CRISTIAN S. CALUDE

SE OPREȘTE SAU NU SE OPREȘTE? ACEASTA ESTE ÎNTREBAREA

Unificare sub Semnul Problemei Opririi de Gheorghe Păun

Cartea Românească
EDUCAȚIONAL

Cuprins

Capitolul 1

De la Entscheidungsproblem la Problema Opririi

În acest capitol discutăm Problema Opririi. Începem cu algoritmi, pseudocod și exemple de programe care se opresc sau nu, apoi formulăm problema, analizăm cazurile decidabile, demonstrăm Teorema Opririi (nedecidabilitate) și explorăm probleme și soluții conexe.

1.1 De ce Problema Opririi?

Problema Opririi este importantă istoric pentru că a fost una dintre primele probleme despre care s-a demonstrat că este nedecidabilă, adică, nu este rezolvabilă de calculatoarele ‚clasice', tipul de dispozitiv de calcul pe care cititorul îl are pe birou. Calculatoarele clasice sunt implementări fizice ale noțiunii matematice de mașină Turing deterministă [151].[1]

Problema Opririi este o formulare matematică relativă la mașinile Turing deterministe.[2] După cum afirmă Copeland [138, p. 40]

> Problema Opririi a fost numită astfel (și, se pare, prima dată enunțată) de Martin Davis [147, pp. 70–71]. Afirmația că Problema Opririi nu poate fi rezolvată de un calculator este cunoscută ca ‚Teorema Opririi'. (S-a spus adesea că Turing a enunțat și demonstrat Teorema

[1] În abordarea clasică, informația este memorată în calculator sub formă de biți reprezentați logic prin 0 (off) sau 1 (on). Secvențele clasice de calcule folosesc procesoare deterministe precum x86 și ARM; repetând calculele cu aceleași intrări se obține aceeași ieșire. În contrast cu acestea, calculatoarele neconvenționale, precum cele cuantice, biologice sau de alte tipuri [70], utilizează alte forme de stocare și procesare a informației.

[2] Turing a introdus mașina care-i poartă numele în [397]. Conform cu [270], Gödel a dezvoltat ideea unei mașini de calcul care este foarte asemănătoare mașinii Turing.

Opririi în ‚Despre Numerele Calculabile',[3] dar acest fapt nu este, cu exactitate, adevărat.)

Rezultatul citat în [147, p. 70] este

Teorema 2.2. *Există o mașină Turing a cărei problemă a opririi este recursiv nerezolvabilă.*[4]

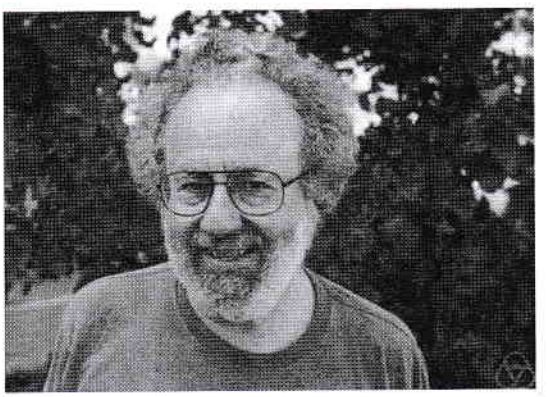

Fig. 1.1: M. Davis[5]

În cartea sa din 1958, Davis creditează monografia lui Klenee [248, p. 382] cu o demonstrație informală. Cum a notat Copeland, articolul [397] este incorect citat ca locul unde a fost prima dată enunțată și demonstrată Teorema Opririi. Care este rațiunea pentru care această afirmație eronată este cu insistență repetată? Vom răspunde la această întrebare în detaliu în Secțiunea 7.5.

1.2 O demonstrație în versuri

G. Pullum, de la Universitatea din Edinburgh, a publicat următoarea demonstrație amuzantă în versuri a Problemei Opririi [329]:[6]

[3] [397].

[4] Recursiv nerezolvabil este vechiul nume pentru efectiv/algoritmic nerezolvabil, nedecidabil sau necalculabil.

[5] Sursa: În Wikipedia. `en.wikipedia.org/wiki/Martin_Davis_(mathematician)#/media/File:Martin_Davis.jpg`.

[6] Publicată inițial în *Mathematics Magazine* 73, 4, 319–320, Octombrie 2000.

Niciun program n-ar putea erorile să depisteze.
Spun asta. Voi îmi cereți: Să ne demonstreze!
Voi arăta că poți munci până la epuizare,
dar nu poți prezice: oprire sau ciclare.

Să presupunem că avem o procedură, să-i spunem P,
care pe fiecare input ne permite să aflăm
dacă acel cod sursă, chiar dacă-i greșit,
definește un program ce din rulare s-a oprit.

Îi introduci programul, cu date potrivite,
P rulează, lucrează și, după o vreme,
(un calcul finit) prezice corect
dacă programul intră în bucle infinite.

De nu există cicluri, P spune Bine.
Înseamnă că rularea pe-acest input s-a oprit în fine.
Dar dacă depistează un ciclu ce nu se mai sfârșește,
P spune Rău!, deci rămâi în coadă de pește.

Adevărul este: P nu poate exista,
Căci dacă l-ai scrie, și mie mi l-ai da,
l-aș folosi ca să creez un logic raționament,
ce-ți zguduie mintea și te vei crede dement.

Iată șmecheria, o poți face și tu.
Voi defini o procedură, o voi numi Q,
succesul lui P de a prezice oprirea
e folosit de Q ca să-ți zdruncine gândirea.

Pentru un program dat, să-i spunem A,
primul pas al programului meu numit Q
e să afle de la P, dacă A rulat pe A
se oprește sau nu.

Dacă P spune Rău!, Q se oprește.
În caz contrar, Q iar de la capăt pornește,
și tot așa, iar și iar o ia de la-nceput
până moare universul și timpul infinit.

Treaba lui Q nu-i terminată, îi voi cere iarăși
să-și prezică purtarea rulând pe el însuși.
Oare ce face Q, cînd își citește sursa, Q?
Va intra în ciclare infinită, sau nu?

Dacă P ne prezice ciclări, Q se-oprește.
Și totuși, am presupus că P nu greșește!
Dacă Q se oprește, P spune Bine,
și Q de la capăt o ia în neștire.

Indiferent ce-ar face P, Q îl păcălește:
îl face de râs, chiar rezultatul lui P îl folosește.
Orice ar face P, nu-l poate prezice pe Q:
are dreptate când greșește, și-i fals când o nimerește!

Am creat un paradox cît se poate de curat
Doar folosind existența lui P cel postulat.
Existența lui P, presupusă, e un lat,
l-a strâns implacabil logica, și... Hat!

Din toată povestea, morala o ghiciți?
Nici nu v-o mai spun. Sunt sigur că știți.
Am arătat că nu poate exista
o procedură să ruleze lui P asemenea.

Nu vom putea găsi mecanism niciodată
Să prezică corect o rulare toată.
Va trebui să ne găsim erorile singuri!
Calculatoarele, vai... doar se sparg în figuri!

Cititorul convins de demonstrația de mai sus poate sări peste restul acestui capitol.

1.3 Programe în limbaj pseudocod

Un algoritm este o secvență de operații care trebuie executate sau efectuate. Algoritmii pot fi executați în memorie, pe hârtie sau de un calculator.

Pseudocodul este un limbaj informal de descriere de nivel înalt având o sintaxă apropiată de a limbii engleze și utilizând convențiile cunoscute ale structurii limbajelor de programare standard, fiind însă destinat mai

degrabă citirii și înțelegerii de către oameni decât execuției pe un calculator.

Pseudocodul nu este folosit doar de începători; programatori experimentați scriu ocazional pseudocod atunci când colaborează în echipă sau chiar pentru necesități proprii, îl utilizează în articole sau cărți. De exemplu, pseudocodul este folosit în știința datelor sau proiectarea web pentru a descrie pașii distincți ai unui algoritm care este ușor de înțeles pentru oricine are cunoștințe de bază de programare. Aspectul său intuitiv este un ajutor în procesele de dezvoltare de software și depanare.

Nu există niciun standard pentru sintaxa pseudocodului; nivelul de detaliu pentru acesta diferă și, în anumite cazuri, se apropie de formatul formal al limbajelor de programare generale, adică un limbaj de programare echivalent în putere de calcul cu o mașină Turing universală [50].

Programele în limbaj pseudocod utilizează instrucțiuni tipice limbajelor de programare standard, precum:

(1) asignări ($=$),
(2) comparații ($=, \neq, \leq, \geq, <, >$),
(3) operații aritmetice ($+, -, \times, /$),
(4) operații logice ($\vee, \wedge$),
(5) cuvinte cheie în programare (**read**, **print**, **do**, **for**, **stop**),
(6) instrucțiuni condiționale (**if-then-else**, **case**), iterative (**while**),
(7) blocuri (**begin**, **end**) și
(8) funcții ($\mathbf{P}(N)$).

De exemplu, $i = 1$ desemnează asignarea valorii 1 variabilei i; $s = s + 1$ este instrucțiunea de incrementare a valorii curente a variabilei s cu 1; **while** $i \leq 100$ **do** este o instrucțiune de control care execută repetitiv un bloc de instrucțiuni cât timp condiția booleană $i \leq 100$ este satisfăcută.

Vom cere ca *orice program să se termine cu o unică instrucțiune* **stop**.

1.4 Programe care se opresc

Programul (pseudocod)

```
PROGRAM(1)
1  i = 1
2  s = 0
3  while i ≤ 100 do
4        s = s + i
5        i = i + 1
6  end while
7  print s
8  stop
```

calculează suma $1 + 2 + \cdots + 100$, produce ieșirea 5 050 și se **oprește**.[7] Decizia privind oprirea programului este simplă: condiția „$i \leq 100$" din iterația **while** (liniile $3, 4, 5, 6$) este satisfăcută numai de 100 de ori, iar în iterația 101 condiția devine falsă. Controlul execuției este transferat la instrucțiunile din liniile 7 și 8, care produc ieșirea 5 050 și opresc calculul. Pentru a facilita înțelegerea programelor, instrucțiunile sunt numerotate. Aceste numere nu fac parte din program, reprezentând meta-informație.

1.5 Programe care nu se opresc

Cum ar putea un program să ruleze la nesfârșit? Figura 1.2 sugerează un astfel de comportament.

[7] Elevul de opt ani Gauss, care a devenit mai târziu *Princeps mathematicorum*, și-a uluit profesorul calculând suma rapid. Gauss nu a utilizat un calculator, dar a observat că suma constă din 50 de perechi de numere $(1 + 100), (2 + 99), (3 + 98), \ldots, (50 + 51)$ cu suma constantă și egală cu 101, adică $50 \times 101 = 5\,050$. Cu cât gândești mai mult, cu atât calculezi mai puțin și reciproc!

Fig. 1.2: După dumneavoastră!

Un exemplu de astfel de program se poate obține modificând PROGRAM(1):

PROGRAM(2)
1 $i = 1$
2 **while** $i \geq 1$ **do**
3 $\quad i = i + 1$
4 $\quad$ **print** i
5 **end while**
6 **stop**

Evident, este simplu să decidem dacă PROGRAM(2) se oprește. Condiția „$i \geq 1$" în **while** (liniile 2, 3, 4 și 5) este totdeauna satisfăcută, iar instrucțiunile 2, 3 și 4 sunt permanent repetate, formând un ciclu; PROGRAM(2) nu se oprește și instrucțiunea **stop** nu este niciodată executată. Programul produce toate valorile întregi pozitive $1, 2, 3, \ldots$

1.6 Cicluri și programe care nu se opresc

Pentru mai multe informații, să analizăm cu mai multă atenție PROGRAM(2): acesta nu se oprește niciodată datorită repetiției infinite a instrucțiunilor de pe liniile 2, 3 și 4. Execuția programului se poate descrie de următoarea secvență *infinită* de instrucțiuni:

$$1, 2, 3, 4, 2, 3, 4, 2, 3, 4, \ldots 2, 3, 4, \ldots$$

Cu alte cuvinte, PROGRAM(2) nu se oprește niciodată deoarece execută *ciclul* 2, 3, 4 la infinit, deci instrucțiunea de pe linia 5 nu este niciodată executată. În matematică, o secvență $u_1, u_2, \ldots, u_n \ldots$ se numește *periodică* dacă se repetă permanent la intervale regulate. Formal, secvența este periodică cu perioada $T > 0$ dacă $u_{n+T} = u_n$, pentru toate valorile $n \geq 1$. Cea mai mică valoare de acest fel, T, se numește cea mai mică perioadă (sau simplu „perioadă"). De exemplu, secvența $0, 1, 0, 1, 0, 1, \ldots$ este periodică cu perioada 2 și secvența $0, 1, 1, 1, 0, 1, 1, 1, \ldots$ este periodică cu perioada 4.

În informatică, un *ciclu infinit* este o secvență de instrucțiuni repetată la infinit. De exemplu, programul care execută instrucțiunile 1, 2, 3, 4, 5, 6 în ordinea următoare

$$1, 2, 3, 4, 5, 6, 3, 4, 3, 4, 3, 4, \ldots$$

conține un ciclu $(3, 4)$ și, în consecință, nu se oprește. Această secvență este periodică. În general, un program conține un ciclu infinit dacă programul execută o secvență periodică de instrucțiuni pentru o anumită intrare. Totuși, în contrast cu PROGRAM(2), se poate scrie un program care conține un ciclu infinit care se oprește pentru anumite intrări.[8]

Un program a cărui secvență de instrucțiuni conține un ciclu infinit nu se oprește niciodată și a decide dacă un program conține un ciclu este simplu.[9] Ce se poate spune despre implicația inversă? Cu alte cuvinte, este adevărat că orice secvență de calcul care nu se oprește niciodată trebuie să conțină un ciclu infinit?

Răspunsul este *negativ* și PROGRAM(3) este un exemplu în acest sens.

[8] Provocare pentru cititor: Scrieți un astfel de program.
[9] Provocare pentru cititor: De ce?

```
PROGRAM(3)
 1  i = 1
 2  while i ≥ 1 do
 3       j = 1
 4       while j ≤ i do
 5            print 0
 6            j = j + 1
 7       end while
 8       i = i + 1
 9  end while
10  stop
```

Într-adevăr, secvența de instrucțiuni executată de PROGRAM(3) începe cu

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8, 9, \ldots$$

și nu se oprește niciodată. Totuși secvența nu conține un ciclu infinit deoarece numărul de repetiții ale secvenței $4, 5, 6, 7$ crește nedefinit.

O *instrucțiune de ciclare* este o instrucțiune de control care permite execuția repetitivă a unui bloc de instrucțiuni. De exemplu, **while** este o astfel de instrucțiune pentru programele utilizate în această carte. Anumite cicluri sunt finite: ciclul din PROGRAM(1) este finit. Ciclul infinit definit de grupul de instrucțiuni $2, 3, 4, 5, 6, 7, 8, 9$ arată că PROGRAM(3) nu se oprește niciodată. Un alt exemplu este programul în care două instrucțiuni, a și b, sunt executate în următoarea secvență:

$$a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \ldots \qquad (1.1)$$

Această secvență binară infinită conține orice cuvânt binar de un număr arbitrar de ori.[10] Se poate scrie un program cu instrucțiunile etichetate a și b a cărui execuție să fie exact secvența (1.1): acest program nu se va opri, dar nu are un ciclu infinit.[11] Aceste tipuri de programe nu doar că nu se opresc, dar formează și „majoritatea" programelor care nu se opresc.[12]

[10] Provocare pentru cititor: De ce?

[11] Provocare pentru cititor: Scrieți un astfel de program.

[12] Provocare pentru cititor: Formalizați noțiunea de „majoritate" și justificați afirmația de deasupra.

1.7 Problema Opririi

După cum am văzut în Secțiunile 1.4 și 1.5, a fost simplu să decidem că PROGRAM(1) se oprește totdeauna și PROGRAM(2) nu se oprește niciodată,[13] deci poate că o analiză mai atentă ar putea conduce la găsirea unei soluții pentru orice program. Dar ce se poate spune despre discuția din Secțiunea 1.6?

> *Problema Opririi* (pentru programe) înseamnă a decide dacă un program arbitrar se oprește.

Pentru a rezolva Problema Opririi (pentru programe) avem nevoie de un program numit **decident de oprire**, care poate prezice corect în timp finit dacă orice alt *program* se va opri sau nu.[14] Nu sunt limitări de memorie sau timp impuse **decidentului de oprire** pentru a ajunge la decizie. Totuși, **decidentul de oprire** ar trebui să ajungă la decizie *în timp finit*; de aceea **decidentul de oprire** se oprește totdeauna. Cum programele includ și intrările lor, **decidentul de oprire** ar trebui să execute un program și intrarea sa.

Confirmarea faptului că un program care se oprește ajunge la instrucțiunea finală se poate obține prin execuția sa, deoarece nu există o limită în timp pentru a finaliza decizia. Ce se întâmplă în cazul în care programul nu se oprește? Pentru cât timp trebuie să-l executăm pentru a decide că nu se oprește niciodată? Intuitiv, acest timp poate fi extrem (arbitrar?) de mare. Dar, este totdeauna *finit*?

1.8 Infinit de multe cazuri decidabile

Problema Opririi este decidabilă pentru un număr infinit de clase de programe și intrări. De exemplu, orice program care nu conține instrucțiuni de ciclare se oprește, iar numărul acestor programe este infinit.

Putem construi multe alte exemple de clase de programe decidabile. Aici, exemplificăm cu două astfel de clase. Începem cu PROGRAM(1) și „parametrizăm" **while**. Înlocuind limita 100 cu variabila N se obține programul

[13] A se nota că niciunul din aceste programe nu folosește intrări.
[14] Timpul viitor indică faptul că decizia trebuie făcută ‚în avans'.