y és un predicat recursiu primitiu, ja que, si $P = \{(u_1, v_1), ..., (u_k, v_k)\}$ són les produccions de la gramàtica G, aleshores Ynm es compleix si i només si

$$V^* n \wedge V^* m \wedge (SBnmgn(u_1)gn(v_1) \vee ... \vee SBnmgn(u_k)gn(v_k))$$

g) *Derivacions*: El predicat D es defineix de la forma següent. Dn si i només si $gn^{-1}(n)$ es una paraula de la forma $\$x_0\$x_1\$...\$x_k\$$, on $k \geq 0$, cada $x_i \in V^*$ i $x_i \Rightarrow_G x_{i+1}$ per $i = 1, ..., k-1$. És a dir, Dn és verdader si i només si n és la codificació d'una derivació en G. El caràcter \$ ha estat introduit únicament per el propòsit de separar les diferents paraules x_i unes respecte de les altres. D'aquesta manera, D és un predicat recursiu primitiu ja que Dn és verdader si i només si

$$\Sigma^* n \wedge \exists m \leq n[n = concat^3(gn(\$), m, gn(\$))] \wedge \forall p \leq n[\forall q \leq n[\forall r \leq n[\forall s \leq n[(n = concat^7(p, gn(\$), q, gn(\$), r, gn(\$), s) \wedge V^* q \wedge V^* r) \rightarrow Yqr]$$

h) *Començament d'una derivació*: El predicat de 2 arguments B està definit per, Bpq si i només si p codifica una derivació com en l'apartat anterior i $q = gn(x_0)$. Aleshores, B és un predicat recursiu primitiu ja que Bpq si i només si

$$Dp \wedge V^* q \wedge (\exists r \leq p[p = concat^4(gn(\$), q, gn(\$), r)])$$

i) *Finalització d'una derivació*: El predicat de 2 arguments E ve definit de la forma següent. Epq si i només si p codifica una derivació com en l'apartat g) i $q = gn(x_k)$. Aleshores, E és un predicat recursiu primitiu ja que Epq si i només si

$$Dp \wedge V^* q \wedge (\exists r \leq p[p = concat^4(r, gn(\$), q, gn(\$))])$$

4.3 Funcions μ-Recursives

En aquesta última secció introduïrem la noció de funció μ-recursiva i demostrarem el teorema que afirma que tota funció gramaticalment computable és també una funció μ-recursiva. El concepte de funció μ-recursiva és una extenció necessària de la noció de funció recursiva primitiva, ja que és possible trobar funcions computables que no són recursives primitives. L'exemple més conegut de funció computable però no recursiva primitiva és el de la funció d'Ackermann, que està definida de la forma següent:

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ i } n = 0 \\ A(m-1, A(m, n-1)) & \text{si } m > 0 \text{ i } n > 0 \end{cases}$$

A causa d'això, anem a definir el concepte de minimalització no acotada, el qual ens serà necessari per a la posterior definició de la noció de funció μ-recursiva.

Definició 4.26. *Sigui $k \geq 0$ i sigui g una funció de k+1 arguments. Llavors la minimalització no acotada de g és la funció de k arguments f tal que, per a tot $\overline{n} \in \mathbb{N}^k$, $f(\overline{n})$ és el mínim nombre m tal que $g(\overline{n}, m) = 0$, si aquest m existeix, i és 0 en cas contrari.*

Escriurem,

$$f(\overline{n}) = \mu m[g(\overline{n}, m) = 0]$$

i diem que f s'obté a partir de g per minimalització no acotada.

Finalment, passarem a donar les definicions de funció recursiva i la de funció μ-recursiva per, a continuació i per finalitzar aquest capítol, donar i demostrar el teorema que afirma que tota funció gramaticalment computable és μ-recursiva.

Definició 4.27. *Sigui $k \geq 0$. Una funció de k+1 arguments g és regular si, per a tot $\overline{n} \in \mathbb{N}^k$ existeix un $m \in \mathbb{N}$ tal que $g(\overline{n}, m) = 0$. Diem que una funció és μ-recursiva si pot ser obtinguda a partir de les funcions inicials ζ, π_l^k i σ usant les següents operacions: composició, recursió primitiva i l'aplicació de la minimalització no acotada a funcions regulars.*

Definició 4.28. *Una funció $f : \Sigma_0^* \to \Sigma_1^*$ és μ-recursiva si la funció $f' : \mathbb{N} \to \mathbb{N}$ és μ-recursiva, on per a tot $n \in \mathbb{N}$, $f'(n)$ és $gn(f(gn^{-1}(n)))$ si $gn^{-1}(n) \in \Sigma_0^*$ i zero en cas contrari.*

Teorema 4.29. *Tota funció gramaticalment computable és μ-recursiva.*

Demostració. Sigui $f : \Sigma_0^* \to \Sigma_1^*$ una funció gramaticalment computable i sigui $G = (V, \Sigma, P, S)$ una gramàtica que computa a f. Siguin x, y, x', y' les paraules associades a G i f tal i com s'especifica en la Definició 3.5. Per a la demostració del teorema, necessitarem definir tres predicats recursius primitius i una funció primitiva recursiva adicionals.

j) *Membre de Σ_0^*, membre de Σ_1^**: El predicat $\Sigma_0^* n$ és cert si i només si $gn^{-1}(n) \in \Sigma_0^*$. Anàlogament, el predicat $\Sigma_1^* m$ és cert si i només si $gn^{-1}(m) \in \Sigma_1^*$. Clarament, aquests dos predicats poden ser escrits a partir d'utilitzar els apartats c) i d) de la secció anterior, demostrant així que són recursius primitius. D'aquesta manera, si $\Sigma \setminus \Sigma_0 = \{a_1, ..., a_k\}$, aleshores $\Sigma_0^* n$ si i només si

$$\forall n_1 \leq n \forall n_2 \leq n[n \neq concat^3(n_1, gn(a_1), n_2) \wedge ... \wedge n \neq concat^3(n_1, gn(a_k), n_2)]$$

De forma anàloga, si $\Sigma \setminus \Sigma_1 = \{a_1, ..., a_l\}$, aleshores $\Sigma_1^* m$ si i només si

$$\forall m_1 \leq m \forall m_2 \leq m[m \neq concat^3(m_1, gn(a_1), m_2) \wedge ... \wedge m \neq concat^3(m_1, gn(a_l), m_2)]$$

k) *Extreure l'última paraula*: La funció *extreure* té com a argument el nombre de Gödel d'una derivació i dóna com a resultat l'última paraula de la derivació. Formalment

$$extreure(p) = \mu n \leq p[\mathrm{Ep}concat^3(gn(x'), n, gn(y'))]$$

l) *Computar*: Sigui C el predicat de 2 arguments recursiu primitiu següent. Cpn si i només si

$$\Sigma_0^* n \to [(Dp \wedge \mathrm{Bp}concat^3(gn(x), n, gn(y)))] \wedge \exists m \leq p[\Sigma_1^* m \wedge \mathrm{Ep}concat^3(gn(x'), m, gn(y'))]$$

Sigui n el nombre de Gödel d'una paraula $u \in \Sigma_0^*$. Aleshores Cnp ens indica que p codifica una derivació de la forma

$$xuy = x_0 \Rightarrow_G x_1 \Rightarrow_G ... \Rightarrow_G x_k = x'vy'$$

on $v \in \Sigma_1^*$. Llavors, tenim que $p = gn(\$x_0\$x_1...\$x_k\$)$.

Sigui γ la funció característica de C. Ja que C és un predicat recursiu primitiu, γ és recursiva primitiva. I, ja que la gramàtica G computa f, per a tota paraula $u \in \Sigma_0^*$, existeix una $v \in \Sigma_1^*$ tal que $xuy \Rightarrow_G^* x'vy'$ (és a dir que hi ha una v tal que $f(u) = v$). D'aquesta manera, per a cada n, existeix algun p tal que $\gamma(n, p) = 1$ (si $gn^{-1}(n) \notin \Sigma_0^*$, llavors $Cn0$ és verdader i $\gamma(n, 0) = 1$). Aleshores, sigui $\overline{\gamma}(n, p) = 1 \dot{-} \gamma(n, p)$ per a tot n i p. Llavors $\overline{\gamma}$ és una funció regular. Per tant, es pot aplicar la minimalització no acotada a $\overline{\gamma}$. Però, si

$$f'(n) = extreure(\mu p[\overline{\gamma}(n,p) = 0])$$

aleshores per a tot $u \in \Sigma_0^*$,

$$f'(gn(u)) = gn(f(u))$$

i per a tota n tal que $gn^{-1}(n) \notin \Sigma_0^*$, $f'(n) = 0$. Llavors, f' està associada amb f tal i com està estipulat en la Definició 4.28 i f' és μ-recursiva. Per tant, f és μ-recursiva. ■

Avans de finalitzar amb aquest capítol, observem que el teorema anterior l'hem demostrat per a funcions les quals tant el seu domini com el seu recorregut són paraules en un alfabet donat. Per veure que les funcions gramaticalment computables les quals el seu domini i recorregut són els nombres naturals també són μ-recursives, simplement em de considerar l'alfabet constituit pel codi unari $\Sigma_0 = \Sigma_1 = \{I\}$

5 Programes de Registres

En aquesta secció veurem la definició de programa de registres, demostrarem que tota funció μ-recursiva és computable per un programa de registres i que tota funció computable per un programa de registres es pot computar per una màquina de Turing.

La idea en la que es basa el concepte de màquines de registre és la de desenvolupar una definició formal d'algorisme el qual pot ser programat per un llenguatge de programació. Per dur a terme aquest propòsit, definirem procediments en el sentit formal per a que aquests puguin ser programats.

Definició 5.1. *Sigui $\Sigma = \{I\}$ l'alfabet unari. Els programes són executats per màquines amb una memòria compartimentada en unitats $R_1, ..., R_m, ...$, anomenades registres. Assumirem que tenim un nombre infinit de registres i que cada registre conté en cada pas de còmput una paraula sobre l'alfabet $\Sigma = \{I\}$.*
A aquestes màquines se les anomena màquines de registres.

Un programa escrit en el llenguatge $\Sigma = \{I\}$ consisteix en instruccions on cada instrucció comença amb un nombre natural L, el qual anomenarem l'etiqueta de la instrucció. A més, únicament estaran permesos els següents tipus d'instruccions:

1) L R_i^+
per a $L, i \in \mathbb{N}$.
Aquesta instrucció s'interpreta com que la màquina afegeix un símbol I a la paraula del registre R_i.

2) L R_i^-
per a $L, i \in \mathbb{N}$.
Aquesta instrucció s'interpreta com que la màquina treu un símbol I de la paraula del registre R_i, sempre i quan aquest no estigui buit. Si el registre R_i està buit, aleshores no s'executa la instrucció.

3) L If $R_i = \square$ Then L' Else L"
per $L, L', L'', i \in \mathbb{N}$.
Aquesta instrucció s'interpreta com que si el registre R_i està buit de contingut (aquest fet es representa amb el símbol $\square$), aleshores s'executa la instrucció etiquetada per L'. Si, en cas contrari, el registre R_i no està buit, aleshores s'executa la instrucció associada a l'etiqueta L".

4) L Go To L'
per $L, L' \in \mathbb{N}$.
Aquesta instrucció s'interpreta com que el programa es dirigeix a la instrucció etiquetada per L'.

5) L Stop
per $L \in \mathbb{N}$.
Aquesta instrucció s'interpreta com que el procés s'atura.

A continuació passarem a definir de manera formal quin és el concepte de programa de registres o, simplement, programa.

Definició 5.2. *Un programa de registres P (o simplement programa) és un seqüència finita $\alpha_0, ..., \alpha_k$ d'instruccions de la forma 1), 2), 3), 4) o 5) de manera que únicament una instrucció és del tipus 5).*

Cada programa P dóna lloc a una forma de procedir: suposem una màquina de registres amb un nombre infinit d'aquests i que s'ha programat amb les regles dictaminades per P. A l'inici del càlcul es té un nombre finit de registres $R_{i_0}, ..., R_{i_k}$ (on $i_0, ..., i_k \leq 1$) cada un dels quals conté una paraula d'entrada. Els còmputs es desenvolupen de manera gradual, i cada pas es correspon amb l'execució d'una instrucció dictaminada per P. Comença amb la primera instrucció i continua línea per línea seguint amb els dictàmens marcats pel programa. I finalment, la màquina s'atura quan arriba a una instrucció Stop.

Tot seguit anem a donar la definició de funció computable per una màquina de registres.

Definició 5.3. *Sigui $f : (\Sigma^*)^n \to \Sigma^*$ una funció de n arguments. Sigui P un programa de registres.*

a) Es diu que el programa P computa f si per a $(x_1, ..., x_n) \in (\Sigma^*)^n$, si $f(x_1, ..., x_n) = y$, aleshores si per a tot $j = 1, ..., n$ el programa P comença el còmput amb x_j en el registre R_j, llavors el programa P s'atura amb y en el registre n+1 i els altres registres contenen les paraules que contenien en l'inici del càlcul.

b) Diem que f és R-computable si existeix un programa de registres que computa f.

A continuació, anem a veure quatre exemples de programes alguns dels quals usarem posteriorment en les demostracions dels dos teoremes que donarem en aquest capítol.

Exemple 5.4. i) Sigui $i \geq 1$. El programa que ve a continuació buida el registre R_i.

1. If $R_i = \square$ Then 4 Else 2
2. R_i^-
3. Go To 1
4. Stop

Aquest programa l'esciurem mitjançant la notació R_i^*.

ii) Siguin $i, j \geq 0$, on $i \neq j$. El programa següent trasllada el contingut del registre R_i al registre R_j i s'atura.

1. R_j^*
2. If $R_i = \square$ Then 6 Else 3
3. R_i^-
4. R_j^+
5. Go To 1
6. Stop

Aquest programa l'esciurem mitjançant la notació $R_i \Rightarrow R_j$.

iii) Siguin $i, j \geq 0$, on $i \neq j$. El programa següent suma el nombre contingut en el registre R_j amb el nombre contingut en el registre R_i, guardant-ne el resultat en el registre R_i i s'atura. En aquest programa s'utilitza un registre auxiliar R_s diferent dels registres R_i i R_j.

1. R_s^*
2. $R_j \Rightarrow R_s$
3. If $R_s = \square$ Then 8 Else 4
4. R_s^-
5. R_i^+
6. R_j^+
7. Go To 3
8. Stop

Aquest programa l'esciurem mitjançant la notació $R_i = R_i + R_j$.

iv) El programa que ve a continuació realitza el producte dels nombres continguts en els registres R_1 i R_2, guardant-ne el resultat en el registre R_3 i s'atura. En aquest programa s'utilitza un registre auxiliar R_l diferent dels registres R_1, R_2 i R_3.

1. R_3^*
2. If $R_2 = \square$ Then 10 Else 3
3. $R_1 \Rightarrow R_l$
4. If $R_l = \square$ Then 8 Else 5
5. R_l^-
6. $R_1 = R_1 + R_2$
7. Go To 4
8. $R_1 \Rightarrow R_3$
9. $R_l \Rightarrow R_1$
10. Stop

De forma anàloga a com em dissenyat el programa que computa el producte, es possible construir un programa que calculi la potècia.

Seguidament, anem a demostrar el següent teorema.

Teorema 5.5. *Tota funció μ-recursiva és computable per un programa de registres.*

Demostració. Demostrerm per inducció que les funcions i els processos que composen la definició de funció μ-recursiva poden ser computables a per programes de registres.

Anem primerament a veure que les funcions inicials ζ, π_i^k i σ poden ser computades per un programa.

a) *Funció zero* ζ: El programa següent computa la funció ζ:

1. R_1^*

b) *Funció i-èsima de k arguments projecció* π_i^k: Siguin $x = (x_1, ..., x_k) \in (\Sigma^*)^k$. Per a cada $1 \leq j \leq k$, la paraula x_j es troba dins del registre R_j. Aleshores $\pi_i^k(x)$ ve definida pel programa següent:

1. R_{k+1}^*
2. $R_{k+1} = R_{k+1} + R_i$

c) *Funció successor* σ: Sigui $x \in \Sigma^*$ i suposem que el contingut del registre R_1 és la paraula x. Aleshores $\sigma(x)$ ve definida pel programa següent:

1. R_2^*
2. $R_2 = R_2 + R_1$
3. R_2^+

A continuació veurem com els processos de composició, recursió primitiva i minimalització no acotada de funcions són preservats per programes de registres.

Composició: Anem primerament a veure un cas particular de composició de funcions per, posteriorment, extendre-ho al cas general. Siguin g_1 i g_2 dues funcions R-computables de 3 arguments i sigui f una funció R-computable de 2 arguments. L'objectiu és provar que la funció h de 3 arguments tal que per a tot $(x_1, x_2, x_3) \in (\Sigma^*)^3$,

$$h(x_1, x_2, x_3) = f(g_1(x_1, x_2, x_3), g_2(x_1, x_2, x_3))$$

és una funció R-computable.

Per fer-ho, considerem els registres R_1, R_2, R_3, R_4, R_{p_1}, R_{p_2}, R_{q_1}, R_{q_2}, R_{q_3} d'una màquina de registres. Siguin $x_1, x_2, x_3 \in \Sigma^*$ les paraules d'entrada. Aplicant la hipòtesis inductiva tenim G_1, G_2 i F programes que computen les funcions g_1, g_2 i f, respectivament. En un inici, per a $j = 1, 2, 3$, la paraula x_j és troba en el regisre R_j. Considerem el programa següent el qual computa la funció h:

1. G_1	10. $R_{p_2} \Rightarrow R_2$
2. $R_4 \Rightarrow R_{p_1}$	11. F
3. G_2	12. $R_3 \Rightarrow R_4$
4. $R_4 \Rightarrow R_{p_2}$	13. R_1^*
5. $R_1 \Rightarrow R_{q_1}$	14. R_2^*
6. $R_2 \Rightarrow R_{q_2}$	15. $R_{q_1} \Rightarrow R_1$
7. $R_3 \Rightarrow R_{q_3}$	16. $R_{q_2} \Rightarrow R_2$

D'aquesta manera veiem com la funció h és R-computable.

Després d'haver vist aquest cas particular, anem a veure l'extenció d'aquest resultat a la composició de funcions de forma més genèrica. Siguin $l > 0$ i $k \geq 0$, g una funció R-computable de l arguments i $h_1, ..., h_l$ funcions R-computables de k arguments. La nostra meta és demostrar que la funció f de k arguments tal que per a tot $(x_1, ..., x_k) \in (\Sigma^*)^k$,

$$f(x_1, ..., x_k) = g(h_1(x_1, ..., x_k), ..., h_l(x_1, ..., x_k))$$

és una funció R-computable.

De forma anàloga a com hem procedit en el cas particular, considerem els registres $R_1, ..., R_k, R_{k+1}, R_{p_1}, ..., R_{p_l}, R_{q_1}, ..., R_{q_k}$ d'una màquina de registres, diferents dos a dos (en cas de que $l > k$, aleshores també tindrem en consideració els registres $R_{k+2}, ..., R_{l+1}$). Siguin $x_1, ..., x_k \in \Sigma^*$ les paraules d'entrada. Aplicant la hipòtesis inductiva, existeixen programes $H_1, ..., H_l$ que computen les funcions $h_1, ..., h_l$, respectivament i novament aplicant la hipòtesis inductiva es té G el programa que computa la funció g. En un inici tindrem que per a $j = 1, ..., k$, la paraula x_j es troba en el registre R_j. Aleshores el programa següent computa la funció f:

1. H_1	3l+k. $R_{p_l} \Rightarrow R_l$
2. $R_{k+1} \Rightarrow R_{p_1}$	3l+k+1. G
...	3l+k+2. $R_{l+1} \Rightarrow R_{k+1}$
2l-1. H_l	3l+k+3. R_1^*
2l. $R_{k+1} \Rightarrow R_{p_l}$	...
2l+1. $R_1 \Rightarrow R_{q_1}$	3l+2k+2. R_k^*
...	3l+2k+3. $R_{q_1} \Rightarrow R_1$
2l+k. $R_k \Rightarrow R_{q_k}$	...
2l+k+1. $R_{p_1} \Rightarrow R_1$	3l+3k+2. $R_{q_k} \Rightarrow R_k$
...	

D'aquesta forma demostrem com la funció f és R-computable i que, per tant, el procés de composició també ho és.

Recursió primitiva: Previament a la demostració del cas general, i de manera il·lustrativa, demostrarem un cas particular de com una funció generada a partir de recursió primitiva és R-computable. Sigui f una funció R-computable de 1 argument i sigui g una funció R-computable de 3 arguments. L'objectiu és provar que la funció h de 2 arguments tal que per a tot $(x_1, x_2) \in (\Sigma^*)^2$,

$$h(x_1, 0) = f(x_1)$$
$$h(x_1, x_2 + 1) = g(x_1, x_2, h(x_1, x_2))$$

és una funció R-computable.

Perquè aquest fet sigui possible, considerem els registres R_1, R_2, R_3, R_4, R_p d'una màquina de registres. Siguin $x_1, x_2 \in \Sigma^*$ les paraules d'entrada. Per la hipòtesis inductiva existeixen programes F i G que computen les funcions f i g, respectivament. En l'inici del còmput, per a $j = 1, 2$, la paraula x_j es troba en el registre R_j. Aleshores, el programa següent computa la funció h:

1. R_3^*	7. G
2. $R_2 \Rightarrow R_p$	8. R_2^+
3. F	9. R_3^*
4. $R_2 \Rightarrow R_3$	10. $R_4 \Rightarrow R_3$
5. If $R_p = \square$ Then 12 Else 6	11. Go To 5
6. R_p^-	12. Stop

A continuació, veurem l'extenció d'aquest cas particular al cas general. Sigui $k \geq 0$. Sigui g una funció R-computable de k arguments i sigui h una funció R-computable de k+2 arguments. La nostra voluntat és demostrar que la funció f de k+1 arguments tal que per a tot $(x_1, ..., x_k) \in (\Sigma^*)^k$ i $x \in \Sigma^*$,

$$f(x_1, ..., x_k, 0) = g(x_1, ..., x_k)$$
$$f(x_1, ..., x_k, x + 1) = h(x_1, ..., x_k, x, f(x_1, ..., x_k, x))$$

és una funció R-computable.

Anàlogament al procediment realitzat en el cas particular, considerem els registres $R_1, ..., R_k$, R_{k+1}, R_{k+2}, R_{k+3}, R_p d'una màquina de registres. Per la hipòtesis inductiva tenim G i H programes que computen a les funcions g i h, respectivament. Siguin $x_1, ..., x_k, x \in \Sigma^*$ les paraules d'entrada. En un inici, per a $j = 1, ..., k$, la paraula x_j es troba en el registre R_j, la paraula x es troba en el registre R_{k+1}. Aleshores, el programa que es troba a continuació computa la funció f:

1. R_{k+3}^*	7. H
2. $R_{k+1} \Rightarrow R_p$	8. R_{k+1}^+
3. G	9. R_{k+2}^*
4. $R_{k+1} \Rightarrow R_{k+2}$	10. $R_{k+3} \Rightarrow R_{k+2}$
5. If $R_p = \square$ Then 12 Else 6	11. Go To 5
6. R_p^-	12. Stop

D'aquesta forma podem concloure que f és una funció R-computable i que, per tant, el procés de recursió primitiva també ho és.

Minimalització no acotada: Anem primerament, com en els casos anteriors, a demostrar un cas particular de minimalització no acotada d'una funció. Sigui f una funció R-computable de 2 arguments. L'objectiu és demostrar que la funció g de 1 argument tal que si $x_1 \in \Sigma^*$, aleshores

$$g(x_1) = \begin{cases} m & \text{on m es el primer nombre tal que f}(x_1, m) = 0; \text{ si un tal m existeix} \\ 0 & \text{altrament} \end{cases}$$

és una funció R-computable.

Per demostrar-ho, considerem els registres R_1, R_2, R_3, R_p d'una màquina de registres. Sigui $x_1 \in \Sigma^*$ la paraula d'entrada. Aplicant la hipòtesis inductiva tenim F un programa

que computa la funció f. En un inici, la paraula x_1 es troba en el registre R_1. El programa següent computa la funció g:

1. R_2^*	6. R_3^*
2. $R_3 \Rightarrow R_p$	7. R_2^+
3. R_3^*	8. Go To 5
4. F	9. $R_p \Rightarrow R_3$
5. If $R_3 = \square$ Then 10 Else 6	10. Stop

D'aquesta manera acabem concloent que g és R-computable.

Anem doncs ara a demostrar el cas general d'aquest fet. Sigui $k > 0$ i sigui f una funció R-computable de k+1. Vegem que la funció h de k arguments i que s'obté a partir de f per minimalització no acotada és R-computable. Per fer-ho, considerem els registres $R_1, ..., R_{k+1}, R_{k+2}, R_p$ d'una màquina de registres. Siguin $x_1, ..., x_k \in \Sigma^*$ les paraules d'entrada. Per hipòtesis inductiva existeix un programa F que computa la funció f. En un inici tindrem que per a cada $1 \leq j \leq k$, la paraula x_j es troba en el registre R_j. El programa següent ens mostra aquest resultat:

1. R_{k+1}^*	6. R_{k+2}^*
2. $R_{k+2} \Rightarrow R_p$	7. R_{k+1}^+
3. R_{k+2}^*	8. Go To 5
4. F	9. $R_p \Rightarrow R_{k+2}$
5. If $R_{k+2} = \square$ Then 10 Else 6	10. Stop

Amb aquest resultat es conclou que h és una funció R-computable i, per tant, el procés de minimalització no acotada també ho és.

Amb aquesta última prova acabem la demostració de que tota funció μ-recursiva és R-computable. ■

Cal observar que la definició de màquina de registres que hem donat i, per conseqüent, la definició de programa, es poden extendre a un alfabet qualsevol $\Sigma = \{a_1, ..., a_n\}$ mitjançant una extenció de les instruccions, donant com a resultat les regles següent:

1) L $R_i = R_i + a_j$
on $L, i, j \in \mathbb{N}$ amb $j \leq n$.
Aquesta instrucció s'interpreta com que la màquina afegeix el símbol a_j a la paraula del registre R_i.

2) L $R_i = R_i - a_j$
on $L, i, j \in \mathbb{N}$ amb $j \leq n$.
Aquesta instrucció s'interpreta com que si la paraula del registre R_i acaba amb el símbol a_j, aleshores aquest és borrat. En cas contrari, la paraula es maté intacta.

3) L If $R_i = \square$ Then L' Else L_0 o bé ... o bé L_n
on $L, L', L_0, ..., L_n \in \mathbb{N}$.
Aquesta instrucció s'interpreta com que si el registre R_i està buit, aleshores el programa va a la instrucció etiquetada per L'. I si, en cas contrari, la paraula del registre R_i acaba amb el símbol a_j, aleshores el programa va a la instrucció etiquetada per L_j, on $0 \leq j \leq n$.

4) L Go To L'
per $L, L' \in \mathbb{N}$.

Aquesta instrucció s'interpreta com que el programa es dirigeix a la instrucció etiquetada per L'.

5) L Stop
per $L \in \mathbb{N}$. Aquesta instrucció s'interpreta com que el procés s'atura.

A més, també és possible obtenir una extenció a qualsevol alfabet $\Sigma = \{a_1, ..., a_k\}$ de la definició de funció computable per un programa registres i, per tant, obtenint així també una extenció del teorema anterior.

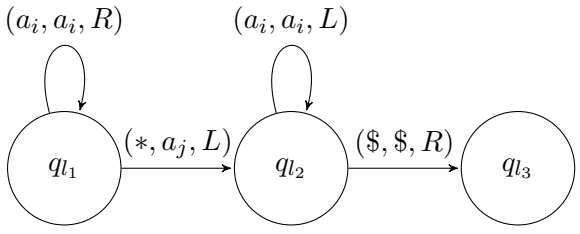
Finalment demostrarem el teorema següent.

Teorema 5.6. *Tota funció computable per un programa de registres és computable per una màquina de Turing.*

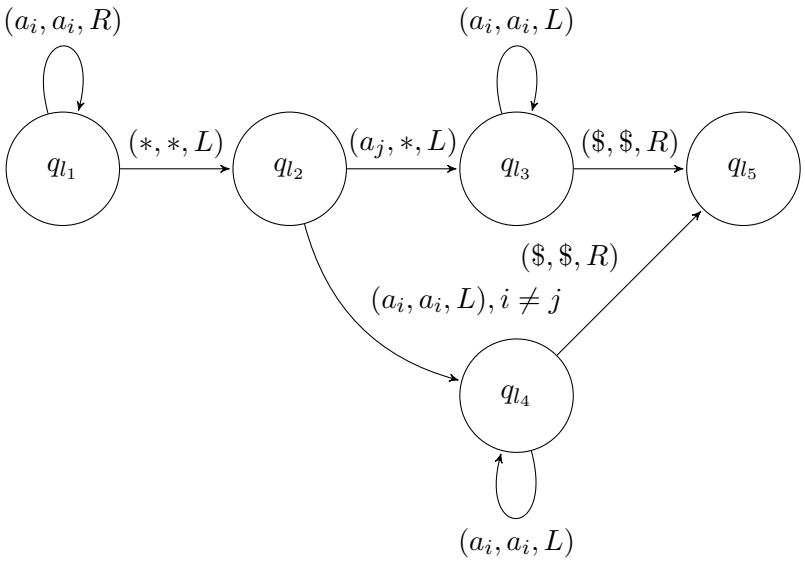
Demostració. Sigui P un programa de registres. Siguin $R_{i_1}, ..., R_{i_n}$ els registres utilitzats per P. Aleshores anem a considerar la següent màquina de Turing de n cintes en la qual per a tot $k = 1, ..., n$, la cinta k representa el registre R_{i_k}. Per tant, el contingut del registre R_{i_k} és el contingut de la cinta k.

Sigui $\Sigma = \{a_1, ..., a_k\}$ l'alfabet del programa de registres. Siguin $x_1, ..., x_n \in \Sigma^*$ els continguts dels registres on per a tot $j = 1, ..., n$, x_j és el contingut del registre R_{i_j}. Aleshores la màquina de Turing següent simula el programa P. Sigui $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una màquina de Turing de n cintes. Per a cada línea l del programa, M té una llista d'estast $q_{l_1}, ..., q_{l_s}$ per simular la instrucció d'aquesta línea on l'estat inicial q_0 coincideix amb el primer estat associat a la primera instrucció, és a dir $q_0 = q_{1_1}$. D'aquesta forma distingirem cinc casos diferents els quals coincideixen amb els cinc tipus diferents d'instruccions que un programa pot executar.

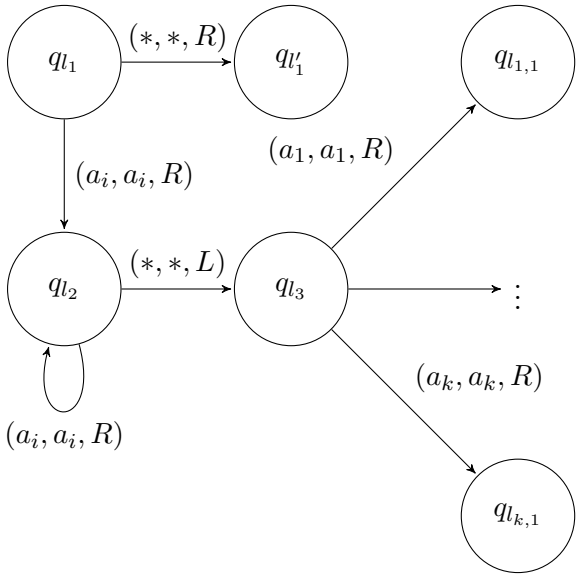
Si en la línea l es troba una instrucció del tipus 1, aleshores M procedeix com segueix: En la cinta i-èsima, la màquina recorre tota la paraula fins al símbol final d'aquesta, substitueix el primer símbol blanc, $*$, que es troba a continuació d'aquest últim caràcter pel símbol a_j i torna a l'inici de la paraula perquè el punter apunti a la segona cel·la de la cinta. A continuació es troba una representació en forma de graf del càlcul que realitza la màquina de Turing en la cinta i-èsima quan en el programa es troba una instrucció del tipus 1.



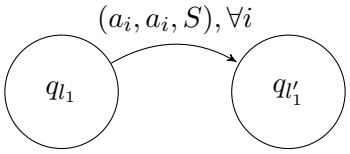
Si en la línea l es troba una instrucció del tipus 2, aleshores M procedeix com segueix: En la cinta i-èsima, la màquina recorre tota la paraula fins a l'últim símbol d'aquesta, en cas de que aquest sigui a_j el canvia pel caràcter blanc $*$, i en cas de que sigui un símbol diferent a a_j, el deixa intacte. Finalment, recorre la paraula de forma inversa fins a que el punter de la cinta torni a apuntar a la segona cel·la d'aquesta. A continuació es troba una representació en forma de graf del càlcul que realitza la màquina de Turing en la cinta i-èsima quan en el programa es troba una instrucció del tipus 2.



Si en la línea l es troba una instrucció del tipus 3, aleshores M procedeix com segueix: La màquina comprova si la cinta i-èsima és buida. En cas de ser així, aleshores la màquina va a l'estat $q_{l'_1}$ (el primer estat associat a la línea del programa etiquetada per L'). En cas contrari, analitza quin és l'últim símbol de la paraula continguda en aquesta cinta, recorrent-la fins a arribar a aquest caràcter (suposem que sigui a_j) i, a continuació M va a l'estat $q_{l_{j,1}}$ (el primer estat associat a la línea del programa etiquetada per L_j). A continuació trobem una representació en forma de graf del càlcul que realitza la màquina de Turing en la cinta i-èsima quan en el programa es troba una instrucció del tipus 3.



Si en la línea l es troba una instrucció del tipus 4, aleshores M procedeix anant a l'estat $q_{l'_1}$ (el primer estat associat a la línea del programa etiquetada per L'). A continuació trobem una representació en forma de graf del càlcul que realitza la màquina de Turing en la cinta i-èsima quan en el programa es troba una instrucció del tipus 4.



Si en la línea l es troba la instrucció del tipus 5, aleshores M procedeix anant a l'estat q_F (l'estat acceptador) que és l'únic estat associat a la instrucció d'aquest tipus.

D'aquesta manera ha quedat demostrat que tots els procediments associats a un programa de registres són computables per una màquina de Turing multicinta i que, per tant, són també computables per una màquina de Turing determinista. ■

Observem que amb aquest últim teorema acabem amb el raonament que avala la veracitat de la tesis de Church-Turing, ja que aquest evidencia l'equivalència entre els quatre conceptes que són acceptats com a definicions de la noció intuïtiva d'algorisme.

6 Conclusions

Aquesta memòria mostra els arguments principals per avalar la certesa de la tesis de Church-Turing, la qual és de gran utilitat en teoria de la computació ja que permet un us formal de la noció d'algorisme i per tant, ens proporciona el fet de poder-li donar un tracte matemàtic. Les formulacions matemàtiques del concepte d'algorisme que em vist en aquesta memòria són els que es plantegen amb més freqüència, però existeixen altres nocions formals associades a la noció d'algorisme que no han estat donades en aquest treball i que també són equivalents al model de màquina de Turing.

Tot i això, al ser la tesis de Church-Turing un enunciat el qual no és possible demostrar, qualsevol raonament que es doni avalant-ne la seva certesa mai serà suficient, ja que cap la possibilitat de que en un futur es trobi un algoritme que no pugui ser descrit mitjançant una màquina de Turing, demostrant així la invalidesa de la tesis. També hi hauria la possibilitat de que es trobés una definició formal de la noció d'algorisme que no fos equivalent a la noció de màquina de Turing, fet que també invalidaria la tesis. Tot i això, cal dir que aquestes són unes possibilitats remotes i que dins de la comunitat matemàtica no es considera que aquests esdeveniments puguin succeir.

Referències

[1] Harry R. Lewis; Christos H. Papadimitriou: *Elements of the theory of computation*, 1a edició, Englewood Cliffs (N.J.) : Prentice-Hall, 1981

[2] Maria Serna; Carme Àlvarez; Rafel Cases; Antoni Lozano: *Els límits de la computació: indecidibilitat i NP-completesa*, Barcelona : Edicions UPC, 2001

[3] George S. Boolos; Richard C. Jeffrey: *Computability and logic*, Cambridge [etc.] : Cambridge University Press, 1989

[4] José F. Prida: *Teorías inseparables*, Madrid : Trotta, 2004

[5] H.-D. Ebbinghaus; J. Flum; W. Thomas: *Mathematical logic*, New York [etc.] : Springer, 1984

[6] Michael Sipser: *Introduction to the theory of computation*, Boston (Mass.) : PWS Publishing Company, 1997

[7] A. M. Turing: *On Computable Numbers, with an Application to the Entscheidungsproblem*, `Proceedings of the London Mathematical Society`. 2 42. pp. 230-265, 1937