Calculabilitate si Complexitate*

Mihai Prunescu†

Contents

*Cea mai mare parte a textului este tradusa din lucrarea lui Uwe Schöning, Theoretische Informatik kurzgefaßt, Spektrum Akademischer Verlag, 1999. Alte pasaje sunt preluate din surse semnalate la paginile respective.

†University of Bucharest, Faculty of Mathematics and Informatics; and Simion Stoilow Institute of Mathematics of the Romanian Academy, Research unit 5, P. O. Box 1-764, RO-014700 Bucharest, Romania.
mihai.prunescu@imar.ro, mihai.prunescu@gmail.com

Part I
Gramatici

1 Ierarhia lui Chomsky

Conceptul formal de gramatica a fost definit de catre Noam Chomsky, un pionier al teoriei limbajelor si al lingvisticii.

Definitie: O gramatica este un tuplu $G = (V, \Sigma, P, S)$ dupa cum urmeaza:

V este o multime finita. Elementele ei se numesc variabile sau simboluri neterminale, si se noteaza cu majuscule.

Σ este o multime finita. Elementele ei se numesc simboluri terminale si se noteaza cu litere mici. Multimile V si Σ sunt disjuncte: $V \cap \Sigma = \emptyset$.

$P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ este multimea regulilor sau a productiilor.

$S \in V$ este simbolul de start.

□

Definitie: Pentru cuvinte $u, v \in (V \cup \Sigma)^*$ definim relatia $u \Rightarrow_G v$. Spunem ca $u \Rightarrow_G v$ daca exista cuvinte $x, z \in (V \cup \Sigma)^*$ si o regula $(y \to y') \in P$ astfel incat $u = xyz$ si $v = xy'z$. □

Definitie: Limbajul generat de G este multimea:

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

Relatia $\Rightarrow_G^*$ este inchiderea reflexiva si tranzitiva a relatiei $\Rightarrow_G$. Un sir de cuvinte $(w_0, w_1, \dots, w_n)$ cu $w_0 = S$, $w_n \in \Sigma^*$ si $w_0 \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_n$ se numeste derivare in G. Observam ca derivarea intr-o gramatica este un proces nedeterminist. □

Definitie: Ierarhia lui Chomsky.

- Orice gramatica este de tip 0.

- O gramatica de tip 0 este de tip 1 (dependenta de context, context-senzitiva) daca pentru orice regula $w_1 \to w_2$ din P este adevarat ca $|w_1| \leq |w_2|$.

- O gramatica de tip 1 este de tip 2 (independenta de context, libera de context) daca pentru orice regula $w_1 \to w_2$ din P este adevarat ca $|w_1| = 1$ si $w_1 \in V$.

- O gramatica de tip 2 este de tip 3 (regulata) daca pentru orice regula $w_1 \to w_2$ din P este adevarat ca $w_2 \in \Sigma \cup \Sigma V$.

□

Definitie: Un limbaj $A \subseteq \Sigma^*$ este de tip t daca exista o gramatica de tip t care il genereaza. Intotdeauna suntem interesati de tipul maximal al unui limbaj. □

Observam ca o gramatica de tip $t \geq 1$ nu poate genera cuvantul vid ε. In unele limbaje insa, cuvantul vid apare in mod natural. De aceea pentru cuvantul vid se introduce o regula speciala $S \to \varepsilon$. Pentru a pastra neschimbat limbajul $L(G)$ se introduce o noua variabila S' iar regulile se modifica in modul urmator:

$S \to$ membrul drept al regulii, cu S inlocuit cu S',

toate regulile cu S inlocuit cu S',

plus regula $S \to \varepsilon$.

In cursul de Limbaje Formale s-a vazut ca limbajele generate de gramatici de tip 3 (regulate) sunt exact cele recunoscute de automate deterministe finite. De asemeni s-a vazut ca limbajele generate de gramatici de tip 2 (independente de context) sunt cele recunoscute de automatele nedeterministe cu stiva. Un scop al acestui curs este introducerea conceptului de calculabilitate. Vom vedea ca o posibila definitie a acestui concept este legata de masinile Turing. De asemeni, vom vedea ca limbajele de tip 0 sunt cele recunoscute de masini Turing deterministe, iar limbajele de tip 1 sunt cele recunoscute de masini Turing nedeterministe liniar marginite. O alta aplicatie interesanta a acestor dezvoltari este Problema Cuvantului.

Definitie: Fie $L \subseteq \Sigma^*$ un limbaj. Problema Cuvantului pentru L este urmatoarea: dat un cuvant $x \in \Sigma^*$, sa se decida daca $x \in L$. □

Teorema: *Problema Cuvantului pentru limbaje de tip* 1 *este decidabila. Cu alte cuvinte, exista un algoritm determinist care primeste o gramatica dependenta de context* $G = (V, \Sigma, P, S)$ *si un cuvant* $x \in \Sigma^*$*, si care decide in timp finit daca* $x \in L(G)$ *sau daca* $x \notin L(G)$*.*

Demonstratie: Pentru $m, n \in \mathbb{N}$ definim multimea:

$T^n_m = \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ si } w \text{ se deriva din } S \text{ in cel mult } m \text{ pasi}\}$.

Mai exact:

$T^n_0 = \{S\}$,

$T^n_{m+1} = T^n_m \cup \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ si } w' \Rightarrow w \text{ pentru un } w' \in T^n_m\}$.

Aceasta reprezentare este corecta numai pentru gramaticile de tip 1, deoarece la gramaticile de tip 0 s-ar putea ca dintr-un cuvant de lungime $> n$ sa rezulte un cuvant de lungime $\leq n$.

Observam ca pentru orice m si n, $T^n_m \subseteq T^n_{m+1}$. Cum exista doar un numar finit de cuvinte de lungime $\leq n$ in $(V \cup \Sigma)^*$ rezulta ca pentru orice n exista un m astfel incat:

$$T^n_m = T^n_{m+1} = T^n_{m+2} = \dots$$

Algoritmul este deci urmatorul. Fie n lungimea lui x. Se calculeaza sirul de multimi (T^n_m) pana cand acest sir devine stationar. Daca la un pas am intalnit cuvantul x, raspunsul este da. Altfel raspunsul este nu. □

In completare:

Teorema[1]: *Exista limbaje decidabile care nu sunt de tip* 1*.*

Demonstratie: Fixam un alfabet terminal Σ cu $|\Sigma| \geq 2$ si consideram toate gramaticile de tip 1 care au Σ ca limbaj terminal. Aceste gramatici pot fi codificate in cuvinte. Cum Σ si S sunt simboluri comune, iar dintre simbolurile neterminale, numai cele care apar in P sunt relevante, cuvantul care codifica o gramatica de tip 1 poate fi pur si simplu lista de reguli de productie P. Aceste coduri se ordoneaza dupa lungime, si la lungime egala, lexicografic, astfel incat toate gramaticile de tip 1 formeaza un sir $G_1, G_2, \dots, G_n, \dots$. Fie limbajul:

$$L = \{w \in \Sigma^* \mid w \notin L(G_{|w|})\}.$$

L este decidabil. Pentru un cuvant x, se gaseste gramatica $G = G_{|x|}$. Se decide daca G produce x. Se pune $x \in L$ daca si numai daca G nu produce x.

L nu este de tip 1. Daca L ar fi de tip 1, ar fi produs de o anumita gramatica de tip 1, fie ea G_m. Dar G_m produce alte cuvinte de lungime m decat cele din L. Contradictie.

□

Ca urmare a caracterizarii limbajelor de tip 0 cu ajutorul masinilor Turing, vom vedea ca, in contrast, Problema Cuvantului pentru limbajele de tip 0 este nedecidabila.

Definitie: O gramatica de tip 1 este in Forma Normala Kuroda daca fiecare regula are una din formele urmatoare:

$$A \to a, \quad A \to B, \quad A \to BC, \quad AB \to CD.$$

Literele mari semnifica variabile, iar litera a semnifica diferite constante.

Teorema: *Pentru orice gramatica* G *de tip* 1 *cu* $\varepsilon \notin L(G)$*, exista o gramatica* G' *in Forma Normala Kuroda cu* $L(G) = L(G')$*.*

Demonstratie: Pentru fiecare simbol terminal a introducem o noua variabila A si regula $A \to a$. Fiecare regula de forma $A \to B_1 B_2 \dots B_k$ se poate inlocui cu un set de reguli de forma $A \to BC$.

[1] Folclor...

Raman regulile de forma $A_1 \dots A_m \to B_1 \dots B_n$ unde $2 \le m \le n$. O asemenea regula se inlocuieste cu urmatoarea multime de reguli:

$$\begin{array}{cc} A_1A_2 \to B_1C_2, & C_m \to B_mC_{m+1} \\ C_2A_3 \to B_2C_3, & C_{m+1} \to B_{m+1}C_{m+2} \\ \vdots & \vdots \\ C_{m-1}A_m \to B_{m-1}C_m, & C_{n-1} \to B_{n-1}C_n \end{array}$$

unde $C_2, \dots, C_{n-1}$ sunt variabile noi.

2 Masini Turing

Masina Turing este un model de calcul mai puternic decat automatul finit sau automatul finit cu stiva. Vom vedea ca masinile Turing liniar marginite nedeterministe sunt exact sistemele care accepta limbajele de tip 1 iar masinile Turing deterministe sunt exact sistemele care accepta limbajele de tip 0. Mai departe vom vedea ca masina Turing duce la o posibila definitie a notiunii de calculabilitate, si ca aceasta definitie se dovedeste echivalenta cu alte definitii justificate. Alan Turing a definit masina Turing abstractizand munca unui contabil. Simbolurile (cifrele) scrise de contabil pot fi puse pe o singura banda iar contabilul le parcurge in mod liniar in ambele sensuri. Functie de *starea* in care se afla, contabilul sterge un simbol, scrie un al simbol, isi schimba starea si se deplasesaza cu un simbol mai la stanga sau mai la dreapta. Definitia corespunde acestei abstractizari:

Definitie: O masina Turing determinista cu o banda este un tuplu $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$, unde:

Z este o multime finita de stari,

Σ este alfabetul de input,

$\Gamma \supset \Sigma$ este alfabetul de lucru,

$\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$ este functia de tranzitie,

$z_0 \in Z$ este starea initiala (de start),

$\Box \in \Gamma \setminus \Sigma$ este simbolul blanc (care marcheaza o casuta goala),

$E \subset Z$ este multimea starilor finale.

Semnificatia acestor simboluri: O banda infinita in ambele sensuri contine inputul, care este un cuvant $w \in \Sigma^*$. Un cap de citire si scriere se afla pe prima litera a inputului iar masina se afla in starea initiala z_0.

$$\dots \ \Box\ \Box\ \Box\ \boxed{a}\ b\ c\ 0\ 1\ 1\ 0\ \Box\ \Box\ \Box\ \dots$$

Aici notatia $\boxed{a}$ indica pozitia capului de citire-scriere pe litera a, prima litera a inputului. Daca la un moment dat masina se afla in starea z si citeste litera a, se evalueaza functia de tranzitie $\delta(z, a) = (z', b, x)$. Deci masina va inlocui a cu b, va trece in starea z' si va face un pas la R (dreapta), L (stanga) respectiv N (nu se va muta). $\Box$

Definitie: O masina Turing nedeterminista cu o banda se defineste asemanator, doar ca functia de tranzitie este o functie:

$$\delta : Z \times \Gamma \to P(Z \times \Gamma \times \{L, R, N\}).$$

In acest caz, dupa ce a citit litera a in starea z, masina alege la intamplare un element $(z', b, x) \in \delta(z, a)$ si il executa. $\Box$

Definitie: O configuratie a masinii Turing este un cuvant $k \in \Gamma^* Z \Gamma^*$. Configuratia $k = \alpha z \beta$ inseamna ca masina este in starea z pe prima litera a lui β, iar in stanga are cuvantul α. Pozitia de start este configuratia $z_0 x$ cu $x \in \Sigma^*$. $\square$

Definitie: Pe multimea configuratiilor se defineste o relatie $\vdash$ dupa cum urmeaza:

$$a_1 \dots a_m z b_1 \dots b_n \vdash \begin{cases} a_1 \dots a_m z' c b_2 \dots b_n, & \delta(z, b_1) = (z', c, N),\ m \geq 0,\ n \geq 1, \\ a_1 \dots a_m c z' b_2 \dots b_n, & \delta(z, b_1) = (z', c, R),\ m \geq 0,\ n \geq 2, \\ a_1 \dots a_{m-1} z' a_m c b_2 \dots b_n, & \delta(z, b_1) = (z', c, L),\ m \geq 1,\ n \geq 1. \end{cases}$$

Exista si doua cazuri speciale. Daca $n = 1$ si masina merge la dreapta, va intalni un blanc:

$$a_1 \dots a_m z b_1 \vdash a_1 \dots a_m c z' \square, \quad \delta(z, b_1) = (z', c, R)$$

Daca $m = 0$ si masina merge catre stanga, va intalni de asemeni un blanc:

$$z b_1 \dots b_n \vdash z' \square c b_2 \dots b_n, \quad \delta(z, b_1) = (z', c, L).$$

Exemplu: Urmatoarea masina Turing primeste un input $x \in \{0, 1\}^*$. Ea interpreteaza acest cuvant ca numar binar, aduna 1 si opreste.

$$M = (\{z_0, z_1, z_2, z_e\}, \{0, 1\}, \{0, 1, \square\}, \delta, z_0, \square, \{z_e\}),$$

$$\begin{array}{rcl} \delta(z_0, 0) & = & (z_0, 0, R) \\ \delta(z_0, 1) & = & (z_0, 1, R) \\ \delta(z_0, \square) & = & (z_1, \square, L) \\ \\ \delta(z_1, 0) & = & (z_2, 1, L) \\ \delta(z_1, 1) & = & (z_1, 0, L) \\ \delta(z_1, \square) & = & (z_e, 1, N) \\ \\ \delta(z_2, 0) & = & (z_2, 0, L) \\ \delta(z_2, 1) & = & (z_2, 1, L) \\ \delta(z_2, \square) & = & (z_e, \square, R) \end{array}$$

Iata un calcul efectuat de catre aceasta masina:

$$z_0 101 \vdash 1 z_0 01 \vdash 10 z_0 1 \vdash 101 z_0 \square \vdash 10 z_1 1 \square \vdash$$

$$\vdash 1 z_1 00 \square \vdash z_2 110 \square \vdash z_2 \square 110 \square \vdash \square z_e 110 \square.$$

$\square$

Definitie: Limbajul acceptat de catre o masina Turing M este:

$$T(M) = \{x \in \Sigma^* \mid z_0 x \vdash^* \alpha z \beta;\ \alpha, \beta \in \Gamma^*;\ z \in E\}.$$

$\square$

3 Limbaje de tip 1

Definitie: O masina Turing este liniar marginita, daca in timpul functionarii ea nu paraseste segmentul determinat de input. In acest scop, se dubleaza alfabetul de input: $\Sigma' = \Sigma \cup \{\bar{a} \mid a \in \Sigma\}$. Literele $\bar{a}$ se folosesc pentru a marca sfarsitul cuvantului. Astfel, masina Turing M (determinista sau nu) este liniar marginita daca si numai daca pentru orice input $a_1 \dots a_n \in \Sigma^+$ si pentru orice configuratie $\alpha z \beta$ astfel incat $a_1 a_2 \dots a_{n-1} \bar{a}_n \vdash^* \alpha z \beta$ are loc $|\alpha\beta| = n$. □

Teorema: *(Kuroda) Limbajele acceptate de catre masini Turing nedeterministe liniar marginite (LBA) sunt limbajele de tip* 1.

Demonstratie: $\Leftarrow$ Fie A un limbaj de ordinul 1, adica $A = L(G)$ unde $G = (V, \Sigma, P, S)$ este o gramatica de tip 1. Descriem o masina Turing nedeterminista M care accepta limbajul A. Fie $x = a_1 \dots a_n$ un input. M alege in mod nedeterminist o regula $u \to v$ din multimea P. Apoi M cauta o aparitie a lui v in x. Daca o gaseste, inlocuieste v cu u. Daca u este mai scurt decat v, restul literelor sunt translatate la stanga. Se poate presupune ca G este in Forma Normala Kuroda, astfel incat niciodata o asemenea translatie nu va fi cu mai mult de o casuta. Cand pe banda ramane doar simbolul de start S, masina se opreste.

Asadar: $x \in L(G)$ **iff** exista o derivare $S \Rightarrow \dots \Rightarrow x$ **iff** exista o evolutie a lui M care simuleaza aceasta derivare in ordine inversa **iff** $x \in T(M)$.

$\Rightarrow$ Fie $A = T(M)$ limbajul recunoscut de o masina Turing nedeterminista liniar marginita. Fie $\Delta = \Gamma \cup (Z \times \Gamma)$ un nou alfabet finit. O δ-tranzitie nedeterminista:

$$\delta(z, a) \ni (z', b, L)$$

va fi inlocuita cu reguli de productie dependente de context, de tipul:

$$c(z, a) \to (z', c)b$$

pentru toate $c \in \Gamma$. Notam cu P' aceasta multime de reguli de productie. Daca in M are loc un calcul de forma $k \vdash^* k'$, atunci in gramatica G este posibila o derivatie corespunzatoare $\tilde{k} \Rightarrow^* \tilde{k}'$, unde $\tilde{k}$ este reprezentarea configuratiei k. Gramatica dependenta de context (context-senzitiva) $G = (V, \Sigma, P, S)$ se defineste in modul urmator:

$$V = \{S, A\} \cup (\Delta \times \Sigma)$$

$$\begin{aligned}
P \;=\; & \{S \to A(\bar{a}, a) \mid a \in \Sigma\} && (1)\\
& \cup \{A \to A(a, a) \mid a \in \Sigma\} && (2)\\
& \cup \{A \to ((z_0, a), a) \mid a \in \Sigma\} && (3)\\
& \cup \{(\alpha_1, a)(\alpha_2, b) \to (\beta_1, a)(\beta_2, b) \mid \alpha_1\alpha_2 \to \beta_1\beta_2 \in P';\ a, b \in \Sigma\} && (4)\\
& \cup \{((z, a), b) \to b \mid z \in E, a \in \Gamma, b \in \Sigma\} && (5)\\
& \cup \{(a, b) \to b \mid a \in \Gamma, b \in \Sigma\} && (6)
\end{aligned}$$

Folosind reguli de tip (1), (2) si (3) se deriva:

$$S \Rightarrow^* ((z_0, a_1), a_1)(a_2, a_2) \dots (a_{n-1}, a_{n-1})(\bar{a}_n, a_n)$$

Primele componente din fiecare pereche definesc configuratia de start a unui calcul Turing. Componentele celelalte definesc cuvantul si nu se vor modifica. Pe componenta 1 se aplica acum regulile (4) si se simuleaza un calcul nedeterminist, pana se atinge o stare finala:

$$\dots \Rightarrow^* (\gamma_1, a_1) \dots (\gamma_{k-1}, a_{k-1})((z, \gamma_k), a_k)(\gamma_{k+1}, a_{k+1}) \dots (\gamma_n, a_n)$$

cu $z \in E$, $\gamma_i \in \Gamma$, $a_i \in \Sigma$. Acum, folosind regulile de productie (5) si (6) este stearsa componenta 1 si se genereaza cuvantul final:

$$\Rightarrow^* a_1 \dots a_n.$$

Se verifica imediat ca toate regulile sunt de tip 1. □

4 Limbaje de tip 0

Teorema: *Limbajele acceptate de catre masini Turing nedeterministe sunt limbajele de tip* 0.

Demonstratie: $\Leftarrow$ Demonstratia este asemanatoare cu cea pentru tipul 1. Masina alege nondeterminist o productie $u \to v$ si gaseste in mod nedeterminist o aparitie a lui v in input ca subcuvant.

- In cazul in care $|u| < |v|$, masina il inlocuieste pe v cu u si deplaseaza partea din input care se afla in dreapta lui v catre stanga, litera cu litera, $|v| - |u|$ casute pentru a forma un cuvant conex.

- In cazul in care $|u| = |v|$, masina il inlocuieste pe v cu u.

- In cazul in care $|u| > |v|$, masina deplaseaza partea din dreapta lui v catre dreapta, litera cu litera, $|u| - |v|$ casute, dupa care il inlocuieste pe v cu u.

Lasam ca exercitiu sa se arate ca aceste operatii deterministe se pot face utilizand un numar finit de stari si eventual o extensie finita a alfabetului gramaticii. Dupa acest pas, se alege din nou in mod nedeterminist o productie, si procesul se reia. Masina se opreste numai in situatia in care la un moment dat pe banda apare doar simbolul initial S. Observati ca masina obtinuta nu este liniar marginita.

$\Rightarrow$[2] Fie $A = T(M)$ limbajul recunoscut de o masina Turing nedeterminista. Facem conventia ca functia δ nu este definita pentru nicio pereche de forma (z, a) unde z este stare finala. Construim urmatoarea gramatica $G = (V, \Sigma, P, S)$ unde:

$$V = ((\Sigma \cup \{\varepsilon\}) \times \Gamma) \cup Z \cup \{S, E_1, E_2, E_3\},$$

unde S, E_1, E_2, E_3 sunt litere noi iar ε este un simbol pentru cuvantul vid. Multimea regulilor de productie P contine urmatoarele:

$$S \to E_1 E_2, \tag{1}$$
$$E_2 \to (a, a) E_2,\ a \in \Sigma \tag{2}$$
$$E_2 \to E_3, \tag{3}$$
$$E_3 \to (\varepsilon, \Box) E_3,\ E_1 \to (\varepsilon, \Box) E_1, \tag{4}$$
$$E_3 \to \varepsilon, E_1 \to z_0, \tag{5}$$
$$z(a, \alpha) \to (a, \beta) z',\ \delta(z, \alpha) \ni (z', \beta, R),\ a \in \Sigma \cup \{\varepsilon\} \tag{6}$$
$$(a, \alpha) z \to z'(a, \beta),\ \delta(z, \alpha) \ni (z', \beta, L),\ a \in \Sigma \cup \{\varepsilon\} \tag{7}$$
$$(a, \alpha) z \to zaz,\ z(a, \alpha) \to zaz,\ z \to \varepsilon,\ a \in \Sigma \cup \{\varepsilon\},\ \alpha \in \Gamma,\ z \in E. \tag{8}$$

De data aceasta cuvantul din Σ^* este mentinut pe prima pozitie iar calculul Turing este simulat pe pozitia a doua. Folosind regulile (1), (2) si (3) se pot construi derivatii de forma:

$$S \Rightarrow^* E_1 (a_1, a_1)(a_2, a_2) \dots (a_{n-1}, a_{n-1})(a_n, a_n) E_3$$

Presupunem ca masina Turing M, pentru a accepta cuvantul $a_1 \dots a_n$, foloseste un numar de l casute in stanga casutei ocupate initial de a_1 si un numar de m casute in dreapta casutei ocupate initial de a_n. Folosind regulile (4) si (5) se obtine o derivatie:

$$S \Rightarrow^* (\varepsilon, \Box)^l z_0 (a_1, a_1)(a_2, a_2) \dots (a_{n-1}, a_{n-1})(a_n, a_n)(\varepsilon, \Box)^m.$$

Folosind (6) si (7) se simuleaza masina M pe componenta a doua, pana cand se ajunge la o stare finala. Acum, folosind (8), se obtine cuvantul terminal $a_1 \dots a_n$. Observam ca aceasta gramatica nu este de tip 1 din cauza regulilor (5) si (8). $\Box$

[2] Ian Chiswell, A course in formal languages, automata and groups, Springer Verlag, 2009.

Urmatorul rezultat intareste puternic teorema precedenta, insa este foarte interesant si in sine.

Teorema: *Pentru orice masina Turing nedeterminista M exista o masina Turing determinista M' care recunoaste acelasi limbaj.*

Demonstratie[3]: La inceputul fiecarei runde de simulare, banda masinii M' contine un cuvant de forma urmatoare:

$$\square\square\square\#\#\alpha_1 z_1 \beta_1 \# \alpha_2 z_2 \beta_2 \# \ldots \# \alpha_k z_k \beta_k \#\#\square\square\square$$

Aici # este un nou simbol, folosit ca separator, iar $\alpha_i z_i \beta_i$ sunt configuratii ale masinii M. La fiecare runda de simulare, masina M' adauga in dreapta toate configuratiile posibile care provin din $\alpha_1 z_1 \beta_1$ intr-un pas nedeterminist al masinii M, dupa care sterge configuratia $\alpha_1 z_1 \beta_1$. Cand intalneste o stare finala a masinii M, masina M' se opreste. □

Teorema: *Limbajele de tip* 0 *sunt exact limbajele acceptate (recunoscute) prin oprire de catre masinile Turing deterministe.*

Demonstratie: Acest lucru rezulta direct din juxtapunerea ultimelor doua rezultate. □

Deci, in cazul general (tip 0), nu are nicio importanta daca vorbim despre o masina Turing determinista sau nedeterminista. In cazul limbajelor de tip 1 nu este deloc clar daca o masina nedeterminista liniar marginita (NLBA = linear bounded automaton) poate fi simulat de catre o masina determinista liniar marginita (LBA = determinist linear bounded automaton). Una dintre cele mai grele probleme din informatica teoretica, inca deschisa in momentul de fata, este daca:

$$LBA = NLBA$$

O alta problema inrudita, numita mult timp "Second LBA Problem", a fost daca limbajele de tip 1 sunt inchise la complementare. Acest lucru a fost demonstrat in lucrari independente de catre Immerman si Szelepcsenyi.

[3] Din memorie...

Part II
Functii calculabile

5 Notiunea intuitiva de functie calculabila

Fie Σ un alfabet finit. Cum exista bijectii calculabile $f : \Sigma^* \to \mathbb{N}$ cu inversa calculabila $f^{-1} : \mathbb{N} \to \Sigma^*$ (exercitiu!), pentru a intelege notiunea de calculabilitate, este suficient sa studiem functii definite pe multimea numerelor naturale cu valori naturale. O functie $f : \mathbb{N}^k \to \mathbb{N}$, eventual partiala, poate fi considerata calculabila, daca exista o procedura efectiva (un algoritm) care, pornita pentru niste valori initiale $(n_1, n_2, \dots, n_k) \in \mathbb{N}^k$, se opreste dupa un numar finit de pasi si produce un rezultat $f(n_1, n_2, \dots, n_k) \in \mathbb{N}$. Asadar obiectul studiului vor fi de fapt algoritmii. Fiecarui algoritm i se asociaza functia calculabila calculata de catre el.

Exemplu: Algoritmul

INPUT(n);

REPEAT UNTIL FALSE;

calculeaza functia Ω care nu este definita nicaieri. Cu alte cuvinte dom $\Omega = \emptyset$. □

Exemplu: Functia

$$f_\pi(n) = \begin{cases} 1 & n \text{ este un segment initial al dezvoltarii lui } \pi \text{ ca fractie zecimala,} \\ 0 & \text{altfel.} \end{cases}$$

astfel incat $f_\pi(314) = 1$, $f_\pi(5) = 0$, este calculabila, deoarece exista metode de aproximare pentru numarul π. □

Exemplu: Functia

$$g(n) = \begin{cases} 1 & n \text{ apare undeva in dezvoltarea lui } \pi \text{ ca fractie zecimala,} \\ 0 & \text{altfel.} \end{cases}$$

s-ar putea sa nu fie calculabila. Ceea ce stim despre numarul π nu este suficient ca sa putem decide acest lucru. In cazul in care fiecare numar ar aparea undeva in dezvoltarea zecimala a lui π, ceea ce nu se stie, functia ar fi $g(n) = 1$ pentru orice n, deci ar fi calculabila. □

Exemplu: Functia

$$h(n) = \begin{cases} 1 & \text{dezvoltarea lui } \pi \text{ ca fractie zecimala contine undeva cifra 7 de } n \text{ ori consecutiv,} \\ 0 & \text{altfel.} \end{cases}$$

este calculabila! Intr-adevar, sau aceasta dezvoltare ca fractie zecimala contine segmente arbitrar de lungi formate din cifra 7, caz in care functia h este constanta si egala cu 1, sau exista un n_0 astfel incat $h(n) = 1$ daca si numai daca $n \leq n_0$. Desi stim ca functia este calculabila, nu cunoastem aceasta functie pentru ca nu cunoastem algoritmul care o calculeaza. Asta deoarece avem prea putine cunostiinte despre numarul π. □

Daca fiecarui numar real $r \in \mathbb{R}$ ii asociem o functie f_r asa cum i-am asociat lui π functia f_π in exemplul de mai sus, marea majoritate a acestor functii nu sunt calculabile. Motivul este simplu: exista doar o infinitate numarabila $\aleph_0$ de algoritmi, dar o infinitate nenumarabila $2^{\aleph_0}$ de numere reale. Mai mult, daca consideram toate submultimile $A \subseteq \mathbb{N}$, marea majoritate a acestor submultimi au functii caracteristice necalculabile si numai o multime numarabila de astfel de submultimi au functii caracteristice calculabile, deoarece exista $2^{\aleph_0}$ astfel de submultimi[4]. Cu alte cuvinte majoritatea submultimilor lui $\mathbb{N}$ sunt nedecidabile algoritmic.

[4]**Teorema**: *(Cantor) Daca A este o multime, iar $P(A) = \{B \mid B \subseteq A\}$ este multimea partilor ei, atunci nu exista nicio functie $f : A \to P(A)$ surjectiva. Cu alte cuvinte, $|A| < |P(A)|$.*

Demonstratie: Fie $f : A \to P(A)$ o functie surjectiva. Definim multimea:

$$C = \{x \in A \mid x \notin f(x)\}.$$

Cum $C \in P(A)$ si f este surjectiva, exista $c \in A$ astfel incat $f(c) = C$. Daca $c \in C$ atunci $c \in f(c)$ deci $c \notin C$, contradictie. Daca $c \notin C$ atunci $c \in C$, contradictie. □

Daca notam cu $\aleph_0$ cardinalitatea lui $\mathbb{N}$ si cu $2^{\aleph_0}$ cardinalitatea lui $P(\mathbb{N})$, rezulta ca $\aleph_0 < 2^{\aleph_0}$. Cu alte cuvinte infinitatea numarabila este strict mai slaba decat puterea continuumului.

Teza lui Church: *Clasa functiilor Turing-calculabile (sau, echivalent, a functiilor WHILE calculabile, GOTO calculabile sau μ-recursive) coincide cu clasa functiilor calculabile in mod intuitiv.*

$\square$

In aceasta parte a cursului vom defini aceste clase de functii si vom demonstra echivalenta lor. Se vor prezenta si doua notiuni mai slabe: functiile LOOP-calculabile si functiile primitiv recursive, care se vor dovedi la randul lor echivalente.

6 Calculabilitate Turing

Prezentam cateva definitii posibile pentru notiunea de functie Turing calculabila. Se va vedea ca aceste definitii sunt echivalente.

Definitie: Fie Σ un alfabet finit. O functie $f : \Sigma^* \to \Sigma^*$ se numeste Turing calculabila, daca exista o masina Turing determinista M astfel incat pentru orice $x, y \in \Sigma^*$, $f(x) = y$ daca si numai daca:

$$z_0 x \vdash^* \square \dots \square z_e y \square \dots \square$$

unde $z_e \in E$. $\square$

Pentru un numar natural n fie $bin(n)$ scrierea binara a lui n, $un(n)$ scrierea lui unara si $dec(n)$ scrierea lui zecimala. De exemplu, $dec(3) = 3$, $un(3) = 111$ iar $bin(3) = 11$.

Definitie: O functie $f : \mathbb{N}^k \to \mathbb{N}$ se numeste Turing calculabila daca exista o masina Turing M astfel incat pentru orice $n_1, n_2, \dots, n_k, m \in \mathbb{N}$ are loc $f(n_1, n_2, \dots, n_k) = m$ daca si numai daca:

$$z_0 bin(n_1)\#bin(n_2)\#\dots\#bin(n_k) \vdash^* \square \dots \square z_e bin(m) \square \dots \square.$$

$\square$

Ultima definitie se poate scrie identic pentru reprezentarea unara $un(n)$, pentru reprezentarea zecimala $dec(n)$, pentru diferite produse $\mathbb{N}^k$ atat la domeniu cat si la codomeniu. Ne putem gandi si la functii $f : \Sigma^2 \to \mathbb{N}^4$, cu output codificat in formatul (bin, un, dec, bin). In realitate, exista bijectii calculabile cu inversa calculabila intre Σ^* si $\mathbb{N}$ si intre $\mathbb{N}^k$ si $\mathbb{N}$. Mai mult, transformarile reprezentarii numerice din baza b_1 in baza b_2 si invers sunt functii calculabile. Fiecare dintre acestea se poate realiza in particular cu masini Turing, care se pot combina cu o masina Turing data atat la prepararea inputului cat si la traducerea outputului (asa cum se va vedea in paginile urmatoare). De aceea avem de a face cu o singura clasa de functii Turing calculabile, iar definitiile de mai sus si alte variante ale lor slujesc mai mult la fixarea unor specificatii legate de reprezentarea functiei, de formatul de input si de formatul de output.

Exemplu: Functia succesor $s(n) = n + 1$ este Turing calculabila, vezi Sectiunea 2 pentru functia succesor in scrierea binara. Daca folosim scrierea unara, masina respectiva este mult mai simpla (exercitiu). $\square$

Exemplu: Functia care nu este definita nicaieri Ω este calculata de masina Turing:

$$\delta(z_0, a) = (z_0, a, R) \text{ pentru toate } a \in \Gamma.$$

$\square$

Exemplu: Am vazut ca un limbaj este de tip 0 daca si numai daca este acceptat de catre o masina Turing. Cu alte cuvinte, exista o masina Turing care calculeaza urmatoarea functie $\chi'_A : \Sigma^* \to \{0, 1\}$,

$$\chi'_A(w) = \begin{cases} 1, & w \in A, \\ \textit{nedefinita}, & w \notin A. \end{cases}$$

Cum aceasta este numai jumatate din functia caracteristica a lui A, spunem despre A ca este o multime semidecidabila. $\square$

Definitie: O masina Turing cu mai multe benzi poate opera pe un numar de $k \geq 1$ benzi. Ea are k capete de citire-scriere. Aceste capete pot citi casute, pot scrie in casute si se misca independent unul de altul, fiecare pe banda lui. Formal, functia de tranzitie este:

$$\delta : Z \times \Gamma^k \to Z \times \Gamma^k \times \{L, R, N\}^k,$$

respectiv

$$\delta : Z \times \Gamma^k \to \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k).$$

in cazul masinilor cu mai multe benzi nedeterministe.

Teorema: *Pentru orice masina Turing M cu mai multe benzi exista o masina Turing M' cu o banda astfel incat $T(M) = T(M')$, respectiv care calculeaza aceeasi functie ca si M.*

Demonstratie: Fie k numarul de benzi ale lui M si fie Γ alfabetul de lucru al lui M. Ideea este sa impartim banda lui M in $2k$ canale. O posibila configuratie a masinii M este:

$$\ldots\ a_1\ a_2\ \boxed{a_3}\ a_4\ a_5\ a_6\ a_7\ a_8\ a_9\ a_{10}\ \ldots$$

$$\ldots\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ \boxed{b_7}\ b_8\ b_9\ b_{10}\ \ldots$$

$$\ldots\ c_1\ \boxed{c_2}\ c_3\ c_4\ c_5\ c_6\ c_7\ c_8\ c_9\ c_{10}\ \ldots$$

Aici $\boxed{a_3}$ inseamna ca primul cap de citire-scriere este pe litera a_3 de pe prima banda, $\boxed{b_7}$ inseamna ca al doilea cap de citire-scriere este pe b_7 si analog pentru al treilea cap de citire-scriere pe c_2.

Cele $2k$ canale ale benzii masinii M' sunt folosite in modul urmator: canalele de indice impar replica benzile masinii M, pe cand canalele de indice par contin un singur simbol (o litera noua). Pozitia lui indica pozitia capului de citire-scriere de pe banda corespunzatoare a masinii M.

$\ldots$	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	$\ldots$
			$\bigstar$								
$\ldots$	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	$\ldots$
							$\bigstar$				
$\ldots$	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	$\ldots$
		$\bigstar$									

Alfabetul de lucru al masinii M' este $\Gamma' = \Gamma \cup (\Gamma \cup \{\bigstar\})^{2k}$. M' o simuleaza pe M dupa cum urmeaza. M' porneste cu un input $x_1 x_2 \ldots x_n \in \Gamma^*$. Apoi M' genereaza configuratia de start a lui M in reprezentarea cu canale. Un pas al lui M este simulat prin mai multi pasi ai lui M'. Presupunem ca capul de citire-scriere al lui M' se afla in stanga tuturor marcajelor cu $\bigstar$. M' merge catre dreapta pana cand a citit toate aceste marcaje. Acum M' "stie" (cu ajutorul starii in care se afla) care argument al functiei δ a lui M trebuie aplicat. Pentru aceasta, $|Z'| > |Z \times \Gamma^k|$. Apoi merge M' catre stanga peste toate marcajele cu $\bigstar$ si produce toate schimbarile necesare, care corespund unui pas al masinii M. $\square$

In randurile urmatoare vom arata cum putem inlantui masini Turing simple in scopul de a demonstra ca anumite functii sunt Turing calculabile. Vom lucra cu masini Turing cu mai multe benzi, deoarece stim datorita propozitiei precedente ca ele sunt echivalente cu niste masini Turing cu o singura banda.

Definitie: Fie M o masina Turing cu o singura banda. Notam cu $M(i, k)$ o masina Turing cu k benzi, astfel incat pe banda i lucreaza masina Turing M iar celelalte benzi raman neschimbate. Mai exact, daca pentru masina M are loc $\delta(z, a) = (z', b, y)$ atunci in masina $M(3, 5)$ de exemplu,

$$\delta(z, c_1, c_2, a, c_3, c_4) = (z', c_1, c_2, b, c_3, c_4, N, N, y, N, N)$$

pentru toate $c_1, c_2, a, c_3, c_4 \in \Gamma$. Daca k nu este important, folosim notatia $M(i)$ in loc de $M(i, k)$. $\square$

Definitie: Masina care aduna 1 poate fi denumita "Tape := Tape + 1". In loc de "Tape := Tape + 1"(i) scriem "Tape(i) := Tape(i) + 1". In mod asemanator putem obtine masinile Turing cu mai multe benzi "Tape(i) := Tape(i) $\dot{-}$ 1", "Tape(i) := 0", "Tape(i) :=Tape(j)". Aici operatia $\dot{-}$, scaderea naturala, este definita in modul urmator:

$$x \dot{-} y = \begin{cases} x - y, & x \geq y \\ 0, & x < y. \end{cases}$$

Definitie: Dorim sa aplicam masini Turing (cu una sau mai multe benzi) una dupa cealalta. Fie $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_i, \Box, E_i)$ cu $i = 1, 2$, doua masini Turing. Masina:

$$\text{start} \to M_1 \to M_2 \to \text{stop},$$

notata si $M_1; M_2$, se defineste ca:

$$M = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_1, \Box, E_2)$$

unde $Z_1 \cap Z_2 = \emptyset$ si

$$\delta = \delta_1 \cup \delta_2 \cup \{(z_e, a, z_2, a, N) \,|\, z_e \in E_1, a \in \Gamma_1\}.$$

$\Box$

Exemplu: Masina

start $\to$ "Tape := Tape + 1" $\to$ "Tape := Tape + 1" $\to$ "Tape := Tape + 1" $\to$ stop,

este o masina Turing, care aduna cu 3. $\Box$

Definitie: Fie M, M_1 si M_2 masini Turing. M are doua stari finale speciale z_{e_1} si z_{e_2}. Se poate imagina usor masina:

IF state(M) = z_{e_1} THEN M_1 ELSE IF state(M) = z_{e_2} THEN M_2

$\Box$

Definitie: Masina Turing cu o banda "Tape=0?" se defineste in modul urmator: $Z = \{z_0, z_1,$ yes, no$\}$, starea initiala z_0, stari finale yes si no. Pentru scrierea binara, δ contine urmatoarele tranzitii:

$$\begin{array}{rcll} \delta(z_0, a) & = & (\text{no}, a, N) & a \neq 0 \\ \delta(z_0, 0) & = & (z_1, 0, R) & \\ \delta(z_1, a) & = & (\text{no}, a, L) & a \neq \Box \\ \delta(z_1, \Box) & = & (\text{yes}, \Box, L) & \end{array}$$

In loc de "Tape=0?"(i) scriem "Tape(i)=0?".

Definitie: Fie M o masina Turing. Folosind ultimele doua definitii, si reinitializarea masinii de test dupa fiecare test efectuat, se poate construi masina:

WHILE Tape(i) $\neq 0$ DO M

7 Limbaje de programare

Definitie: O masina cu registre este formata dintr-un numar potential infinit de registre x_0, x_1, x_2, ... Fiecare registru poate contine un numar arbitrar de bete de chibrit |. Daca nu contine niciun bat de chibrit, valoarea lui x_i este 0. Prin conventie, numai un numar finit de registre pot avea o valoare diferita de 0. De exemplu, in configuratia:

$$\begin{array}{ccl} x_0 & = & ||| \\ x_1 & = & \\ x_2 & = & ||||| \\ x_3 & = & || \\ x_4 & = & \\ x_5 & = & \\ \vdots & \vdots & \vdots \end{array}$$

$x_0 = 3$, $x_1 = 0$, $x_2 = 5$, $x_3 = 2$, iar celelalte x_i sunt 0. □

Definitie: Simbolurile cu care se scrie un program in limbajul LOOP sunt de urmatoarele tipuri:

Variabile: x_0, x_1, x_2, x_3, ...

Constante: 0, 1, 2, 3, 4, ...

Separatoare: ; :=

Semne de operatie: + -

Cuvinte cheie: LOOP DO END

□

Definitie: Sintaxa programelor LOOP este definita inductiv.

- Atat $x_i := x_j + 1$ cat si $x_i := x_j - 1$ sunt programe LOOP.

- Daca P_1 si P_2 sunt programe LOOP, atunci $P_1; P_2$ este un program LOOP.

- Daca P este un program LOOP si x_i este o variabila, atunci LOOP x_i DO P END este un program LOOP.

□

Definitie: Semantica programelor LOOP se defineste inductiv in mod asemanator cu sintaxa.

- Programele $x_i := x_j + 1$ si $x_i := x_j - 1$ au semnificatia intuitiva, cu completarea ca scaderea folosita aici este scaderea naturala, definita in sectiunea precedenta si notata acolo cu "$\dot{-}$". Pentru simplitate folosim doar semnul minus.

- Atunci cand se ruleaza programul $P_1; P_2$, se ruleaza programul P_1 iar pe configuratia rezultata se ruleaza programul P_2.

- Atunci cand se ruleaza programul LOOP x_i DO P END, programul P se ruleaza in mod repetat un numar de ori egal cu valoarea lui x_i **la inceput**, adica inainte de prima rulare a lui P.

□

Limbajul LOOP a fost introdus in 1967 de catre Albert R. Meyer si Dennis M. Ritchie, cu scopul de a defini asa-zisele functii primitiv recursive. Sensul acestei notiuni va fi explicat mai tarziu.

Definitie: O functie $f : \mathbb{N}^k \to \mathbb{N}$ se numeste LOOP calculabila daca exista un program LOOP P astfel incat, pentru orice $n_1, \ldots, n_k \in \mathbb{N}$, daca este pornit cu valorile $n_1, \ldots, n_k \in \mathbb{N}$ in registrele $x_1, \ldots, x_k$ si cu valoarea 0 in celelalte registre, opreste cu valoarea $f(n_1, \ldots, n_k)$ in registrul x_0.

□

Este usor de observat ca urmatoarele programe pot fi scrise in LOOP: $x_i := x_j + c$, $x_i := x_j - c$, $x_i := x_j$, $x_i := c$. Aici putem avea si situatia $i = j$, iar c poate fi orice constanta.

Programul IF $x = 0$ THEN A END este urmatorul:

$y := 1;$

LOOP x DO $y := 0$ END;

LOOP y DO A END;

Programul $x_0 := x_1 + x_2$ se scrie:

$x_0 := x_1$;

LOOP x_2 DO $x_0 := x_0 + 1$ END

Analog se poate scrie programul $x_0 := x_1 - x_2$. Cu ajutorul adunarii, putem scrie programul $x_0 := x_1 * x_2$ dupa cum urmeaza:

$x_0 := 0$;

LOOP x_2 DO $x_0 := x_0 + x_1$ END

Putin mai complicat este programul $x_0 = x_1$ DIV x_2, prezentat mai jos:

```
x0 := 0;
LOOP x1 DO
        IF x1 ≥ x2 THEN
                x1 := x1 − x2;
                x0 := x0 + 1;
        END
END
```

Pentru $x_0 := x_1$ MOD x_2 putem considera programul $x_0 := x_1 - (x_1$ DIV $x_2) * x_2$.

Teorema: *Rularea unui program* LOOP *se termina intotdeauna, indiferent de valoarea initiala a argumentelor. In particular, toate functiile* LOOP *calculabile sunt functii totale.*

Demonstratie: Demonstratia se face prin inductie dupa definitia inductiva a programelor LOOP. □

Aceasta teorema arata deja ca nu toate functiile calculabile sunt LOOP calculabile. Astfel, programele LOOP nu sunt indicate pentru recunoasterea multimilor prin oprire. Vom vedea intr-o sectiune viitoare ca exista si functii calculabile total definite care nu sunt LOOP calculabile.

Definitie: Programele WHILE sunt programele LOOP imbogatite cu urmatorul pas de constructie: Daca P este un program WHILE, atunci

$$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

este un program WHILE. Semantica acestui construct este urmatoarea: programul P se repeta pana cand se intampla ca valoarea x_i sa fie egala cu 0. Spre deosebire de programele LOOP, valoarea lui x_i se verifica dupa fiecare rulare a programului P.

Teorema: *Orice functie* LOOP *calculabila este* WHILE *calculabila. Mai mult, pentru orice program* WHILE *exista un program* WHILE *echivalent, care nu contine nicio comanda* LOOP.

Demonstratie: Orice subprogram care are structura

$$\text{LOOP } x \text{ DO } P \text{ END}$$

poate fi inlocuit cu subprogramul:

$y := x$;

WHILE $y \neq 0$ DO $y := y - 1; P$ END

unde y este o variabila care nu apare in P. □

Definitie: O functie $f : \mathbb{N}^k \to \mathbb{N}$ se numeste WHILE calculabila daca exista un program WHILE P astfel incat, pentru orice $n_1, \ldots, n_k \in \mathbb{N}$, daca este pornit cu valorile $n_1, \ldots, n_k \in \mathbb{N}$ in registrele

$x_1, \ldots, x_k$ si cu valoarea 0 in celelalte registre, opreste cu valoarea $f(n_1, \ldots, n_k)$ in registrul x_0 daca si numai daca valoarea $f(n_1, \ldots, n_k)$ este definita, iar in caz contrar nu se opreste niciodata.

$\square$

Teorema: *Masinile Turing deterministe cu o banda pot simula programe* WHILE. *In particular orice functie* WHILE *calculabila este Turing calculabila.*

Demonstratie: In sectiunea precedenta am aratat cum fiecare comanda si constructie din programele WHILE poate fi facuta cu masini Turing deterministe cu mai multe benzi. Am aratat de asemeni ca aceste masini Turing pot fi simulate de masini Turing cu o banda. $\square$

Definitie: Un program GOTO pentru masina cu registre, este un sir de comenzi indexate cu etichete (labels):

$$M_1 : A_1;\ M_2 : A_2;\ \ldots\ M_n : A_n;$$

Oricare doua etichete sunt diferite. Comenzile A_i pot fi de urmatoarele tipuri:

Atribuiri: $x_i := x_j \pm 1$

Salt neconditionat: GOTO M_i

Salt conditionat: IF $x_i = c$ THEN GOTO M_j

Oprire: STOP

$\square$

Semantica programelor GOTO se defineste asemanator cu cea a programelor WHILE. Programele GOTO pot intra si ele in bucle infinite, precum si programele WHILE, de exemplu M : GOTO M;. Calculabilitatea GOTO se defineste exact ca si calculabilitatea WHILE - conditia necesara si suficienta ca functia sa fie definita pentru un anumit argument, este ca programul GOTO corespunzator sa opreasca. Datorita structurii care poate deveni foarte complicata, programele GOTO se mai numesc si programe *spaghetti*.

Teorema: *Orice* WHILE *program poate fi simulat printr-un program* GOTO. *In particular orice functie* WHILE *calculabila este* GOTO *calculabila.*

Demonstratie: Orice subprogram de forma

$$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

poate fi simulat de un subprogram de forma:

M_1 : IF $x_i = 0$ THEN GOTO M_4;

M_2 : P;

....

M_3 : GOTO M_1;

M_4 : ....

$\square$

Reciproca este adevarata:

Teorema: *Orice* GOTO *program poate fi simulat printr-un program* WHILE. *In particular orice functie* GOTO *calculabila este* WHILE *calculabila.*

Demonstratie: Fie un GOTO program:

$$M_1 : A_1;\ M_2 : A_2;\ \ldots\ M_k : A_k;$$

Vom simula acest program printr-un WHILE program care are o singura comanda WHILE, in modul urmator:

$count := 1;$

WHILE $count \neq 0$ DO

IF $count = 1$ THEN A'_1 END;

IF $count = 2$ THEN A'_2 END;

$\vdots$

IF $count = k$ THEN A'_k END;

END

Aici programele A' sunt definite in modul urmator:

$$A' = \begin{cases} x_i := x_j \pm c; count := count + 1 & \text{daca } A = \{x_i := x_j \pm c\}, \\ count := n & \text{daca } A = \{ \text{ GOTO } M_n\}, \\ \text{IF } x_i = c \text{ THEN } count := n & \text{daca } A = \{ \text{ IF } x_i = c \\ \ \text{ELSE } count := count + 1 \text{ END} & \ \text{THEN GOTO } M_n\}, \\ count := 0 & \text{daca } A = \{ \text{ STOP } \}. \end{cases}$$

□

Cele doua demonstratii de mai sus au o consecinta neasteptata, dupa cum urmeaza:

Teorema: *(Forma Normala Kleene a Programlor* WHILE*) Orice functie* WHILE *calculabila poate fi calculata printr-un program* WHILE *care contine o singura bucla* WHILE.

Demonstratie: Se folosesc demonstratiile ultimelor doua teoreme. Programul WHILE se transforma intr-un program GOTO, iar acesta se transforma intr-un program WHILE care contine o singura bucla WHILE. □

In continuare vom nota cu LOOP, WHILE, GOTO si Turing clasa functiilor calculabile prin respectivele modele de calcul. Am vazut ca LOOP $\subset$ WHILE. De asemeni am vazut ca WHILE = GOTO $\subset$ Turing. Scopul nostru urmator este sa aratam ca Turing $\subset$ GOTO.

Teorema: *Programele* GOTO *pot simula masinile Turing. In particular orice functie Turing calculabila este* GOTO *calculabila.*

Demonstratie: Fie $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ o masina Turing determinista cu o banda. Programul GOTO care o simuleaza este format din trei parti

$$M_1 : P_1;\ M_2 : P_2;\ M_3 : P_3$$

dupa cum urmeaza:

- P_1 transforma configuratia de start k_1 a masinii Turing intr-un triplet de numere naturale (x_1, y_1, z_1).

- P_2 simuleaza pas cu pas functionarea masinii Turing. De fiecare data o configuratie k este codificata intr-un triplet de numere naturale (x, y, z). Daca $k \vdash k'$ atunci in programul GOTO, $(x, y, z) \vdash (x', y', z')$.

- P_3 primeste forma codificata a configuratiei finale (x_f, y_f, z_f) si genereaza outputul masinii Turing.

Numai P_2 foloseste functia δ, asa ca ne vom concentra pe aceasta parte a programului. Iata cum se face codificarea. Fie $b \in \mathbb{N}$ un numar natural astfel incat $b > |\Gamma|$. O configuratie a masinii Turing este un cuvant de forma:

$$a_{i_1} \dots a_{i_p} z_t a_{j_1} \dots a_{j_q}.$$

Aici este important ca nicio litera din alfabetul Γ si nicio stare din multimea Z sa nu fie indexata cu 0. De aceea am si denumit starea initiala z_1. Numerele naturale (x, y, z) care codifica configuratia sunt:

$$\begin{aligned} x &= (i_1 \dots i_p)_b \\ y &= (j_q \dots j_1)_b \\ z &= t \end{aligned}$$

Aici $(i_1 \dots i_p)_b$ semnifica acel numar natural care in baza b are ca reprezentare "$i_1 \dots i_p$". Faptul ca in y cifrele apar in ordine inversa este important. Cifra unitatilor este cea mai accesibila, si reprezinta intotdeauna litera cea mai apropiata de capul de citire-scriere. Secventa de program $M_2 : P_2$ are urmatoarea structura:

$$\begin{array}{rcl} M_2 & : & a := y \text{ MOD } b; \\ & & \text{IF } z = 1 \text{ AND } a = 1 \text{ THEN GOTO } M_{11}; \\ & & \text{IF } z = 1 \text{ AND } a = 2 \text{ THEN GOTO } M_{12}; \\ & & \vdots \\ & & \text{IF } z = k \text{ AND } a = m \text{ THEN GOTO } M_{km}; \\ M_{11} & : & \diamond \\ & & \text{GOTO } M_2; \\ M_{12} & : & \diamond \\ & & \text{GOTO } M_2; \\ & & \vdots \\ M_{km} & : & \diamond \\ & & \text{GOTO } M_2; \end{array}$$

Acum ne ocupam cu partile marcate cu $\diamond$. Fie M_{ij} eticheta respectiva, presupunem ca valoarea functiei de tranzitie este:

$$\delta(z_i, a_j) = (z_{i'}, a_{j'}, L).$$

Actiunea ei poate fi simulata de urmatoarea succesiune de comenzi:

$$\begin{aligned} z &:= i'; \\ y &:= y \text{ DIV } b; \\ y &:= b * y + j'; \\ y &:= b * y + (x \text{ MOD } b); \\ x &:= x \text{ DIV } b; \end{aligned}$$

O succesiune similara de comenzi simuleaza tranzitiile la stanga. In cazul in care $z_{i'}$ este o stare finala, se adauga la sfarsit comanda GOTO M_3. Subprogramele P_1 si P_3 sunt standard si pot fi realizate in multe moduri. $\square$

Concuzia este ca clasele de functii (partiale sau totale) Turing calculabile, WHILE calculabile si GOTO calculabile coincid. Clasa functiilor LOOP calculabile este inclusa in aceasta clasa si este strict mai mica deoarece contine numai functii total definite. Vom vedea mai tarziu ca exista si functii totale care sunt WHILE calculabile dar nu sunt LOOP calculabile.

8 Busy beavers

Conform Tezei lui Church, vom numi de acum inainte clasa functiilor Turing, WHILE sau GOTO calculabile, pur si simplu functii calculabile. In acest paragraf vom construi:

- O functie total definita care este calculabila dar nu este LOOP calculabila.

- O functie total definita care nu este calculabila.

In acest scop, va trebui sa definim notiunea de lungime a unui program. In acest scop, pentru fiecare limbaj de programare prezentat, trebuie sa definim alfabetul in care sunt scrise programele. Pentru limbajul LOOP definim alfabetul:

$$A_{\text{LOOP}} = \{x \; ; \; := \; + \; - \; | \; \text{LOOP DO END}\}$$

care are 9 litere. Variabilele $x_0, x_1, x_2, \dots$ se noteaza $x, x|, x||, \dots$. Expresiile $x+1$ si $x-1$ se scriu $x + |$ si $x - |$. Se accepta numai comenzile din definitia limbajului. Pentru programele WHILE, alfabetul este asemanator, doar ca litera LOOP se inlocuieste cu litera WHILE si se mai adauga literele $\neq$ si 0. Cum avem literele 0 si $|$, variabilele se pot nota in limbajul WHILE cu x urmat de un cuvant binar. Asadar $x00$ si $x||0|$ sunt exemple de variabile.

Ne reamintim definitia functiei calculabile de o variabila. Un program P care poate fi LOOP, WHILE sau GOTO calculeaza o valoare $P(n)$ daca si numai daca programul porneste cu $x_1 = n$ si toate celelalte registre goale, si la un moment dat se opreste. Atunci valoarea din registrul x_0 este valoarea functiei $P(n)$ care in acest caz este definita.

Notiunea "busy beaver" (castori ocupati, castori vrednici) se refera istoric la masini Turing. Asa zisa Busy Beaver Competition, conceputa de catre Tibor Rado in 1962, este intrebarea: Care este numarul cel mai mare de betigase "|" scrise pe banda de catre o masina Turing cu cel mult n stari si care foloseste alfabetul format dintr-o singura litera, si anume betigasul?

Definitie: Pentru un tip de program TYPE in multimea { LOOP, WHILE } definim functia totala BB_{TYPE} in modul urmator:

$$BB_{\text{TYPE}}(n) = \max \; \{P(0) \,|\, P \text{ este un TYPE program de lungime } \leq n$$

$$\text{care este pornit cu toate registrele goale si opreste } \}$$

In cazul in care niciun asemenea program nu exista, se ia $BB_{\text{TYPE}}(n) = 0$. □

Observam ca functia este monotona, adica pentru orice n, $BB_{\text{TYPE}}(n+1) \geq BB_{\text{TYPE}}(n)$ si tinde la infinit.

Teorema[5]: *Fie $f : \mathbb{N} \to \mathbb{N}$ o functie total definita* TYPE *calculabila. Atunci exista $n_f \in \mathbb{N}$ astfel incat $BB_{TYPE}(n) > f(n)$ pentru orice $n \geq n_f$.*

Demonstratie: Fie P_f acel program TYPE care il calculeaza pe f. Presupunem ca in alfabetul A_{TYPE}, programul respectiv are lungimea c. Atunci, pentru orice $n \in \mathbb{N}$, exista un program $P_{n,f}$ care porneste pe registrele goale, scrie numarul n in registrul x_1, apoi lucreaza precum P_f si la oprire a scris valoarea lui $f(n)$ in x_0. In final programul $P_{n,f}$ mai adauga un betigas in registrul x_0. Orice numar poate fi format fintr-o alternanta de programe de tipul $x1 := x1 + 1$ respectiv $x1 := x1 + x1$. Daca lungimea totala a combinatiei acestor doua programe scrise in sintaxa din definitie este k, lungimea programului $P_{n,f}$ este $c + k \log n + 8$. Deci:

$$BB_{\text{TYPE}}(c + 8 + k \log n) > f(n)$$

pentru toti $n \in \mathbb{N}$. Dar pentru un anumit n_f, daca $n \geq n_f$, atunci $n > c + 8 + k \log n$. Deci:

$$BB_{\text{TYPE}}(n) \geq BB_{\text{TYPE}}(c + 8 + k \log n) > f(n)$$

pentru $n \geq n_f$. □

Teorema: *Functia BB_{LOOP} nu este* LOOP *calculabila, dar este o functie calculabila. Functia BB_{WHILE} nu este o functie calculabila.*

[5] Un rezultat asemanator, formulat pentru masini Turing, se gaseste in articolul lui Scott Aaronson, The Busy Beaver Frontier, Electronic Colloquium on Computational Complexity, Report No. 115 (2020)

Demonstratie: Functia Busy Beaver pentru limbajul LOOP nu este LOOP calculabila, conform teoremei precedente. Intr-adevar, daca ar fi LOOP calculabila, de la un rang incolo ar trebui sa fie strict dominata de catre ea insasi, ceea ce este imposibil. Functia Busy Beaver LOOP creste asadar mai repede decat orice functie LOOP calculabila.

Totusi, aceasta functie este intuitiv calculabila. Un posibil algoritm arata in felul urmator. Se genereaza toate cuvintele de lungime $\leq n$ in alfabetul limbajelor LOOP. Daca un asemenea cuvant se dovedeste a fi un program corect scris in LOOP, programul se ruleaza pe configuratia initiala vida a registrelor. Cum orice program LOOP opreste cu orice input, si acest program va opri si va calcula o valoare. Cea mai mare dintre aceste valori este rezultatul functiei Busy Beaver LOOP. Acest algoritm ar putea fi implementat, de exemplu, pe o masina Turing cu mai multe benzi, deci si in WHILE sau GOTO.

Functia Busy Beaver pentru limbajul WHILE nu este WHILE calculabila. Dar cum identificam multimea functiilor WHILE calculabile cu multimea functiilor calculabile in general, tragem concluzia ca Busy Beaver pentru WHILE nu este calculabila deloc. De data aceasta putem spune ca functia Busy Beaver WHILE creste mai repede decat orice functie calculabila. □

In capitolul urmator vom vedea ca functiile LOOP calculabile mai au o caracterizare - cea de functii primitiv recursive, pe cand functiile calculabile sunt acelasi lucru cu functiile recursive sau μ-recursive. Istoric, primul exemplu de functie primitiv recursiva care nu este recursiva a fost dat de Gabriel Sudan in 1927. Cel mai cunoscut exemplu de asemenea functie este functia lui Wilhelm Ackermann din 1928. Functia Busy Beaver LOOP este un exemplu relativ recent.

9 Functii primitiv recursive si recursive

Mai jos sunt definite functiile primitiv recursive. Este vorba despre functii $f : \mathbb{N}^k \to \mathbb{N}$ cu un numar arbitrar de variabile. Principiul acestei definitii este urmatorul: intai sunt mentionate cateva functii simple care fac parte din aceasta clasa, apoi sunt mentionate operatiile cu functii la care este inchisa aceasta clasa sau, mai bine zis, care genereaza aceasta clasa.

Definitie: Functiile primitiv recursive se definesc in modul urmator.

- Toate functiile constante $f : \mathbb{N}^k \to \mathbb{N}$ date de $c(n_1, \dots, n_k) = c$ sunt primitiv recursive.

- Toate proiectiile $\pi^k_m : \mathbb{N}^k \to \mathbb{N}$ date de $\pi^k_m(n_1, \dots, n_m, \dots, n_k) = n_m$ sunt primitiv recursive.

- Functia succesor $s : \mathbb{N} \to \mathbb{N}$ data de $s(n) = n + 1$ este primitiv recursiva.

- Orice compunere de functii primitiv recursive este o functie primitiv recursiva. Mai exact, daca $f : \mathbb{N}^k \to \mathbb{N}$ si functiile $g_1, \dots, g_k : \mathbb{N}^m \to \mathbb{N}$ sunt primitiv recursive, atunci functia $h : \mathbb{N}^m \to \mathbb{N}$ data de:

$$h(n_1, \dots, n_m) := f(g_1(n_1, \dots, n_m), \dots, g_k(n_1, \dots, n_m))$$

este primitiv recursiva.

- Orice functie obtinuta prin operatia de recursie primitiva din functii primitiv recursive este primitiv recursiva. Mai exact, daca $f : \mathbb{N}^k \to \mathbb{N}$ si $g : \mathbb{N}^{k+2} \to \mathbb{N}$ sunt functii primitiv recursuve, atunci functia $h : \mathbb{N}^{k+1} \to \mathbb{N}$ data de:

$$\begin{aligned} h(0, n_1, \dots, n_k) &= f(n_1, \dots, n_k) \\ h(n+1, n_1, \dots, n_k) &= g(h(n, n_1, \dots, n_k), n, n_1, \dots, n_k) \end{aligned}$$

este primitiv recursiva. □

Se observa ca toate functiile primitiv recursive sunt functii total definite. Iata cateva exemple. Functia adunare $add : \mathbb{N}^2 \to \mathbb{N}$ data de $add(n, x) = n + x$ este primitiv recursiva, deoarece se defineste:

$$\begin{aligned} add(0, x) &= x \\ add(n+1, x) &= s(add(n, x)) \end{aligned}$$

La fel functia inmultire $mult : \mathbb{N}^2 \to \mathbb{N}$, $mult(n, x) = n * x$ este primitiv recursiva, cu definitia:

$$\begin{aligned} mult(0, x) &= 0 \\ mult(n+1, x) &= add(mult(n, x), x) \end{aligned}$$

Functia predecesor, $u : \mathbb{N} \to \mathbb{N}$, data de $u(x) = \max(0, n-1)$, este primitiv recursiva, deoarece:

$$\begin{aligned} u(0) &= 0 \\ u(n+1) &= n \end{aligned}$$

Cu ajutorul lui u putem arata ca si scaderea naturala $sub : \mathbb{N}^2 \to \mathbb{N}$, $sub(x, y) = x - y$, este primitiv recursiva:

$$\begin{aligned} sub(x, 0) &= x \\ sub(x, y+1) &= u(sub(x, y)) \end{aligned}$$

Functia $\binom{n}{2} = \frac{n(n-1)}{2}$ este primitiv recursiva deoarece se poate defini prin recursie primitiva bazata pe adunare:

$$\begin{aligned} \binom{0}{2} &= 0 \\ \binom{n+1}{2} &= \binom{n}{2} + n \end{aligned}$$

Folosind proiectiile si compunerea, observam ca si functia urmatoare este primitiv recursiva:

$$c(x, y) = \binom{x+y+1}{2} + x$$

este la randul ei primitiv recursiva. Aceasta functie se numeste functia de imperechere a lui Cantor.

Exercitiu: Sa se arate ca functia de imperechere a lui Cantor $c : \mathbb{N}^2 \to \mathbb{N}$ este o bijectie. □

Folosind aceasta functie, putem codifica tupluri de lungime $k+1$ in numere, pentru orice k dat:

$$\langle n_0, n_1, \dots, n_k \rangle := c(n_0, c(n_1, \dots, c(n_k, 0) \dots)).$$

Aceasta functie este si ea primitiv recursiva. Codificarea obtinuta nu este bijectiva datorita ultimului 0. Ea a fost insa aleasa asa pentru simplitatea functiei inverse.

Fie e si f reciprocele functiei c, definite de relatiile:

$$e(c(x, y)) = x, \quad f(c(x, y)) = y, \quad c(e(n), f(n)) = n.$$

Atunci functia de codificare a tuplurilor are urmatoarele functii inverse (functii de proiectie):

$$\begin{aligned} d_0(n) &= e(n) \\ d_1(n) &= e(f(n)) \\ &\vdots \\ d_k(n) &= e(f(f(\dots f(n) \dots)) \end{aligned}$$

unde litera f apare de k ori.

Urmatoarele randuri au ca scop justificarea faptului ca functiile inverse e si f sunt primitiv recursive.

Definitie: Fie $P \subseteq \mathbb{N}$ un predicat unar. Notatia $P(x)$ inseamna ca $x \in P$. P se numeste primitiv recursiv daca functia lui caracteristica (o notam la fel) $P : \mathbb{N} \to \mathbb{N}$ este primitiv recursiva. □

Definitie: Dat fiind predicatul P, se defineste o functie $q(x)$ cu ajutorul asa-zisului operator max marginit, adica:

$$q(n) = \begin{cases} \max\{x \leq n \mid P(x)\}, & \text{daca acest maxim exista,} \\ 0, & \text{altfel.} \end{cases}$$

Observam ca:

$$\begin{aligned} q(0) &= 0 \\ q(n+1) &= \begin{cases} n+1, & \text{daca } P(n+1), \\ q(n) & \text{altfel.} \end{cases} \end{aligned}$$

ceea ce este echivalent cu

$$\begin{aligned} q(0) &= 0, \\ q(n+1) &= q(n) + P(n+1) * (n+1-q(n)). \end{aligned}$$

Asadar functia q este primitiv recursiva.

Analog se poate defini quantificatorul existential marginit. Dat fiind un predicat $P(x)$, se defineste un nou predicat $Q(n)$ care e adevarat daca si numai daca exista un $x \leq n$ cu proprietatea $P(x)$. Asadar:

$$\begin{aligned} Q(0) &= P(0), \\ Q(n+1) &= P(n+1) + Q(n) - P(n+1) * Q(n). \end{aligned}$$

Asadar, daca P este primitiv recursiv, asa este si Q.

Acum ne putem intoarce la functiile e si f. Observam ca:

$$\begin{aligned} e(n) &= \max\{x \leq n \mid \exists y \leq n : c(x,y) = n\}, \\ f(n) &= \max\{y \leq n \mid \exists x \leq n : c(x,y) = n\}. \end{aligned}$$

Cu aceasta e clar ca ambele functii sunt primitiv recursive.

Teorema: *Clasa functiilor primitiv recursive coincide exact cu clasa functiilor* LOOP *calculabile.*

Demonstratie: $\Leftarrow$ Fie $f : \mathbb{N}^r \to \mathbb{N}$ o functie LOOP calculabila. Fie P acel program LOOP, care calculeaza f. Presupunem fara a restrange generalitatea ca variabilele care apar in P sunt $x_0, x_1, \ldots, x_k$ cu $k \geq r$. Aratam prin inductie dupa structura lui P ca exista o functie primitiv recursiva $g_P : \mathbb{N} \to \mathbb{N}$ care descrie actiunea lui P in urmatorul sens. Daca $a_1, \ldots a_k$ sunt valorile variabilelor la inceput, atunci:

$$g_P(\langle a_0, a_1, \ldots, a_k \rangle) = \langle b_0, b_1, \ldots, b_k \rangle$$

unde $b_0, b_1, \ldots, b_k$ sunt valorile registrelor la oprirea programului.

Daca P are structura $x_i := x_j \pm 1$ atunci:

$$g_P(n) = \langle d_0(n), \ldots, d_{i-1}(n), d_j(n) \pm 1, d_{i+1}(n), \ldots, d_k(n) \rangle.$$

Daca P are forma $Q; R$ atunci $g_P(n) = g_R(g_Q(n))$, unde g_R si g_Q exista datorita ipotezei de inductie.

Daca P are forma LOOP x_i DO Q END, atunci se defineste initial o functie h prin recursie primitiva, dupa cum urmeaza:

$$\begin{aligned} h(0, x) &= x, \\ h(n+1, x) &= g_Q(h(n, x)). \end{aligned}$$

Este clar ca $h(n, x)$ reflecta starea variabilelor $x = \langle x_0, \ldots, x_k \rangle$ dupa n aplicari ale programului Q. Deci functia cautata g_P este in mod natural:

$$g_P(n) = h(d_i(n), n).$$

Cu asta am aratat ca orice functie LOOP calculabila este primitiv recursiva. Mai exact, functia calculata de programul P va fi:

$$f(n_1, \ldots, n_r) = d_0(g_P(\langle 0, n_1, n_2, \ldots, n_r, 0_{r+1}, \ldots, 0_k \rangle)),$$

si este clar o functie primitiv recursiva.

$\Rightarrow$ Cealalta directie se arata prin inductie dupa structura functiilor primitiv recursive. Functiile de baza (constante, proiectii si functia succesor) sunt evident LOOP calculabile. De asemeni compozitia functiilor LOOP calculabile este LOOP calculabila. Singura problema ar fi la recursia primitiva. Daca f este definita prin relatiile:

$$\begin{aligned} f(0, x_1, \ldots, x_r) &= g(x_1, \ldots, x_r) \\ f(n+1, x_1, \ldots, x_k) &= h(f(n, x_1, \ldots, x_k), n, x_1, \ldots, x_k) \end{aligned}$$

atunci f poate fi calculat de urmatorul program LOOP:

$y := g(x_1, \ldots, x_r); k := 0;$
LOOP n DO $y := h(y, k, x_1, \ldots, x_r); k := k + 1$ END.

□

Observam ca toate functiile primitiv recursive sunt functii totale, adica functii definite pentru toate valorile argumentelor lor. O extensie stricta a clasei functiilor primitiv recursive se obtine prin adaugarea operatorului μ.

Definitie: Fie $f : \mathbb{N}^{k+1} \to \mathbb{N}^k$ o functie, eventual partiala. Operatorul μ aplicat lui f este o functie $\mu f = g : \mathbb{N}^k \to \mathbb{N}$ data de:

$$g(x_1, \ldots, x_k) = \min\{n \,|\, f(n, x_1, \ldots, x_n) = 0 \text{ si } \forall\ m < n,\ f(m, x_1, \ldots, x_k) \text{ este definita}\}.$$

In aceasta constructie, $\min \emptyset$ este o valoare nedefinita. □

Prin urmare, ca urmare a aplicarii operatorului μ unor functii totale, pot rezulta functii partiale. De exemplu, daca operatorul μ se aplica functiei totale constante $f(x, y) = 1$, se obtine functia partiala Ω, care nu este definita nicaieri.

Definitie: Clasa functiilor μ-recursive (denumite si functii recursive) este cea mai mica clasa de functii (eventual partiale) care functiile constante, proiectiile si functia succesor si care este inchisa la compunerea de functii, la recursia primitiva si la actiunea operatorului μ. □

Teorema: *Clasa functiilor μ-recursive coincide exact cu clasa functiilor* WHILE *calculabile.*

Demonstratie: Trebuie sa adaugam operatorul μ in demonstratia precedenta. Daca P este un program WHILE de forma:

$$\text{WHILE } x_i \neq 0 \text{ DO } Q \text{ END}$$

outem defini o functie h astfel incat valoarea $h(n, x)$ reflecta starea variabilelor $\langle x_0, \ldots, x_k \rangle$ dupa ce Q a rulat de n ori pe masina cu registre. Apoi definim:

$$g_P(x) = (\mu(d_i(h))(x), x).$$

Observam ca $\mu(d_i(h))(x)$ este numarul minim de repetitii ale lui Q pana cand x_i ia valoarea 0.

Reciproc, fie $g = \mu f$ unde functia f este calculata de un program WHILE. Atuncui urmatorul program calculeaza functia g:

$$
\begin{array}{l}
x_0 = 0; y := f(0, x_1, \ldots, x_n); \\
\text{WHILE } y \neq 0 \text{ DO} \\
\quad x_0 := x_0 + 1; y := f(x_0, x_1, \ldots, x_n); \\
\text{END}
\end{array}
$$

□

Aceasta demonstratie are si o implicatie neasteptata:

Teorema: *(Kleene) Pentru orice functie recursiva f de aritate n exista doua functii primitiv recursive p si q de aritate $n+1$ astfel incat f se reprezinta sub forma:*

$$f(x_1, \ldots, x_n) = p(x_1, \ldots, x_n, (\mu q)(x_1, \ldots, x_n)).$$

Demonstratie: Orice functie recursiva se calculeaza cu ajutorul unui program WHILE care contine o singura data cuvantul WHILE. Daca acest program este transformat in functie recursiva, apare o reprezentare de aceasta forma. □

Part III
Problema Opririi

10 Multimi recursiv enumerabile

Notiunea de calculabilitate a fost definita pentru functii. Vom introduce aici notiunea corespunzatoare pentru multimi de cuvinte, respectiv pentru sumbultimile lui $\mathbb{N}$.

Definitie: O submultime $A \subseteq \Sigma^*$ se numeste decidabila, daca functia $\chi_A : \Sigma^* \to \{0,1\}$ ei caracteristica este calculabila. Asta inseamna ca pentru toate $w \in \Sigma^*$,

$$\chi_A(w) = \begin{cases} 1, & w \in A, \\ 0, & w \notin A. \end{cases}$$

O submultime $A \subseteq \Sigma^*$ se numeste semidecidabila daca functia partiala $\chi'_A : \Sigma^* \to \{0,1\}$ este calculabila:

$$\chi'_A(w) = \begin{cases} 1, & w \in A, \\ \text{nedefinita} & w \notin A. \end{cases}$$

O multime semidecidabila este asadar multimea de valori pentru care o masina Turing (sau program WHILE, program GOTO, functie μ-recursiva) se opreste. In cazul multimii decidabile, dimpotriva, exista un algoritm de decizie care se opreste pentru orice input posibil si da intotdeauna raspunsul adecvat.

Exact aceleasi definitii sunt valabile si pentru submultimile lui $\mathbb{N}$. □

Teorema: *O multime A este decidabila daca si numai daca atat multimea A cat si complementara ei $\overline{A}$ sunt semidecidabile. Aici complementara se refera la Σ^* sau la $\mathbb{N}$, in functie de multimea in care traieste A.*

Demonstratie: Este clar ca daca o multime este decidabila atunci atat ea cat si complementara ei sunt semidecidabile. Pentru directia cealalta: Fie T_1 si T_0 masini Turing cu o banda astfel incat A este multimea de valori pentru care T_1 se opreste iar $\overline{A}$ este multimea de valori pentru care T_0 se opreste. Se poate construi usor o masina Turing cu doua benzi T astfel incat capul de pe banda 1 o simuleaza pe T_1, capul de pe banda 0 o simuleaza pe T_0 si cele doua capete muta alternativ. In momentul in care unul dintre capete ajunge intr-o stare finala, intreaga masina se opreste si da output 1 in cazul in care T_1 s-a oprit, respectiv 0 daca T_0 s-a oprit. □

Definitie: O multime $A \subseteq \Sigma^*$ respectiv $A \subseteq \mathbb{N}$ se numeste recursiv enumerabila daca $A = \emptyset$ sau daca exista o functie calculabila totala $f : \Sigma^* \to \mathbb{N}$ respectiv $f : \mathbb{N} \to \mathbb{N}$ astfel incat

$$A = \{f(0), f(1), f(2), \dots\}.$$

Nu este insa exclusa posibilitatea ca f sa enumere A cu repetitii, adica pentru $i \neq j$ sa avem $f(i) = f(j)$. □

Teorema: *O multime nevida este recursiv enumerabila daca si numai daca este semidecidabila.*

Demonstratie: Fara a restrange generalitatea, schitam demonstratia doar pentru submultimile lui $\mathbb{N}$. Directia de la stanga la dreapta este simpla. Fie $f : \mathbb{N} \to \mathbb{N}$ o functie calculabila total definita care o enumera pe A. Iata un program care opreste doar daca inputul x este in A:

1 : INPUT (x);

2: $n := 0$;

3: IF $f(n) = x$ THEN OUTPUT 1 END

4: $n := n + 1$;

5: GOTO 3

Pentru directia cealalta ne reamintim functia de imperechere a lui Cantor $c(x, y) = n$ si inversele ei $d(n)$ si $e(n)$. Cum functia c este bijectiva, inseamna ca $(d(n), e(n))$ parcurge toate perechile de numere naturale cand n il parcurge pe n. Fie A o multime semidecidabila si M o masina Turing cu o banda astfel incat $A = T(M)$. Fixam un element $a \in A$. Urmatoarea masina Turing

M' calculeaza o functie total definita care are ca imagine multimea A. M' primeste o valoare n, calculeaza $x = d(n)$ si $y = e(n)$, apoi simuleaza masina M cu input x cel mult y pasi. (Observati ca masina M' trebuie sa isi numere proprii pasi. Acest lucru se realizeaza usor daca masina M' este o masina Turing cu mai multe benzi.) Daca aceasta simulare ajunge intr-o stare finala a masinii M, atunci M' returneaza x, altfel M' returneaza a. □

Concluzie: Daca $A \subseteq \Sigma^*$ sau $A \subseteq \mathbb{N}$, si $A \neq \emptyset$, atunci sunt echivalente:

1. A este recursiv enumerabila.
2. A este semidecidabila.
3. A este de tip 0.
4. $A = T(M)$ pentru o masina Turing M.
5. χ'_A este o functie partiala calculabila.
6. A este domeniul de definitie al unei functii partiale calculabile.
7. A este imaginea unei functii calculabile totale.

□

In paragraful urmator vom vedea ca exista multimi recursiv enumerabile care nu sunt decidabile (nu sunt recursive).

11 Self-reference Problem

Pentru a produce o multime semidecidabila dar nedecidabila, trebuie sa asociem fiecarei masini Turing cu o banda un nume (un cod). Desi nu este foarte important cum se face acest lucru, prezentam acum o metoda simpla. Elementele din $\Gamma = \{a_0, \ldots, a_k\}$ si elementele din $Z = \{z_0, \ldots, z_n\}$ sunt numerotate. Sunt fixate numerele simbolurilor □, 0, 1, #, al starii initiale si ale starilor finale. Fiecarei δ-reguli de forma:

$$\delta(z_i, a_j) = (z_{i'}, a_{j'}, y).$$

i se asociaza cuvantul

$$w_{i,j,i',j'y} = \#\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m),$$

unde $bin(x)$ este reprezentarea binara a lui x iar $m = 0$ daca $y = L$, $m = 1$ daca $y = R$, $m = 2$ daca $y = N$.

Acestui cuvant i se poate in sfarsit asocia un cuvant peste alfabetul $\{0, 1\}$ daca recurgem la codificarea $0 \to 00$, $1 \to 01$ si $\# \to 11$. Este clar, ca nu orice cuvant binar va fi codul unei masini Turing. Acest lucru se poate face insa prin conventie. Pentru aceasta fixam o masina Turing $\overline{M}$ si definim pentru orice cuvant $w \in \{0, 1\}^*$:

$$M_w = \begin{cases} M, & \text{daca } w \text{ este codul lui } M, \\ \overline{M}, & \text{in caz contrar.} \end{cases}$$

Definitie: Problema Recunoasterii Propriului Nume (sau Self-reference Problem) este urmatoarea multime:

$$K = \{w \in \{0, 1\}^* \mid w \in T(M_w)\}.$$

Intrebarea asociata este urmatoarea: *Care sunt masinile Turing care isi recunosc propriul nume, in sensul ca ating o stare finala in timpul prelucrarii acestui cuvant?* □

Teorema: *Problema Recunoasterii Propriului Nume este nedecidabila.*

Demonstratie[6]: Sa presupunem ca problema K este decidabila. Atunci functia caracteristica χ_K este calculabila si exista o masina Turing M care calculeaza aceasta functie. Aceasta masina M poate fi transformata usor intr-o masina M' care are urmatorul comportament: Daca M returneaza 0, M' returneaza 0 si se opreste. Daca M returneaza 1, M' intra intr-o bucla infinita si nu se opreste niciodata. Fie w' numele (codul) lui M'.

$$\begin{aligned} M'(w') \text{ opreste} \quad &\Leftrightarrow \quad M(w') = 0 \\ &\Leftrightarrow \quad \chi_K(w') = 0 \\ &\Leftrightarrow \quad w' \notin K \\ &\Leftrightarrow \quad M_{w'}(w') \text{ nu opreste} \\ &\Leftrightarrow \quad M'(w') \text{ nu opreste} \end{aligned}$$

Aceasta contradictie arata ca multimea K nu poate fi decidabila. □

Pentru a arata ca alte probleme sunt nedecidabile, se recurge la reductii.

Definitie: Fie $A \subseteq \Sigma^*$ si $B \subseteq \Gamma^*$ limbaje. Spunem ca A se reduce la B si scriem $A \leq B$ daca exista o functie calculabila totala $f : \Sigma^* \to \Gamma^*$ astfel incat pentru toti $x \in \Sigma^*$ are loc:

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

Bineinteles $\mathbb{N}$ il poate inlocui in definitie pe Σ^*, pe Γ^* sau pe ambele[7]. □

Lema: *Daca $A \leq B$ si B este decidabila, atunci A este decidabila. Daca $A \leq B$ si B este semidecidabila, atunci A este semidecidabila.*

Demonstratie: Presupunem ca f este functia totala calculabila care il reduce pe A la B. Daca χ_B calculabila, atunci si $\chi_B \circ f$ este calculabila. Dar in virtutea echivalentei din definitie, pentru orice x, $\chi_A(x) = \chi_B(f(x))$, χ_A este calculabila si A este decidabila. Acelasi rationament in cazul in care B este semidecidabila. □

Definitie: Problema (generala a) Opririi este multimea:

$$H = \{w\#x \,|\, M_w(x) \text{ opreste }\}.$$

□

Teorema: *Problema Opririi H este nedecidabila.*

Demonstratie: Este suficient sa aratam $K \leq H$. Alegem $f(w) = w\#w$. Atunci $w \in K \Leftrightarrow f(w) \in H$. □

Definitie: Problema Opririi Pe Banda Goala este multimea:

$$H_0 = \{w \,|\, M_w(\square) \text{ opreste }\}.$$

□

Teorema: *Problema Opririi Pe Banda Goala H_0 este nedecidabila.*

Demonstratie: Aratam ca $H \leq H_0$. Fiecarui cuvant $w\#x$ ii asociem o masina Turing M dupa cum urmeaza: M porneste cu banda goala, scrie x, dupa care se comporta ca M_w cu input x. Fie $f(w\#x) = code(M)$. Este evident ca:

$$w\#x \in H \Longleftrightarrow f(w\#x) \in H_0.$$

□

[6] Aceasta demonstratie se aseamana cu demonstratia faptului ca pentru orice multime A, $A < P(A)$. Metoda se numeste *diagonalizare*.

[7] Asemenea indicatii sunt in realitate superflue, deoarece daca $\Sigma = \{|\}$ este alfabetul cu o litera, atunci $\Sigma^* = \mathbb{N}$. Totusi voi repeta din cand in cand genul acesta de indicatie pentru a sublinia faptul ca nu este nicio diferenta intre calculabilitatea cu numere si calculabilitatea cu cuvinte.

12 Teorema lui Rice

Cele trei probleme nedecidabile din sectiunea precedenta se refera la proprietati ale masinilor Turing. Teorema urmatoare arata ca aproape orice proprietate a masinilor Turing este nedecidabila.

Teorema: *(Rice) Fie $\mathcal{R}$ clasa tuturor functiilor Turing calculabile, inclusiv cele partial definite. Fie $\mathcal{S} \subset \mathcal{R}$ o submultime oarecare, astfel incat $\mathcal{S} \neq \emptyset$ si $\mathcal{S} \neq \mathcal{R}$. Atunci multimea:*

$$C(\mathcal{S}) = \{w \,|\, \textit{functia calculata de } M_w \textit{ este in } \mathcal{S}\}.$$

este nedecidabila.

Demonstratie: Fie $\Omega \in \mathcal{R}$ functia total nedefinita. Ω poate fi in $\mathcal{S}$ sau nu.

Cazul 1: $\Omega \in \mathcal{S}$.

Cum $\mathcal{S} \neq \mathcal{R}$, exista o functie $q \in \mathcal{R} \setminus \mathcal{S}$. Fie Q o masina Turing care o calculeaza pe q. Fiecarui cuvant $w \in \{0,1\}^*$ ii ordonam o masina Turing M care se comporta in modul urmator:

> M este pornita pe un input y. M ignora initial acest input si se comporta precum M_w pornita pe banda goala. Daca acest calcul atinge o stare finala, atunci M se comporta precum Q cu input y.

Fie g functia calculata de masina M.

$$g = \begin{cases} \Omega, & \text{daca } M_w \text{ nu opreste pe banda goala,} \\ q, & \text{altfel.} \end{cases}$$

Functia $f(w) = \mathit{code}(M)$ este total definita si calculabila. Au loc urmatoarele implicatii:

$$\begin{aligned} w \in H_0 &\Rightarrow M_w \text{ opreste pe banda goala} \\ &\Rightarrow M \text{ calculeaza functia } q \\ &\Rightarrow \text{ functia calculata de } M_{f(w)} \text{ nu este in } \mathcal{S} \\ &\Rightarrow f(w) \notin C(\mathcal{S}) \end{aligned}$$

Reciproc este valabil:

$$\begin{aligned} w \notin H_0 &\Rightarrow M_w \text{ nu opreste pe banda goala} \\ &\Rightarrow M \text{ calculeaza functia } \Omega \\ &\Rightarrow \text{ functia calculata de } M_{f(w)} \text{ este in } \mathcal{S} \\ &\Rightarrow f(w) \in C(\mathcal{S}) \end{aligned}$$

Cu alte cuvinte functia f este o reductie $\overline{H_0} \leq C(\mathcal{S})$. Cum H_0 este nedecidabila, asa este si $\overline{H_0}$, deci $C(\mathcal{S})$ este nedecidabila.

Cazul 2: $\Omega \in \mathcal{S}$.

In acest caz se demonstreaza analog $H_0 \leq C(\mathcal{S})$. □

Teorema lui Rice are multiple aplicatii. De exemplu, nu este decidabil daca o masina Turing calculeaza o functie total definita. Nu este decidabil daca o masina Turing calculeaza o functie constanta, o functie marginita, o functie nemarginita, etc.

Exista probleme algoritmice care sunt *si mai nedecidabile* decat Problema Opririi. Fie Problema Echivalentei pentru Masini Turing:

$$E = \{u\#v \,|\, M_u \text{ calculeaza aceeasi functie precum } M_v\}.$$

Atunci $H \leq E$ dar $E \not\leq H$. In realitate se poate defini un sir infinit de probleme A_1, A_2, A_3, ... astfel incat mereu $A_i < A_{i+1}$. Se vorbeste despre grade de nerezolvabilitate.

13 Problema Corespondentei (Post)

Post's Correspondence Problem, prescurtat PCP, are urmatoarea definitie:

Se da: Un sir finit de perechi de cuvinte (x_1, y_1), (x_2, y_2), ..., (x_k, y_k), unde toti $x_i, y_i \in \Sigma^+$.

Se cere: Exista $n \geq 1$ si un sir de numere naurale $i_1, \ldots, i_n \in \{1, 2, \ldots, k\}$ astfel incat:

$$x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n} \text{ ?}$$

Exemplu: Instanta

$$C = ((1, 101), (10, 00), (011, 11))$$

adica $x_1 = 1$, $x_2 = 10$, $x_3 = 011$, $y_1 = 101$, $y_2 = 00$ si $y_3 = 11$, are o solutie $(1, 3, 2, 3)$ deoarece:

$$x_1 x_3 x_2 x_3 = 101110011 = y_1 y_3 y_2 y_3.$$

Observatie: PCP este recursiv enumerabila (echivalent, semidecidabila). Intr-adevar, se pot incerca toate combinatiile de lungine 1, apoi toate combinatiile de lungime 2, apoi toate combinatiile de lungime 3, etc. Daca la un moment dat se gaseste o solutie, procedura se opreste. $\Box$

Pentru a arata ca PCP este nedecidabila, o vom reduce intai la o problema inrudita, MPCP (modified PCP), dupa care vom reduce MPCP la Problema Opririi. In problema MPCP se cauta solutii $i_1, \ldots, i_n$ unde $i_1 = 1$.

Lema: MPCP $\leq$ PCP.

Demonstratie: Fie \$ si # litere noi, care nu apar in alfabetul Σ in care este formulata o instanta a problemei MPCP. Pentru un cuvant $w = a_1 a_2 \ldots a_n \in \Sigma^+$ definim:

$$\begin{aligned} cw &= \#a_1\#a_2\#\ldots\#a_m\# \\ sw &= \#a_1\#a_2\#\ldots\#a_m \\ ew &= a_1\#a_2\#\ldots\#a_m\# \end{aligned}$$

Fiecarei instante $C = ((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k))$ a lui MPCP ii asociem urmatoarea instanta:

$$f(C) = ((cx_1, sy_1), (ex_1, sy_1), (ex_2, sy_2), \ldots, (ex_k, sy_k), (\$, \#\$)).$$

Functia f este evident calculabila. Aratam ca f este o reductie de la MPCP la PCP. Mai exact aratam: C are o solutie cu $i_1 = 1$ daca si numai daca $f(C)$ are o solutie.

Daca C are o solutie $(1, i_2, \ldots, i_n)$ atunci $(1, i_2 + 1, \ldots, i_n + 1, k + 2)$ este o solutie a lui $f(C)$.

Daca $f(C)$ are o solutie $i_1, \ldots, i_n \in \{1, \ldots, k + 2\}$, neaparat $i_1 = 1$, $i_n = k + 2$, si ceilalti indici sunt in multimea $\{2, \ldots, k + 1\}$. In acest caz $(1, i_2 - 1, \ldots, i_{n-1} - 1)$ este o solutie pentru C. $\Box$

Lema: H $\leq$ MPCP.

Demonstratie: Fie o masina Turing $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$ o masina Turing si un input $w \in \Sigma^*$. Prezentam o metoda algoritmica de generare a unui sir de perechi (x_1, y_1), ..., (x_k, y_k) astfel incat:

$$M(w) \text{ opreste } \iff (x_1, y_1), \ldots, (x_k, y_k) \text{ are o solutie cu } i_1 = 1.$$

Alfabetul instantei MPCP pe care o construim este $\Gamma \cup Z \cup \{\#\}$. Prima pereche de cuvinte este $(\#, \#z_0 w\#)$. Solutia trebuie sa inceapa cu aceasta pereche. Celelalte perechi se impart in niste grupe dupa cum urmeaza:

Reguli de copiere: (a, a) pentru toate $a \in \Gamma \cup \{\#\}$.

Reguli de tranzitie:

$$\begin{aligned} (za, z'c) &\quad \text{daca } \delta(z, a) = (z', c, N) \\ (za, cz') &\quad \text{daca } \delta(z, a) = (z', c, R) \\ (bza, z'bc) &\quad \text{daca } \delta(z, a) = (z', c, L), \text{ pentru toti } b \in \Gamma \end{aligned}$$

$$\begin{array}{rl} (\#za, \#z'\Box c) & \text{daca } \delta(z,a) = (z', c, L) \\ (z\#, z'c\#) & \text{daca } \delta(z,\Box) = (z', c, N) \\ (z\#, cz'\#) & \text{daca } \delta(z,\Box) = (z', c, R) \\ (bz\#, z'bc\#) & \text{daca } \delta(z,\Box) = (z', c, L), \text{ pentru toti } b \in \Gamma \end{array}$$

Reguli de stergere: (az_e, z_e), $(z_e a, z_e)$ pentru toate $a \in \Gamma$ si $z_e \in E$.

Regula finala: $(z_e\#\#, \#)$ pentru toate $z_e \in E$.

Daca masina Turing M opreste cu input w, exista un sir de configuratii $(k_0, k_1, \ldots k_t)$ cu $k_0 = z_0 w$, $k_t = uz_e v$ cu $u, v \in \Gamma^*$ si $z_e \in E$ si $k_i \vdash k_{i+1}$ pentru $i = 0, 1, \ldots, t-1$. Problema MPCP are atunci o solutie de forma:

$$\#k_0\#k_1\#\ldots\#k_t\#k_t'\#k_t''\#\ldots\#z_e\#\#.$$

Cuvintele k_t', k_t'', $\ldots$ rezulta din $k_t = uz_e v$ prin stergerea succesiva a simbolurilor in jurul lui z_e. In timpul constructiei solutiei, cuvantul format din x_{i_j} este mereu cu o configuratie in urma cuvantului format din y_{i_j}.

Metoda de constructie prezentata este o reductie de la H la MPCP. $\Box$

Teorema: *Problema Corespondentei lui Post este nedecidabila.*

Demonstratie: In lemele anterioare am vazut ca H $\leq$ MPCP $\leq$ PCP. Deci H $\leq$ PCP. Cum H este nedecidabila, asa este si PCP. $\Box$

Definitie: Notam cu 01PCP varianta Problemei Corespondentei peste alfabetul $\{0, 1\}$. $\Box$

Teorema: *Problema* 01PCP *este nedecidabila.*

Demonstratie: Aratam ca PCP $\leq$ 01PCP. Fie $\Sigma = \{a_1, \ldots, a_m\}$ alfabetul in care e scrisa o instanta a problemei PCP. Fiecarei litere a_j din Σ ii asociem stringul $\hat{a_j} = 01^j$. Pentru un cuvant $w = x_1 \ldots x_n \in \Sigma^+$ fie $\hat{w} = \hat{b_1} \ldots \hat{b_n}$. Este clar ca (x_1, y_1), $\ldots$, (x_n, y_n) are o solutie daca si numai daca $(\hat{x_1}, \hat{y_1})$, $\ldots$, $(\hat{x_n}, \hat{y_n})$ are o solutie. $\Box$

14 Masina Turing Universala

Am aratat ca H $\leq$ PCP si am vazut anterior ca PCP este recursiv enumerabila. Asadar si H este recursiv enumerabila. Asta inseamna urmatorul lucru: exista o masina Turing care, daca primeste un input $w\#x$, va opri daca si numai daca masina M_w opreste cand calculeaza inputul x. De fapt se poate demonstra mai mult, si anume existenta unei masini Turing universale.

Definitie: O masina Turing universala U este o masina Turing care pentru fiecare input de forma $w\#x$ simuleaza functionarea masinii M_w pentru inputul x. $\Box$

Teorema: *(Turing) Exista masini Turing universale.*

Demonstratie[8]: Masina universala pe care o descriem aici este o masina Turing cu mai multe benzi. In final se poate aplica teorema de echivalenta si se poate trage concluzia ca exista masini Turing universale cu o singura banda. Presupunem ca masina Turing care trebuie simulata are un alfabet de lucru de n litere si un numar de m stari. Literele se codifica in stringuri de forma $\#code(a_i)$ iar starile in stringuri de forma $code(z_j)$, unde aceste coduri sunt cuvinte binare iar separatorul $\#$ indica inceputul unei noi litere.

In procesul de simulare sunt folosite mai multe benzi dupa cum urmeaza:

Banda C contine codul masinii care este simulata. Aceasta banda este read only. Codul este dat de o succesiune de tranzitii δ scrise sub forma:

$$\#\#code(z)\#code(a)\#code(z')\#code(a')\#code(m)$$

[8] Folclor.

unde fara a restrange generalitatea presupunem ca toate starile sunt codificate cu stringuri binare de aceeasi lungime si la fel toate literele sunt codificate cu stringuri binare de aceeasi lungime.

Banda S este banda pe care are loc simularea. Daca la un moment dat masina M_w are pe banda cuvantul $a_1 \dots \boxed{a_i} \dots a_n$ cu capul de citire-scriere centrat pe a_i, pe banda S a masinii U se citeste

$$\#code(a_1) \dots \boxed{\#} code(a_i) \dots \#code(a_n).$$

Banda Z este banda pe care masina U retine starea actuala z a masinii M_w. La orice schimbare de stare a masinii M_w, cuvantul de pe banda Z este inlocuit: $code(z)$ este sters si inlocuit cu $code(z')$.

Cum are loc simularea: capul de pe banda C cauta perechea (z, a) corespunzatoare literei scanate pe banda S si a starii notate pe banda Z. Capetele S si Z scaneaza codurile respective de mai multe ori in timpul cautarii pentru a gasi tranzitia potrivita. In momentul in care ea a fost gasita, se inlocuieste starea de pe banda Z si litera corespuunzatoare de pe banda S, dupa care capul de pe banda S face miscarea m care poate fi la dreapta, la stanga sau pe loc.

Important este faptul ca aceasta simulare se poate face cu un numar finit de stari, care nu depinde de lungimea codurilor pentru litere si stari simulate. □

Masina care se obtine din transformarea masinii de mai sus intr-o masina cu o banda nu este performanta, si este departe de a avea cel mai mic numar posibil de stari sau de litere. Aceasta este o doar o demonstratie pentru existenta masinii universale. Marvin Minsky a descoperit in 1962 o masina universala cu 7 stari si 4 litere. Yurii Rogozhin a mers mai departe si a gasit masini cu urmatoarele caracteristici (numar de stari, numar de litere): (15, 2), (9, 3), (6, 4), (5, 5), (4, 6), (3, 9) si (2, 18). In anul 2007 studentul Alex Smith (20 de ani) de la Universitatea din Birmingham a castigat Premiul Wolfram de 25.000 de $ aratand ca o masina cu 2 stari si 3 litere este universala si, mai mult, este cea mai mica masina Turing universala posibila. Demonstratia este foarte complexa, si anumite detalii inca sunt in dezbatere.

Ideea lui Turing - existenta unei masini universale - a prefigurat aparitia calculatorului in sine si a compilerelor si interpreterelor pentru diferite limbaje de programare. Notiunea de completitudine Turing descrie capacitatea unui sistem de a fi programabil si universal. Masinile Turing, programele WHILE si GOTO, functiile recursive, sunt sisteme Turing complete. Un sistem de calcul este Turing complet daca poate simula o masina Turing universala si poate fi simulat de o masina Turing universala.

15 Teorema lui Gödel

Teorema de Incompletitudine a lui Gödel spune ca orice tentativa de a axiomatiza Aritmetica este incompleta. Cu alte cuvinte, vor exista intotdeauna propozitii adevarate in structura algebrica $(\mathbb{N}, +, \cdot)$ care nu sunt demonstrabile in sistemul de axiome expus. Vom vedea ca teorema are o demonstratie alternativa in teoria calculabilitatii, diferita de demonstratia initiala a lui Gödel. Totusi, descoperirea teoremei in 1935 a fost un pas important catre actuala teorie a calculabilitatii.

Pentru inceput vom defini inductiv notiunile de termen si formula.

Definitie: Notiunea de termen este definita dupa cum urmeaza:

1. Orice numar $n \in \mathbb{N}$ este un termen.
2. Orice variabila x_i, $i \in \mathbb{N}$ este un termen.
3. Daca t_1 si t_2 sunt termeni, atunci si $(t_1 + t_2)$ respectiv $(t_1 \cdot t_2)$ sunt termeni.

□

Definitie: Notiunea de formula este definita dupa cum urmeaza:

1. Daca t_1 si t_2 sunt termeni, atunci $(t_1 = t_2)$ este o formula.
2. Daca F si G sunt formule, atunci $\neg F$, $(F \wedge G)$ si $(F \vee G)$ sunt formule.
3. Daca x este o variabila si F este o formula, atunci $\exists x\, F$ si $\forall x\, F$ sunt formule.

$\square$

Definitie: Fie V multimea variabilelor. O functie $\varphi : V \to \mathbb{N}$ se numeste evaluare. Actiunea lui φ se poate extinde de la variabile la termeni, dupa cum urmeaza:

$$\begin{aligned} \varphi(n) &= n \\ \varphi((t_1 + t_2)) &= \varphi(t_1) + \varphi(t_2) \\ \varphi((t_1 \cdot t_2)) &= \varphi(t_1) \cdot \varphi(t_2) \end{aligned}$$

$\square$

Definitie: O formula care nu contine variabile libere se numeste propozitie. $\square$

Definitie: Pentru formule aritmetice definim notiunea de adevar dupa cum urmeaza:

1. $(t_1 = t_2)$ este adevarata, daca pentru orice evaluare $\varphi : V \to \mathbb{N}$, $\varphi(t_1) = \varphi(t_2)$. In particular, rezulta ca formula $x = x$ este adevarata, desi nu este o propozitie.
2. $\neg F$ este adevarata, daca F nu este adevarata.
3. $(F \wedge G)$ este adevarata, daca atat F cat si G sunt adevarate.
4. $(F \vee G)$ este adevarata, daca F este adevarata sau G este adevarata.
5. $\exists x\ F$ este adevarata, daca pentru un $n \in \mathbb{N}$, $F(x/n)$ este adevarata. Aici x/n inseamna ca orice aparitie libera a lui x este inlocuita cu constanta n.
6. $\forall x\ F$ este adevarata, daca pentru orice $n \in \mathbb{N}$, $F(x/n)$ este adevarata.

$\square$

Exemple:

$$\forall x\ \exists y\ ((x + y) = (x \cdot (x + 1))$$

este o formula (propozitie) adevarata, deoarece pentru orice $x \in \mathbb{N}$ putem alege $y = x \cdot x$.

$$\forall x\ \exists y\ ((x = 0) \vee ((x \cdot y) = 1))$$

este o formula (propozitie) falsa in multimea numerelor naturale. Ea este adevarata peste alte multimi in care s-a definit o operatie de inmultire, cum ar fi cea a numerelor rationale sau cea a numerelor reale. $\square$

Definitie: O functie $f : \mathbb{N}^k \to \mathbb{N}$ se numeste functie definibila in limbajul aritmeticii daca exista o formula aritmetica $F(x_1, \dots, x_k, y)$ cu proprietatea ca pentru toate valorile $n_1, \dots, n_k, m \in \mathbb{N}$,

$$f(n_1, \dots, n_k) = m \Leftrightarrow F(n_1, \dots, n_k, m).$$

$\square$

Exemple: Functia de adunare este definita de formula:

$$y = x_1 + x_2$$

si analog este definita functia de inmultire. Daca se introduce relatia:

$$a < b :\longleftrightarrow \exists z\ a + z + 1 = b,$$

functia DIV este definita de formula:

$$\exists r\ (r < x_2) \wedge (x_1 = y \cdot x_2 + r),$$

iar functia MOD este definita de formula:

$$\exists k\ (y < x_2) \wedge (x_1 = k \cdot x_2 + y).$$

□

Definitie: Pentru o mai usoara scriere a formulelor, introducem urmatoarele notatii:

$$\forall x < k(\dots) \text{ pentru } \forall x\ (\neg(x < k) \vee (\dots)),$$

$$\exists x < k(\dots) \text{ pentru } \exists x\ ((x < k) \wedge (\dots)).$$

Cuantificatorii introdusi se numesc cuantificatori marginiti. □

In cele ce urmeaza, ne ocupam cu codificarea sirurilor finite de numere naturale. Un asemenea sir $(n_0, \dots, n_k)$ se poate obtine dintr-o pereche de numere (a, b) in modul urmator:

$$n_i = a \text{ MOD } (1 + (i+1) \cdot b).$$

Aceasta relatie poate fi reprezentata printr-o formula aritmetica. Formula:

$$G(a, b, i, y) := y < (1 + (i+1) \cdot b) \wedge \exists k\ (a = y + k \cdot (1 + (i+1) \cdot b))$$

este adevarata daca si numai daca $y = a \text{ MOD } (1 + (i+1) \cdot b)$.

Lema: *Pentru orice sir finit* $(n_0, \dots, n_k) \in \mathbb{N}^k$ *exista* $a \in \mathbb{N}$ *si* $b \in \mathbb{N}$ *astfel incat pentru orice* $i = 0, 1, \dots, k$ *are loc:*

$$n_i = a \text{ MOD } (1 + (i+1)b).$$

Demonstratie: Fie $s = \max(k, n_0, \dots, n_k)$ si alegem $b = s!$. Aratam intai ca numerele $b_i = 1 + (i+1) \cdot b$ sunt doua cate doua prime intre ele. Fie $i < j$ si presupunem ca exista un numar prim p care il divide atat pe b_i cat si pe b_j. Atunci p divide diferenta $b_j - b_i = (j-i)b$. Dar cum $(j-i) \leq k \leq s$ rezulta ca $(j-i)$ il divide pe b. Deci p il divide pe b. Dar cum p il divide pe b_i, p divide 1, contradictie.

In continuare aratam ca pentru fiecare doua numere a si a' cu $0 \leq a < a' < b_0 b_1 \dots b_k$ solutiile sistemelor de congruente:

$$\begin{aligned} n_i &= a \text{ MOD } b_i \ \ (i = 0, \dots, k) \\ n'_i &= a' \text{ MOD } b_i \ \ (i = 0, \dots, k) \end{aligned}$$

sunt diferite. Presupunem ca $(n_0, \dots, n_k) = (n'_0, \dots, n'_k)$. Atunci fiecare b_i divide numarul $a' - a$. Cum numerele b_i sunt prime intre ele, rezulta ca produsul $b_0 b_1 \dots b_k$ divide $a' - a$. Dar $|a' - a| < b_0 b_1 \dots b_k$, deci $a' = a$. Contradictie.

Numerele $a \in \{0, \dots, b_0 b_1 \dots b_k - 1\}$ genereaza $b_0 b_1 \dots b_k$ sisteme de solutii $(n_0, \dots, n_k)$ cu proprietatea ca $b_i < n_i$ pentru toti i. Deci fiecare sir finit apare ca solutie a sistemului de congruente. Deci exista un a care livreaza exact sirul dat. □

Teorema: *Orice functie calculabila este aritmetic definibila.*

Demonstratie: Pentru functiile calculabile alegem metoda de calcul data de limbajul WHILE. Aratam ca pentru orice program WHILE P, care contine variabilele $x_0, \dots, x_k$, exista o formula F_P care contine variabilele libere $x_0, \dots, x_k$ si $y_0, \dots, y_k$ astfel incat pentru toate $m_i, n_i \in \mathbb{N}$:

$$F_P(m_0, \dots, m_k, n_0, \dots, n_k) \Longleftrightarrow P(m_0, \dots, m_k) = (n_0, \dots, n_k),$$

adica programul P pornit cu valorile $(n_0, \dots, n_k)$ in registre, va opri la un moment dat si va lasa in registre valorile $(n_0, \dots, n_k)$. Odata ce vom fi demonstrat acest lucru, daca P calculeaza o functie $f : \mathbb{N}^n \to \mathbb{N}$ cu $n \le k$. atunci f este definita de urmatoarea formula:

$$F(x_1, \dots, x_n, y) = \exists w_1 \exists w_2 \dots \exists w_k \; F_P(0, x_1, \dots, x_n, 0_{n+1}, \dots, 0_k, y, w_1, \dots, w_k).$$

Existenta formulei F_P se demonstreaza inductiv dupa structura programelor WHILE.

Daca P are forma $x_i := x_j + 1$, atunci:

$$F_P = (y_i = x_j + 1) \wedge \bigwedge_{l \neq i} (y_l = x_l).$$

Daca P are forma $x_i := x_j - 1$, atunci:

$$F_P = [(x_j = 0) \vee (x_j = y_i + 1)] \wedge [(x_j \ge 1) \vee (y_i = 0)] \wedge \bigwedge_{l \neq i} (y_l = x_l).$$

Daca P are forma $Q; R$, atunci exista conform ipotezei de inductie formulele F_Q si F_R care satisfac cerinta de mai sus pentru programele respective. Definim:

$$F_P = \exists z_0 \dots \exists z_k \; (F_Q(x_0, \dots, x_k, z_0, \dots, z_k) \wedge F_R(z_0, \dots, z_k, y_0, \dots, y_k)).$$

Daca P are forma WHILE $x_i \neq 0$ DO Q END, exista din ipoteza de inductie formula corespunzatoare F_Q. Fie F_P urmatoarea formula:

$$\begin{aligned}
f_P \;\; = \;\; & \exists a_0 \exists b_0 \dots \exists a_k \exists b_k \exists t && (1)\\
& [\; G(a_0, b_0, 0, x_0) \wedge \cdots \wedge G(a_k, b_k, 0, x_k) \wedge && (2)\\
& \;\; G(a_0, b_0, t, y_0) \wedge \cdots \wedge G(a_k, b_k, t, y_k) \wedge && (3)\\
& \;\; \forall j < t \; \exists w \; G(a_i, b_i, j, w) \wedge (w > 0) \wedge && (4)\\
& \;\; G(a_i, b_i, t, 0) \wedge && (5)\\
& \;\; \forall j < t \; \exists w_0 \dots \exists w_k, \exists w'_0 \dots \exists w'_k && (6)\\
& \;\;\; [\; F_Q(w_0, \dots, w_k, w'_0, \dots, w'_k) \wedge && (7)\\
& \;\;\;\; G(a_0, b_0, j, w_0) \wedge \cdots \wedge G(a_k, b_k, j, w_k) \wedge && (8)\\
& \;\;\;\; G(a_0, b_0, j+1, w'_0) \wedge \cdots \wedge G(a_k, b_k, j+1, w'_k) \;]\;] && (9)
\end{aligned}$$

In randul (1) se enunta existenta unor perechi de numere a_n si b_n care codifica siruri de lungime t. In randul (2) se scrie conditia ca valorile initiale ale sirurilor sa fie valorile initiale din registre, adica variabilele libere x_j. In randul (3) se scrie conditia ca valorile finale din registre sa fie valorile finale ale sirurilor, adica variabilele libere y_j. In randurile (4) si (5) se pune conditia ca valorile din registrul i sunt diferite de 0 pana la pasul t, cand valoarea acestui registru devine 0. Incepand cu randul (6) se pune conditia ca valorile succesive din registre sa fie data de aplicarea programului Q. Asadar programul Q se aplica de t ori, pana cand valoarea din registrul i devine 0. □

Teorema: *Multimea formulelor aritmetice adevarate nu este recursiv enumerabila. De asemeni, multimea propozitiilor adevarate nu este recursiv enumerabila.*

Demonstratie: Cum propozitiile adevarate sunt incluse in formulele aritmetice adevarate, si este decidabil daca o expresie este propozitie sau nu (deoarece propozitiile nu au variabile libere), este suficient sa demonstram teorema pentru propozitii. Pentru orice propozitie F este adevarat ca sau F, sau $\neg F$ este adevarata in structura algebrica $(\mathbb{N}, +, \cdot)$. Daca multimea de propozitii:

$$Th(\mathbb{N}) = \{F \mid (\mathbb{N}, +, \cdot) \models F\}$$

(numita si teoria numerelor naturale) ar fi recursiv enumerabila, atunci ea ar fi decidabila. Algoritmul de decizie este urmatorul: fie $F_0, F_1, F_2, \ldots$ o enumerare algoritmica a lui $Th(\mathbb{N})$, eventual cu repetitii. La un moment dat, $F_i = F$, caz in care F este adevarata, sau $F_i = \neg F$, caz in care F este falsa. Decizia se va lua in timp finit.

Acum aratam ca $Th(\mathbb{N})$ nu este decidabila, si cu asta vom fi aratat ca aceasta multime nu este nici macar recursiv enumerabila. Fie A o multime recursiv enumerabila, care nu este decidabila, de exemplu K, H, H_0, PCP. Cum A este recursiv-enumerabila, functia:

$$\chi'_A(n) = \begin{cases} 1, & n \in A, \\ \text{nedefinita}, & n \notin A. \end{cases}$$

este WHILE calculabila, deci conform teoremei anterioare, este aritmetic definibila cu ajutorul unei formule $F(x, y)$. Asadar:

$$\begin{aligned} n \in A \quad & \Longleftrightarrow \quad \chi'_A(n) = 1 \\ & \Longleftrightarrow \quad F(n, 1) \text{ e adevarata} \\ & \Longleftrightarrow \quad F(n, 1) \in Th(\mathbb{N}). \end{aligned}$$

Aplicatia $n \rightsquigarrow F(n, 1)$ este o reductie de la A la $Th(\mathbb{N})$. Cum A nu este decidabil, nu este nici $Th(\mathbb{N})$, si datorita discutiei anterioare, $Th(\mathbb{N})$ nu este recursiv enumerabila. $\square$

Pentru a formula si demonstra Teorema de Incompletitudine a lui Gödel in forma ei initiala, trebuie sa ne referim la notiunea de demonstratie. Teoriile la care se refera Teorema de incompletitudine a lui Gödel nu pornesc cu o structura matematica si cu propozitiile adevarate in structura respectiva. Aceste teorii pornesc cu o multime de propozitii numite axiome si cu niste reguli de a deriva noi propozitii din axiome. Retinem faptul ca multimea axiomelor - care sunt cuvinte finite peste un anumit alfabet - este decidabila. Fiecare demonstratie este un sir finit de cuvinte in care fiecare cuvant este sau o axioma, sau rezulta din cuvinte deja existente in sir in virtutea unei reguli de derivatie. Daca se introduce un simbol nou cu rol de separator, multimea demonstratiilor intr-un sistem formal este o multime decidabila de cuvinte, pentru ca se poate verifica mecanic daca un cuvant este sau nu este o demonstratie. In final exista o functie calculabila care asociaza fiecarei demonstratii propozitia care a fost demonstrata. De exemplu, ultimul cuvant dedus in demonstratie este teorema demonstrata.

Definitie: Un sistem formal pentru o multime $A \subset \Gamma^*$ este o pereche (B, F) astfel incat:

1. $B \subseteq \Sigma^*$ este o multime decidabila.
2. $F : B \to A$ este o functie totala calculabila.

Fie $Dem(B, F) := F(B) \subseteq A$. Sistemul formal se numeste complet daca $Dem(B, F) = A$. $\square$

Teorema: *(De incompletitudine a lui Gödel) Orice sistem formal pentru $Th(\mathbb{N})$ este in mod necesar incomplet. Raman intotdeauna propozitii adevarate care nu sunt demonstrate de catre sistem.*

Demonstratie: $Dem(B, F) = F(B)$ este imaginea unei multimi decidabile printr-o functie calculabila totala, deci este o multime recursiv enumerabila. Cum $Th(\mathbb{N})$ nu este recursiv enumerabila, $Dem(B, F) \neq Th(\mathbb{N})$. $\square$

Retinem faptul ca teoria completa a unei structuri este sau decidabila, sau nici macar recursiv enumerabila. Alfred Tarski a aratat ca $Th(\mathbb{R}, +, \cdot)$ si ca $Th(\mathbb{C}, +, \cdot)$ sunt in prima categorie. Kurt Gödel a aratat ca $Th(\mathbb{N}, +, \cdot)$ este in a doua categorie, si de aici s-a dedus ca $Th(\mathbb{Z}, +, \cdot)$ si $Th(\mathbb{Q}, +, \cdot)$ sunt in a doua categorie. Ultimul rezultat a fost obtinut de Julia Robinson.

Unele probleme specifice sunt la randul lor nedecidabile. Yuri Matiasievici a aratat ca multimea ecuatiilor rezolvabile:

$$D = \{P \in \mathbb{Z}[X_1, \ldots, X_n] \,|\, n \geq 1, \exists x_1, \ldots, x_n \in \mathbb{N} \ \ P(x_1, \ldots, x_n) = 0\}$$

este recursiv enumerabila dar nedecidabila. Cu aceasta Matiasievici a rezolvat negativ problema a 10-a a lui Hilbert, aratand ca nu exista niciun algoritm de decizie pentru rezolvarea ecuatiilor diofantice.

Part IV

P versus NP

16 Notatia O mare

Conditiile asimptotice de crestere a functiilor se scriu cu usurinta folosind urmatoarele notatii:

Definitie: Fie $f, g : \mathbb{N} \to \mathbb{N}$ functii. Spunem ca:

1. $f = O(g)$ daca pentru un $c \in \mathbb{N}$, $f(n) \leq cg(n)$ pentru orice n suficient de mare.
2. $f = \Omega(g)$ daca $g = O(f)$.
3. $f = \Theta(g)$ daca $f = O(g)$ si $g = O(f)$.
4. $f = o(g)$ daca pentru orice $\epsilon > 0$, $f(n) \leq \epsilon g(n)$ pentru orice n suficient de mare.
5. $f = \omega(g)$ faca $g = o(f)$.

Cel mai frecvent se foloseste notatia $f(n) = O(g(n))$ pentru a sublinia si denumirea variabilei careia i se aplica functiile. $\square$

17 P si NP

In toate problemele de complexitatea calculului se considera alfabete Σ cu $|\Sigma| \geq 2$. La un moment dat se va pune si problema unor limbaje unare, dar asta numai pentru a sublinia diferentele.

Definitie: Fie $f : \mathbb{N} \to \mathbb{N}$ o functie. Clasa $TIME(f(n))$ este formata din toate multimile $A \subset \Sigma^*$ pentru care exista o masina Turing determinista cu mai multe benzi M astfel incat $A = T(M)$ si $time_M(x) \leq g(|x|)$ unde $g = O(f)$. Aici $time_M(x)$ este numarul de pasi efectuat de masina M pentru inputul x. Functia $time_M$ este definita pe Σ^* si ia valori in $\mathbb{N}$. $\square$

Observam ca din definitie, masina Turing M opreste pentru toate valorile de input x.

De asemeni, putem analiza demonstratia teoremei conform careia exista intotdeauna o masina Turing determinista cu o banda care simuleaza o masina Turing determinista cu mai multe benzi. Din analiza simularii se observa ca daca masina cu mai multe benzi are timpul de lucru marginit de o functie $f(n)$, atunci masina cu o banda are timpul de lucru $f(n)^2$. Definitia de mai sus a fost data folosind masini cu mai multe benzi, pentru ca complexitatea calculata este mai realista, mai apropiata de complexitatea algoritmului intuitiv.

Definitie: Un polinom este o functie $p : \mathbb{N} \to \mathbb{N}$ de forma:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

unde $a_i, k \in \mathbb{N}$. $\square$

Definitie: Clasa de complexitate P se defineste in modul urmator:

$$\begin{aligned} \mathrm{P} &= \{A \mid \text{ exista o masina Turing } M \\ &\quad \text{si un polinom } p \text{ cu} \\ &\quad T(M) = A \text{ si } time_M(x) \leq p(|x|)\} \\ &= \bigcup_{p \text{ polinom}} TIME(p(n)) \end{aligned}$$

Observatie: Daca functia $f(n)$ este LOOP calculabila (primitiv recursiva), atunci orice multime A din $TIME(f(n))$ are functia caracteristica LOOP calculabila (primitiv recursiva).

Demonstratie: Dupa cum am vazut, masina Turing cu mai multe benzi este simulata de un program GOTO, iar programul GOTO este simulat de un program WHILE care contine o singura bucla WHILE. Cum timpul de lucru este marginit de o functie LOOP calculabila, putem inlocui aceasta bucla WHILE cu o bucla LOOP. Adaugam buclele LOOP necesare calcularii functiei $f(n)$, si gata. Trebuie doar sa ne incredintam ca simularea masinii Turing printr-un program GOTO nu scoate timpul de lucru din clasa functiilor LOOP calculabile. $\square$

Definitie: Pentru masini Turing nedeterministe M fie:

$$ntime_M(x) = \begin{cases} \text{min } [\text{ Lungimea unui calcul al lui } M(x) & \\ \text{care se termina cu o stare acceptanta }], & x \in T(M) \\ 0 & x \notin T(M) \end{cases}$$

Fie $f : \mathbb{N} \to \mathbb{N}$ o functie. Clasa $NTIME(f(n))$ este formata din multimile $A \subseteq \Sigma^*$ pentru care exista o masina nedeterminista cu mai multe benzi M cu $A = T(M)$ si $ntime_M(x) \leq g(|x|)$ unde $g = O(f)$. Mai departe, definim:

$$\text{NP} = \bigcup_{p \text{ polinom}} NTIME(p(n)).$$

Observatie: Pentru toate multimile $A \in$ NP, functia caracteristica este LOOP calculabila (primitiv recursiva).

Demonstratie: Masinile Turing cu mai multe benzi se simuleaza cu masini Turing cu o banda, indiferent daca sunt deterministe sau nu. Timpul de calcul $f(n)$ este inlocuit cu $O(f(n)^2)$, care este polinomial daca f este polinom. Masinile Turing nedeterministe cu o banda sunt simulate de masini Turing deterministe cu o banda. Daca analizam aceasta simulare, timpul de calcul $f(n)$ este inlocuit cu timpul de calcul $f(n)^2 r^{2f(n)}$, unde r este numarul maxim se variante nedeterministe pentru un $\delta(z, a)$. Rezulta ca orice problema din NP se poate decide determinist in timp de $O(2^{p(n)})$, unde p este un polinom. Cum aceasta functie este primitiv recursiva, se aplica observatia precedenta. $\square$

Observatie: P $\subseteq$ NP. Pana acum a ramas o problema deschisa daca P = NP sau P $\neq$ NP. Aceasta este una din cele mai importante probleme ale matematicii. A fost pusa prima pe lista celor 7 Probleme ale Mileniului de catre Clay Institute din USA. De pe aceasta lista, numai Conjectura lui Poicarre a fost rezolvata pana in prezent. $\square$

Observatie: Au loc urmatoarele incluziuni:

$$\text{P} \subseteq \text{NP} \subsetneq \text{Primitiv Recursiv} \subsetneq \text{Decidabil} \subsetneq \text{Semidecidabil}.$$

18 Probleme NP-complete

Pentru a defini problemele NP-complete, avem nevoie de o notiune de recuctie adaptata pentru masinile cu timp de calcul marginit de o functie care depinde de lungimea inputului. Dam aceasta definitie aici direct in contextul timpului polinomial.

Definitie: Fie $A \subseteq \Sigma^*$ si $B \subseteq \Gamma^*$ doua multimi. Se spune ca A este polinomial reductibil la B si se scrie $A \leq_p B$ daca exista o functie totala, calculabila in timp polinomial, $f : \Sigma^* \to \Gamma^*$ astfel incat pentru orice $x \in \Sigma^*$ are loc:

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

$\square$

Lema: *Daca* $A \leq_p B$ *si* $B \in$ NP, *atunci* $A \in$ NP. *Daca* $A \leq_p B$ *si* $B \in$ P, *atunci* $A \in$ P.

Demonstratie: Fie $A \leq_p B$ datorita unei functii f, calculata de o masina Turing M_f. Fie polinomul p cel care margineste timpul de calcul al lui M_f. Fie $B \in$ P, recunoscut de o masina Turing M, al carui timp de calul este marginit de un polinom q. Masina Turing combinata $M_f; M$ este un algoritm in timp polinomial. Timpul lui de calcul pentru input x este marginit de:

$$p(|x|) + q(|f(x)|) \leq p(|x|) + q(p(|x|)).$$

Aceasta functie este un polinom. Enuntul despre NP se demonstreaza asemanator, considerand o masina nedeterminista M. $\square$

Definitie: O multime A este NP-hard daca pentru toate multimile $L \in \mathrm{NP}$ are loc: $L \leq_p A$. O multime A se numeste NP-completa daca A este NP-hard si $A \in \mathrm{NP}$. $\square$

Teorema: *Fie A NP-completa. Atunci:*

$$A \in \mathrm{P} \Longleftrightarrow \mathrm{P} = \mathrm{NP}.$$

Demonstratie: Fie $A \in \mathrm{P}$ si fie L o multime oarecare din NP. Cum A este NP-hard, $L \leq_p A$. Cu lema precedenta, avem $L \in \mathrm{P}$. Reciproc, daca $\mathrm{P} = \mathrm{NP}$, atunci $A \in \mathrm{P}$. $\square$

In cele ce urmeaza vom aborda urmatoarea strategie. Vom gasi o anumita problema NP-completa. Prin reduceri succesive, vom gasi apoi o intreaga pleiada de probleme NP-complete.

19 SAT

Definitie: Formulele din logica propozitionala (pe scurt, formule propozitionale) se definesc in modul urmator:

1. Variabilele x_i sunt formule.
2. Daca F si G sunt formule, atunci $(F) \vee (G)$, $(F) \wedge (G)$ si $\neg(F)$ sunt formule.

$\square$

Definitie: Consideram cunoscute tabelele de adevar ale operatiilor booleene de disjunctie, conjunctie si negatie. Pentru orice valori ale variabilelor $\vec{x} \in \{0,1\}^n$ formula propozitionala are o valoare $F(\vec{x}) \in \{0,1\}$. Formula propozitionala se numeste contradictorie daca $\forall \vec{x} \in \{0,1\}^n$, $F(\vec{x}) = 0$. Pentru formula necontradictorie are loc asadar $\exists \vec{x} \in \{0,1\}^n$, $F(\vec{x}) = 1$. Aceste formule se numesc si satisfiabile. $\square$

Definitie: Problema Satisfiabilitatii Formulelor in Logica Propozitionala, notata SAT, se defineste in modul urmator:

$$SAT = \{code(F) \in \Sigma^* \,|\, F \text{ formula necontradictorie de logica propozitionala}\}.$$

Se considera $code(F)$ in loc de F, pentru ca problema trebuie expusa intr-un alfabet finit. Variabilele se codifica de exemplu in modul urmator. Variabila x_n e codificata de cuvantul $x\, bin(n)$. $\square$

Teorema: *(Cook) Problema SAT a Satisfiabilitatii Formulelor Propozitionale este* NP*-completa.*

Demonstratie: Intai aratam ca $SAT \in \mathrm{NP}$. O masina nedeterminista care recunoaste formule necontradictorii functioneaza in modul urmator. Intai, intr-o faza determinista, formula este citita si se stabilese ce variabile apar in formula. Acestea sunt copiate eventual pe alta banda. Presupunem ca aceste variabile sunt $x_1, \dots, x_k$. Masina trece intr-o faza nedeterminista si alege la intamplare valori $a_1, \dots, a_k \in \{0,1\}$ pentru aceste variabile. In acest moment exista 2^k evolutii posibile ale masinii Turing. Apoi masina trece in ultima faza, determinista. Masina inlocuieste in formula fiecare aparitie a variabilei x_i cu constanta aleasa a_i. In final masina evalueaza formula. Pentru aceasta formula va fi scanata de un numar de ori egal cu profunzimea formulei (depth, adica numarul maxim de perechi de paranteze imbratisate in care se afla un simbol) - numar care este mai mic decat lungimea ei n. De asemeni, numarul de variabile k este strict mai mic decat n. Timpul total de lucru este $O(n^2)$. Masina accepta $code(F)$ daca la sfarsit are pe banda numai valoarea 1.

Acum aratam ca SAT este NP-hard. Pentru aceasta avem nevoie de urmatoarea pregatire:

Pentru orice $m \geq 1$ exista o formula propozitionala $G(x_1, \ldots, x_m)$ care este adevarata daca si numai daca exact una din variabilele x_i este adevarata iar celelalte sunt false. Lungimea acestei formule este $O(m^2)$.

Fie formula:

$$G(x_1, \ldots, x_m) = \Big(\bigvee_{i=1}^{m} x_i \Big) \ \wedge \ \Big(\bigwedge_{j=1}^{m-1} \bigwedge_{l=j+1}^{m} \neg(x_j \wedge x_l) \Big).$$

Prima parte a formulei este adevarata daca cel putin o variabila este adevarata. A doua parte a formulei este adevarata daca cel mult una dintre variabile este adevarata. Fiind vorba despre o conjunctie a celor doua parti, ambele trebuie sa fie adevarate.

Fie L o problema NP oarecare. Atunci $L = T(M)$ unde M este o masina Turing nedeterminista care accepta in timp polinomial. Putem presupune ca functia δ contine comanda $\delta(z_e, a) \ni (z_e, a, N)$, astfel incat odata ce o stare finala a fost atinsa, masina sa poata ramane in aceasta stare. Fie p un polinom care margineste timpul de calcul al lui M. Fie $x = x_1 x_2 \ldots x_n \in \Sigma^*$ un input pentru M. Vom construi o formula propozitionala F astfel incat:

$$x \in L \Longleftrightarrow F \text{ este satisfiabila.}$$

Fie $\Gamma = \{a_1, \ldots, a_l\}$ alfabetul de lucru al lui M si $Z = \{z_0, z_1, \ldots, z_k\}$ multimea de stari a lui M. Formula F contine urmatoarele variabile propozitionale (booleene):

1. $sta_{t,z}$, cu $t = 0, 1, \ldots, p(n)$ si $z \in Z$. Semnificatie: $sta_{t,z} = 1$ daca si numai daca la pasul t masina M este in starea z.

2. $poz_{t,i}$, cu $t = 0, 1, \ldots, p(n)$ si $i = -p(n), \ldots, p(n)$. Semnificatie: $poz_{t,i} = 1$ daca si numai daca masina M se gaseste la pasul t in pozitia i.

3. $ban_{t,i,a}$, cu $t = 0, 1, \ldots, p(n)$, $i = -p(n), \ldots, p(n)$ si $a \in \Gamma$. Semnificatie: $ban_{t,i,a} = 1$ daca si numai daca la pasul t in pozitia i este litera a.

Formula F va avea structura:

$$F = U \wedge I \wedge D \wedge N \wedge E,$$

unde:

- U exprima faptul ca in fiecare moment masina se afla intr-o singura stare, capul de citire-scriere se afla intr-o singura celula, si in fiecare celula este o singura litera.

- I exprima configuratia initiala a masinii, deci momentul $t = 0$.

- D exprima tranzitiile posibile, conform functiei δ.

- N exprima faptul ca celulele in care capul de citire-scriere nu este prezent isi pastreaza litera de la un pas la altul.

- E exprima conditia de acceptare, si anume ca o stare finala sa fi fost atinsa.

Asadar:

$$U = \bigwedge_{t} \ [\ G(sta_{t,z_1}, \ldots, sta_{t,z_k}) \wedge G(poz_{t,-p(n)}, \ldots, poz_{t,p(n)}) \wedge \bigwedge_{i} G(ban_{t,i,a_1}, \ldots, ban_{t,i,a_l}) \ ].$$

$$I = sta_{0,z_0} \wedge poz_{0,1} \wedge \bigwedge_{j=1}^{n} ban_{0,j,x_j} \wedge \bigwedge_{j=-p(n)}^{0} ban_{0,j,\Box} \wedge \bigwedge_{j=n+1}^{p(n)} ban_{0,j,\Box}.$$

$$D = \bigwedge_{t,z,i,a} [\ (sta_{t,z} \wedge poz_{t,i} \wedge ban_{t,i,a}) \longrightarrow \bigvee_{(z',a',y) \in \delta(z,a)} (sta_{t+1,z'} \wedge poz_{t+1,i+y} \wedge ban_{t+1,i,a'}) \ ].$$

$$N = \bigwedge_{t,i,a} [\ (\neg poz_{t,i} \wedge ban_{t,i,a}) \longrightarrow ban_{t+1,i,a} \ ].$$

$$E = \bigvee_{z \in E} sta_{p(n),z}.$$

Se verifica usor ca lungimea acestei formule este polinomiala in n. $\square$

20 3SAT

Definitie: Problema *3SAT* este un caz particular al lui *SAT*. In aceasta problema este vorba numai despre satisfiabilitatea formulelor propozitionale scrise in forma normala conjunctiva care au cel mult 3 literale in fiecare clauza, prescurtat 3CNF. Concret, o formula 3CNF este de forma urmatoare:

$$F = \bigwedge_i (z_{i1} \vee z_{i2} \vee z_{i3}),$$

unde fiecare (i, j) exista o variabila $x \in \{x_1, \dots, x_n\}$ astfel incat $z_{ij} = x$ sau $z_{ij} = \neg x$. Problema *3SAT* este asadar:

$$\mathit{3SAT} = \{code(F) \in \Sigma^* \mid F \text{ formula 3CNF necontradictorie}\}.$$

$\square$

Teorema: *Problema 3SAT este* NP*-completa.*

Demonstratie: Bineinteles ca $\mathit{3SAT} \in \mathrm{NP}$, deoarece se testeaza in timp nedeterminist polinomial satisfiabilitatea unor formule propozitionale. Fie FP multimea formulelor propozitionale. Pentru a arata ca *3SAT* este NP-hard, trebuie sa construim o functie f totala, calculabila in timp polinomial, de la FP la 3CNF, astfel incat pentru orice formula propozitionala F:

$$F \text{ este satifiabila} \Longleftrightarrow f(F) \text{ este satifiabila.}$$

In paralel cu descrierea formala a algoritmului care calculeaza functia f, vom urmari ce se intampla cu exemplul:

$$F = \neg(\neg(x_1 \vee \neg x_3) \vee x_2).$$

1. Se folosesc relatiile lui De Morgan si dubla negatie, pentru a duce negatiile la variabile.

$$\begin{aligned} \neg(a \vee b) &\leftrightarrow \neg a \wedge \neg b \\ \neg(a \wedge b) &\leftrightarrow \neg a \vee \neg b \\ \neg(\neg a) &\leftrightarrow a \end{aligned}$$

   Ca urmare a acestei operatii, formula propozitionala F este inlocuita de:

$$F_1 = (x_1 \vee \neg x_3) \wedge \neg x_2.$$

2. Fiecarei operatii de conjunctie sau disjunctie i se atribuie o noua variabila. In cazul nostru atribuirile sunt:

$$\begin{aligned} y_1 &\leftrightarrow (x_1 \vee \neg x_3) \\ y_0 &\leftrightarrow (y_1 \wedge \neg x_2) \end{aligned}$$

   Variabila introdusa la sfarsit, pentru intreaga formula, trebuie sa aiba si ea valoarea de adevar 1. Toate relatiile obtinute se combina conjunctiv. In cazul nostru, se obtine formula propozitionala:

$$F_2 = [y_0] \wedge [y_0 \leftrightarrow (y_1 \wedge \neg x_2)] \wedge [y_1 \leftrightarrow (x_1 \vee \neg x_3)].$$

3. Cu ajutorul urmatoarelor tautologii:

$$(a \leftrightarrow (b \vee c)) \quad \leftrightarrow \quad (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)$$
$$(a \leftrightarrow (b \wedge c)) \quad \leftrightarrow \quad (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$$

se inlocuieste fiecare conditie cu o conjunctie de 3 clause, fiecare continand cel mult 3 literale pro clausa. Exemplul nostru devine:

$$F_3 = y_0 \wedge (\neg y_0 \vee y_1) \wedge (\neg y_0 \vee \neg x_2) \wedge (y_0 \vee \neg y_1 \vee x_2) \wedge$$
$$\wedge (y_1 \vee \neg x_1) \wedge (\neg y_1 \vee x_1 \vee \neg x_3) \wedge (y_1 \vee x_3).$$

Intregul algoritm are nevoie de timp liniar (exercitiu!). Daca definim $f(F) = F_3$, observam ca $SAT \leq_p 3SAT$, deci *3SAT* este o problema NP-hard. □

Urmatorul rezultat arata ca numarul 3 din *3SAT* nu poate fi imbunatatit, decat poate daca P = NP.

Observatie: *Problema 2SAT este in* P.

Presupunem ca intr-o instanta apar variabilele $x_1, \dots, x_n$. Se construieste un graf cu $2n$ varfuri care au etichetele $x_1, \dots, x_n, \neg x_1, \dots, \neg x_n$. Fiecare clauza de forma $(a_1 \vee a_2)$, unde a_1 si a_2 sunt variabile negate sau nu, poate fi privita atat ca implicatia $\neg a_1 \to a_2$ cat si ca implicatia $\neg a_2 \to a_1$. Toate aceste implicatii se deseneaza ca muchii orientate in graf. Se dovedeste ca problema este rezolvabila daca si numai daca pentru nicio variabila x nu exista vreun drum orientat de la x la $\neg x$ SI de la $\neg x$ la x, adica x si $\neg x$ nu se afla in acelasi ciclu orientat. Intr-adevar, in acest caz nu putem avea solutie. Pe de alta parte, daca un asemenea ciclu orientat nu exista, daca $\neg x$ apare dupa x, facem $x = 0$, iar daca x apare dupa $\neg x$, facem $x = 1$. De la algoritmii de grafuri se stie ca gasirea ciclurilor orientate se poate face in timp polinomial.

Acest lucru poate fi dedus si prin rezolutie. Inlocuim o pereche de clauze de forma $(a \vee \neg b)$ respectiv $(b \vee c)$ cu rezolventa $(a \vee c)$. Cu aceasta ocazie obtinem numai rezolvente cu 2 literale - ceea ce nu este cazul in problema *3SAT*, unde rezolventa poate avea 4 literale, si ar putea creste in continuare in procesul de rezolutie. Daca rezolutia se opreste la un moment dat, si inca exista clauze care nu contin literale opuse, instanta este satisfiabila. Daca la un moment dat s-au terminat clauzele, instanta nu poate fi satisfacuta. Timpul este evident polinomial. □

21 CLIQUE si VERTEX COVER

Definitie: Un graf neorientat este o structura $G = (V, E)$ unde $E \subseteq V^2$ este o relatie cu proprietatea ca $\forall x, y \; E(x, y) \leftrightarrow E(y, x)$. Spunem ca relatia este simetrica.

Definitie: Problema *CLIQUE* se defineste in modul urmator:

Input: Un graf neorientat $G = (V, E)$ si un numar $k \in \mathbb{N}$.

Question: Exista o clica de marime cel putin k?

O asemenea clica este $V' \subseteq V$ astfel incat pentru orice $u, v \in V'$ cu $u \neq v$ are loc $E(u, v)$. □

Teorema: *CLIQUE este* NP-*completa.*

Demonstratie: Problema este in NP. Putem alege in mod nondeterminist o submultime cu k elemente V' si verificam in timp patratic, citind matricea care codifica graful, ca oricare doua varfuri distincte din V' sunt conectate.

Problema este NP-hard. Aratam ca $3SAT \leq_p CLIQUE$. Fie F o formula 3CNF cu exact 3 literale pe clauza. Asta se poate presupune intotdeauna, fiindca putem repeta un literal o data sau de doua ori in aceeasi clauza. Asadar:

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$$

unde:

$$z_{ij} \in \{x_1, x_2, \dots\} \cup \{\neg x_1, \neg x_2, \dots\}.$$

Observam ca cel mult $3m$ variabile diferite pot sa apara in F.

Acestei formule propozitionale i se asociaza un graf $G = (V, E)$ si un numar $k \in \mathbb{N}$:

$$\begin{aligned} V &= \{(1,1), (1,2), (1,3), \dots, (m,1), (m,2), (m,3)\} \\ E &= \{[(i,j), (p,q)] \mid i \neq p \wedge z_{ij} \neq \neg z_{pq}\} \\ k &= m \end{aligned}$$

F poate fi satisfacuta de niste valori $\vec{x} \in \{0,1\}^{3m}$

$\Leftrightarrow$ In fiecare clauza exista un literal care ia valoarea 1, de exemplu $z_{1j_1}, z_{2j_2}, \dots, z_{m,j_m}$.

$\Leftrightarrow$ Exista literale $z_{1j_1}, z_{2j_2}, \dots, z_{m,j_m}$ care nu sunt doua cate doua complementare.

$\Leftrightarrow$ Exista varfuri $(1, j_1), (2, j_2), \dots, (m, j_m)$ in G conectate doua cate doua.

$\Leftrightarrow$ G are o clica de marime m. □

Definitie: Problema *VERTEX COVER* se defineste in modul urmator:

Input: Un graf neorientat $G = (V, E)$ si un numar $k \in \mathbb{N}$.

Question: Exista o acoperire de marime cel mult k?

O asemenea acoperire este o submultime $V' \subseteq V$ astfel incat pentru orice $u, v \in V$ cu $E(u, v)$ avem $u \in V'$ sau $v \in V'$. □

Teorema: *VERTEX COVER este* NP*-completa.*

Demonstratie: Problema este in NP. Putem alege in mod nondeterminist o submultime cu k elemente V' si verificam in timp patratic, citind matricea care codifica graful, ca orice muchie contine un varf din V'.

Problema este NP-hard. Aratam ca *CLIQUE* $\leq_p$ *VERTEX COVER*. Pentru graful $G = (V, E)$ si numarul k construim graful complementar:

$$\overline{G} = (V, \overline{E} = \{(u, v) \mid u, v \in V, \ \neg E(u, v)\}).$$

si numarul $|V| - k$. Intr-adevar, daca in G exista o clica K cu cel putin k varfuri, atunci fie C complementara $V \setminus K$. C are cel mult $|V| - k$ varfuri. Daca doua varfuri au proprietatea ca $\overline{E}(u, v)$, atunci $\neg E(u, v)$ deci $u \notin K$ sau $v \notin K$, deci $u \in C$ sau $v \in C$. Asadar $\overline{G}$ are o acoperire cu cel mult $|V| - k$ varfuri. Reciproca este de asemeni adevarata. □

22 SUBSET SUM, PARTITION si BIN PACKING

Definitie: Problema *SUBSET SUM* se defineste in modul urmator:

Input: Numere $a_1, a_2, \dots, a_k, b \in \mathbb{N}$.

Question: Exista o submultime $I \subseteq \{1, 2, \dots, k\}$ cu $\sum_{i \in I} a_i = b$?

□

Mentionez ca aceasta problema se mai numeste si *RUCKSACK* sau *KNAPSACK*.

Teorema: *SUBSET SUM este* NP*-completa.*

Demonstratie: Problema este in NP. Putem alege la intamplare o submultime $I \subseteq \{1, 2, \dots, k\}$. In mod determinist se calculeaza suma elementelor a_i cu $i \in I$ si se compara cu b. In caz de egalitate, se accepta instanta.

Problema este NP-hard. Vom arata ca *3SAT* $\leq_p$ *SUBSET SUM*. Fie o formula $F \in$ 3CNF:

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$$

unde:

$$z_{ij} \in \{x_1, x_2, \ldots, x_n\} \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}.$$

Definim numarul $b \in \mathbb{N}$. El este:

$$b = \underbrace{444\ldots44}_{m}\underbrace{11\ldots11}_{n}.$$

unde m este numarul de clauze iar n este numarul de variabile. Pentru a ilustra mai bine constructia, ea va fi prezentata in paralel cu un exemplu. Fie asadar o formula cu 3 clauze si cu 5 variabile, anume:

$$F = (x_1 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee x_5 \vee x_4) \wedge (\neg x_2 \vee \neg x_2 \vee \neg x_5).$$

In acest caz numarul b este:

$$b = 44411111.$$

In continuare se construiesc $k = 2n + 2m$ numere, dupa cum urmeaza.

Numerele t_i cu $i = 1, \ldots, n$. In numarul t_1 este pe pozitia i cifra $m \in \{0, 1, 2, 3\}$ in cazul in care variabila x_1 are in clauza i un numar de m ocurente **pozitive**. In al doilea bloc numeric, numarul t_1 are un 1 pe prima pozitie si in rest 0. La fel se defineste t_2, doar ca in al doilea bloc numeric acest numar are un 1 pe pozitia 2 si in rest 0. In cazul nostru, numerele t_i sunt urmatoarele:

$$\begin{aligned} t_1 &= 100\ 10000 \\ t_2 &= 000\ 01000 \\ t_3 &= 000\ 00100 \\ t_4 &= 010\ 00010 \\ t_5 &= 110\ 00001 \end{aligned}$$

Numerele f_i cu $i = 1, \ldots, n$. In numarul f_1 este pe pozitia i cifra $m \in \{0, 1, 2, 3\}$ in cazul in care variabila x_1 are in clauza i un numar de m ocurente **negative**. In al doilea bloc numeric, numarul f_1 are un 1 pe prima pozitie si in rest 0. La fel se defineste f_2, doar ca in al doilea bloc numeric acest numar are un 1 pe pozitia 2 si in rest 0. In cazul nostru, numerele f_i sunt urmatoarele:

$$\begin{aligned} f_1 &= 010\ 10000 \\ f_2 &= 002\ 01000 \\ f_3 &= 100\ 00100 \\ f_4 &= 000\ 00010 \\ f_5 &= 001\ 00001 \end{aligned}$$

Numerele c_j si d_j cu $j = 1, \ldots, m$. Numarul c_j are pe pozitia j un 1 si in rest 0, iar $d_j = 2c_j$.

$$\begin{aligned} c_1 &= 100\ 00000 \\ c_2 &= 010\ 00000 \\ c_3 &= 001\ 00000 \end{aligned}$$

$$\begin{aligned} d_1 &= 200\ 00000 \\ d_2 &= 020\ 00000 \\ d_3 &= 002\ 00000 \end{aligned}$$

Vom arata ca daca exista o solutie $\vec{x}$ astfel incat $F(\vec{x}) = 1$, atunci exista o solutie in care daca $x_i = 1$, se gaseste numarul t_i iar daca $x_i = 0$, se gaseste numarul f_i.

In cazul nostru, o solutie este data de $\vec{x} = (1, 0, 0, 1, 0)$. Asta inseamna ca alegem numerele t_1, f_2, f_3, t_4, f_5. Suma lor este numarul 21311111. Aceasta inseamna ca in clauza 1, doua literale sunt adevarate, in clauza 2, un literal este adevarat, iar in clauza 3, trei literale sunt adevarate. Putem completa aceasta suma cu numerele de umplutura d_1, c_2, d_2, c_3 si obtinem suma dorita $b = 44411111$.

Presupunem ca avem o alegere de numere care adunate dau b. Al doilea bloc este de asa natura, incat pentru fiecare i de la 1 la n, exact unul dintre numerele t_i si f_i a fost ales. Consideram vectorul binar $\vec{x}$ corespunzator cu aceasta alegere. Presupunem ca clauza j nu este satisfacuta de $\vec{x}$. Atunci in suma acestor t_i si f_i, pozitia j ramane egala cu 0 si nicio alegere a numerelor de umplutura nu mai poate aduce aceasta pozitie la valoarea ceruta 4. □

Definitie: Problema *PARTITION* se defineste in modul urmator:

> **Input**: Numere $a_1, a_2, \dots, a_k \in \mathbb{N}$.
>
> **Question**: Exista o submultime $J \subseteq \{1, 2, \dots, k\}$ cu $\sum_{i \in J} a_i = \sum_{i \notin J} a_i$?

□

Teorema: *PARTITION este* NP-*completa.*

Demonstratie: Problema este in NP. Putem alege la intamplare o submultime $J \subset \{1, 2, \dots, k\}$. In mod determinist se calculeaza suma elementelor a_i cu $i \in J$, suma elementelor a_i cu $i \notin J$ si se compara aceste sume. In caz de egalitate, se accepta instanta.

Problema este NP-hard. Aratam ca *SUBSET SUM* $\leq_p$ *PARTITION*. Fie $(a_1, a_2, \dots, a_k, b)$ o instanta a problemei *SUBSET SUM*. Fie

$$M = \sum_{i=1}^{k} a_i$$

si definim urmatoarea aplicatie:

$$(a_1, a_2, \dots, a_k, b) \rightsquigarrow (a_1, a_2, \dots, a_k, M - b + 1, b + 1).$$

Fie $I \subseteq \{1, 2, \dots, k\}$ o solutie a problemei *SUBSET SUM* pentru instanta din stanga. Deci:

$$\sum_{i \in I} a_i = b.$$

Atunci:

$$\sum_{i \in I \cup \{k+1\}} a_i = b + (M - b + 1) = M + 1, \qquad \sum_{i \notin I \cup \{k+1\}} a_i = (M - b) + (b + 1) = M + 1,$$

deci $I \cup \{k + 1\}$ este o solutie a problemei *PARTITION* pentru instanta din dreapta.

Reciproc, fie J o solutie a problemei *PARTITION* pentru instanta din dreeapta. Atunci numerele $M - b + 1$ si $b + 1$ nu pot fi ambele in J pentru ca suma lor $M + 2$ este strict mai mare decat suma celorlalte a_i. Din acelasi motiv, ele nu pot fi ambele in complementara lui J. Daca $M + b - 1$, fara a restrange generalitatea, este in J, atunci este J fara acest numar o solutie a problemei *SUBSET SUM*, instanta din stanga, deoarece suma acestei multimi de numere va fi $M + 1 - (M - b + 1) = b$. □

Urmatoarea problema are importanta practica:

Definitie: Problema *BIN PACKING* se defineste in modul urmator:

> **Input**: O marime de container $b \in \mathbb{N}$, un numar de containere $k \geq 2$, obiecte de marimi $a_1, a_2, \dots, a_k \leq b$.
>
> **Question**: Pot fi distribuite obiectele in containere?

$\square$

Riguros, cautam o functie $f : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, k\}$ astfel incat pentru orice $j = 1, \ldots, k$ are loc:

$$\sum_{f(i)=j} a_i \leq b.$$

Teorema: *BIN PACKING este* NP*-completa.*

Demonstratie: Problema este in NP, din nou printr-un argument guess and check.

Problema este NP-hard. Aratam ca *PARTITION* $\leq_p$ *BIN PACKING*. Reductia functioneaza in modul urmator:

$$(a_1, \ldots, a_k) \rightsquigarrow \begin{cases} \text{Marimea containerelor:} & b = \left(\sum_{i=1}^{k} a_i\right)/2, \\ \text{Numarul containerelor:} & k = 2, \\ \text{Obiecte de marimi:} & a_1, \ldots, a_k. \end{cases}$$

$\square$

23 HAMILTON PATH si TRAVELING SALESMAN

Definitie: Problema *ORIENTED HAMILTON PATH* se defineste in modul urmator:

Input: Un graf $G = (V, E)$ si doua varfuri $s, e \in V$ cu $s \neq e$.

Question: Exista un drum hamiltonian de la s la e in G?

$\square$

Cu alte cuvinte, daca $|V| = n$, sa existe o permutare $\pi \in S_n$ astfel incat $v_{\pi(1)} = s$, $v_{\pi(n)} = e$ si pentru orice $i = 1, \ldots, n-1$,

$$(v_{\pi(i)}, v_{\pi(i+1)}) \in E.$$

Teorema: *ORIENTED HAMILTON PATH este* NP*-completa.*

Demonstratie[9]: Problema este in NP. O masina genereaza aleator un sir de $n-1$ numere din multimea $\{1, 2, \ldots, n\}$ apoi verifica in mod determinist daca sirul respectiv este un drum hamiltonian de la s la e pentru graful orientat G.

Problema este NP-hard. Vom arata ca *3SAT* $\leq_p$ *ORIENTED HAMILTON PATH*. Fie o formula $F \in$ 3CNF:

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \cdots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$$

unde:

$$z_{ij} \in \{x_1, x_2, \ldots, x_n\} \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}.$$

Vom construi un graf orientat (G, s, e) care contine un drum hamiltonian de la s la e daca si numai daca formula 3CNF este satisfiabila. Graful are $4mn + m + 2$ varfuri, dupa cum urmeaza. Exista un varf de start s si un varf terminal e. Exista m varfuri $c_1, \ldots, c_m$ asociati clauzelor. Fiecarei variabile x_i i se asociaza un lant liniar de $4m$ varfuri. Cele $4m$ varfuri dintr-un asemenea lant sunt conectate prin muchii neorientate, adica $(v_k, v_{k+1}) \in E$ si $(v_{k+1}, v_k) \in E$. Un asemenea lant poate fi parcurs de drumul hamiltonian atat de la stanga la dreapta, in cazul in care variabila corespunzatoare este adevarata, cat si de la dreapta la stanga, daca variabila corespunzatoare este falsa. Lantul L_i corespunzator variabilei x_i arata in modul urmator:

$$L_i \ : \ A_i = \odot \cdots \odot \cdots \odot \cdots\cdots\cdots\cdots\cdots \odot \cdots \odot \cdots \odot = B_i$$

[9]Sanjeev Arora si Boaz Barak: Computational Complexity, A Modern Approach. Cambridge University Press, 2009.

$$x_i = 1 \; : \; \odot \hookrightarrow \odot \hookrightarrow \odot \hookrightarrow \odot$$

$$x_i = 0 \; : \; \odot \hookleftarrow \odot \hookleftarrow \odot \hookleftarrow \odot$$

Cele doua varfuri extreme ale lui L_i se numesc A_i (in stanga) si B_i (in dreapta).

Varfurile se conecteaza prin muchii orientate in modul urmator:

Varful s se conecteaza prin muchii orientate cu cele doua extremitati ale lui L_1: sA_1 si sB_1.

Fiecare lant L_i se conecteaza cu fiecare lant L_{i+1} pentru $i < n$ prin 4 muchii orientate: A_iA_{i+1}, A_iB_{i+1}, B_iA_{i+1}, B_iB_{i+1}.

Cele doua extremitati ale lui L_n se conecteaza cu muchii orientate cu varful e: A_ne si B_ne.

Un varf c_j care corespunde clauzei j se conecteaza cu muchii orientate cu un lant L_i daca si numai daca x_i apare in clauza j. Pentru fiecare aparitie a lui x_i in clauza j se aleg doua varfuri consecutive u si v din L_i cu u in stanga lui v. Daca aparitia lui x_i in clauza j este **pozitiva** se introduc muchiile orientate uc_i si c_iv. Daca aparitia lui x_i in clauza j este **negativa** se introduc muchiile orientate vc_i si c_iu.

$$\begin{array}{ccccc} & \text{c} & & & \\ & \uparrow & \searrow & & \\ \cdots & \text{u} & \cdots & \text{v} & \cdots \end{array} \qquad (x_i \text{ apare in } c)$$

$$\begin{array}{ccccc} & \text{c} & & & \\ & \downarrow & \nwarrow & & \\ \cdots & \text{u} & \cdots & \text{v} & \cdots \end{array} \qquad (\neg x_i \text{ apare in } c)$$

Doua conexiuni de clauza cu acelasi lant trebuie sa aiba cel putin o muchie orizontala intre ele. Lanturile sunt suficient de lungi pentru a satisface aceasta conditie.

$\Rightarrow$ Presupunem ca formula 3CNF este satisfacuta de valorile $\vec{x} \in \{0, 1\}$. Drumul hamiltonian incepe in s si se termina in e. Lanturile pentru care variabila corespunzatoare este $x_i = 1$ se vor parcurge in sens pozitiv iar cele pentru care $x_i = 0$ in sens negativ. Cum in fiecare clauza cel putin un literal este egal cu 1, clauza respectiva va putea fi vizitata de pe lantul corespunzator acelui literal. Daca acest lucru se poate face de mai multe ori, o vom face o singura data pentru a respecta conditia drumului hamiltonian.

$\Leftarrow$ Presupunem ca graful orientat G contine un drum hamiltonian. Acest drum trebuie sa inceapa in s, la care nu se poate ajunge venind din alt varf, si trebuie sa se termine in e, fiindca din e nu se poate pleca in alta parte. Fiecare lant va fi parcurs in sens pozitiv sau negativ. In mod corespunzator se dau valori variabilelor x_i. Fie c o clauza oarecare. Cum c este vizitata dintr-un anumit lant, literalul corespunzator are valoarea 1, deci clauza este satisfacuta. Deoarece asta se intampla cu toate clauzele, formula este satisfacuta.

Constructia are loc in timp polinomial in lungimea formulei. $\square$

Definitie: Problema *ORIENTED HAMILTON CYCLE* se defineste in modul urmator:

Input: Un graf $G = (V, E)$.

Question: Exista un ciclu hamiltonian in G?

$\square$

Cu alte cuvinte, daca $|V| = n$, sa existe o permutare $\pi \in S_n$ astfel incat pentru orice $i = 1, \ldots, n-1$,

$$(v_{\pi(i)}, v_{\pi(i+1)}) \in E$$

si de asemeni $(v_{\pi(n)}, v_{\pi(1)}) \in E$.

Teorema: *ORIENTED HAMILTON CYCLE este* NP*-completa.*

Demonstratie[10]: Problema este in NP. O masina genereaza aleator un sir de n numere din multimea $\{1, 2, \ldots, n\}$ apoi verifica in mod determinist daca sirul respectiv este un ciclu hamiltonian pentru graful orientat G.

Problema este NP-hard. Aratam ca *ORIENTED HAMILTON PATH* $\leq_p$ *ORIENTED HAMILTON CYCLE*. Pentru o instanta (G, s, e) a problemei *ORIENTED HAMILTON PATH* construim o instanta $\overline{G} = (\overline{V}, \overline{E})$ a problemei *ORIENTED HAMILTON CYCLE* astfel incat rezolvabilitatea se prezerva in ambele directii. Fie $\overline{V} = V \setminus \{e\}$ cu

$$\overline{E} = E \setminus \{(x, e) \,|\, x \in V\} \cup \{(x, s) \,|\, x \neq e \wedge (x, e) \in E\}.$$

Evident (G, s, e) admite un drum hamiltonian de la s la e daca si numai daca $\overline{G}$ admite un ciclu hamiltonian. $\square$

Definitie: Problema *HAMILTON CYCLE* se defineste in modul urmator:

Input: Un graf neorientat $G = (V, E)$.

Question: Exista un ciclu hamiltonian in G?

$\square$

Cu alte cuvinte, relatia de muchie este simetrica, adica pentru orice $x, y \in V$, $(x, y) \in E$ daca si numai daca $(y, x) \in E$. Definitia ciclului hamiltonian este aceeasi.

Teorema: *HAMILTON CYCLE este* NP*-completa.*

Demonstratie: Problema este in NP. O masina genereaza aleator un sir de n numere din multimea $\{1, 2, \ldots, n\}$ apoi verifica in mod determinist daca sirul respectiv este un ciclu hamiltonian pentru graful neorientat G.

Problema este NP-hard. Aratam ca *ORIENTED HAMILTON CYCLE* $\leq_p$ *HAMILTON CYCLE*. Iata cum se transforma o instanta de graf orientat intr-una de graf neorientat. Fie v un varf al lui G (in care intra cel putin 2 sageti) $\vee$ (din care ies cel putin doua sageti). De exemplu, varful urmator are 3 sageti de intrare si 2 sageti de iesire:

$$\begin{array}{ccc} \searrow & v & \nearrow \\ \longrightarrow & \odot & \\ \nearrow & & \searrow \end{array}$$

Acesta este inlocuit cu un subgraf neorientat format din 3 noi noduri, dupa cum urmeaza:

$$\begin{array}{ccccccc} \ddots & a & & b & & c & \cdot^{\cdot^{\cdot}} \\ \cdots & \odot & \cdots & \odot & \cdots & \odot & \\ \cdot^{\cdot^{\cdot}} & & & & & & \ddots \end{array}$$

Fie $\overline{G}$ graful neorientat care rezulta din aceasta constructie. Este evident ca un circuit hamiltonian in G se transpune fara probleme intr-un circuit hamiltonian in $\overline{G}$. Presupunem acum ca graful neorientat G are un circuit hamiltonian. Daca acest ciclu ajunge in a, el nu poate parasi a printr-o alta muchie, fara sa mearga in b, pentru ca atunci b devine o fundatura, si nu vom mai avea ciclu hamiltonian in $\overline{G}$. Deci ciclul va trebui sa treaca prin b si prin c. Modulo o alegere de sens, acest ciclu hamiltonian poate fi factorizat la un ciclu hamiltonian al lui G. $\square$

Definitie: Problema *EULER CYCLE* se defineste asemanator cu problema *HAMILTON CYCLE*. Se da un graf neorientat G si se cere un drum ciclic in graf care sa contina *toate muchiile* si *fiecare muchie numai o data*. Atentie: ciclul poate sa treaca de mai multe ori prin acelasi varf. O instanta a acestei probleme, Problema Podurilor din Königsberg, a fost rezolvata de Euler. Aceasta a fost prima problema de teoria grafurilor. $\square$

[10]Folclor.

Observatie: *Problema EULER CYCLE este rezolvabila in timp polinomial.*

Intr-adevar, se stie ca un graf neorientat admite un ciclu Euler daca si numai daca fiecare varf este conectat cu un numar par de varfuri. Acest lucru se poate decide in timp liniar citind matricea grafului. □

Urmatoarea problema are importanta practica:

Definitie: Problema *TRAVELING SALESMAN* se defineste in modul urmator:

> **Input**: O matrice $M \in \mathcal{M}_{n\times n}(\mathbb{N})$ a distantelor dintre n orase si un $k \in \mathbb{N}$.
>
> **Question**: Exista o permutare $\pi \in S_n$ astfel incat:

$$\sum_{i=1}^{n-1} M_{\pi(i),\pi(i+1)} + M_{\pi(n),\pi(1)} \leq k \quad ?$$

□

Cu alte cuvinte, un negustor trebuie sa isi planifice un turneu prin aceste orase, dar doreste ca pretul calatoriei sa fie cat mai mic posibil.

Teorema: *TRAVELING SALESMAN este* NP*-completa.*

Demonstratie: Problema este in NP. O masina genereaza aleator un sir de n numere din multimea $\{1, 2, \dots, n\}$ apoi verifica in mod determinist daca sirul respectiv este un turneu inchis si ca lungimea totala satisface conditia din enunt.

Problema este NP-hard. Aratam ca *HAMILTON PATH* $\leq_p$ *TRAVELING SALESMAN*. Acest lucru este dat de urmatoarea reductie:

$$G = (\{1, \dots, n\}, E) \quad \rightsquigarrow \quad \begin{cases} \text{Matrice:} & M_{ij} = \begin{cases} 1, & (i,j) \in E, \\ 2, & (i,j) \notin E. \end{cases} \\ \text{Lungimea maxima:} & n. \end{cases}$$

Este clar ca exista un turneu de lungime n daca si numai daca graful neorientat G admite un ciclu hamiltonian. □

24 3COLORING

Definitie: Problema *3COLORING* se defineste in modul urmator:

> **Input**: Un graf neorientat $G = (V, E)$.
>
> **Question**: Exista o colorare a lui V in 3 culori, astfel incat fiecare muchie sa aiba capetele de culori diferite?

Riguros, se intreaba daca exista o functie $f : V \to \{1, 2, 3\}$ astfel incat pentru orice $u, v \in V$, daca $(u, v) \in E$ atunci $f(u) \neq f(v)$.

Teorema: *3COLORING este* NP*-completa.*

Demonstratie[11]: Problema este in NP. O masina genereaza aleator o functie $f : V \to \{1, 2, 3\}$ apoi verifica in mod determinist conditia din enunt.

Problema este NP-hard. Vom construi o reductie *3SAT* $\leq_p$ *3COLORING*. Consideram o formula propozitionala $F \in$ 3CNF, de forma:

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$$

unde:

$$z_{ij} \in \{x_1, x_2, \dots, x_n\} \cup \{\neg x_1, \neg x_2, \dots, \neg x_n\}.$$

[11] Folclor.

Vom construi un graf neorientat G cu $2n + 6m + 3$ varfuri, care admite o colorare cu 3 culori conform enuntului daca si numai daca F este necontradictorie.

- Graful contine in primul rand 3 varfuri b, t si f si muchiile neorientate bt, tf si fb. Cum aceste muchii formeaza un triunghi, cele trei varfuri trebuie sa aiba culori diferite. Aceste culori se numesc BASE, TRUE si FALSE. Fiecare culoare se asociaza cu varful notat cu litera ei initiala. Ultimele doua culori se vor interpreta ca valori de adevar al unor expresii propozitionale.

$$\begin{array}{ccccc} t & & & & x_i \\ \vdots & \ddots & & \iddots & \vdots \\ \vdots & & b & & \vdots \\ \vdots & \iddots & & \ddots & \vdots \\ f & & & & \overline{x_i} \end{array}$$

- Pentru fiecare variabila propozitionala x_i introducem doua varfuri: x_i si $\overline{x_i}$. Varfurile $\overline{x_i}$ corespund negatiei $\neg x_i$. Pentru a forta aceste doua varfuri sa capete una din culorile TRUE si FALSE, se introduc muchiile bx_i, $x_i\overline{x_i}$ si $\overline{x_i}b$.

- Se considera urmatorul graf:

$$\begin{array}{ccccc} u & \cdots & \alpha & & \\ & & \vdots & \ddots & \\ & & \vdots & & \gamma \\ & & \vdots & \iddots & \\ v & \cdots & \beta & & \end{array}$$

Acest graf are urmatoarele proprietati. Daca u si v sunt FALSE, atunci si c este FALSE. Daca cel putin unul dintre varfurile u si v este TRUE, atunci exista o colorare a grafului astfel incat γ este TRUE. Spunem ca acest graf modeleaza conjunctia $u \wedge w$.

Iesirea c a acestui graf se poate conecta la un graf asemanator, dupa cum urmeaza:

$$\begin{array}{ccccccccc} u & \cdots & \alpha & & & & & & \\ & & \vdots & \ddots & & & & & \\ & & \vdots & & \gamma & \cdots & \delta & & \\ & & \vdots & \iddots & & & \vdots & \ddots & \\ v & \cdots & \beta & & & & \vdots & & \mu \\ & & & & & & \vdots & \iddots & \\ w & \cdots & \cdots & \cdots & \cdots & \cdots & \varepsilon & & \end{array}$$

Acest graf modeleaza o clauza $u \wedge v \wedge w$ in modul urmator: daca varfurile u, v si w au toate culoarea FALSE, atunci varful μ trebuie sa aiba culoarea FALSE. Daca cel putin unul dintre cele trei varfuri are culoarea TRUE, va exista o colorare a grafului in care μ are culoarea TRUE.

- Pentru fiecare dintre cele m clauze se deseneaza o copie a acestui graf. In loc de u, v si w se fac legaturi cu literalele x_i respectiv $\overline{x_i}$, din definitia clauzei. Pentru a forta varfurile μ sa fie TRUE, pentru varful μ corespunzator fiecarei clauze, se introduc muchiile μf si μb.

Este clar ca satisfiabilitatea formulei propozitionale este echivalenta cu faptul ca graful construit are o 3-colorare. $\square$

Definitie: Problema *PLANAR 3COLORING* se defineste in modul urmator:

Input: Un graf planar neorientat $G = (V, E)$.

Question: Exista o colorare a lui V in 3 culori, astfel incat fiecare muchie sa aiba capetele de culori diferite?

Teorema: *PLANAR 3COLORING este* NP*-completa.*

Demonstratie[12]: Problema este in NP. O masina genereaza aleator o functie $f : V \to \{1, 2, 3\}$ apoi verifica in mod determinist conditia din enunt.

Problema este NP-hard. Pentru a costrui o reductie *3COLORING* $\leq_p$ *PLANAR 3COLORING*, vom examina graful urmator H. Iata o colorare a lui H in care varfurile Nord si Sud au aceeasi culoare, iar varfurile EST si Vest au aceeasi culoare.

$$\begin{array}{ccccccccc}
 & & & & b & & & & \\
 & & & \iddots & \vdots & \ddots & & & \\
 & & r & \cdots & g & & r & & \\
 & \iddots & & \iddots & \vdots & \ddots & \vdots & \ddots & \\
g & \cdots & b & \cdots & r & \cdots & b & \cdots & g \\
 & \ddots & \vdots & \ddots & \vdots & \iddots & & \iddots & \\
 & & r & & g & \cdots & r & & \\
 & & & \ddots & \vdots & \iddots & & & \\
 & & & & b & & & &
\end{array}$$

Iata si o alta colorare, in care toate cele patru varfuri au aceeasi culoare:

$$\begin{array}{ccccccccc}
 & & & & r & & & & \\
 & & & \iddots & \vdots & \ddots & & & \\
 & & b & \cdots & g & & g & & \\
 & \iddots & & \iddots & \vdots & \ddots & \vdots & \ddots & \\
r & \cdots & b & \cdots & r & \cdots & b & \cdots & r \\
 & \ddots & \vdots & \ddots & \vdots & \iddots & & \iddots & \\
 & & g & & g & \cdots & b & & \\
 & & & \ddots & \vdots & \iddots & & & \\
 & & & & r & & & &
\end{array}$$

Se poate arata usor ca orice colorare valida a grafului H in 3 culori are proprietatea ca cele doua perechi de varfuri opuse au aceeasi culoare.

Fie G un graf oarecare. Daca G nu este planar, fiecare intersectie de muchii care nu este varf se inlocuieste cu un subgraf H. Mai exact, daca intersectia are loc intre doua muchii Nord-Sud si Vest-Est din G, se identifica varful Nord din G cu varful Nord din H, varful Vest din G cu varful Vest din H, si se traseaza noile muchii Est(H) - Est(G) si Sud(H)-Sud(G). In cazul in care o muchie intalneste mai multe muchii si intersectiile nu sunt varfuri in G, se pot lipi acolo mai multe copii ale lui H. In final se obtine un graf planar $\overline{G}$ care este 3-colorabil daca si numai daca G este 3-colorabil. Ambele colorari ale lui H sunt in general necesare. $\square$

Definitie: Problema *2COLORING* se defineste in modul urmator:

Input: Un graf neorientat $G = (V, E)$.

Question: Exista o colorare a lui V in 2 culori, astfel incat fiecare muchie sa aiba capetele de culori diferite?

Observatie: *2COLORING este in* P.

[12]Michael R. Garey, David E. Johnson: COMPUTERS & INTRACTABILITY, A GUIDE to the Theory of NP-Completeness.

Demonstratie: Fara a restrange generalitatea, consideram ca graful G este conex. Fie v un varf in G. Coloram v in alb. Coloram toti vecinii lui v in negru. Coloram toti vecinii lor in alb. Etc. Acest procedeu se termina sau cu un conflict, caz in care oprim si dam raspunsul ”nu”, sau cu colorarea corecta a intregului graf, caz in care dam raspunsul ”da”. Cum fiecare muchie se considera o singura data, timpul este polinomial in lungimea totala a matricii grafului. $\square$

25 MATH

Fie AX un sistem fundamental de axiome pentru Matematica, de exemplu ZFC (Zermelo-Fraenkel with the Axiom of Choice) sau NBG (Neumann-Bernays-Gödel Axiom System).

Definitie: Problema *MATH* se defineste in modul urmator:

$$MATH = \{\varphi\#^r \mid \mathrm{AX} \vdash \varphi \text{ si exista o demonstratie de lungime } \leq r\}.$$

Teorema: *MATH este* NP*-completa.*

Demonstratie: Problema este in NP. Se ghiceste un sir de caractere de lungime r si se verifica faptul ca acest sir de caractere este o demonstratie si ca ultima formula dedusa este φ. Structura demonstratiei poate fi data prin definitie astfel incat verificarea sa dureze un timp cel mult patratic. Astfel, fiecare rand din demonstratia standard poate fi numerotat. Un rand poate incepe cu:

$$(8)\ \mathrm{Ax}\colon \dots$$

in cazul in care avem de a face cu o axioma, sau cu:

$$(8)\ \mathrm{Taut}\colon \dots$$

daca avem de a face cu una din tautologiile necesare in sistemul respectiv, sau cu

$$(8)\ \mathrm{MP}(5,7) : \dots$$

daca avem de a face cu aplicarea regulii Modus Ponens pentru formulele 5 si 7, sau cu:

$$(8)\ \mathrm{G}(5,x) : \dots$$

daca avem de a face cu aplicarea regulii de generalizare.

Problema este NP-hard. Aratam ca *3SAT* $\leq_p$ *MATH*. Fie F o formula 3CNF. Daca formula este satisfiabila, pentru un anumit $\vec{A} \in \{0,1\}$, $F(\vec{A})$ se poate demonstra in AX. Din $F(\vec{A})$ se poate deduce in AX $\exists\, x_1, \dots, x_n \in \{0,1\} F(\vec{x})$. Aceasta demonstratie nu este mai lunga decat $K|F|^2$, unde $K \in \mathbb{N}$ este o constanta care depinde doar de conventiile alese. Asadar, reductia este:

$$F \leadsto (\exists\, x_1, \dots, x_n \in \{0,1\} \quad F(\vec{x}))\#^{K|F|^2}.$$

$\square$

Desi niciun sistem axiomatic nu poate demonstra intreaga matematica, conform Teoremei lui Gödel, se considera ca problema *MATH* incorporeaza cea mai dificila parte a procesului de gandire si descoperire in matematica[13]. Desi autorul acestui curs nu este neapart de acord cu aceasta idee[14], este adevarat ca in cazul in care P = NP ar exista urmatoarea metoda automata de **semidecizie** pentru a afla daca o teorema φ este demonstrabila in ZFC. Sirul de cuvnte $(\varphi\#^k)_{k\in\mathbb{N}}$ ar fi introdus in masina determinista in timp polinomial care decide *MATH* pana cand aceasta ar

[13] Neil Immerman, Descriptive Complexity, Springer Verlag 1998.

[14] Neil Barton, Moritz Müller, Mihai Prunescu, On Representations of Intended Structures in Foundational Theories. Journal of Philosophical Logic 2021.

accepta cuvantul respectiv. Intregul procedeu ar dura un timp polinomial in lungimea celei mai scurte demonstratii a lui φ.

Intr-o scrisoare din 1956, Kurt Gödel ii scria lui John von Neumann: *daca aceasta problema ar putea fi rezolvata in timp patratic, indecidabilitatea problemei de decizie ar fi mai putin descurajatoare, deoarece matematica se preocupa in general cu demonstratii scurte, care incap in cateva carti...* Intr-adevar, o rezolvare a problemei *MATH* in timp patratic ar face din Matematica o adevarata *computational Utopia*. Se poate spune ca Kurt Gödel a formulat cu aceasta ocazie prima data problema P versus NP. Din pacate el nu a publicat problema, si ea a ramas necunoscuta pana la redescoperirea ei de catre Cook la sfarsitul anilor 1960. Totusi, pentru a sublinia contributia lui Gödel in aceasta directie, unul dintre cele mai importante premii care se acorda pentru rezultate importante in Informatica Teoretica se numeste Premiul Gödel.

Part V

Alte clase de complexitate

Cea mai mare parte a informatiilor cuprinse in aceasta parte a cursului sunt luate din cartea lui Sanjeev Arora si Boaz Barak, Computational Complexity, A Modern Approach.

26 Algoritmi pseudopolinomiali

Notiunea de algoritm pseudopolinomial exprima faptul ca timpul de lucru al algoritmului este polinomial in anumiti parametrii de care depinde problema. Acesti parametri nu reprezinta masura clasica de complexitatea a problemei, adica lungimea inputului, dar pot avea totusi relevanta practica. In cazurile unor probleme NP-complete, parametrii respectivi sunt exponentiali in lungimea inputului. Ilustram aceasta situatie cu problema *SUBSET SUM*.

Consideram urmatoarea varianta a problemei *SUBSET SUM*. Inputul are forma:

$$x_1, x_2, \dots, x_k, b$$

cu $x_1, x_2, \dots, x_k, b \in \mathbb{Z}$. Fie A suma tuturor valorilor negative x_i si fie B suma tuturor valorilor pozitive x_i. Urmatorul algoritm, bazat pe programarea dinamica, rezolva aceasta problema. Fie s o noua variabila. Definim:

$$Q(i,s) = \begin{cases} 1, & \exists\, J \subseteq \{1,\dots,i\} \quad J \neq \emptyset \wedge s = \sum_{j\in J} x_j, \\ 0, & \text{altfel.} \end{cases}$$

Observam ca $Q(i,s) = 0$ daca $s < A$ sau daca $s > B$. Folosind acest lucru, se poate calcula $Q(i,s)$ inductiv. Pentru $A \leq s \leq B$,

$$Q(1,s) = \begin{cases} 1, & s = x_1, \\ 0, & \text{altfel.} \end{cases}$$

Pentru $2 \leq i \leq k$ si $A \leq s \leq B$, definim:

$$Q(i,s) = Q(i-1,s) \vee (x_i = s) \vee Q(i-1, s-x_i).$$

Raspunsul problemei este $Q(k,b)$. Acest algoritm necesita $O(k(B-A))$ operatii aritmetice. Timpul se considera pseudopolinomial in ordinul de marime 10^k al numerelor care apar in problema, dar este exponential in lungimea inputului, privit ca sir de cifre, virgule si semne algebrice. $\square$

Acest fapt are si o consecinta surprinzatoare:

Definitie Problema *UNARY SUBSET SUM* are inputs de forma:

$$un(x_1)\#un(x_2)\#\dots\#un(x_k)\#un(b).$$

Cuvantul este acceptat daca si numai daca exista $J \neq \emptyset$, $J \subseteq \{1,\dots,i\}$ astfel incat $b = \sum_{j\in J} x_j$.

Desi problema inrudita *SUBSET SUM* este NP-completa, situatia problemei *UNARY SUBSET SUM* este cu totul alta:

Teorema: *Problema UNARY SUBSET SUM este in clasa* P.

Demonstratie: Fie n lungimea inputului. Cu notatiile din algoritmul pseudopolinomial de mai sus, $A = 0$, $B = \sum_{i=1}^{k} x_i < n$, $k < n$ si algoritmul necesita $O(n^2)$ operatii algebrice. Cum fiecare operatie algebrica unara se face in timp $O(n)$, algoritmul determinist de decizie necesita timp polinomial $O(n^3)$. $\square$

27 Limbaje unare

Problema *UNARY SUBSET SUM* din paragraful precedent nu este un limbaj unar, intrucat scrierea instantelor are nevoie de doua simboluri, 1 si #. Totusi ea indica faptul ca in cazul limbajelor unare apar anumite exceptii si particularitati.

Un limbaj unar L este o submultime $L \subseteq \{1\}^*$. La inceputul capitolului despre P si NP am spus ca in probleme de complexitate, alfabetele au cel putin doua litere. De data aceasta vom face o exceptie. Urmatorul fapt este remarcabil: limbajele unare nu pot fi NP-complete daca P $\neq$ NP.

Teorema: *(Berman) Daca un limbaj unar L este* NP*-complet, atunci* P = NP.

Demonstratie: Fie L un limbaj unar NP-complet. In particular exista o reductie polinomiala $SAT \leq_p L$, fie g functia care realizeaza aceasta reductie, deci $F \in SAT$ daca si numai daca $g(F) \in L$. Fie $F = F(x_1, \dots, x_n)$ o formula propozitionala, astfel incat $x_1, \dots, x_n$ sunt toate variabilele propozitionale care apar in aceasta formula. Fie $m = |F|$. Cum g este calculabila in timp polinomial, $g(F) = |g(F)| \leq Cm^k$ pentru anumite constante C si k care depind numai de g. Notam Cm^k cu M.

Vom descrie un algoritm de decizie determinist in timp polinomial pentru SAT.

Efectuam urmatoarele operatii. Scriem formulele $F_0 = F(0, x_2, \dots, x_n)$ si $F_1 = F(1, x_2, \dots, x_n)$. Daca $g(F_0) = g(F_1)$ atunci aceste formule sunt in aceeasi masura satisfiabile, astfel incat o stergem pe F_1 si continuam numai cu F_0. Observam ca ambele formule F_0 si F_1 sunt mai scurte decat F, astfel incat valorile corespunzatoare ale lui g sunt $\leq M$.

Daca dupa inlocuirea valorilor $x_1, \dots, x_i$ am obtinut o lista de formule $F_{w_1}, \dots, F_{w_k}$, unde w_j sunt cuvinte binare de lungime i. Acum pentru fiecare formula inlocuim x_{i+1} cu valorile 0 si 1. Se obtine lista $F_{w_1 0}, F_{w_1 1}, \dots, F_{w_k 0}, F_{w_k 1}$. Pentru aceste formule se calculeaza toti $g(F_{w_j b})$ si se obtin valori $< M$. Pentru fiecare valoare a lui g obtinuta se lasa pe lista un singur reprezentant F_{w_k}, iar celelalte se sterg.

Dupa inlocuirea lui x_n cu 0 si 1 si dupa curatirea listei astfel incat pentru fiecare valoare $g(F_w)$ (unde $w \in \{0,1\}^n$) sa ramana pe lista un singur reprezentant, lista va fi formata doar din expresii booleene constante, care se pot evalua. Daca cel putin una dintre ele are valoarea 1, F este satisfiabila, altfel nu.

La fiecare substitutie de variabila, lista are cel mult $2M$ elemente, si dupa fiecare proces de stergere a unor formule, lista are cel mult M elemente. Asadar timpul de calcul este polinomial. □

In concluzie, limbajele unare trebuiesc tratate separat:

Definitie: Pentru alfabetul $\Sigma = \{1\}^*$ definim P_1 (sau tally P) clasa limbajelor unare care sunt decise in timp polinomial de masini Turing deterministe si NP_1 (sau tally NP) clasa limbajelor recunoscute in timp polinomial de masini Turing nedeterministe. Problema deschisa corespunzatoare, daca:

$$\mathrm{P}_1 \neq \mathrm{NP}_1,$$

se numeste *tally P versus NP problem.* □

28 Clasa coNP

Definitie: Fie $\Sigma = \{0,1\}$. Definim clasa de multimi coNP in felul urmator:

$$\mathrm{coNP} = \{L \subset \Sigma^* \,|\, (\Sigma^* \setminus L) \in \mathrm{NP}\}.$$

In realitate clasa se poate defini folosind orice alt alfabet finit. □

Clasa coNP nu este complementul clasei NP. De fapt, NP ∩ coNP ≠ ∅ deoarece P ⊆ NP ∩ coNP. Urmatoarea problema este un exemplu de problema in coNP:

$$\overline{SAT} = \{F \,|\, F \text{ nu este satisfiabila}\}.$$

Aceasta ne permite urmatoarea definitie alternativa:

Definitie: Pentru orice $L \subseteq \{0,1\}^*$, spunem ca $L \in$ coNP daca exista un polinom $p : \mathbb{N} \to \mathbb{N}$ si o masina M in timp polinomial astfel incat pentru orice $x \in \{0,1\}^*$,

$$x \in L \Longleftrightarrow \forall u \in \{0,1\}^{p(|x|)} \quad M(x,u) = 1.$$

□

Definitie: O formula propozitionala $F(\vec{x})$ se numeste tautologie daca ia valoarea 1 pentru orice valori ale variabilelor $\vec{x} \in \{0,1\}^n$. Definim problema:

$$TAUTOLOGY = \{F \mid F \text{ este o tautologie}\}.$$

Teorema: *Problema TAUTOLOGY este* coNP-*completa.*

Demonstratie: Din definitie este clar ca $TAUTOLOGY$ este in coNP. Tot ce trebuie sa aratam este ca daca $L \in$ coNP, atunci $L \leq_p TAUTOLOGY$. Pentru aceasta trebuie sa modificam reductia Cook de la $\{0,1\}^* \setminus L$, care este in NP, la SAT. Aceasta reductie produce pentru fiecare $x \in \{0,1\}^*$ o formula propozitionala F_x care este satisfiabila daca si numai daca $x \in \{0,1\}^* \setminus L$. Consideram formula $\neg F_x$. Aceasta este in $TAUTOLOGY$ daca si numai daca $x \in L$. □

Observatie: Daca P = NP atunci NP = coNP = P. Acest lucru se poate interpreta si in urmatorul mod: daca aratam ca NP $\neq$ coNP atunci am aratat automat si P $\neq$ NP. □

29 Clasele EXP si NEXP

Definitie: Clasele EXP si NEXP sunt analogul exponential al claselor P si NP. Mai exact:

$$\text{EXP} = \bigcup_{c \geq 1} \text{DTIME}\Big(2^{n^c}\Big),$$

$$\text{NEXP} = \bigcup_{c \geq 1} \text{NTIME}\Big(2^{n^c}\Big).$$

□

Desi la prima vedere nu merita sa ne ocupam de asemenea obiecte, din moment ce problema P versus NP, care este mult mai importanta din punct de vedere practic, nu a fost rezolvata, urmatorul rezultat teoretic indica o legatura cu problema P versus NP.

Teorema: *Daca* EXP $\neq$ NEXP *atunci* P $\neq$ NP.

Demonstratie: Presupunem ca P = NP si aratam ca in acest caz EXP = NEXP. Fie $L \in$ NTIME$\Big(2^{n^c}\Big)$ si ca masina Turing nedeterminista M il accepta pe L. Fie problema:

$$L_{\text{pad}} = \Big\{ \Big(x, 1^{2^{|x|^c}}\Big) \mid x \in L \Big\}.$$

Iata o masina Turing nedeterminista pentru problema L_{pad}. Dat un cuvant y, verifica daca exista un subcuvant z astfel incat:

$$y = \Big(z, 1^{2^{|x|^c}}\Big).$$

Daca nu exista, intoarce 0 si opreste. Daca y este de forma aceasta, simuleaza M pentru inputul z un numar de $2^{|z|^c}$ pasi si intoarce starea lui M. Timpul de lucru este polinomial in $|y|$ si deci L_{pad} este in NP. Daca P = NP, atunci L_{pad} este in P. Aratam ca L este in EXP. Ca sa decidem in mod determinist daca $x \in L$, il concatenam pe x cu $2^{|x|^c}$ de 1 si decidem folosind o masina Turing determinista in timp polinomial daca cuvantul astfel obtinut este in L_{pad}. Timpul total de lucru este exponential. □

30 Time and Space Hierarchy Theorems

Definitie: O functie $T : \mathbb{N} \to \mathbb{N}$ se numeste timp-construibila daca $T(n) \geq n$ si exista o masina Turing M care calculeaza functia $x \rightsquigarrow bin(T(|x|))$ in timp $T(n)$. Exemple de functii timp-construibile sunt n, $n \log n$, n^2, 2^n. Timpul de lucru se exprima de obicei in functii timp-constribile. Restrictia $T(n) \geq n$ se pune pentru a ii permite masinii sa citeasca inputul. □

Ne reamintim ca clasa DTIME($f(n)$) se defineste ca multimea limbajelor decise de masini Turing deterministe in timp $O(f(n))$.

Teorema: *(Time Hierarchy Theorem) Daca functiile f si g sunt timp-construibile si $f(n)\log f(n) = o(g(n))$, atunci:*

$$\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n)).$$

Demonstratie: Pentru a ilustra ideea demonstratiei fara a complica prea mult notatiile, demonstram aici doar cazul particular DTIME(n) $\subsetneq$ DTIME($n^{1.5}$).

Urmatorul lucru este cunoscut: Exista o masina Turing universala U cu mai multe benzi care simuleaza orice masina Turing (cu mai multe benzi) M_α. Mai mult, daca M_α opreste cu inputul x dupa T pasi, atunci $U(x,\alpha)$ opreste dupa $CT\log T$ pasi. Aici C este o constanta care depinde numai de numarul de benzi, numarul de stari si alfabetul lui M_α dar nu si de x.

Consideram urmatoarea masina Turing D: *Cu input x, functioneaza precum masina universala U un numar de $|x|^{1.4}$ pasi pentru a simula executia lui M_x pe inputul x. Daca U opreste si da o valoare $b \in \{0,1\}$ in acest timp, atunci da raspunsul opus $1-b$. Altfel da raspunsul 0.*

Prin definitie, masina D opreste in $\leq n^{1.4}$ pasi, deci limbajul L decis de D este in DTIME($n^{1.5}$). Vom arata ca $L \notin$ DTIME(n). Pentru a ajunge la o contradictie, presupunem ca exista o masina Turing M si o constanta c astfel incat pentru orice input $x \in \{0,1\}^*$, M opreste in cel mult $c|x|$ pasi si calculeaza raspunsul $D(x)$.

Timpul necesar simularii masinii M de catre masina universala U este cel mult $c'c|x|\log(c|x|)$, unde c' nu depinde de x. Exista un numar n_0 astfel incat $n^{1.4} > c'cn\log(cn)$ pentru orice $n \geq c_0$. Fie x un cuvant care codifica masina M si are o lungime $n \geq n_0$. Un asemenea cuvant exista fiindca orice masina este codificata de o infinitate de cuvinte. (De exemplu, se pot introduce stari inaccesibile, si masina nu se schimba.) Deci $D(x)$ va obtine outputul $b = M(x)$ in cel mult $|x^{1.4}|$ pasi, dar din definitia lui D avem $D(x) = 1 - b \neq M(x)$. Contradictie. $\square$

Definitie: Fie $S : \mathbb{N} \to \mathbb{N}$ o functie si $L \subset \{0,1\}^*$ o multime. Spunem ca $L \in$ SPACE($S(n)$) daca exista o constanta c si o masina Turing determinista M care decide L fara sa foloseasca mai mult de $cS(n)$ casute pe fiecare banda de lucru. [In aceasta definitie se exclude banda de input, dar se face conventia ca pe banda de input evolueaza doar un cap de citire.] Similar se defineste NSPACE($S(n)$), doar ca M poate fi si o masina nedeterminista. $\square$

Definitie: O functie $S : \mathbb{N} \to \mathbb{N}$ se numeste spatiu-construibila, daca exista o masina Turing determinista care calculeaza $S(|x|)$ in spatiu $O(S(|x|)$ pentru orice input x. Functiile $\log n$, n si 2^n sunt spatiu construibile. $\square$

Teorema: *(Space Hierarchy Theorem) Daca functiile f si g sunt spatiu-construibile si $f(n) = o(g(n))$, atunci:*

$$\mathrm{DSPACE}(f(n)) \subsetneq \mathrm{DSPACE}(g(n)).$$

Demonstratie: Demonstratia este complet analoga cu demonstratia de la Time Hierarchy Theorem, cu diferenta ca se poate obtine mai usor o masina universala care sa aiba nevoie de o expansiune a spatiului cu un factor constant c. De aceea nu este nevoie de factorul logaritmic de mai sus. $\square$

Observatie: Exista rezultate similare (Non-determinist Hierarchy Theorems) pentru clasele nedeterministe NTIME($f(n)$) si NSPACE($f(n)$). $\square$

31 Spatiu marginit

Definitie: O configuratie a unei masini Turing cu k benzi este un tuplu $(z, x_1y_1, \dots, x_ky_k)$. Aici $z \in Z$ este o stare a masinii iar x_iy_i este cuvantul scris pe banda i la un moment dat. Capul de citire, citire-scriere sau scriere de pe banda i indica prima litera a cuvantului y_i. $\square$

Definitie: Pentru orice masina Turing M care lucreaza pe spatiu marginit $S(n)$ si primeste un input $x \in \{0,1\}^*$, graful configuratiilor $G_{M,x}$ este un graf orientat ale carui varfuri sunt posibilele configuratii ale masinii Turing, in care orice banda are cel mult $S(|x|)$ celule diferite de blanc. Exista o muchie de la configuratia C la configuratia C' daca si numai daca exista o tranzitie a masinii Turing de la C la C'. In cazul in care masina este determinista, toate varfurile grafului au fan-out cel mult 1, adica cel mult o sageata iese din fiecare varf. In cazul in care masina este nedeterminista, ea poate fi reformulata (normata) astfel incat fiecare varf sa aiba fan-out cel mult 2. Mai mult, M poate fi modificata sa isi stearga toate benzile inainte de a opri - observati ca acest lucru nu modifica conditia de spatiu marginit. Astfel se poate presupune ca exista o singura configuratie C_{accept} cu care masina opreste si returneaza valoarea 1. Asadar masina accepta inputul x daca si numai daca exista un drum ordonat de la C_{start} la C_{accept}. □

Lema: *Fie $G_{M,x}$ graful configuratiilor unei masini Turing cu spatiu $S(n)$ pentru un input x de lungime n. Atunci:*

1. *Fiecare varf in graful $G_{M,x}$ poate fi descris folosind $cS(n)$ biti pentru o constanta c care depinde doar de alfabetul lui M, numarul de stari si numarul ei de benzi. In particular acest graf are cel mult $2^{cS(n)}$ varfuri.*

2. *Exista o formula propozitionala* CNF *$\varphi_{M,x}$ de marime $O(S(n))$ astfel incat pentru orice doua cuvinte binare C si C', $\varphi_{M,x}(C,C') = 1$ daca si numai daca C si C' codifica doua configuratii ale masinii M care se invecineaza in graful $G_{M,x}$.*

□

Prima afirmatie este standard. A doua se demonstreaza ca la Teorema lui Cook.

Lema: *Se considera problema urmatoare: fiind dat un graf orientat G si doua varfuri s, t ale lui G, sa se decida daca exista un drum ordonat de la s la t. Aceasta problema este rezolvata de un algoritm determinist in timp polinomial in $n = |G|$.*

Demonstratie: Este vorba de un algoritm breadth-first search. Se marcheaza varful s, multimea $V_0 := \{s\}$. La pasul urmator se marcheaza varfurile la care se poate ajunge din s intr-un pas, rezulta multimea $V_1 \supseteq V_0$. Data fiind multimea V_k, se adauga acele varfuri care se pot vizita din V_k si nu sunt inca in V_k. Se obtine multimea $V_{k+1} \supseteq V_k$. Daca varful $t \in V_{k+1}$, intoarce 1. Daca $V_{k+1} = V_k \not\ni t$, intoarce 0. Altfel continua. □

Teorema: *Pentru orice functie spatiu construibila $S : \mathbb{N} \to \mathbb{N}$,*

$$\mathrm{DTIME}(S(n)) \subseteq \mathrm{SPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{O(S(n))}).$$

Demonstratie: Primele doua incluziuni sunt usor de explicat, deci ne referim la a treia. Graful orientat $G_{M,x}$ se poate construi in timpul $2^{O(S(n))}$. Dupa aceea se decide in timp $p(2^{O(S(n))}) \subset 2^{O(S(n))}$ daca exista un drum orientat intre C_{start} si C_{accept}. □

32 PSPACE

Definitie:

$$\mathrm{PSPACE} = \bigcup_{c>0} \mathrm{SPACE}(n^c),$$

$$\mathrm{NPSPACE} = \bigcup_{c>0} \mathrm{NSPACE}(n^c).$$

Observam ca P $\subseteq$ PSPACE, de asemeni *SAT* $\in$ PSPACE, ba chiar NP $\subseteq$ PSPACE. Deci P $=$ PSPACE ar implica P $=$ NP. Pentru reductii in PSPACE se foloseste aceeasi definitie de reductie in *timp* polinomial care s-a folosit si in cazul clasei NP.

Definitie: O problema L' este PSPACE-hard daca pentru orice problema $L \in$ PSPACE , $L \leq_p L'$. Daca $L' \in$ PSPACE, spunem ca L' este PSPACE-completa. $\square$

Un exemplu standard de problema PSPACE-completa este *SPACE TMSAT*, definita ca:

$$SPACE\ TMSAT = \{(M, w, 1^n) \,|\, \text{masina Turing determinsta } M \text{ accepta } w \text{ in spatiu } n\},$$

in sensul ca foloseste cel mult n casute pe fiecare banda de lucru, cu exceptia benzii de input.

O problema PSPACE-completa mai interesanta va fi definita mai jos.

Definitie: O formula booleeana quantificata (QBF) este o formula de forma:

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \ \varphi(x_1, x_2, \dots, x_n)$$

unde fiecare Q_i semnifica un cuantificator $Q_i \in \{\forall, \exists\}$, toate variabilele x_i se refera la domeniul binar $\{0, 1\}$ iar $\varphi(x_1, x_2, \dots, x_n)$ este o formula din logica propozitionala. $\square$

Desi se cere ca quantificatorii sa fie la inceputul formulei, acest lucru nu este cu adevarat o restrictie, deoarece ei pot fi adusi in timp polinomial la inceputul formulei folosind identitati de genul $\neg\forall x\varphi(x) = \exists x \neg\varphi(x)$ sau $\psi \vee \exists x \varphi(x) = \exists x\ \psi \vee \varphi(x)$. De asemeni, nu se cere ca φ sa fie intr-o forma speciala, precum CNF sau 3CNF, deoarece se stie ca se pot introduce variabile noi si φ se poate aduce intr-o forma echivalenta din punctul de vedere al satisfiabilitatii, care sa fie 3CNF.

Exemplu: Formula cuantificata:

$$\forall\, x\, \exists\, y\ (x \wedge y) \vee (\neg x \wedge \neg y)$$

inseamna *pentru orice* $x \in \{0, 1\}$ *exista un* $y \in \{0, 1\}$ *astfel incat* $x = y$ si este adevarata. Formula:

$$\forall\, x\, \forall\, y\ (x \wedge y) \vee (\neg x \wedge \neg y)$$

este falsa. Alegand $x = 0$ si $y = 1$, producem o demonstratie a faptului ca formula este falsa.

Definitie: Problema *TQBF*, adica True Quantified Boolean Formulas, este multimea acelor QBF care sunt adevarate. $\square$

Teorema: *Problema TQBF este* PSPACE-*completa.*

Demonstratie: Problema *TQBF* este in PSPACE. Fie:

$$\psi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \ \varphi(x_1, x_2, \dots, x_n)$$

o instanta a problemei *TQBF*. Notam lungimea ei cu m. Vom arata ca valoarea de adevar a cestei propozitii se poate decide in spatiu $O(n+m) = O(m)$. Rezolvam cazul putin mai general in care in afara de variabile, negatii, conjunctii, disjunctii si paranteze, formula poate contine si constantele 0 si 1. Daca $n = 0$ formula nu contine variabile si poate fi evaluata in timp si spatiu $O(m)$. Fie $n > 0$. Pentru $b \in \{0, 1\}$ notam cu $\psi_{|x_1=b}$ formula obtinuta din ψ prin stergerea literelor $Q_1 x_1$ din prefix si inlocuirea oricarei ocurente ale lui x_1 cu b. Daca $Q_1 = \forall$, atunci:

$$\psi \leftrightarrow \psi_{|x_1=0} \wedge \psi_{|x_1=1}.$$

Daca $Q_1 = \exists$, atunci:

$$\psi \leftrightarrow \psi_{|x_1=0} \vee \psi_{|x_1=1}.$$

Punctul esential este ca ambele evaluari $\psi_{|x_1=0}$ si $\psi_{|x_1=1}$ pot fi facute refolosind acelasi spatiu de calcul. Mai exact, dupa ce a fost calculata valoarea de adevar a lui $\psi_{|x_1=0}$, algoritmul trebuie sa retina un singur bit si poate refolosi restul spatiului pe care il are la dispozitie.

Problema *TQBF* este PSPACE-hard. Fie $L \in$ PSPACE. Vrem sa aratam ca $L \leq_p$ *TQBF*. Fie M o masina Turing determinista care decide L in spatiu $S(n)$ si fie $x \in \{0, 1\}^n$. Vom arata cum se construieste o formula booleeana cuantificata de lungime $O(S(n)^2)$ care este adevarata daca si numai daca M accepta x. Fie $m = O(S(n))$ numarul de biti necesari codificarii unei configuratii a masinii M pentru inputs de lungime n. Stim ca exista o formula booleeana $\varphi_{M,x}$ astfel incat pentru orice doua tupluri $C, C' \in \{0,1\}^m$, $\varphi_{M,x}(C, C') = 1$ daca si numai daca exista o sageata $C \to C'$ in graful orientat al configuratiilor asociat perechii (M, x). Folosim $\varphi_{M,x}$ pentru a genera o formula booleeana cuantificata $\psi(C, C')$ care este adevarata daca si numai daca exista un drum orientat de la C la C'. In aceasta formula se pot substitui C cu C_{start} si C' cu C_{accept} si gasim formula adevarata daca si numai daca M il accepta pe x.

Definim formula ψ prin inductie. Punem $\psi_0(C, C') = \varphi_{M,x}(C, C') \vee C = C'$ si $\psi_i(C, C')$ sa fie adevarat daca si numai daca exista un drum de lungime cel mult 2^i de la C la C' in $G_{M,x}$. Atunci formula cautata este $\psi = \psi_m$. Ideea este sa construim $\psi_i(C, C')$ in modul urmator:

$$\psi_i(C, C') = \exists C'' \ \psi_{i-1}(C, C'') \wedge \psi_{i-1}(C'', C').$$

Aceasta formula este corecta prin prisma a ceea ce exprima, dar nu poate fi folosita pentru inductie, pentru ca ar duce la o formula ψ de lungime $O(2^m)$, ceea ce este prea mult. In loc de asta folosim niste cuantificatori suplimentari, si definim $\psi_i(C, C')$:

$$\exists C'' \forall D^1 \forall D^2 \Big((D^1 = C \wedge D^2 = C'') \vee (D^1 = C'' \wedge D^2 = C') \Big) \to \psi_{i-1}(D^1, D^2).$$

Observam ca $|\psi_i| \leq |\psi_{i-1}| + O(m)$ ceea ce face ca $|\psi_m| \leq O(m^2)$. Formula finala poate fi usor transformata intr-o formula cu toti cuantificatorii la inceput. □

33 PSPACE = NPSPACE

Cand am demonstrat ca problema *TQBF* este PSPACE-completa, nu am folosit nicaieri presupunerea ca graful $G_{M,x}$ este format din noduri cu fan-out egal cu 1! Asadar teorema este mai generala si arata de fapt ca *TQBF* este NPSPACE-completa. Dar fiind o problema rezolvabila de masini deterministe in spatiu polinomial, aceasta implica urmatorul corolar:

Corolar: PSPACE = NPSPACE.

Acest fapt se poate generaliza imediat:

Teorema: *(lui Savitch) Pentru orice functie spatiu-construibila* $S : \mathbb{N} \to \mathbb{N}$ *cu* $S(n) \geq \log n$*,* NSPACE$(S(n)) \subseteq$ SPACE$(S(n)^2)$.

Demonstratie: Demonstratia este foarte asemanatoare cu cea a PSPACE-completitudinii lui *TQBF*. Fie $L \in$ NSPACE$(S(n)$ un limbaj recunoscut de o masina Turing nedeterminista M astfel incat pentru fiecare $x \in \{0,1\}^*$ graful configuratiilor $G_{M,x}$ are cel mult $N = 2^{O(S(n))}$ varfuri iar $x \in L$ este echivalent cu existenta unui drum orientat de la C_{start} la C_{accept}. Iata un program REACH(u, v, i) care da un raspuns boolean la intrebarea daca de la u la v exista un drum orientat de lungime $\leq 2^i$. Daca $i = 0$, programul returneaza 1 daca $u = v$ sau daca $\varphi_{M,x}(u, v)$ este adevarata, adica daca muchia orientata uv este in graf, altfel returneaza 0. Pentru $i > 0$, programul enumera toate varfurile w din graf folosind spatiu $\log N = O(S(n))$ si va calcula REACH$(u, w, i-1)$ si REACH$(w, v, i-1)$. In acest caz programul returneaza 1 daca si numai daca ambele subrutine returneaza 1. Desi algoritmul face n invocatii recursive, el poate refolosi spatiul la fiecare dintre invocatii. Daca $s_{M,i}$ este spatiul folosit de catre REACH(u, v, i), $s_{M,i} = s_{M,i-1} + O(\log N)$ si deci $s_{M,\log N} = O(\log^2 N) = O(S(N)^2)$. Dar de la C_{start} la C_{accept} poate sa existe un drum de lungime cel mult N. □

Concluzie: Stim ca:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXP} \subseteq \text{NEXP}.$$

Din Time Hierarchy Theorem stim ca P $\subsetneq$ EXP. Deci unele din incluziunile dintre ele ar trebui sa fie stricte. De asemenea stim ca daca EXP $\neq$ NEXP atunci P $\neq$ NP. Toate celelalte probleme sunt deschise. □

34 L si NL

Definitie:

$$\mathrm{L} = \mathrm{SPACE}(\log n),$$
$$\mathrm{NL} = \mathrm{NSPACE}(\log n).$$

Despre clasele PSPACE si NPSPACE gandim ca fiind analogul spatial al claselor temporale P si NP. Pentru clasele L si NL nu exista analog temporal, deoarece nu are sens ca timpul de lucru sa fie mai mic decat lungimea inputului. Inputul trebuie in primul rand citit. De aceea s-a convenit ca pentru clasele SPACE modelul de calcul sa contina o banda de input, care este numai read-only, si sa se margineasca doar spatiul pe benzile de lucru, in cazul acesta la $c \log n$, unde c este o constanta care depinde de masina.

Urmatoarele multimi sunt in L:

$$EVEN = \{x \mid x \text{ contine un numar par de cifre } 1\},$$

$$MULT = \{bin(n)\#bin(m)\#bin(nm) \mid m, n \in \mathbb{N}\}.$$

Nu stim multe lucruri despre L, de exemplu nu stim daca L $\neq$ NP.

Pentru a vorbi despre NL, trebuie sa definim un tip special de reductie. Problema care ne preocupa in primul rand este daca L $\neq$ NL. Avand in vedere ca L $\subseteq$ NL $\subseteq$ P, nu putem consider reductii in timp polinomial, fiindca reductiile nu au voie sa fie mai puternice decat cea mai slaba dintre clase, care este L. De aceea trebuie sa definim notiunea de reductie *logspace*, adica o functie care poate fi calculata de o masina determinista in spatiu logaritmic. Problema este insa ca o masina ar putea avea prea putina memorie ca sa poata sa scrie valoarea pe care a calculat-o. De aceea se cere ca masina sa fie capabila sa calculeze un anumit bit al valorii functiei, care este cerut explicit, in spatiu logaritmic.

Definitie: O functie $f : \{0,1\}^* \to \{0,1\}^*$ se numeste implicit logspace calculabila daca f este polinomial marginita (adica exista un c astfel incat $|f(x)| \leq |x|^c$ pentru orice $x \in \{0,1\}^*$) si limbajele $L_f = \{(x,i) \mid f(x)_i = 1\}$ si $L'_f = \{(x,i) \mid i \leq |f(x)|\}$ sunt in L. $\square$

Definitie: Se spune ca limbajul B este logspace reductibil la limbajul C, si se noteaza $B \leq_l C$, daca exista o functie implicit logspace calculabila $f : \{0,1\}^* \to \{0,1\}^*$ si pentru orice $x \in \{0,1\}^*$, $x \in B$ daca si numai daca $f(x) \in C$. $\square$

Definitie: Spunem ca C este NL-complet daca si numai daca este in NL si orice problema din NL este logspace recuctibila la C. $\square$

Lema: *Compunerea a doua functii implicit logspace calculabile este o functie implicit logspace calculabila.*

Demonstratie: Fie M_f si M_g masini logspace deterministe care calculeaza $(x,i) \rightsquigarrow f(x)_i$ respectiv $(y,j) \rightsquigarrow g(y)_j$. Construim o masina M_h care pentru input (x,j) cu $j \leq |g(f(x))|$ calculeaza $g(f(x))_j$. Masina M_h pretinde ca are o a doua banda fictiva de input pe care este scris $f(x)$ si simuleaza M_g pentru acest "input". Adevarata banda de input contine perechea (x,j). Pentru a isi intretine fictiunea, masina M_h pastreaza pe o banda de lucru indicele i al celulei de pe banda de input fictionala pe care M_g o citeste la momentul prezent. Aceasta informatie necesita spatiu $\log |f(x)|$. Pentru a calcula un pas, masina M_g are nevoie de continutul acestei celule, adica de $f(x)_i$. Pentru aceasta M_h suspenda temporar simularea lui M_g (si pentru aceasta pastreaza continutul benzilor de lucru ale lui M_g pe niste benzi de lucru proprii) si il simuleaza (invoca) pe M_f pentru inputul (x,i) pentru a il calcula pe $f(x)_i$. Apoi reincepe simularea lui M_g folosind acest bit. Spatiul total folosit de M_h este $O(\log |g(f(x))| + s(|x|) + s'(|f(x)|))$. Cum $|f(x)| \leq poly(|x|)$, aceasta expresie este $O(\log |x|)$. $\square$

Lema:

1. *Daca $B \leq_l C$ si $C \leq_l D$ atunci $B \leq_l D$.*

2. Daca $B \leq_l C$ si $C \in$ L atunci $B \in$ L.

Demonstratie: Punctul (1) rezulta imediat din lema anterioara. Punctul (2) rezulta luand pentru f reductia de la B la C si pentru g functia caracteristica a lui C. □

Ne reamintim problema *PATH*, al carei input avea forma (G, s, t), unde G este un graf orientat si s, t sunt doua varfuri, si se punea intrebarea daca exista un drum orientat de la s la t.

Teorema: *PATH este* NL-*completa.*

Demonstratie: *PATH* este in NL. Daca exista un drum de la s la t, va exista si un drum de lungime cel mult n. Masina nedeterminista poate sa se fixeze pe un varf v, sa ii scrie numele cu $\log n$ biti pe o banda de lucru, si apoi in mod nedeterminist sa aleaga un varf v' dintre varfurile pentru care sageata vv' exista. Ea poate scrie numele varfurilor acestea in mod alternativ pe doua benzi de lucru. Pe o a treia banda de lucru, masina numara prin cate varfuri a trecut, si ii trebuie din nou un spatiu de cel mult $\log n$ biti pentru asta. Masina porneste din varful s. Daca intalneste varful t in cel mult n pasi, accepta.

PATH este NL-hard. Fie L un limbaj in NL si fie M masina nedeterminista care il recunoaste pe L in spatiu logaritmic. Descriem o functie implicit logspace calculabila care il reduce pe L la *PATH*. Pentru orice input x de marime n, $f(x)$ va fi graful configuratiilor $G_{M,x}$ ale carui varfuri sunt toate cele $2^{O(\log n)}$ de configuratii posibile ale masinii cu input x, inclusiv configuratia C_{start} si configuratia acceptanta C_{acc}. In acest graf exista un drum ordonat de la C_{start} la C_{acc} daca si numai daca $x \in L$. Graful este reprezentat printr-o matrice de adiacenta care are un 1 in pozitia (C, C') daca si numai daca exista o sageata de la C la C'. Asadar trebuie sa aratam ca exista o masina determinista logspace care poate calcula la cerere orice bit din matricea de adiacenta. Intr-adevar, o masina determinista poate verifica in spatiu $O(|C| + |C'|) = O(\log |x|)$ daca C' este una din cele doua configuratii care ii poate urma lui C aplicand functia de tranzitie δ are masinii nedeterministe. □

Asadar L = NL daca si numai daca *PATH* este in L. Acest lucru este pana acum necunoscut. Surprinzator, s-a demonstrat de curand ca restrictia lui *PATH* la grafuri neorientate este in L. De asemeni se stie ca NL = coNL.

Concluzie: Stim ca:

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP}.$$

- Din Time Hierarchy Theorem stim ca P $\subsetneq$ EXP.
- Din Space Hierachy Theorem se stie ca L $\subsetneq$ PSPACE. Acest lucru se poate imbunatati in modul urmator:
- Din varianta nedeterminista a lui Space Hierarchy Theorem se stie ca NL $\subseteq$ NPSPACE. Combinat cu faptul ca NPSPACE = PSPACE, tragem concluzia ca NL $\subsetneq$ PSPACE.
- De asemenea stim ca daca EXP $\neq$ NEXP atunci P $\neq$ NP.
- Despre celelalte incluziuni nu se stie daca sunt stricte sau nu. □