Metode de a demonstra (ne)decidabilitatea unor probleme

19 octombrie 2022

1 De ce studiem aceste metode?

În capitolul anterior am aflat că nu toate problemele sunt decidabile. Există probleme *decidabile* (recursive), *semidecidabile* (recursive enumerabile, dar nu recursive - cărora le putem identifica instanțele-*da*, dar nu le putem identifica instanțele-*nu*), și probleme care nu sunt nici măcar semi-decidabile (nu sunt recursive enumerabile). În capitolul curent:

- dăm exemple de probleme semidecidabile, și înțelegem natura lor: sunt probleme de căutare într-un spațiu infinit, care, atunci când lucrul căutat nu există, nu au cum să știe că ar trebui să înceteze căutarea;
- introducem mecanismul de reducere prin mapare (a intrărilor unei probleme în intrări ale altei probleme), ilustrat prin reducerea Turing; această tehnică ne permite să demonstrăm că anumite probleme nu sunt recursive sau recursiv enumerabile; reducerile joacă un rol important atât în teoria decidabilității (capitolul curent), cât și în teoria complexității (vom vedea mai târziu);
- realizăm o serie de reduceri Turing, un antrenament pentru a recunoaște șabloanele comune unor probleme aparent diferite; abilitatea de a recunoaște astfel de șabloane în contexte neevidente se traduce în abilitatea de a recunoaște într-o problemă nouă o problemă pe care am mai întâlnit-o și rezolvat-o, adică în mai multă eficiență în rezolvarea problemelor;

2 Reducerea Turing

2.1 Definiție

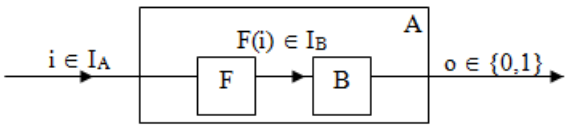

Figura 1: Reducerea Turing a unei probleme A la o problemă B

Spunem că problema A se reduce Turing la problema B ($A \leq_T B$) dacă și numai dacă există o funcție (implementabilă) F care transformă orice intrare i a problemei A într-o intrare $F(i)$ pentru problema B, astfel încât $A(i) = 1 \Leftrightarrow B(F(i)) = 1$. Deci pentru ca transformarea F să fie o reducere Turing valabilă, trebuie să existe un program sau o mașină Turing care într-un număr finit de pași să o efectueze. Altfel spus, o demonstrație matematică (neconstructivă) ca o asemenea funcție există nu este îndeajuns.

Demonstrația că $A \leq_T B$ se realizează în 3 pași: găsirea funcției de transformare F, demonstrarea implicației de la stânga la dreapta, și demonstrarea implicației de la dreapta la stânga.

Intuiția informală este că A este mai ușoară (sau la fel de ușoară) ca B, în sensul că, dacă știm să rezolvăm B, automat știm și să rezolvăm A. $\leq_T$ se comportă ca o relație de ordine între dificultatea problemelor, din punct de vedere al decidabilității.

Observația 1: Reducerea Turing nu este o relație simetrică, deci dacă A se reduce la B nu

înseamnă că și B se reduce la A. Fiecărei instanțe i (aleasă aleator) din A îi corespunde o instanță (construită convenabil, în funcție de i) din B. Întregului set de intrări pentru A i se asociază doar un subset din intrările lui B.

Observația 2: Echivalența este o dublă implicație, care trebuie demonstrată în ambele sensuri (întâi $A(i) = 1 \Rightarrow B(F(i)) = 1$, apoi $B(F(i)) = 1 \Rightarrow A(i) = 1$; demonstrând doar prima parte, arătăm că B nu transformă instanțele-*da* ale lui A în instanțe-*nu*, dar nu garantăm că B nu inventează instanțe-*da* noi).

Observația 3: Ca orice relație de ordine, $\leq_T$ este reflexivă, antisimetrică ($A \leq_T B$ și $B \leq_T A \Leftrightarrow A \equiv_T B$), și tranzitivă. Aceste proprietăți sunt ușor de demonstrat și sunt lăsate ca exercițiu.

2.2 Consecințe

1. $A \leq_T B$ și $B \in \mathcal{R} \Rightarrow A \in \mathcal{R}$ (rezultă ușor din definiția reducerii Turing)
2. $A \leq_T B$ și $B \in \mathcal{RE} \Rightarrow A \in \mathcal{RE}$ (rezultă ușor din definiția reducerii Turing)
3. $A \leq_T B$ și $A \notin \mathcal{R} \Rightarrow B \notin \mathcal{R}$ (altfel ar deveni și A recursivă, conform 1.)
4. $A \leq_T B$ și $A \notin \mathcal{RE} \Rightarrow B \notin \mathcal{RE}$ (altfel ar deveni și A recursiv enumerabilă, conform 2.)

Aceste consecințe se folosesc pentru a demonstra (ne)decidabilitatea unor probleme, folosind reduceri (de) la probleme a căror (ne)decidabilitate este cunoscută.

3 Demonstrarea nedecidabilității unor probleme

3.1 Exemple de probleme nedecidabile

- Problema opririi (se oprește un program arbitrar P pe un input arbitrar w?);
- Determinarea validității unei propoziții arbitrare în logica cu predicate de ordinul întâi;
- Rezolvarea unei ecuații diofantice arbitrare;
- Post Correspondence Problem.

3.2 Post Correspondence Problem (PCP)

3.2.1 Enunț și exemple

Dându-se 2 liste W și X a câte n cuvinte fiecare ($W = [w_1, w_2, ..., w_n]$ și $X = [x_1, x_2, ..., x_n]$), există o secvență nevidă de k întregi $i_1, i_2, ..., i_k$ astfel încât $w_{i_1} w_{i_2} ... w_{i_k} = x_{i_1} x_{i_2} ... x_{i_k}$?(unde $w_{i_1} w_{i_2} ... w_{i_k}$ reprezintă concatenarea cuvintelor $w_{i_1}, w_{i_2}, ..., w_{i_k}$, iar secvența nu trebuie să fie de k întregi diferiți între ei, ci același indice poate fi folosit de oricâte ori)

Intuitiv, pentru fiecare k (1, 2, 3, ...), putem încerca toate combinațiile de indici. Dacă există o soluție, o vom găsi mai devreme sau mai târziu; dacă însă nu există, nu vom avea niciodată garanția că nu vom găsi o soluție pentru un k mai mare decât cel curent.

Putem să ne imaginăm perechile cu același indice ca pe niște dominouri pe care trebuie să le așezăm unul lângă altul până când partea de sus arată ca partea de jos (cu precizarea că putem folosi același domino de oricâte ori). De exemplu, pentru intrarea $W = [a, bb, a], X = [aa, b, bb]$, rezultă dominourile de mai jos, iar o soluție posibilă este *(1, 3, 2, 2)*, care produce, și "sus" și "jos", cuvântul *aabbbb*.

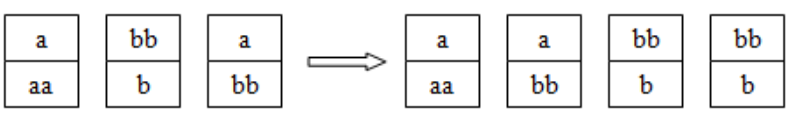

Figura 2: O pereche intrare-ieșire pentru PCP

Exerciții: Găsiți (sau argumentați că nu există) soluții pentru următoarele instanțe PCP:

1. $W = [ab, baa, aba], X = [aba, aa, baa]$
2. $W = [bb, ab, c], X = [b, ba, bc]$

3.2.2 Exemplu de reducere Turing: MPCP $\leq_T$ PCP

Prin MPCP (Modified PCP), înțelegem problema PCP cu restricția că soluțiile trebuie să înceapă cu indicele 1 (cu perechea (w_1, x_1)). Demonstrăm MPCP $\leq_T$ PCP.

Pasul 1: pentru o intrare oarecare W, X a lui MPCP, construim o intrare convenabilă W', X' a lui PCP, astfel încât $MPCP(W, X) = 1 \Leftrightarrow PCP(W', X') = 1$.
Truc: Introducem în alfabet două simboluri noi, * și \$.

- față de cum arată W în MPCP, în PCP introducem * după fiecare simbol;
- față de cum arată X în MPCP, în PCP introducem * înainte de fiecare simbol (de exemplu, dominoul (ab, bb) devine $(a * b*, *b * b)$);
- corespunzând dominoului cu indice 1 ((w_1, x_1)), adăugăm în plus un domino cu simbolurile * plasate după regula de mai sus, plus un simbol * în fața lui w_1, și numim aceast domino dominoul 0 (de exemplu, dominoul (baa, ab) generează dominoul 0 $(*b * a * a*, *a * b)$);
- adăugăm dominoul final, $(\$, *\$)$.

Astfel ne-am asigurat că orice soluție pentru PCP începe cu dominoul 0 (singurul în care cuvintele încep cu același simbol) și se termină cu dominoul final (singurul în care cuvintele se termină cu același simbol).

Pasul 2: $MPCP(W, X) = 1 \Rightarrow PCP(W', X') = 1$.
Dacă MPCP are o soluție, ea este de forma $(1, i_2, i_3, ..., i_k)$, ceea ce face ca $(0, i_2, i_3, ..., i_k, i_{final})$ să fie soluție pentru PCP.

Pasul 3: $PCP(W', X') = 1 \Rightarrow MPCP(W, X) = 1$.
Dacă PCP are o soluție, ea este de forma $(0, i_2, i_3, ..., i_k, i_{final})$, ceea ce face ca $(1, i_2, i_3, ..., i_k)$ să fie soluție pentru MPCP (practic, se obține ștergând * și \$ din soluția pentru PCP).

Exerciții:

1. Este PCP decidabilă pe un alfabet unar?
2. Este PCP decidabilă pe un alfabet binar?

3.3 Probleme înrudite cu problema opririi

3.3.1 Șabloane generale de reducere

Dacă vrem să demonstrăm că o problemă nu este recursivă (decidabilă), vom face o reducere Turing de la o problemă despre care știm că nu este recursivă (cum ar fi problema opririi) la problema noastră. Dacă vrem să demonstrăm că problema nu este nici recursiv enumerabilă, vom face reducerea de la complementul problemei opririi. Aceste șabloane se vor înțelege mai ușor după rezolvarea câtorva exerciții):

- Dacă ne interesează doar proprietăți ale setului de intrări pe care un program P' se termină, vom construi P' ca pe un program care își ignoră inputul și rulează la un moment dat $P(w)$, astfel încât proprietatea căutată să se respecte doar când $P(w)$ se termină.
- Dacă ne interesează un comportament al programului P' (în afara proprietății de a se termina sau nu pe anumite intrări), vom încerca să modificăm P astfel încât să nu manifeste niciodată acel comportament, apoi vom construi P' astfel încât să manifeste comportamentul în cauză doar după ce $P(w)$ se termină.
- Ocazional, inputul lui P' nu este ignorat complet, ci este testat pentru anumite proprietăți ușor de detectat, caz în care programul returnează, sau intră în buclă infinită, sau rulează $P(w)$ returnând doar când acesta returnează.

3.3.2 VID: Are un program arbitrar *P'* proprietatea de a nu se opri pe niciun input?

Demonstrăm că VID nu este recursiv enumerabilă demonstrând că $\overline{PO} \leq_T VID$.

Pasul 1: pentru o intrare oarecare P, w pentru $\overline{PO}$, construim o intrare convenabilă P' a lui VID, astfel încât $\overline{PO}(P, w) = 1 \Leftrightarrow VID(P') = 1$. Ne încadrăm în șablonul 1 de mai sus.

```
P'(x) {
  P(w);
  return 1;
}
```

Conform definiției lui P', acesta se oprește pe orice input ($\Rightarrow VID(P') = 0$) dacă $P(w)$ se oprește, și niciodată ($\Rightarrow VID(P') = 1$) altfel.

Pasul 2: $\overline{PO}(P, w) = 1 \Rightarrow VID(P') = 1$.
Când P ciclează pentru w ($\overline{PO}(P, w) = 1$), P' va cicla și el pentru orice input x ($VID(P') = 1$).

Pasul 3: $VID(P') = 1 \Rightarrow \overline{PO}(P, w) = 1$.
Dacă P' nu se oprește pentru niciun input, înseamnă că $P(w)$ nu se oprește nici el.

Exerciții: Determinați pentru următoarelor probleme dacă ele sunt decidabile (recursive), semi-decidabile (recursiv enumerabile dar nu recursive) sau nu sunt nici semidecidabile (nu sunt recursiv enumerabile):

1. P5: Se termină un program arbitrar P' pe intrarea 5?
2. P5': Se termină un program arbitrar **doar** pe intrarea 5?
3. EXP: Calculează un program arbitrar 2^x pentru orice intrare $x \in \mathbb{N}$?
4. STOP3: Se oprește un program arbitrar P' fix pe 3 intrari?
5. EQP: Se opresc două programe arbitrare, P_1 și P_2, pe aceleași inputuri? (reduceți atât PO, cât și VID la EQP)

4 Concluzii și lecturi recomandate

- Noțiunile de calculabilitate și decidabilitate au un istoric mai complicat decât forma în care ni se prezintă nouă astăzi (vezi www.panspermia.org/egyptians2008.doc);
- Noțiunea abstractă de calculabilitate nu se traduce direct într-o noțiune practică de calculabilitate (vezi teza Church-Turing);
- Cunoașterea nedecidabilității unei probleme ne poate conduce la a căuta cazuri particulare pe care problema să fie decidabilă; adesea problemele nedecidabile au cazuri particulare a căror decidabilitate este încă în dubiu (vezi PCP limitat la 3-6 dominouri);
- Există funcții necalculabile; un exemplu spectaculos este *busy beaver* (vezi http://www.scottaaronson.com/writings/bignumbers.html pentru o discuție interesantă despre puterea paradigmelor în matematică și știința calculatoarelor, cu referire inclusiv la busy beaver);
- Nu există reguli generale pentru identificarea felului în care o problemă se poate transforma într-alta, însă experiența și imaginația ajută foarte mult (vezi http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html pentru un mod distractiv de a prezenta problema opririi).