III. Calculabilitate la nivel intuitiv

October 22, 2007

Teoria calculabilităţii, sau studiul algoritmilor, este un domeniu matematic foarte vast, în care se studiază câteva modele matematice ale ideii intuitive de algoritm. In acest capitol, urmând o sugestie a lui C. Calude, arătăm că unele rezultate din teoria calculabilităţii, culminând cu nedecidabilitatea problemei opririi, pot fi stabilite lucrând numai cu noţiunea intuitivă de algoritm. Ca o aplicaţie se obţin trei variante axiomatice ale teoremei de incompletitudine a lui Gödel din logica matematică.

1 Funcţii şi mulţimi calculabile

Presupunem cunoscută noţiunea intuitivă de *algoritm*. O putem asimila cu ideea de *program* de calculator. In cele ce urmează vom lucra cu o idealizare a acestei noţiuni, în sensul că ignorăm nu numai limitele performanţelor oricărei generaţii de calculatoare în ceea ce priveşte spaţiul de memorie şi timpul de execuţie (performanţe care se îmbunătăţesc continuu), dar şi limitele teoretice deduse din speculaţiile asupra numărului finit de particule din Univers şi asupra duratei finite a Universului. H.B. Enderton [1977] observă că aceasta revine la a stabili ,,o margine superioară teoretică a ceea ce se poate considera în orice secol ca fiind calculabil" (prin algoritmi NN). Mai trebuie spus că programele sunt scrise într-un limbaj de programare arbitrar, dar fixat, şi că sunt corecte din punct de vedere sintactic (ceea ce revine la faptul că au fost compilate cu succes).

Limbajul de programare foloseşte un *alfabet finit*. Fie A acest alfabet îmbogăţit cu un blanc separator de instrucţiuni. In felul acesta orice program, gândit în mod obişnuit ca o succesiune de cuvinte-instrucţiuni, devine un singur cuvânt peste alfabetul A. Această *compactificare* o putem aplica şi datelor de intrare, precum şi celor de ieşire: fiecare din ele este în mod curent văzută ca un şir de cuvinte din A^*, dar poate fi de asemenea gândită ca un singur cuvânt din A^*. Dacă nu menţionăm altfel, vom presupune făcută acestă compactificare.

Astfel, un program – sau algoritm – devine un cuvânt din A^*, care asociază anumitor cuvinte din A^* (semnificând diverse date de intrare), câte un cuvânt

tot din A^* (semnificând datele de ieşire corespunzătoare). Cu alte cuvinte, algoritmul determină o funcţie parţială

(1) $$f : A^* \stackrel{\circ}{\longrightarrow} A^*,$$

numită *funcţia calculată* de acel algoritm. Vom numi *funcţii intuitiv calculabile* sau, mai scurt, *funcţii calculabile*, acele funcţii f care sunt calculate de algoritmi. Observăm că o aceeaşi funcţie calculabilă poate fi calculată de mai mulţi algoritmi. Să ne amintim că o funcţie parţială (1) poate fi asimilată cu o funcţie totală

(2) $$f : \mathrm{dom} f \longrightarrow A^* \ , \ \mathrm{dom} f \subseteq A^*.$$

In cazul nostru semnificaţia domeniului $\mathrm{dom} f$ al funcţiei f este că el constă din acele date de intrare $w \in A^*$ pentru care ieşirea $f(w)$ este obţinută de algoritm în timp finit, după care algoritmul se opreşte. Pentru un cuvânt $w \in A^* \backslash \mathrm{dom} f$, semnificaţia acestei apartenenţe este că dacă se aplică la intrare w, atunci algoritmul intră într-o ciclare infinită şi deci calculul nu se opreşte niciodată. Putem presupune, fără a micşora generalitatea, că nu există cazuri de blocare a calculelor, ele fiind convertite la o ciclare infinită (de exemplu, dacă se aplică un semnal de intrare inadecvat). Reţinem că ori de câte ori spunem că *un algoritm produce o anumită ieşire*, aceasta se face *în timp finit* (într-un număr finit de paşi).

Inainte de a merge mai departe, vom adopta o standardizare care face teoria independentă de alfabetul A. Construim o bijecţie

(3) $$\beta : A^* \longrightarrow \boldsymbol{N}$$

în felul următor. Utilizăm descompunerea

$$A^* = \bigcup_{n \in \mathbb{N}} A^n \ ,$$

unde A^n este mulţimea cuvintelor de lungime n şi introducem pe A^* următoarea ordine totală: ordonăm alfabetul $A = \{a_0 < a_1 < \ldots < a_{k-1}\}$, apoi

$$A^0 < A^1 < A^2 < \ldots < A^n < \ldots ,$$

unde $A^0 = \{\lambda\}$, $A^1 = A$, iar $A^i < A^{i+1}$ înseamnă că fiecare cuvânt de lungime i este anterior fiecărui cuvânt de lungime $i+1$, iar pe fiecare mulţime A^n introducem ordinea lexicografică. Aceasta din urmă revine la isomorfismul de ordine de la A^n la $\{0, 1, 2, \ldots, k^n - 1\}$ care duce fiecare cuvânt $a_{i_1} a_{i_2} \ldots a_{i_n}$ în numărul natural care se reprezintă sub forma $i_1 i_2 \ldots i_n$ în baza k, adică

$$i_1 k^{n-1} + i_2 k^{n-2} + \ldots + i_{n-1} k + i_n \ .$$

Atunci $\beta(w)$ este pasul la care se obţine cuvântul w atunci când se parcurge A^* în ordinea de mai sus, adică

$$\beta(\lambda) = 0 \ ,$$

$$\beta(a_i) = 1 + i \qquad (i \in \{0, 1, \ldots, k-1\}) \ ,$$

$$\beta(a_{i_1} a_{i_2}) = 1 + k + i_1 k + i_2 \qquad ((i_1, i_2) \in \{0, 1, \ldots, k^2 - 1\}) \ ,$$

$$\ldots$$

$$\beta(a_{i_1}a_{i_2}\ldots a_{i_n}) = \sum_{r=0}^{n-1} k^r + i_1k^{n-1} + i_2k^{n-2} + \ldots + i_n$$

$$((i_1, i_2, \ldots, i_n) \in \{0, 1, \ldots, k^n - 1\}) \ ,$$

$$\ldots$$

Atât β, cât şi funcţia inversă β^{-1}, se calculează prin câte un algoritm. Intr-adevăr, $\beta^{-1}(0) = \lambda$, iar dacă $p \in \boldsymbol{N}\backslash\{0\}$, atunci $\beta^{-1}(p)$ se determină în felul următor. Calculăm succesiv

$$1, 1 + k, 1 + k + k^2, 1 + k + k^2 + k^3, \ldots,$$

până când găsim n astfel încât

$$\sum_{r=0}^{n-1} k^r \leq p < \sum_{s=0}^{n} k^s \ , \text{ deci } \ 0 \leq p - \sum_{r=0}^{n-1} k^r < k^n \ ,$$

de unde rezultă în mod unic $i_1, i_2, \ldots, i_n \in \{0, 1, \ldots, k-1\}$ astfel încât

$$p - \sum_{r=0}^{n-1} k^r = i_1k^{n-1} + i_2k^{n-2} + \ldots + i_{n-1}k + i_n \ ,$$

deci $\beta^{-1}(p) = a_{i_1}a_{i_2}\ldots a_{i_n}$.

Rezultă imediat că se obţine o bijecţie între $[A^*, A^*]$ şi $[\boldsymbol{N}, \boldsymbol{N}]$, asociind fiecărei funcţii $f : A^* \stackrel{\circ}{\longrightarrow} A^*$, funcţia $\beta \circ f \circ \beta^{-1}$; funcţia inversă este dată de $g \mapsto \beta^{-1} \circ g \circ \beta$. Mai mult, această bijecţie păstrează funcţiile calculabile. Am putea rezuma spunând că există un ,,dicţionar calculabil" între teoria calculabilităţii pe A^* şi teoria calculabilităţii pe $\boldsymbol{N}$. Ne vom concentra asupra acesteia din urmă.

Faptul 1. *Algoritmii formează o mulţime numărabilă.*

Argument. Codul care asociază fiecărui algoritm un număr natural (iniţial un cuvânt din A^*) este o injecţie: doi algoritmi distincţi diferă prin cel puţin o instrucţiune, deci şirurile de instrucţiuni sunt diferite, deci cuvintele-şiruri sunt diferite. □

Faptul 2. *Funcţiile calculabile formează o mulţime numărabilă, inclusă strict în mulţimea $[\boldsymbol{N}, \boldsymbol{N}]$ a funcţiilor parţiale.*

Argument. Asociind fiecărui algoritm funcţia pe care o calculează, obţinem o surjecţie de la mulţimea numărabilă a algoritmilor (cf. Fapt 1) la mulţimea funcţiilor calculabile. Aceasta demonstrează prima afirmaţie, ceea ce încheie argumentarea în virtutea observaţiei că mulţimea $[\boldsymbol{N}, \boldsymbol{N}]$ este nenumărabilă, deoarece submulţimea $\boldsymbol{N}^{\boldsymbol{N}}$ este nenumărabilă. Această ultimă afirmaţie este cunoscută şi ea se demonstrează cu tehnica diagonalei a lui Cantor: se presupune

că există o enumerare f_n , $n \in \boldsymbol{N}$, a funcţiilor de la $\boldsymbol{N}$ la $\boldsymbol{N}$ şi se construeşte funcţia

$$g : \boldsymbol{N} \longrightarrow \boldsymbol{N} \ , \ g(n) = f_n(n) + 1 \qquad (\forall n \in \boldsymbol{N}) \ ,$$

obţinându-se contradicţia $g \neq f_n$ $(\forall n \in \boldsymbol{N})$. □

*

In practica scrierii programelor de calculator dorim să ne asigurăm că programul pe care îl scriem este corect. Situaţia ideală ar fi ca răspunsul la această întrebare să fie dat tot de calculator, adică să existe un program care, primind la intrare un program oarecare, să afle dacă programul-intrare calculează funcţia dorită. Aceasta nu se poate realiza, deoarece nu se poate afla algoritmic nici măcar dacă executarea unui program se termină într-un număr finit de paşi sau continuă nedefinit. Vom demonstra această imposibilitate.

Mai întâi introducem un uşor abuz de limbaj.

Denumirea 1. Spunem că un program m cu datele de intrare n *se termină*, dacă se termină într-un număr finit de paşi calculele declanşate de m cu datele n; cu alte cuvinte, dacă n aparţine domeniului funcţiei calculate de m. *Problema opririi* este problema de a afla, pentru *orice* program m şi *orice* date de intrare n, dacă programul m cu datele n se termină.

Faptul 3 (Turing). *Problema opririi nu este decidabilă*, adică *nu este rezolvabilă algoritmic*, adică *nu există un algoritm care, primind la intrare un program m cu date de intrare n, să afle dacă programul m cu datele n se termină.*

Argument (C. Calude, A. Salomaa [1994]). Presupunem că există un *program al opririi*, contrar afirmaţiei din enunţ. Construim următorul program:

- citeşte un număr natural N ;
- generează toate programele m şi datele de intrare n care au împreună cel mult N biţi ;
- pentru fiecare pereche generată (m, n), activează programul opririi şi elimină perechile (m, n) care nu se termină ;
- pentru fiecare din perechile (m, n) rămase, activează programul m şi determină ieşirea $o(m, n)$;
- determină cea mai mare dintre ieşirile $o(m, n)$ produse şi scoate la ieşire dublul acestui maxim ;
- stop .

Observăm mai întâi că există un număr finit de perechi (m, n) de dimensiune $\leq N$, iar atât programul opririi, cât şi programele (m, n) activate se termină, prin urmare programul de mai sus se termină oricare ar fi N.

Să evaluăm numărul de biţi necesar pentru a putea scrie programul de mai sus împreună cu intrarea N. Fie k numărul minim de biţi cu care se poate scrie N. Atunci $2^{k-1} \leq N < 2^k$, deci $k - 1 \leq \log_2 N$, sau $k \leq 1 + \log_2 N$, iar programul are un număr constant de biţi. Rezultă că numărul de biţi al programului împreună cu intrarea N este mărginit de $c + \log_2 N$, unde c este un

număr natural fixat. Pe de altă parte se vede uşor că $\lim_{x\to\infty}(x - \log_2 x) = \infty$, deci pentru N suficient de mare avem $c + \log_2 N \leq N$, prin urmare programul m_0 de mai sus împreună cu intrarea N se află printre programele activate de el însuşi, deci ieşirea sa este $o(m_0, N)$. Acum obţinem contradicţia

$$o(m_0, N) = 2 \max o(m, n) > \max o(m, n) \geq o(m_0, N) \ .$$

□

Revenind la observaţia lui H.B. Enderton [1977] citată în introducere, constatăm că am pus în evidenţă o problemă care nu poate fi rezolvată cu calculatorul, oricare ar fi performanţele tehnice ale acestuia!

*

In continuare studiem descrierea algoritmică a mulţimilor. *Toate mulţimile cu care lucrăm sunt mulţimi de numere naturale.*

Denumirea 2. Spunem că o mulţime $M \subseteq \boldsymbol{N}$ este *intuitiv calculabilă* sau, mai simplu, *calculabilă*, dacă funcţia ei caracteristică

$$\chi_M : \boldsymbol{N} \longrightarrow \boldsymbol{N} \ , \ \chi_M(n) = \begin{cases} 1 & , \quad n \in M \ , \\ 0 & , \quad n \in \boldsymbol{N} \backslash M \ , \end{cases}$$

este calculabilă. Cu alte cuvinte, aceasta înseamnă că există un algoritm care, primind la intrare $n \in \boldsymbol{N}$, decide dacă $n \in M$ sau nu.

Faptul 4. *Mulţimile finite (inclusiv Ø) şi mulţimea $\boldsymbol{N}$ sunt calculabile.*

Argument. Apartenenţa la mulţimea Ø este calculată de algoritmul care, pentru orice $n \in \boldsymbol{N}$, produce ră spunsul 0 (nu).

Apartenenţa la o mulţime $\{i_1, i_2, \ldots, i_n\} \subseteq \boldsymbol{N}$ este calculată de algoritmul care citeşte $n \in \boldsymbol{N}$ şi îl compară pe rând cu $i_1, i_2, \ldots, i_n$, până când sau la un pas k găseşte $n = i_k$ şi se opreşte cu răspunsul 1 (da), sau găseşte $n \neq i_k$ pentru $k = 1, .2, \ldots, n$ şi se opreşte cu răspunsul 0 (nu).

Apartenenţa la mulţimea $\boldsymbol{N}$ este calculată de algoritmul care, pentru orice $n \in \boldsymbol{N}$ produce răspunsul 1 (da). □

Faptul 5. *Mulţimile calculabile formează un corp de mulţimi* (adică o algebră booleană).

Argument. Evident. □

Denumirea 3. Spunem că o mulţime $M \subseteq \boldsymbol{N}$ este *intuitiv enumerabilă* sau, mai simplu, *enumerabilă*,[1] dacă există un algoritm care produce şi stochează în memorie numai numere din M şi orice număr din M este produs după un număr finit de paşi.

Faptul 6. *Dacă mulţimea enumerabilă M este infinită, algoritmul corespunzător din denumirea 3 nu se opreşte niciodată.*

[1] A nu se confunda cu *numărabilă*.

Argument. Evident. □

Să notăm că denumirea 3 *nu* cere ca algoritmul să se oprească în cazul când mulţimea M este finită.

Faptul 7. *Orice mulţime calculabilă este enumerabilă.*

Argument. Dacă mulţimea M este calculabilă, atunci ea este listată de următorul algoritm.

Se generează pe rând toate numerele naturale şi pentru fiecare $n \in \boldsymbol{N}$ se activează algoritmul apartenenţei la M; dacă răspunsul este $n \in M$, atunci se reţine numărul n. Algoritmul continuă nedefinit, chiar dacă mulţimea M este finită. □

Faptul 8. *O mulţime este calculabilă dacă şi numai dacă atât ea, cât şi complementara ei sunt enumerabile.*

Argument. Fie M o mulţime calculabilă. Atunci M este enumerabilă conform faptului 7. Pe de altă parte este evident că şi $\boldsymbol{N}\backslash M$ este calculabilă (faptul 5), deci şi $\boldsymbol{N}\backslash M$ este enumerabilă.

Reciproc, să presupunem că M şi $\boldsymbol{N}\backslash M$ sunt enumerabile. Dacă $M = \emptyset$ sau $M = \boldsymbol{N}$, atunci M este calculabilă. Altfel apartenenţa la M este decisă de următorul algoritm.

Se citeşte $n \in \boldsymbol{N}$. Se activează algoritmul care listează M până când găseşte un element $a \in M$. Dacă $n = a$, algoritmul se opreşte cu răspunsul $n \in M$. Dacă nu, se activează algoritmul care listează $\boldsymbol{N}\backslash M$, până când găseşte un element $b \in \boldsymbol{N}\backslash M$. Dacă $n = b$, algoritmul se opreşte cu răspunsul $b \notin M$. Dacă nu, se reia ciclul, fiecare din cei doi algoritmi de enumerare fiind reluat de unde fusese oprit.

După un număr finit de paşi, numărul n va apărea fie pe lista produsă de primul algoritm, fie pe lista celui de-al doilea algoritm. Deci algoritmul de mai sus se opreşte. □

Faptul 9. *Dacă A şi B sunt mulţimi enumerabile, atunci mulţimile $A \cup B$ şi $A \cap B$ sunt enumerabile.*

Argument. Pentru $A \cup B$ construim următorul algoritm. Activăm algoritmul care listează A până când produce un element a_0, apoi activăm algoritmul care enumeră B până când produce un element b_0. La pasul general reluăm primul algoritm de unde a fost întrerupt, până când produce un nou element a_k, apoi al doilea algoritm până când produce un nou element b_k; etc.

Pentru $A \cap B$ modificăm algoritmul precedent în felul următor. Lista-rezultat nu mai coincide cu listele produse de algoritmul precedent, ci este construită de următoarea clauză suplimentară. La pasul general avem listele

$$a_0 \qquad b_0$$

$$a_1 \qquad b_1$$

...

a_k b_k

Comparăm a_k cu elementele $b_0, b_1, \ldots b_k$ şi dacă găsim i cu $a_k = b_i$, scriem a_k; tot aşa, dacă găsim j cu $b_k = a_j$, scriem b_k. □

Faptul 10. *Dacă mulţimea A este enumerabilă, atunci există un algoritm care enumeră elementele ei fără repetiţii* .

Argument. Considerăm un algoritm care enumeră elementele mulţimii A şi îl modficăm în felul următor: elementul a_k găsit la pasul general este scris numai dacă nu coincide cu nici unul din elementele $a_0, a_1, \ldots, a_{k-1}$ scrise anterior. □

2 Aplicaţii în logica matematică

Logica matematică lucrează cu sisteme formale. Ideea de a construi sisteme logice formale a apărut ca urmare a dorinţei de a avea atât o evidenţiere clară a ipotezelor care sunt folosite într-o demonstraţie, cât şi certitudinea corectitudinii unei demonstraţii. Mai exact, corectitudinea trebuie să poată fi verificată de către cineva care să nu apeleze la idei mai mult sau mai puţin ingenioase, ci să urmeze nişte reguli precise care să ofere garanţia că într-un număr finit (oricât de mare!) de paşi se va afla răspunsul da/nu dacă demonstraţia a fost corectă. Cu alte cuvinte, corectitudinea unei demonstraţii trebuie să poată fi verificată algoritmic. Astfel, sistemele formale sunt legate de calculabilitate şi putem aprecia că acesta este punctul de plecare al legăturii strânse dintre logica matematică a zilelor noastre şi informatică.

Observaţiile de mai sus privind sitemele formale l-au condus pe Hilbert la formularea dezideratului ca întreaga matematică să fie construită ca un sistem formal. Ulterior, Gödel a demonstrat că aceasta este o imposibilitate: nu numai întreaga matematică, dar nici măcar aritmetica nu poate fi descrisă printr-un sistem formal. Aceasta este în esenţă *teorema de incompletitudine* a lui Gödel, care a fost demonstrată pentru diverse sisteme formale cunoscute şi considerate că descriu în mod adecvat raţionamentele matematice; nici unul din aceste sisteme nu poate descrie întreaga aritmetică.

Să observăm însă că această ultimă constatare pare insuficientă pentru a conchide eşecul programului lui Hilbert: de unde ştim că nu ar putea fi inventat un nou sistem formal care să cuprindă întreaga matematică? Apare astfel ideea de a vedea care sunt trăsăturile comune pe care este rezonabil să le atribuim oricărui sistem formal şi de a demonstra teorema de tip Gödel pentru orice sistem formal cu aceste caracteristici. Altfel spus, trebuie dată o definiţie axiomatică a noţiunii de sistem formal în aşa fel încât definiţia să fie cât mai cuprinzătoare, iar teorema lui Gödel să poată fi demonstrată în acest cadru. Există mai multe astfel de axiomatizări, ceea ce transferă la nivelul axiomatizărilor întrebarea de

mai sus asupra eşecului programului lui Hilbert. Suntem astfel conduşi la ideea că afirmarea acestui eşec este o *teză* şi nu o (meta)teoremă.

In cele ce urmează vom prezenta trei asemenea axiomatizări, cu teoremele de incompletitudine corespunzătoare.

Primele două axiomatizări folosesc, pe lângă noţiunea de algoritm, încă un concept intuitiv: mulţimea

$$Ari$$

a afirmaţiilor aritmetice. De asemenea, este folosit faptul că propoziţia ,,programul (m, n) se opreşte" se poate traduce într-o afirmaţie din Ari şi această codificare este algoritmică. Acest fapt se poate demonstra folosind teoria funcţiilor primitiv recursive; aici îl vom admite ca axiomă.

Intrucât lucrăm cu trei noţiuni de sistem formal, le vom distinge prin trei etichete ad-hoc.

Incepem cu axiomatizarea propusă de Calude şi Salomaa [1994].

Denumirea 4. Un $T-sistem$ este un triplet (L, T, cod), unde $\emptyset \neq T \subset L \subset \boldsymbol{N}$ şi $cod : Ari \stackrel{\circ}{\longrightarrow} L$. Sistemul se numeşte:

– *formal*, dacă mulţimea T şi funcţia cod sunt calculabile;

– *consistent*, dacă din $cod(a) \in T$ rezultă că afirmaţia a este adevărată;

– *complet*, dacă din adevărul afirmaţiei a rezultă $cod(a) \in T$;

– *universal*, dacă funcţia cod este totală.

Mulţimea L semnifică limbajul sistemului, T este mulţimea tezelor din sistem (propoziţiile demonstrabile în sistem), iar funcţia cod exprimă în sistem unele afirmaţii aritmetice.

Faptul 11 (,,teorema de incompletitudine"). *Nu există nici un $T-$sistem formal consistent, complet şi universal.*

Argument. Presupunem că (L, T, cod) ar fi un asemenea sistem. Atunci putem construi următorul algoritm:

1. Codifică propoziţia ,,programul (m, n) se opreşte" într-o afirmaţie $a \in Ari$.
2. Calculează $cod(a)$.
3. Calculează $\chi_T(cod(a))$ (funcţia caracteristică).
4. Ieşirea este ,,da" sau ,,nu" după cum rezultatul din pasul 3 este 1 sau 0.

Se vede imediat că acest algoritm rezolvă problema opririi, în contradicţie cu faptul 3. □

Nu cunoaştem sursa axiomatizării care urmează.

Denumirea 5. Un $TF - sistem$ este un triplet (L, T, F), unde $\emptyset \neq T, F \subset L \subset \boldsymbol{N}$. Elementele mulţimilor T şi F se numesc *teze*, respectiv *falsităţi*, iar L este *limbajul* sistemului. Sistemul se numeşte:

– *formal*, dacă mulţimile L, T, F sunt enumerabile;

– *consistent sintactic*, dacă $T \cap F = \emptyset$;

– *complet sintactic*, dacă $T \cup F = L$;

– *consistent semantic*, dacă există o funcţie $int : L \longrightarrow Ari$ astfel încât pentru orice $\alpha \in L$, dacă $\alpha \in T / \alpha \in F$ atunci $int(\alpha)$ este o afirmaţie adevărată/falsă. Funcţia int se numeşte *interpretare* .

Această axiomatizare permite să se obţină atât un rezultat de incompletitudine, cât şi o ,,teoremă de completitudine".

Denumirea 6. Fie (L, T, F) un $TF-$sistem consistent semantic. Un element $\alpha \in L$ se numeşte *tautologie* dacă $int(\alpha)$ este adevărată pentru orice interpretare int. Se spune că sistemul este *sănătos* dacă orice teză este tautologie, *adecvat* dacă orice tautologie este teză, şi *complet semantic* dacă este sănătos şi adecvat.

Faptul 12 (,,teorema de completitudine"). *Orice $TF-$sistem complet sintactic şi consistent semantic este consistent sintactic şi complet semantic.*

Argument. Dacă ar exista $\alpha \in T \cap F$, ar rezulta că $int(\alpha)$ este adevărată şi falsă. Contradicţie.

Dacă $\alpha \in T$ atunci $int(\alpha)$ este adevărată.

Fie α o tautologie. Dacă $\alpha \notin T$ atunci $\alpha \in F$, deci $int(\alpha)$ este falsă. Contradicţie. □

Faptul 13. *In orice $TF-$sistem formal consistent sintactic şi complet sintactic, mulţimile T şi F sunt calculabile.*

Argument. Din faptul 8. □

Introducem acum o întărire a proprietăţii de consistenţă semantică.

Denumirea 7. Spunem că un $TF-$sistem este *universal* dacă are o interpretare calculabilă şi surjectivă.

Faptul 14 (,,teorema de incompletitudine"). *Nu există nici un $TF-$sistem formal consistent sintactic, complet sintactic şi universal.*

Argument. Presupunem că (L, T, F) are fi un asemenea sistem. Fie int o interpretare surjectivă. In virtutea faptului 13, mulţimile T şi F sunt calculabile. Atunci putem construi următorul algoritm:

1. Codifică propoziţia ,,programul (m, n) se opreşte" într-o afirmaţie $a \in Ari$.

2. Activează algoritmul care enumeră L şi pentru fiecare $x \in L$ calculează $int(x)$, până când se obţine $x_0 \in L$ pentru care $int(x_0) = a$.

3. Calculează $\chi_T(x_0)$.

4. Ieşirea este ,,da" sau ,,nu" după cum rezultatul de la pasul 3 este 1 sau 0.

Se vede imediat că acest algoritm rezolvă problema opririi, în contradicţie cu faptul 3. □

Cea de-a treia axiomatizare a noţiunii de sistem formal pe care o prezentăm are trei trăsături distinctive.

In primul rând este o axiomatizare mai detaliată, calchiată după modelul calculului predicatelor.

Apoi, axiomatizarea este realizată în termeni pur sintactici, recursul la semantică fiind implicit. Putem înţelege aceasta în felul următor. Sistemul operează cu mulţimi enumerabile de obiecte; aceste mulţimi pot fi interpretate ca sferele unor noţiuni aritmetice. Cerinţa ca aceste mulţimi, respective aceste noţiuni, să fie exprimabile în sistem, reprezintă o condiţie slabă de universalitate, întrucât nu este suficient să reprezentăm în sistem doar noţiunile aritmeticii: trebuie reprezentate şi teoremele care leagă aceste noţiuni, împreună cu demonstraţiile respective. Teorema de incompletitudine va fi prin urmare o teoremă tare, deoarece va arăta că nici această condiţie slabă de universalitate nu poate fi realizată.

A treia, dar nu cea mai puţin importantă, caracteristică a sistemului este că foloseşte anumite proprietăţi bine precizate ale mulţimilor şi funcţiilor enumerabile. Aceste proprietăţi pot fi luate ca axiome, astfel încât cadrul intuitiv este depăşit, definiţia $TF\mu-$sistemelor şi teorema de incompletitudine corespunzătoare sunt chiar o definiţie şi o teoremă în sensul matematic obişnuit.

Această abordare este datorată lui A. S. Davis [1961/65].

Definiţia 1. Lucrăm cu mulţimi incluse în $\boldsymbol{N}$ şi mulţimi incluse în produse carteziene finite de mulţimi incluse în $\boldsymbol{N}$, precum şi cu funcţii între astfel de mulţimi. Se consideră o familie de asemenea mulţimi şi funcţii, pe care o numim clasa *mulţimilor şi funcţiilor efective*, despre care presupunem că satisfac următoarele proprietăţi:

(1) mulţimile finite sunt efective;
(2) $\boldsymbol{N}$ este efectivă;
(3) dacă X şi Y sunt efective, atunci $X \cap Y$ este efectivă;
(4) o mulţime X este efectivă dacă şi numai dacă există o bijecţie efectivă $\beta : X \longrightarrow D$, unde $D = \emptyset$ sau D este de forma $D = \{0, 1, \ldots, n-1\}$ sau $D = \boldsymbol{N}$;
(5) dacă $f : A \longrightarrow B$ este efectivă, $Y \subseteq B$ şi Y este efectivă, atunci $f^{-1}(Y)$ este efectivă;
(6) dacă X este efectivă, atunci funcţiile incluziune $\iota : X \longrightarrow Y$ şi funcţiile constante $k : X \longrightarrow Z$ sunt efective;
(7) dacă f este bijecţie efectivă, atunci f^{-1} este bijecţie efectivă;
(8) dacă $f : A \longrightarrow B$ şi $g : B \longrightarrow C$ sunt efective, atunci $g \circ f$ este efectivă;
(9) dacă $f : A \longrightarrow B$, $g : A \longrightarrow C$ şi $h : B \times C \longrightarrow D$ sunt efective, atunci $h(f, g) : A \longrightarrow D$ este efectivă.

Axiomele (1) şi (2) au rolul de a asigura existenţa mulţimilor efective, atât finite, cât şi infinite. Celelalte axiome vor fi folosite în demonstraţiile rezultatelor care urmează.

Se poate arăta că mulţimile şi funcţiile intuitiv enumerabile satisfac axiomele (1)–(9). Aceasta presupune o uşoară extindere a ideii de enumerabilitate intuitivă, în sensul că luăm în consideraţie şi mulţimile incluse în produse carteziene

finite de mulţimi incluse în $\boldsymbol{N}$, precum şi funcţiile având graficul enumerabil.

Definiţia 2. Un $TF\mu-sistem$ este un şir (W, O, P, C, μ, T, F), în care: W este o mulţime infinită de numere naturale, pe care le numim *expresii*;

$$\emptyset \neq O, P, C \subset W,$$

iar elementele din O, P, C le numim respectiv *obiecte, propoziţii* şi *clauze*; μ este o funcţie

$$\mu : W^2 \longrightarrow W \text{ cu proprietatea } \mu(C \times O) \subseteq P,$$

pe care o numim *cuplare* (engl. *mating*);

$$\emptyset \neq T, F \subset P,$$

iar elementele din T şi F le numim respectiv *teze* şi *falsităţi*.

Spunem că un $TF\mu-$sistem este
– *formal*, dacă toate componentele sale sunt efective în sensul definiţiei 1;
– *consistent*, dacă $T \cap F = \emptyset$;
– *complet*, dacă $T \cup F = P$.

Definiţia 3. Intr-un $TF\mu-$sistem spunem că o mulţime de obiecte $X \subseteq O$ este *reprezentată de o clauză* $\alpha \in C$ dacă

$$\forall o \in O : o \in X \Longleftrightarrow \mu(\alpha, o) \in T.$$

Spunem că sistemul este
– *universal*, dacă are o infinitate de obiecte şi orice mulţime efectivă de obiecte este reprezentată de o clauză.

Modelul concret avut în vedere în definiţiile de mai sus este acela în care W este mulţimea expresiilor unui calculul cu predicate, propoziţiile fiind formulele închise, adică formulele fără variabile libere, clauzele sunt formulele având o singură variabilă liberă, obiectele sunt constantele individuale, iar pentru o clauză $\alpha \in C$ şi un obiect $o \in O$, cuplarea $\mu(\alpha, o)$ este rezultatul înlocuirii variabilei libere din α cu constanta o.

Propoziţia 1. *In orice $TF\mu-$sistem formal cu o infinitate de obiecte există o bijecţie efectivă*

$$\theta : O \longrightarrow W.$$

Demonstraţie. Deoarece W şi O sunt mulţimi efective, proprietatea (4) ne asigură existenţa unor bijecţii efective $\theta_1 : W \longrightarrow \boldsymbol{N}$ şi $\theta_2 : O \longrightarrow \boldsymbol{N}$. Atunci $\theta_1^{-1} : \boldsymbol{N} \longrightarrow W$ este bijecţie efectivă conform proprietăţii (7), deci $\theta = \theta_1^{-1} \circ \theta_2$ este o bijecţie efectivă cf. (8). □

Propoziţia 2. *In orice $TF\mu-$ sistem formal, orice mulţime de obiecte reprezentată de o clauză este efectivă.*

Demonstraţie. Fie $X \subseteq O$ reprezentată de clauza α. Fie $\sigma : W \longrightarrow$

W , $\sigma(\beta) = \mu(\alpha, \beta)$. Funcţia $k : W \longrightarrow W$, $k(\gamma) = \alpha$ $(\forall \gamma \in W)$ şi identitatea $1_W : W \longrightarrow W$ sunt efective cf. (6), iar $\sigma = \mu(k, 1_W)$, deci σ este efectivă cf. (9). Pentru orice $\beta \in W$ avem

$$\beta \in X \Longleftrightarrow \beta \in O \ \& \ \mu(\alpha, \beta) \in T \Longleftrightarrow \beta \in O \ \& \ \sigma(\beta) \in T \Longleftrightarrow \beta \in O \cap \sigma^{-1}(T),$$

adică $X = O \cap \sigma^{-1}(T)$, deci X este efectivă cf. (5) şi (3) . □

Teoremă (de incompletitudine). *Nu există nici un $TF\mu-$sistem formal consistent, complet şi universal.*

Demonstraţie. Presupunem că sistemul este formal, consistent şi universal; vom arăta că el nu este complet.

Fiind universal, are o infinitate de obiecte. In virtutea propoziţiei 1, rezultă că există o bijecţie efectivă $\theta : O \longrightarrow W$. Incluziunea $\iota : O \longrightarrow W$ este efectivă cf. (6). Fie $\delta = \mu(\theta, \iota) : O \longrightarrow W$. Din (9) rezultă că δ este efectivă, deci $\delta^{-1}(F)$ este efectivă cf. (5). Conform universalităţii, rezultă că $\delta^{-1}(F)$ este reprezentată de o clauză α. Atunci pentru orice $o \in O$ avem

$$\mu(\alpha, o) \in T \Longleftrightarrow o \in \delta^{-1}(F) \Longleftrightarrow \delta(o) \in F \Longleftrightarrow \mu(\theta(o), o) \in F.$$

Luând $o := \theta^{-1}(\alpha)$, obţinem

$$\mu(\alpha, \theta^{-1}(\alpha)) \in T \Longleftrightarrow \mu(\alpha, \theta^{-1}(\alpha)) \in F.$$

Dar sistemul este consistent, deci cele două afirmaţii din echivalenţa de mai sus nu pot fi amândouă adevărate, deci amândouă sunt false, prin urmare

$$\mu(\alpha, \theta^{-1}(\alpha)) \notin T \cup F$$

şi cum $\mu(\alpha, \theta^{-1}(\alpha)) \in P$, sistemul nu este complet. □