ALGORITMI - CALCULABILITATE

Horia Georgescu

În cadrul acestui articol vom aborda problema existenței algoritmilor de rezolvare pentru diferite probleme și vom insista asupra calculabilității funcțiilor.

În cei 12 ani de existență, în *Gazeta de Informatică* (apoi ***GInfo***) au apărut nenumărate articole privind algoritmii, sub toate aspectele lor. Dar toate aceste articole au avut un numitor comun și anume au presupus existența algoritmilor pentru problemele studiate.

Există algoritmi pentru orice problemă?

Această problemă a stat în atenția matematicienilor chiar înainte de apariția calculatoarelor și face obiectul *teoriei calculabilității*. La bază stau cercetările din al patrulea deceniu al secolului trecut ale unor personalități precum *Church*, *Gödel*, *Kleene*, *Post* și *Turing*. Ei au propus diferite definiții ale noțiunii de algoritm, bazate fie pe limbaje de programare, fie pe mașini matematice, definiții care s-au dovedit a fi echivalente.

Articolul de față își propune să prezinte cititorilor elementele de bază ale teoriei calculabilității, cu scopul de a dovedi că există probleme pentru care nu pot fi elaborați algoritmi. Prezentarea nu este originală, ci urmează întocmai o parte din minunata carte *Computability, Complexity and Languages, Academic Press* 1983, a lui ***Martin Davis*** și ***Elaine Weyuker***.

Ne exprimăm speranța că cititorii ***GInfo*** vor fi interesați de acest subiect, care din păcate lipsește chiar din programa multor facultăți cu profilul informatică.

Limbajul de programare S

O primă definiție a noțiunii de algoritm se bazează pe limbajul *S*. Într-o primă etapă se va face prezentarea acestui limbaj foarte simplu, urmând ca apoi să căpătăm încredere în posibilitățile sale.

Un program *P* scris în limbajul *S* este o secvență de instrucțiuni (un număr finit de instrucțiuni scrise într-o anumită ordine), program în care intervin:
- numere naturale;
- variabile de intrare, notate de obicei prin x_1, x_2, ...;
- variabila de ieșire y;
- variabile intermediare (de lucru), notate de obicei prin z_1, z_2, ...

cu mențiunea că valorile luate de variabile pot fi orice numere naturale.

Deoarece lucrăm numai cu numere naturale, vom spune, uneori, pur și simplu număr în loc de număr natural.

Înainte de a prezenta instrucțiunile limbajului *S*, precizăm că ele pot fi precedate, opțional, de etichete. Acestea sunt reprezentate prin litere mari (eventual indexate cu numere naturale), cuprinse între paranteze drepte și plasate la începutul instrucțiunii. Se admite ca mai multe instrucțiuni să aibă aceeași etichetă; rațiunea acestui fapt va fi prezentată în alt articol.

Instrucțiunile admise de limbajul *S* și semnificația lor sunt următoarele:
- $v \leftarrow v + 1$ — valoarea curentă a variabilei v crește cu o unitate;
- $v \leftarrow v - 1$ — valoarea curentă a variabilei v scade cu o unitate (dacă era strict pozitivă), respectiv rămâne `0` (dacă era `0`);
- **`if`** $v \neq$ `0` **`goto`** L — dacă valoarea curentă a variabilei v este nenulă, atunci se face transfer necondiționat la *prima* instrucțiune din program etichetată cu L; dacă nici o in-

strucțiune nu este etichetată cu L, atunci programul se termină; dacă valoarea curentă a lui v este 0, atunci se trece la instrucțiunea următoare;
- $v \leftarrow v$ — este instrucțiunea cu efect nul, a cărei utilitate va apărea ulterior.

Exemplul 1
Programul:

```
[A]  x ← x - 1
     y ← y + 1
     if x ≠ 0 goto A
```

se termină pentru orice valoare inițială α a lui x. La ieșire vom avea:

$$y = \begin{cases} 1, \alpha = 0 \\ \alpha, \alpha > 0 \end{cases}$$

Exemplul 2
Programul:

```
[A]  x ← x + 1
     if x ≠ 0 goto A
```

nu se termină niciodată, indiferent de valoarea inițială a lui x.

Observație
Faptul că programele în limbajul S produc o singură valoare de ieșire nu este o restricție. Într-adevăr, dacă dorim să obținem mai multe valori de ieșire, vom scrie câte un program pentru fiecare și vom executa succesiv aceste programe. Menționăm că, în acest studiu, suntem interesați *numai de existența algoritmilor, nu și de eficiența lor*.

Fie P un program scris în limbajul S. Fie V mulțimea tuturor variabilelor (de intrare, de lucru și de ieșire) care apar în P. Vom da în continuare o definiție riguroasă a *semanticii* programului P.

Definim ***starea*** programului la un moment oarecare a executării sale ca fiind o funcție s: $V \to N$, unde N este mulțimea numerelor naturale. Deci, o stare este dată de valorile curente ale variabilelor din program.

O *configurație* a programului P de lungime n (având n instrucțiuni) este o pereche (i, s) cu $i \in \{1, 2, ..., n, n + 1\}$ și s stare.

O *configurație inițială* are forma $(1, s_0)$, unde:
- $s_0(u) = r_i$, dacă $u = x_i$ și r_i este valoarea inițială a lui x_i și
- $s_0(u) = 0$, dacă u nu este o variabilă de intrare.

O *configurație terminală* are forma $(n + 1, s)$. Vom asimila transferul la o etichetă inexistentă cu transferul la instrucțiunea cu numărul $n + 1$.

Fie (i, s) o configurație neterminală. Succesoarea ei (j, t) este definită astfel:

1) Dacă instrucțiunea i este $v \leftarrow v + 1$, atunci $j = i + 1$, iar:
   - $t(u) = u$, dacă $u \neq v$ și
   - $t(u) = u + 1$, dacă $u = v$.
2) Dacă instrucțiunea i este $v \leftarrow v - 1$, atunci $j = i + 1$, iar:
   - $t(u) = s(v) - 1$, dacă $u = v$ și $s(v) > 0$ și
   - $t(u) = s(v)$, în caz contrar.
3) Dacă instrucțiunea i este $v \leftarrow v$, atunci $j = i + 1$, și $t = s$.
4) Dacă instrucțiunea i este **if** $v \neq 0$ **goto** L , atunci $t = s$, iar:
   - $j = i + 1$, $s(v) = 0$;
   - $j = n + 1$, $s(v) > 0$ și nu există instrucțiuni etichetate cu L, și
   - $j = k$, dacă $s(v) > 0$ și instrucțiunea k este prima instrucțiune cu eticheta L.

Se observă că succesoarea oricărei configurații neterminale este unic determinată și nu depinde de eventuala etichetare a sa.

Un *calcul* al programului P este o *secvență de configurații consecutive* $c_0, c_1, \ldots, c_k$ cu c_k configurație finală (terminală).

În continuare vom lucra cu funcții parțiale (parțial definite). Mai precis, pentru f: $N^m \to N$ vom nota prin *Dom f* elementele din N^m pentru care funcția f este definită.

Fie P un program în care variabilele de intrare sunt $x_1, x_2, \ldots, x_m$. Atunci *funcția (parțială) calculată de programul* P este $\psi_P^{(m)} : N^m \to N$ definită astfel:

1) $\psi_P^{(m)}(x_1, x_2, \ldots, x_m) = y'$ dacă există un calcul $c_0, c_1, \ldots, c_k$ al lui P cu $c_0 = (1, s_0)$ și $s_k(y) = y'$;
2) $\psi_P^{(m)}(x_1, x_2, \ldots, x_m) = \uparrow$ (nedefinit) dacă există un șir infinit de configurații $c_0, c_1, \ldots$ al lui P cu $c_0 = (1, s_0)$, deci dacă programul P nu se termină.

Observație
Fie P un program în care apar variabilele de intrare $x_1, x_2, \ldots, x_m$. Pentru orice număr natural n, îi putem asocia o funcție $\psi_P^{(n)}$ astfel:
- dacă $n \leq m$, atunci $\psi_P^{(n)}(x_1, x_2, \ldots, x_n) = \psi_P^{(m)}(x_1, x_2, \ldots, x_m)$, adică $x_{n+1}, \ldots\ldots, x_m$ sunt ignorate;
- dacă $n > m$, atunci $\psi_P^{(n)}(x_1, \ldots, x_n) = \psi_P^{(m)}(x_1, \ldots, x_m, 0, \ldots, 0)$.

Fie f: $N^m \to N$. Funcția f se numește ***parțial calculabilă*** dacă există un program P în limbajul S cu $\psi_P^{(m)} = f$. Dacă *Dom* $f = N^m$, atunci f se numește ***calculabilă***.

Exemplul 1 arată că funcția f: $N \to N$ dată de:

$$f(x) = \begin{cases} 1, x = 0 \\ x, x > 0 \end{cases}$$, este *calculabilă*.

Exemplul 2 arată că funcția unară cu domeniul vid este *parțial calculabilă*.

Macroinstrucțiuni
Această secțiune urmărește să căpătăm încredere în limbajul S. Vom arăta că principalele prelucrări uzuale asupra numerelor naturale pot fi realizate prin programe scrise în acest limbaj.

Macroinstrucțiunile sunt abrevieri pentru secvențe de instrucțiuni în limbajul S. Menirea lor este de a scrie programe cât mai inteligibile. Prezența unei macroinstrucțiuni

într-un program trebuie interpretată ca prezența în acel punct al programului a unei *dezvoltări* ale sale.

Chiar dacă în același program apare în diferite locuri o aceeași macroinstrucțiune, aceasta nu înseamnă că va fi inserată aceeași dezvoltare a sa. Mai mult, chiar impunem ca aparițiile unei aceleiași macroinstrucțiuni pe poziții diferite din program să presupună dezvoltări diferite ale macroinstrucțiunii (variabile de lucru și etichete distincte). De asemenea trebuie respectată regula ca valoarea variabilelor care apar în macroinstrucțiune, cu excepția celor care apar în membrul stâng al unei atribuiri, să nu fie modificate ca efect al executării macroinstrucțiunii.

Macroinstrucțiunea **goto** *L* are următoarea dezvoltare posibilă:

```
[A] z ← z + 1
    if z ≠ 0 goto A
```

Macroinstrucțiunea $v \leftarrow 0$ are dezvoltarea:

```
[A] v ← v - 1
    if v ≠ 0 goto A
```

unde eticheta *A* nu mai apare nicăieri în programul care conține această macroinstrucțiune.

3) Macroinstrucțiunea $v \leftarrow k$ are dezvoltarea:

```
v ← 0
v ← v + 1
...
v ← v + 1
```

unde instrucțiunea $v \leftarrow v + 1$ apare de exact *k* ori.

Exemplul 3

Funcția $f: N \to N$ dată de $f(x) = x$ este *calculabilă*.

Pentru demonstrație, vom scrie următorul program *P* cu $\psi_P^{(1)} = f$, program care va asigura păstrarea valorii inițiale a lui *x*:

```
[A] if x ≠ 0 goto B
    goto C
[B] x ← x - 1
    y ← y + 1
    z ← z + 1
    goto A
[C] if z ≠ 0 goto D
    goto E
[D] z ← z - 1
    x ← x + 1
    goto C
```

unde primele 6 instrucțiuni copiază valoarea lui *x* în variabilele *y* și z, iar următoarele recopiază valoarea lui *z* în *x*.

Putem acum introduce macroinstrucțiunea $v \leftarrow u$ cu dezvoltarea:

```
    v ← 0
[A] if u ≠ 0 goto B
    goto C
[B] u ← u - 1
    v ← v + 1
    z ← z + 1
    goto A
[C] if z ≠ 0 goto D
    goto E
[D] z ← z - 1
    u ← u + 1
    goto C
[E] v ← v
```

care "transcrie" funcția din exemplul 3.

Se impun următoarele remarci:

- nu am inițializat pe *z* cu 0, deoarece este clar că după orice executare a macroinstrucțiunii, valoarea sa va fi 0;
- apare necesitatea instrucțiunii $v \leftarrow v$;
- ca de obicei, variabila *z* nu mai apare nicăieri în program (este o variabilă de lucru "nouă"), iar etichetele *A*, *B*, *C*, *D*, *E* nu mai apar nicăieri în program (sunt etichete "noi"). Având convingerea că cititorul s-a obișnuit deja cu aceste reguli, nu le vom mai aminti în continuare, presupunându-le subînțelese.

Exemplul 4

Funcția $f: N \times N \to N$ definită prin $f(x_1, x_2) = x_1 + x_2$ este *calculabilă*.

Un program care calculează această funcție este următorul:

```
    y ← x1
    z ← x2
[B] if z ≠ 0 goto A
    goto E
[A] z ← z - 1
    y ← y + 1
    goto B
```

iar macroinstrucțiunea $z \leftarrow z_1 + z_2$ are următoarea dezvoltare posibilă:

```
    u ← z1
    v ← z2
[B] if v ≠ 0 goto A
    goto E
[A] v ← v - 1
    u ← u + 1
    goto B
[E] z ← u
```

Exemplul 5

Funcția $f: N \times N \to N$ definită prin $f(x_1, x_2) = x_1 \cdot x_2$ este *calculabilă*.

Un program care calculează această funcție este următorul:

```
    z ← x2
[B] if z ≠ 0 goto A
    goto E
[A] z ← z - 1
```

```
    y ← y + x1
    goto B
```

iar macroinstrucțiunea $z \leftarrow z_1 \cdot z_2$ are următoarea dezvoltare posibilă:

```
    u ← 0
    v ← z2
[B] if v ≠ 0 goto A
    goto E
[A] v ← v - 1
    u ← u + z1
    goto B
[E] z ← u
```

Exemplul 6

Fie f: $N \times N \rightarrow N$ funcția parțială definită prin:

$$f(x_1, x_2) = \begin{cases} x_1 - x_2, x_1 \geq x_2 \\ \uparrow, x_1 < x_2 \end{cases}.$$

Un program care calculează această funcție este următorul:

```
    y ← x1
    z ← x2
[C] if z ≠ 0 goto A
    goto E
[A] if y = 0 goto B
    goto A    {y = 0 și z = 0}
[B] y ← y - 1
    z ← z - 1
    goto C
```

Macroinstrucțiunea asociată este $z \leftarrow z_1 - z_2$ și are următoarea dezvoltare posibilă:

```
    u1 ← z1
    u2 ← z2
[C] if u2 ≠ 0 goto A
    goto E
[A] if u1 = 0 goto B
    goto A    {y = 0 și z = 0}
[B] u1 ← u1 - 1
    u2 ← u2 - 1
    goto C
[E] z ← u1
```

Exemplul 7

Fie f: $N^m \rightarrow N$ o funcție (parțial) calculabilă, calculată de un program P. Vom introduce macroinstrucțiunea:

$$z \leftarrow f(z_1, \ldots, z_m).$$

Pentru a obține o dezvoltare a acestei macroinstrucțiuni, să presupunem că în programul P variabilele de intrare sunt $x_1, \ldots, x_m$, variabilele de lucru sunt $z_1, \ldots, z_k$, iar etichetele sunt $A_1, \ldots, A_p$, unde (cu eventuala excepție a lui A_p), orice etichetă care apare într-o instrucțiune **if** este eticheta a cel puțin uneia dintre instrucțiunile din P. Atunci o dezvoltare a macroinstrucțiunii $z \leftarrow f(z_1, \ldots, z_m)$ este următoarea:

```
y' ← 0
x1' ← x1
…
xm' ← xm
z1' ← 0
…
zk' ← 0
P'
I
```

unde:

- $y', x'_1, \ldots, x'_m, z'_1, \ldots, z'_k$ sunt variabile de lucru noi;
- $A'_1, \ldots, A'_p$ sunt etichete noi;
- P' se obține din P înlocuind pe $y, x_1, \ldots, x_m, z_1, \ldots, z_k, A_1, \ldots, A_p$ cu $y', x'_1, \ldots, x'_m, z'_1, \ldots, z'_k, A'_1, \ldots, A'_p$.
- I este una dintre instrucțiunile:
  - z ← y', dacă eticheta A_p este definită;
  - [A'p] z ← y', dacă eticheta A_p nu este definită.

Se observă imediat că dacă pentru anumite valori inițiale $r_1, \ldots, r_m$ ale lui $x_1, \ldots, x_m$ programul P nu se termină, atunci nici executarea macroinstrucțiunii $z \leftarrow f(z_1, \ldots, z_m)$ nu se termină pentru $z_1, \ldots, z_m$ având înainte de executarea macroinstrucțiunii valorile $r_1, \ldots, r_m$.

Dacă o macroinstrucțiune dintr-un program nu se termină, atunci nici programul care o folosește nu se termină.

Numim predicat o funcție f: $N^m \rightarrow \{0, 1\}$, parțial sau total definită, unde 0 corespunde falsului, iar 1 corespunde adevărului.

Exemplul 8

Fie P: $N^m \rightarrow \{0, 1\}$ un predicat calculabil. Atunci putem introduce macroinstrucțiunea:

```
if P(z1, …, zm) goto L
```

a cărei dezvoltare este:

```
z ← P(z1, ..., zm)
if z = 0 goto L
```

Un caz particular îl constituie predicatul:

$$P(x) = \begin{cases} 1, x = 0 \\ 0, x \neq 0 \end{cases},$$

care este calculabil conform programului:

```
if x ≠ 0 goto E
y ← y + 1
```

ceea ce permite introducerea următoarei macroinstrucțiuni: **if** v = 0 **goto** L.

Va urma...

În următoarele articole care constituie continuarea celui prezentat vom interacționa cu câteva probleme nedecidabile și vom vedea cum problema de față poate fi tratată folosind alte modele de abordare.

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.