Metode de a demonstra (ne)decidabilitatea unor probleme

October 2017

1 De ce studiem aceste metode?

În capitolul anterior am aflat ca nu toate problemele sunt decidabile. Există probleme *decidabile*, *semidecidabile* (cărora le putem identifica instanțele-da, dar nu le putem identifica instanțele-nu), și probleme care nu sunt nici măcar semidecidabile. Problemele *nedecidabile* includ ultimele două categorii (adică toate problemele care nu sunt *decidabile*). În capitolul curent:

- dăm exemple de probleme semidecidabile dar nedecidabile, și înțelegem natura lor: sunt probleme de căutare într-un spațiu infinit, care, atunci când lucrul căutat nu există, nu au cum sa știe că ar trebui sa înceteze căutarea;
- introducem mecanismul de reducere prin mapare (a intrarilor unei probleme în intrari ale altei probleme), ilustrat prin reducerea Turing; această tehnică ne permite să demonstrăm că anumite probleme sunt nedecidabile; 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;
- introducem, pe scurt, noțiunea de gramatică independentă de context, un mecanism matematic simplu folosit cu succes în descrierea limbajelor formale și, într-o bună măsură, și a limbajului natural (al comunicării între oameni).

2 Reducerea Turing

2.1 Definiție

Spunem că problema A se reduce Turing la problema B (A $\leq_T$ B) dacă și numai dacă există o funcție F care transformă orice intrare i a problemei A într-o intrare *F(i)* pentru problema B, astfel încât A(i) = 1 , B(F(i)) = 1.

Demonstrația că A $\leq_T$ B se realizeaza 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, daca știm să rezolvam 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.

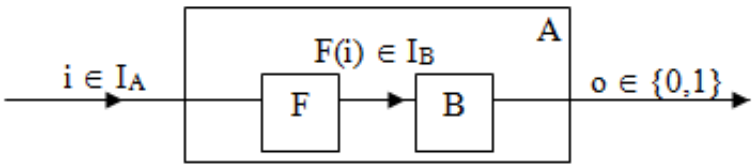

Figura 1: Reducerea Turing a unei probleme A la o problema B

Observația 1: Reducerea Turing nu este o relație simetrică. Faptul că dacă A se reduce la B nu înseamnă că și B se reduce la A este pus în evidență de sublinierea cuvintelor *orice* și *o*. Fiecărei instanțe i (aleasă aleator) din A îi corespunde o instanță (construită convenabil, în funcție de) din B. Întregului set de intrări pentru A i se asociaza 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 ca 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 $\equiv$ 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 decidabilă $\Rightarrow$ A decidabilă (rezultă ușor din definiția reducerii Turing)
2. A $\leq_T$ B și A nedecidabilă $\Rightarrow$ B nedecidabilă (altfel ar deveni și A decidabilă, conform 1.)

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

Exerciții: Ținând cont de faptul că $\leq_T$ se comportă ca o relație de ordine, și cunoscând că A $\leq_T$ B și B $\leq_T$ C, spuneți despre următoarele afirmații dacă sunt sigur adevărate (A), sigur false (F), sau uneori adevărate (A/F) (observație: acolo unde apar operații pe mulțimi, considerați că fiecare problemă este reprezentată prin mulțimea ei de instanțe-da):

- A este semidecidabilă dar nu decidabilă, iar C este decidabilă.
- Daca C este semidecidabilă, atunci A $\cup$ B este semidecidabilă.
- A este decidabilă, B este semidecidabilă, iar C nu este semidecidabilă.
- Daca B este decidabilă, atunci complementul lui A este (o problema) decidabilă.
- C este semidecidabilă dar nu decidabilă, iar A este decidabilă.
- Daca B este semidecidabilă, atunci B $\cap$ C este semidecidabilă.
- C este decidabilă, iar complementul lui B nu este (o problemă) semidecidabilă.
- Daca B este decidabilă, atunci complementul lui C este (o problemă) decidabilă.

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 = $[w_1, w_2, ..., w_n]$ s, i 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 sa fie de k întregi diferiți între ei, același indice putând 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*.

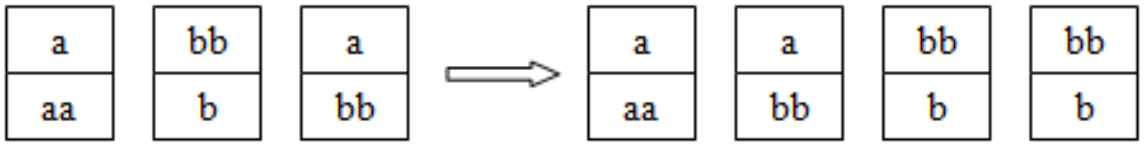

Figura 2: O pereche intrare-ieșire pentru PCP

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 (w1, x1)). 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 * dupa 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 ((w1, x1)), adăugam în plus un domino cu simbolurile * plasate dupa regula de mai sus, plus un simbol * în fața lui w1, 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 Proprietați ale gramaticilor independente de context

3.4 Gramatici independente de context

PCP se folosește frecvent pentru demonstrarea nedecidabilității unor proprietăți ale gramaticilor, unde prin gramatică înțelegem un set de reguli de formare a cuvintelor într-un anumit limbaj. O gramatică formală oarecare se definește printr-un tuplu (N, T, S, P):

- N = neterminali (variabile care se pot înlocui conform regulilor de producție din P);
- T = terminali (constante, simboluri dintr-un alfabet, de exemplu literele de la a la z);
- S = simbol de start (un neterminal care va evolua conform regulilor din P);
- P = reguli de producție.

Gramaticile independente de context sunt gramatici cu proprietatea că toate regulile din P sunt de forma $V \to w$, unde V reprezintă un singur neterminal, iar w reprezinta un șir de terminali și/sau neterminali, posibil vid. Independența de context se referă la faptul că neterminalii pot fi rescriși

indiferent de contextul în care apar (ceea ce nu se întâmplă la o regulă de genul $w_1Vw_2 \to w_3$, de exemplu).

Limbajul generat de o gramatică G (notat L(G)) reprezintă mulțimea tuturor cuvintelor care se pot obține plecând de la simbolul de start și aplicând reguli de producție până când se ajunge la un șir care nu mai conține niciun neterminal. De exemplu, limbajul generat de gramatica $S \to a|aS|bS$ (S trece în a, sau în aS, sau în bS) este mulțimea tuturor cuvintelor nevide formate din simbolurile a și b, care se termină obligatoriu în a.

3.4.1 Problema intersecției a doua gramatici independente de context (PI) ($L(G1) \cap L(G2) \neq \emptyset$)

Pentru a demonstra ca PI este nedecidabilă, demonstrăm PCP $\leq_T$ PI. Cum PCP este nedecidabilă, folosind consecința 2, rezultă că PI este nedecidabilă.

Pasul 1: pentru o intrare oarecare *W, X* a lui PCP, construim o intrare convenabilă G_1,G_2 a lui PI, astfel încât $PCP(W, X) = 1 \Leftrightarrow PI(G_1, G_2) = 1$.

Truc: Introducem n (cu n = lungimea listelor W și X) noi simboluri (care nu se află în alfabetul instanței PCP): $a_1, a_2, ..., a_n$.

- definim G1 prin regulile $S1 \to w_1S_1a_1|w_2S_1a_2|...|w_nS_1a_n$ și $S1 \to w_1a_1|w_2a_2|...|w_na_n$;
- definim G2 prin regulile $S2 \to x_1S_2a_1|x_2S_2a_2|...|x_nS_2a_n$ și $S2 \to x_1a_1|x_2a_2|...|x_na_n$.

Pasul 2: $PCP(W, X) = 1 \Leftrightarrow PI(G_1, G_2) = 1$. Dacă PCP are o soluție de forma $(i_1, i_2, i_3, ..., i_k)$, atunci aplicăm pe rând regulile de producție corespunzătoare acestor indici, $S_1 \to w_{i_1}S_1a_{i_1} \to w_{i_1}w_{i_2}S_1a_{i_2}a_{i_1} \to ... \to w_{i_1}w_{i_2}...w_{i_k}a_{i_k}...a_{i_2}a_{i_1}$, s, i $S_2 \to x_{i_1}S_2a_{i_1} \to x_{i_1}x_{i_2}S_2a_{i_2}a_{i_1} \to ... \to x_{i_1}x_{i_2}...x_{i_k}a_{i_k}...a_{i_2}a_{i_1}$, și observăm că obținem același cuvânt, deci intersecția e nevidă ($PI(G_1, G_2) = 1$).

Pasul 3: $PI(G_1, G_2) = 1 \Leftrightarrow PCP(W, X) = 1$. Dacă $L(G_1) \cap L(G_2) \neq \emptyset$, înseamnă că există o serie de indici $i_1, i_2, ..., i_k$ ai regulilor de producție alese, astfel încât $w_{i_1}w_{i_2}...w_{i_k}a_{i_k}...a_{i_2}a_{i_1} = x_{i_1}x_{i_2}...x_{i_k}a_{i_k}...a_{i_2}a_{i_1}$ (trebuie să fie aceiași indici, pentru ca cele 2 cuvinte să se termine în același șir de simboluri a). Acest șir de indici devine soluție pentru PCP.

Exercițiu (similar cu demonstrația anterioară): Demonstrați nedecidabilitatea problemei ambiguității gramaticilor independente de context (PA) (o gramatică G este ambiguă dacă există un cuvânt $w \in G$ care poate fi generat, folosind regulile de producție din G, în două moduri diferite).

3.5 Probleme înrudite cu problema opririi

3.5.1 Șabloane generale de reducere

Nedecidabilitatea multor probleme se demonstrează prin reducere de la problema opririi (PO(P,w)). În general (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 intrari), vom încerca să modificăm P astfel încât sa nu manifeste niciodata acel comportament, apoi vom construi P' astfel încât să manifeste comportamentul în cauză doar dupa 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 returneaza, sau intră în buclă infinită, sau ruleaza P(w) returnând doar când acesta returnează.

Observație: Uneori, în demonstrarea nedecidabilității unei probleme P_2, vom reduce o problemă nedecidabilă P_1 la complementul lui P_2, construind o intrare *F(i)* pentru P_2 pe baza unei intrări i pentru P_1 astfel încât $P_1(i) = 1 \Leftrightarrow P_2(F(i)) = 0$. Dacă P_2 ar fi decidabilă, n-ar conta că inversăm 1 și 0, în continuare am putea folosi ieșirea lui P_2 pentru a determina ieșirea lui P_1.

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

Demonstrăm că VID este nedecidabilă demonstrând că $PO \leq_T \neg VID$.

Pasul 1: pentru o intrare oarecare *P, w* pentru PO, construim o intrare convenabilă P' a lui VID, astfel încât $PO(P, w) = 1 \Leftrightarrow VID(P_0) = 0$. Ne încadrăm în șablonul 1 de mai sus.

```
1: function P'(x)
2:     P(w)
3:     return 1
4: end function
```

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

Pasul 2: $PO(P, w) = 1 \Rightarrow VID(P') = 0$. Când P se oprește pe w (PO(P,w) = 1), P' se va opri și el pe orice input x (VID(P0) = 0).

Pasul 3: $VID(P') = 0 \Rightarrow PO(P, w) = 1$. Când există inputuri pe care P' se oprește, înseamnă că P(w) s-a oprit.

Exerciții: Demonstrați decidabilitatea sau nedecidabilitatea următoarelor probleme:

1. P5: Se termină un program arbitrar P' pe inputul 5?
2. P3*0: Printează un program arbitrar P' 3 de 0 la rând, atunci când este rulat pe inputul 1?
3. STOP3: Se oprește un program arbitrar P' fix pe 3 inputuri?
4. 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 cauta cazuri particulare pe care problema să fie decidabilă; adesea problemele nedecidabile au cazuri particulare a caror 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, însa 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).