ALGORITMI - CALCULABILITATE

Horia Georgescu

În cadrul acestui articol, ultimul de altfel din această serie, vom aborda problema decidabilității funcțiilor folosind alte metode de calculabilitate și vom prezenta o serie de probleme nedecidabile fără însă a insista asupra demonstrațiilor acestora.

Alte modele de calculabilitate

Pentru a scurta drumul către rezultatele importante ale *teoriei calculabilității*, vom omite în continuare demonstrațiile. Cei interesați într-o expunere completă se pot adresa autorului sau pot consulta bibliografia.

Calcule pe cuvinte

Fie $A = \{s_1, s_2, .., s_n\}$ un alfabet cu n litere. Introducem următoarea funcție $\phi_n: A^* \to N$ care stabilește pentru fiecare cuvânt $w \in A$ codificarea sa $\phi_n(w)$ ca număr natural, astfel:

$$\phi_n(w) = \begin{cases} 0, & w = \lambda \\ i_k \cdot n^k + \ldots + i_1 \cdot n + i_0, & w = s_{i_k} s_{i_{k-1}} \ldots s_{i_1} s_{i_0} \end{cases}$$

Exemplul 23

Fie $A = \{a, b, c\}$ cu literele precizate în această ordine. Atunci vom avea:

$\phi_3(baacb) = 2 \cdot 3^4 + 1 \cdot 3^3 + 1 \cdot 3^2 + 3 \cdot 3 + 2 = 209$.

Observăm că este vorba de ceva asemănător cu scrierea în baza n, cu deosebirea că în loc de cifrele 0, 1, ..., n - 1 se folosesc "cifrele" 1, 2, ..., n.

Motivațiile sunt următoarele:

- se evită lipsa unicității reprezentării uzuale în baza n; astfel 23 = 023 = 0023 = ... (în noua scriere nu mai apare cifra 0);
- se poate lucra și în baza 1. Astfel, dacă $A = \{a\}$, atunci: $\phi_1(a^n) = 1 \cdot 1 + ... + 1 \cdot 1 + 1 = n$.

Se impun următoarele observații:

- funcția ϕ_n depinde în mod esențial de n. Astfel, pentru exemplul de mai sus, dacă considerăm $w = baacb \in \{a, b, c, d\}$ cuvântul atașat va fi: $\phi_4(w) = 2 \cdot 4^4 + 1 \cdot 4^3 + 1 \cdot 4^2 + 3 \cdot 4 + 2 \neq \phi_3(w)$;
- funcția ϕ_n depinde și de ordinea în care apar literele în A. Nu am precizat explicit acest lucru pentru a nu încărca notația;
- aplicația ϕ_n este bijectivă.

Exemplul 24

Fie $x = 209$. Căutăm $w \in \{a, b, c\}$ cu $w = \varphi_3^{-1}(x)$.

$209 = 3 \cdot 69 + \mathbf{2}$
$69 = 3 \cdot 22 + \mathbf{3}$!!!
$22 = 3 \cdot 7 + \mathbf{1}$
$7 = 3 \cdot 2 + \mathbf{1}$
$2 = 3 \cdot 0 + \mathbf{2}$

Deci $209 \to (2, 1, 1, 3, 2) \to baacb$.

Prezentăm în continuare, fără demonstrație, următoarea propoziție:

Propoziția 8

Trecerea de la orice $x \in N$ la $w = \varphi_n^{-1}(x)$ se face în mod calculabil.

Fie $f: N^m \to N$. Atunci pentru orice alfabet A cu n litere într-o ordine dată, putem considera funcția $f_n = \phi_n^{-1} \circ f \circ \phi_n^m$, adică $f_n(w_1, ..., w_m) = \phi_n^{-1}(f(\phi_n(w_1), ..., \phi_n(w_m)))$.

Fie A un alfabet cu n litere într-o ordine dată și fie $f_n: (A^*)^m \to A^*$. Atunci este bine determinată funcția $f = \phi_n \circ f_n \circ (\phi_n^{-1})^m$ dată de $f(w_1, ..., w_m) = \phi_n(f_n(\phi_n^{-1}(w_1), ..., \phi_n^{-1}(w_m)))$.

Se observă în cele de mai sus că, pentru un alfabet dat cu n litere precizate într-o anumită ordine, funcțiile f_n și f sunt bine determinate una de către cealaltă, aceste determinări fiind realizate prin funcții calculabile.

Rezultă că o funcție $f: (A^*)^m \to A^*$ poate fi privită ca o funcție $f: N^m \to N$. Aceasta ne permite să vorbim de *funcții* (*parțial*) *calculabile pe cuvinte*. Mai precis, $f_n: (A^*)^m \to A^*$ se numește (*parțial*) *calculabilă* dacă f este (parțial) calculabilă.

Exemplul 25

Funcția $|\ |_n: N \to N$, corespunzătoare funcției: $A^* \to N$, care calculează lungimea unui cuvânt, este calculabilă:

$$|u|_n = \min_{k \le u}\left[\sum_{j=0}^{k} n^j > u\right].$$

Limbajul S_n

Limbajul S nu este adecvat pentru lucrul cu cuvinte. Astfel, ștergerea ultimei litere a unui cuvânt se face anevoios, iar instrucțiunea $v \leftarrow v + 1$ (esențială pentru lucrul cu numere) este lipsită de semnificație în lucrul pe cuvinte (de exemplu, pentru $A = \{a, b, c\}$, cuvântul *bacc* este transformat în cuvântul *bbaa*).

De aceea, pentru fiecare $n > 0$, introducem limbajul S_n, care lucrează cu variabile care sunt cuvinte într-un alfabet A cu $|A| = n$. Instrucțiunile acestui limbaj sunt următoarele:

- $v \leftarrow sv$ – se adaugă la stânga cuvântului v litera s;
- $v \leftarrow v^-$ – se șterge ultima literă a lui v; $\lambda^- = \lambda$;
- **if** v **ends** s **goto** L – dacă v se termină cu litera s, se face transfer la prima instrucțiune din program cu eticheta L; în caz contrar se trece la instrucțiuniea următoare.

Evident în instrucțiunile de mai sus trebuie să avem $v \in A^*$, $s \in A$. Prin analogie cu limbajul S, variabilele diferite de variabilele de intrare se consideră inițializate cu λ (corespunzător lui 0), se folosesc macroinstrucțiuni etc. Diferența față de S constă numai în aceea că variabilele sunt gândite ca având valori în A^*, iar forma instrucțiunilor diferă, fiind acum adecvată lucrului cu cuvinte.

Spunem că o funcție $f: (A^*)^m \to A^*$ este (*parțial*) *calculabilă* în S_n dacă este calculată de un program în limbajul S_n.

Deși instrucțiunile din S_n se referă la cuvinte, putem să le gândim ca referindu-se la numerele "în baza n" pe care le reprezintă acele cuvinte; de exemplu, efectul numeric al instrucțiunii $v \leftarrow s_i v$ este $v \leftarrow i \cdot n^{|v|_n} + v$.

Instrucțiunile din S_n sunt naturale pentru cuvinte și nenaturale pentru numere, așa cum instrucțiunile din S sunt naturale pentru numere, dar nu și pentru cuvinte.

Prezentăm câteva macroinstrucțiuni în limbajul S_n, împreună cu dezvoltările corespunzătoare (se consideră $A = \{s_1, ..., s_n\}$):

- **if** $v \neq \lambda$ **goto** L — **if** v **ends** s_i **goto** L $(1 \le i \le n)$
- $v \leftarrow \lambda$ — $[A]$ $v \leftarrow v^-$; **if** $v \neq \lambda$ **goto** A
- **goto** L — $z \leftarrow s_1 z$; **if** $z \neq \lambda$ **goto** L
- $v' \leftarrow v$ — $z \leftarrow \lambda$; $v' \leftarrow \lambda$; $[A]$ **if** v **ends** s_i **goto** B_i $(1 \le i \le n)$; **goto** C; $[B_i]$ $(1 \le i \le n)$: $v \leftarrow v^-$; $v' \leftarrow s_i v'$; $z' \leftarrow s_i z'$; **goto** A; $[C]$ **if** v **ends** s_i **goto** D_i $(1 \le i \le n)$; **goto** E; $[D_i]$ $(1 \le i \le n)$: $z \leftarrow z^-$; $v \leftarrow s_i v$; **goto** C

(se copiază mai întâi v în v' și z, iar apoi se copiază z în v).

Prezentăm în continuare, fără demonstrație, următoarea teoremă:

Teorema 2

Fie $f: N^m \to N$ (parțial) calculabilă $\Leftrightarrow \forall n$ și $A = \{s_1, ..., s_n\}$, funcția $f_n : (A^*)^m \to A^*$ este (parțial) calculabilă în S_n.

Programe Post-Turing

Fie $A = \{s_1, ..., s_n\}$ un alfabet format din n litere. Pe baza lui, vom introduce un nou limbaj: *limbajul Post-Turing* T_n.

Se folosește o bandă infinită, atât la dreapta cât și la stânga, formată din celule care pot conține fie o literă din A, fie un blanc (notat prin b sau s_0). Mai există un cap de citire / scriere care poate explora, la fiecare moment de timp, o singură celulă. Întotdeauna banda va conține numai un număr finit de celule în care apar litere din A.

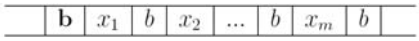
Instrucțiunile limbajului *Post-Turing* T_n sunt următoarele:

- **print** s – în celula din dreptul capului de citire / scriere se scrie litera s;
- **if** s **goto** L – dacă celula din dreptul capului de citire / scriere conține litera s, atunci se face transfer la prima instrucțiune din program cu eticheta L; în caz contrar, se trece la instrucțiunea următoare;
- **right** – capul de citire / scriere este deplasat cu o poziție la dreapta;
- **left** – capul de citire / scriere este deplasat cu o poziție la stânga.

Prin *configurația* benzii de intrare la un moment de timp oarecare înțelegem conținutul ei, împreună cu poziția capului de citire / scriere; dacă va apărea numai o porțiune din bandă, se va considera că în celulele din dreapta și stânga sa apar numai celule goale (blancuri). Pentru a indica po-

ziția capului de citire/scriere, vom figura îngroșat (bold) simbolul din celula în dreptul căreia se află.

Configurația inițială pentru un program în limbajul T_n care prelucrează datele de intrare $x_1, ..., x_m \in A^*$ este următoarea:



unde, așa cum am convenit, poziția capului de citire / scriere este în stânga celulei care conține valoarea x_1.

Dacă, de exemplu, $x_2 = \lambda$, atunci după x_1 vor apărea două blancuri.

Să observăm că nu putem face distincție între această configurație și cea cu $m + 1$ intrări cu $x_{m+1} = \lambda$.

Exemplul 26

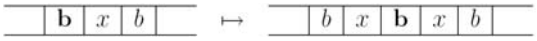
Fie $A = \{s_1, s_2, s_3\}$ și $x \in A^*$. Următoarea schimbare de configurație:

este realizată, de exemplu, de programul următor, scris în limbajul T_3:

```
[A] right
    if s_i goto A (1 ≤ i ≤ 3)
    print s_1
    right
    print s_1
[B] left
    if s_i goto B (1 ≤ i ≤ 3)
```

Exemplul 27

Fie $A = \{s_1\}$ și $x \in A^*$. Următorul program realizează schimbarea de configurație:



Vom folosi, în afară de s_1 și $b = s_0$, un simbol auxiliar $m \notin A$ (numit *marcaj*):

```
[A] right
    if b goto E
    print m
[B] right
    if s_1 goto B
[C] right
    if s_1 goto C
    print s_1
[D] left
    if s_1 goto D
    if b goto D
    print s_1
    if s_1 goto A
```

Acest program necesită următoarele explicații:
- primele trei instrucțiuni înlocuiesc primul s_1 cu m;
- următoarele patru instrucțiuni determină deplasarea la dreapta peste toți s_1, apoi peste b, apoi peste toți s_1;
- cele patru instrucțiuni care încep cu cea etichetată cu D determină deplasarea la stânga peste toți s_1 și b, până la m;
- penultima instrucțiune înlocuiește pe m cu s_1;
- ultima instrucțiune face transfer la prima, dar acum capul de citire/scriere este cu o poziție mai la dreapta.

Pentru $x = s_1s_1$, se trece succesiv prin următoarele configurații:

$$\mathbf{b}s_1s_1b \overset{1}{\mapsto} b\mathbf{m}s_1b \overset{2}{\mapsto} bms_1b\mathbf{b} \overset{3}{\mapsto} bms_1b\mathbf{s_1}b \overset{4}{\mapsto} b\mathbf{m}s_1bs_1b \overset{5}{\mapsto}$$

$$b\mathbf{s_1}s_1bs_1b \overset{1}{\mapsto} bs_1\mathbf{m}bs_1b \overset{2}{\mapsto} bs_1mbs_1\mathbf{b} \overset{3}{\mapsto} bs_1mbs_1\mathbf{s_1}b \overset{4}{\mapsto}$$

$$bs_1\mathbf{m}bs_1s_1b \overset{5}{\mapsto} bs_1\mathbf{s_1}bs_1s_1b \overset{1}{\mapsto} bs_1s_1\mathbf{b}s_1s_1b$$

Exemplul de mai sus arată necesitatea (sau cel puțin utilitatea) folosirii unor simboluri auxiliare (marcaje), pe lângă simbolurile din $A \cup \{s_0\}$.

Fie funcția parțială f_n: $(A^*)^m \to A^*$.

Spunem că un program P în limbajul T_n calculează funcția f_n dacă:
- plecându-se din configurația inițială cu valorile de intrare $x_1, ..., x_m$, programul se oprește $\Leftrightarrow f_n(x_1, ..., x_m)\downarrow$;
- la oprire, prin ștergerea de pe bandă a simbolurilor care nu sunt în A (adică $s_0 = b$ sau marcaje), rămâne $f_n(x_1, ..., x_m)$.

Prezentăm în continuare câteva macroinstrucțiuni ale limbajului T_n, împreună cu dezvoltările lor.

- **goto** L — **if** s_i **goto** L $(0 \le i \le n)$

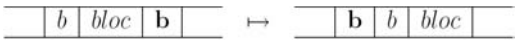
- **right to next blank**

```
[A] right
    if b goto E
    goto A
```

- **left to next blank**

```
[A] left
    if b goto E
    goto A
```

- **move block right**, care realizează schimbarea de configurație:



unde blocul nu conține b:

```
[C] left
    if s_i goto A_i (0 ≤ i ≤ n)
[A_i] (1 ≤ i ≤ n)
    right
    print s_i
    left
    goto C

[A_0] right
    print b
    left
```

- **erase a block**

```
[A] right
    if b goto E
    print b
    goto A
```

(se merge la dreapta, ștergând (înlocuind cu b) toate simbolurile până la primul b).

Prezentăm în continuare, fără demonstrație, următoarea teoremă:

Teorema 3
Pentru orice funcție parțială f_n: $(A^*)^m \to A^*$, f_n este calculată de un program în T_n dacă și numai dacă este parțial calculabilă în S_n.

Mașini Turing

La fel ca în cazul programelor *Post-Turing*, vom folosi o bandă infinită atât la dreapta, cât și la stânga, formată din celule care conțin simboluri dintr-un alfabet T, precum și simbolul b. La fiecare moment de timp pe bandă se vor afla doar un număr finit de simboluri din T. Alfabetul T conține o submulțime A. Ne va interesa în continuare cum, din cuvintele $x_1, ..., x_m \in A^*$, se poate obține un cuvânt $y \in A^*$; simbolurile din $T \setminus A$ se numesc tot *marcaje*. Capul de citire / scriere poate explora de asemenea o singură celulă. Deosebirea va consta în faptul că trecerea de la o configurație la alta nu se va mai realiza pe baza unor instrucțiuni, ci pe baza unor așa numite *stări* ale mașinii *Turing*.

O *mașină Turing* este un quadruplu $M = (K, T, q_1, Q)$ unde:

- K este alfabetul stărilor;
- T este alfabetul mașinii *Turing*;
- $q_1 \in K$ este starea inițială;
- Q este o mulțime finită de quadruple pe baza cărora se trece de la o configurație la alta.

O configurație are forma: $\overset{\alpha s \beta}{q}$, unde $\alpha, \beta \in (T \cup \{b\}^*)$, $s \in T \cup \{b\}$ este simbolul din dreptul capului de citire / scriere, iar q este starea curentă.

Prezentăm în continuare forma quadruplelor din mulțimea Q, împreună cu schimbările de configurație corespunzătoare:

- (q_i, s_j, s_k, q_l) $\quad \overset{\alpha \mathbf{s_j} \beta}{q_i} \mapsto \overset{\alpha \mathbf{s_k} \beta}{q_l}$
- (q_i, s_j, R, q_l) $\quad \overset{\alpha \mathbf{s_j} s_k \beta}{q_i} \mapsto \overset{\alpha s_j \mathbf{s_k} \beta}{q_l}$
- (q_i, s_j, L, q_l) $\quad \overset{\alpha s_k \mathbf{s_j} \beta}{q_i} \mapsto \overset{\alpha \mathbf{s_k} s_j \beta}{q_l}$

unde q_i este starea din care se pleacă, s_j este simbolul explorat de capul de citire / scriere, iar q_l este starea în care se ajunge; cele trei forme de quadruple determină înlocuirea simbolului s_j explorat prin s_k, deplasarea la dreapta și deplasarea la stânga a capului de citire / scriere. Vom nota în continuare $s_0 = b$.

Vom considera numai *mașini Turing deterministe*, adică în care nu există două quadruple care încep cu aceeași pereche (q_i, s_j).

Configurația inițială a benzii este următoarea:

$\overset{\mathbf{b} x_1 b x_2 b \ldots b x_m}{q_1}$, unde $x_1, ..., x_m \in A^*$.

Schimbările de configurație se fac conform quadruplelor din K. Mașina *Turing se oprește* dacă se ajunge într-o configurație de forma $\overset{\alpha \mathbf{s_j} \beta}{q}$ și nici un quadruplu din Q nu începe cu (q_i, s_j).

Spunem că M *calculează* funcția f parțială de aritate m pe A^*, dacă plecând de la configurația inițială, M se oprește pentru $(x_1, ..., x_m) \in (A^*)^m$ dacă și numai dacă $f(x_1, ..., x_m)$ este definită; în plus, la oprire pe bandă se obține, ștergându-se toate simbolurile care nu sunt în A, tocmai $f(x_1, ..., x_m)$.

Exemplul 28
Fie $M = ((q_1, q_2, q_3), \{1\}, q_1, Q)$ unde: $Q = \{(q_1, s_0, R, q_2), (q_2, 1, R, q_2), (q_2, s_0, 1, q_3), (q_3, 1, R, q_3), (q_3, q_0, 1, q_1)\}$.

Cum $s_0 = b$, avem de exemplu:

$$\overset{\mathbf{b}111}{q_1} \mapsto \overset{1\mathbf{1}1}{q_2} \mapsto \overset{11\mathbf{1}}{q_2} \mapsto \overset{111\mathbf{b}}{q_2} \mapsto \overset{1111}{q_3} \mapsto \overset{1111\mathbf{b}}{q_3} \mapsto \overset{11111}{q_1}$$

(se observă că putem spune că această mașină *Turing* calculează funcția dată de $f(x) = x + 2$).

O altă reprezentare pentru o mașină *Turing* este cea sub formă de tabel, pe care ne mărginim a o prezenta pentru exemplul considerat.

	q_1	q_2	q_3
b	R, q_2	$1, q_3$	$1, q_1$
1		R, q_2	R, q_3

Prezentăm, fără demonstrație, următoarea teoremă:

Teorema 4
Orice funcție parțială este calculată de un program *Post-Turing* dacă și numai dacă este calculată și de o mașină *Turing* cu același alfabet.

Concluzie
Am introdus în acest paragraf mai multe modele de calculabilitate (de definire a noțiunii de algoritm) care au folosit fie limbaje de programare (pe numere naturale sau pe cuvinte), fie mașini matematice. Teoremele enunțate au scos în evidență echivalența lor, ceea ce confirmă *teza lui Church*.

Alte probleme nedecidabile

Includem în această secțiune alte probleme nedecidabile, neînsoțite însă de demonstrațiile corespunzătoare.

Teorema 5
Nu există un algoritm care să determine pentru orice program P din limbajul S dacă $\psi_P^{(1)}$ este sau nu total calculabilă.

Teorema spune că problema "*să se determine pentru un program oarecare dacă el se termină pentru orice valoare de intrare*" este nedecidabilă.

Observație
Rezultatul este diferit de cel din *Teorema* 1 (predicatul *HALT* nu este calculabil) deoarece aici problema are ca date de intrare programe, pe când în *Teorema* 1 problema avea ca date de intrare programe însoțite de valori de intrare.

Teorema 6
Fie f o funcție (parțială) calculabilă de o variabilă. Atunci nu există un algoritm pentru următoarea problemă: *să se determine pentru orice program P dacă* $\psi_P^{(1)} = f$.

Facem următoarea observație legată de această teoremă: să considerăm o funcție calculabilă simplă, de exemplu, cea dată de $f(x) = x + 1$. Presupunem că cerem mai multor persoane să scrie programe în limbajul S care să calculeze această funcție. Teorema de mai sus ne spune că *nu există un algoritm care să verifice că aceste programe produc rezultatul dorit. Deci, cel puțin din punct de vedere al corectării programelor, profesorul nu poate fi înlocuit de calculator.*

Teorema 7
Nu există un algoritm pentru următoarea problemă: fiind date $u, v \in N$, să se determine dacă $\Phi_u^{(1)} = \Phi_v^{(1)}$.

Teorema de mai sus arată *că problema echivalenței a două programe este nedecidabilă*.

Încheiem lista problemelor nedecidabile cu *problema de corespondență a lui Post*, formulată în anul 1946, problemă cu aplicații în teoria limbajelor formale.

Un sistem Post peste un alfabet A constă dintr-un număr finit k de tipuri de dominouri de forma:

$$\begin{array}{|c|}\hline \uparrow \\ \hline u_i \\ \hline v_i \\ \hline \end{array},$$

cu $u_i, v_i \in A^*$, $i = 1, ..., k$ (săgeata de pe dominiouri arată că nu le putem răsturna; nu o vom mai figura în continuare).

Există o infinitate de dominouri de fiecare tip.

Un sistem Post are soluție dacă există o înșiruire de dominouri, astfel încât concatenarea cuvintelor aflate în partea superioară și concatenarea cuvintelor aflate în partea inferioară să producă același cuvânt.

Exemplul 29
Fie $A = \{a, b\}$. Atunci sistemul *Post* următor:

$$\left\{ \begin{array}{|c|}\hline a \\ \hline aa \\ \hline \end{array}, \begin{array}{|c|}\hline bb \\ \hline b \\ \hline \end{array}, \begin{array}{|c|}\hline a \\ \hline bb \\ \hline \end{array} \right\},$$

are soluția a^2b^4 dată de:

$$\begin{array}{|c|c|c|c|}\hline a & a & bb & bb \\ \hline aa & bb & b & b \\ \hline \end{array}.$$

Problema de corespondență a lui *Post* are următorul enunț: există un algoritm care să determine pentru orice sistem Post dacă el are sau nu soluții?

Teorema de mai jos arată că *problema de corespondență a lui Post nu este decidabilă*.

Teorema 8
Fie A un alfabet cu $|A| \geq 2$. Atunci nu există un algoritm care pentru orice sistem *Post P* peste A să determine dacă P are soluție.

Scurt istoric al teoriei calculabilității

Noțiunea de algoritm în accepțiunea uzuală, cunoscută de sute de ani, a fost și este suficientă pentru a elabora algoritmi pentru diferite probleme, inclusiv de a-i implementa în diferite limbaje de programare.

Totuși, în anii 1930, dezvoltarea logicii matematice, precum și perspectiva imediată a apariției calculatoarelor, a făcut ca matematicienii să încerce să găsească o definiție riguroasă a conceptului de algoritm. Necesitatea acestui lucru a apărut deci prin prisma evoluției matematicii, dar și pentru a răspunde la o întrebare naturală: "e foarte bine că vor fi construite calculatoare, dar care sunt limitele prelucrărilor pe care le pot realiza?".

Charles Babbage, plecând de la o idee a lui *Leibniz*, a încercat să construiască un raționament matematic care să folosească în mod mecanic simbolurile algebrice. Această încercare a fost concretizată în mașina sa în greutate de 3 tone, bazată pe metoda diferențelor finite.

Punctul de plecare al cercetărilor ce au condus la definirea conceptului de algoritm este considerat programul matematic inițiat de ***David Hilbert***. *Hilbert* a propus o metodă axiomatică de lucru cu numerele naturale, în care acestea sunt gândite ca obiecte între care există relați descrise printr-un sistem necontradictoriu de axiome. Axiomele și teoremele sunt reprezentate prin secvențe finite de simboluri, iar demonstrațiile trebuiesc făcute conform unor "reguli mecanice". În 1922 a propus celebra *Entscheidungsproblem* (problema deciziei) care constă în a găsi o modalitate de a testa într-un număr finit de pași dacă o expresie formală poate fi dedusă dintr-un sistem dat de axiome.

Zece ani mai târziu, s-a arătat însă că problema deciziei nu poate avea un răspuns afirmativ; era necesar să se precizeze ce se înțelege prin funcție calculabilă și prin procedură de decizie.

În anul 1931, ***Kurt Gödel*** a dovedit incompletitudinea aritmeticii: "există propoziții din aritmetică pentru care nu se poate demonstra dacă sunt adevărate sau false". Pentru aceasta, el a introdus noțiunile de funcție recursivă și funcție calculabilă.

Denumirile au variat de-a lungul anilor, de aceea prezentăm în continuare accepțiunile actuale.

Vom numi funcții inițiale următoarele trei funcții, evident calculabile:
- *funcția succesor s*: $N \to N$ dată de $s(x) = x + 1$;
- *funcția nulă n*: $N \to N$ dată de $n(x) = 0$;
- *funcțiile proiecții* u_i^n: $N^n \to N$ (i = 1, 2, ..., n) date de $u_i^n(x_1, x_2, ..., x_n) = x_i$.

Definiție
O funcție se numește *primitiv recursivă* dacă se obține din funcțiile inițiale aplicând de un număr finit de ori operația de compunere și recursia (1).

Evident, orice funcție primitiv recursivă este calculabilă.

Teorema 9
Există funcții calculabile, care nu sunt primitiv recursive.

Lucrările lui *Gödel* au stat la baza cercetărilor întreprinse de ***Alonzo Church***, ***Stephen Kleene***, ***Alan Turing*** și ***Emil Post***. Fiecare dintre ei a definit un concept de calculabilitate, adică un concept de algoritm. Pe baza acestui concept au abordat problema de decizie a lui *Hilbert*, demonstrând că există probleme nedecidabile. Unele dintre aceste abordări au fost prezentate în acest articol, într-o formă posibilă după decenii de simplificări și rafinări.

Echivalența diferitelor concepte de algoritm introduse a fost esențială în acceptarea *tezei lui Church*, care ține de domeniul *metamatematicii*.

Note biografice

Charles Babbage (*Teignmouth* 1792 - *Londra* 1871)
Absolvent al Universității Cambridge. A avut preocupări de calcul diferențial și integral, dar în special de logică matematică. A conceput două mașini de calcul, una bazată pe diferențe finite, iar cealaltă analitică, aceasta din urmă fiind unanim considerată ca precursoarea calculatoarelor moderne. Membru al *Societății Regale*.

Augusta Ada King, contesă de Lovelace (1815 - 1852)
Fiica poetului *Lord George Gordon Byron*, colaboratoare a lui *Charles Babbage*. Este considerată prima femeie informatician din lume. A dat numele limbajului de programare *Ada*, care a introdus concepte noi în teoria programării.

David Hilbert (*Königsberg* 1862 - *Götingen* 1943)
Considerat de mulți ca cel mai mare matematician al epocii moderne, a avut contribuții importante în domeniile teoriei algebrice a numerelor, fundamentelor geometriei, analizei, fizicii teoretice și fundamentelor matematicii. Cele 23 de probleme propuse la *Congresul matematicienilor de la Paris* (1900), dintre care ultima a primit răspuns doar în 1970, continuă să stimuleze studiile teoretice.

Kurt Gödel (*Brno* 1906 - *Princeton* 1978)
Lucrările sale cuprind arii vaste: fizică, logică și filozofia matematicii. Celebru în special pentru teorema de incompletitudine (1933) care îi poartă numele. Membru al *Cercului de Filozofie* de la *Viena*, fuge la *Princeton* după instalarea regimului nazist.

Alonzo Church (*Washington* 1903 - *Götingen* 1995)
A urmat studiile la *Princeton*, apoi a profesat la *UCLA*. Cunoscut ca matematician, logician și filozof. Cele mai importante rezultate obținute sunt teorema care îi poartă numele (care afirmă că nu există o procedură de decizie pentru aritmetică) și *teza lui Church*, prezentată în acest articol.

Stephen Cole Kleene (*Hartford* 1909 - *Madison* 1994)
Matematician ilustru, cu contribuții majore în domeniile logicii matematice, calculabilității și limbajelor formale. Contribuțiile sale în domeniul funcțiilor recursive au avut consecințe notabile, dintre care menționăm demonstrația lui *Matijasevics* (1970) a nedecidabilității celebrei probleme a zecea a lui *Hilbert* privind existența unei soluții întregi pentru polinoamele cu coeficienți întregi.

Emil Leon Post (*Augustow* 1897 - *New York* 1954)
Își obține doctoratul la *Universitatea din Columbia* și apoi predă la *Universitatea Cornell* și la alte universități din *New York*. Membru în *American Mathematical Society* și fondator al *Association for Symbolic Logic*. Este considerat ca precursor al teoriei moderne a demonstrației.

Alan Turing (*Londra* 1912 - *Wilmslow* 1954)
A absolvit *King's College*, *Cambridge*, apoi s-a mutat la *Princeton* pentru a lucra cu *Church*. Aici a publicat celebrul său articol privind problema de decizie a lui *Hilbert* în care a introdus modelul de mașină *Turing*, model de referință și astăzi. Întors în *Anglia*, a lucrat ca specialist în descifrarea codurilor la *Foreign Oce* în timpul celui de al doilea război mondial. După război a participat la producerea unei mașini automate de calcul.

A formulat întrebarea "Presupunând că un calculator poate să gândească, cum putem să ne dăm seama (If a computer could think, how would we tell)?". Întrebarea s-a concretizat în *testul lui Turing*: "se pun întrebări unei persoane și unui calculator, la care trebuie să răspundă printr-un text. Poate un arbitru să nu fie capabil să identifice sursa răspunsurilor?". Deocamdată nici un calculator nu a trecut *testul lui Turing* (premiul de 100.000 lire sterline își așteaptă în continuare câștigătorul...). Mai menționăm că *Turing* este considerat părintele inteligenței artificiale.

Bibliografie

1. **Martin D. Davis, Elaine J. Weyuker**, *Computability, Complexity and Languages,* Academic Press, 1983
2. **Christos H. Papadimitriou**, *Computational Complexity*, Addison-Wesley, 1994
3. **Jean Luc Chambert,** *A History of Algorithms,* Springer Verlag, 1999
4. **http://turnbull.mcs.st-and.ac.uk/history/Mathematicians**

Dl. prof. dr. Horia Georgescu este cadru didactic la Universitatea București, fondator și director științific al GInfo și poate fi contactat prin e-mail la adresa **g_horia@hotmail.com**.