**Nedecidabilitate**

1. Ce este nedecidabilitatea?

- O problemă este nedecidabilă dacă nu există niciun algoritm care să o poată rezolva pentru toate intrările (de exemplu, verificarea software)

- Deși ar părea că avem computere suficient de puternice să rezolve orice problemă folosind algoritmi, există totuși probleme, chiar din viața de zi cu zi, care nu pot fi rezolvate computațional

- O problemă relevantă este problema ATM: pentru o TM M și un string w, putem decide dacă M acceptă inputul w?

- Mașina Turing Universală (UTM), propusă de Alan Turing în 1936, avea ca scop fix rezolvarea acestei probleme și funcționează astfel:

- TM U primește ca input reprezentarea sub formă de string a TM M și stringului w, notată ⟨M,w⟩

- U simulează M pe inputul w

- Dacă M intră în stare de accept, acceptă și U

- Dacă M intră în stare de reject, rejectă și U

- Totuși, nu putem spune dacă această mașină se va opri. Pe unele inputuri, va merge la nesfârșit și un algoritm nu poate afla răspunsul la această întrebare

- Distincție importantă: ATM este Turing-recognoscibil dar nu decidabil

- O TM U poate recunoaște ATM prin simularea lui M pe w

- Dacă M acceptă w, U va accepta eventual

- Dacă M nu acceptă w, U poate rula la infinit

- Această limitare demonstrează diferența fundamentală între recunoaștere și decizie

2. Metoda Diagonalizării

- Tehnică descoperită de Cantor în 1873 folosită pentru a demonstra că anumite mulțimi sunt mai mari decât altele

- Cantor a observat că două mulțimi finite au aceeași dimensiune dacă elementele unei mulțimi pot fi asociate cu elementele celeilalte mulțimi

- Această metodă poate fi extinsă la mulțimi infinite

- Pentru mulțimi A și B și o funcție f de la A la B:

- f este one-to-one dacă nu mapează niciodată elemente diferite în același loc (f(a) ≠ f(b) când a ≠ b)

- f este onto dacă atinge fiecare element din B (pentru fiecare b din B există un a din A unde f(a) = b)

- A și B au aceeași dimensiune dacă există o funcție one-to-one, onto f:A→B

- O funcție care este atât one-to-one cât și onto se numește corespondență

3. Mulțimi Numărabile

- O mulțime este numărabilă dacă este finită sau poate fi pusă în corespondență cu N (numerele naturale)

- Dacă o mulțime infinită nu poate fi pusă în corespondență cu N atunci este nenumărabilă

4. Q este Numărabilă

- Fie Q = {m/n | m, n din N} mulțimea numerelor raționale pozitive

- Deși Q pare mult mai mare decât N, aceste mulțimi au aceeași dimensiune

- Metoda demonstrației:

- Creăm o matrice infinită conținând toate numerele raționale pozitive

- Numărul i/j apare pe linia i și coloana j

- Convertim matricea în listă folosind diagonalizarea

- Prima diagonală: 1/1

- A doua diagonală: 2/1, 1/2

- A treia diagonală: 3/1, 2/2 (sărim deoarece este echivalent cu 1/1), 1/3

- Continuăm acest proces pentru a lista toate elementele din Q

- Aceasta creează o corespondență cu N

5. R este Nenumărabilă

- Fie R mulțimea numerelor reale

- Demonstrație prin contradicție:

- Presupunem că există o corespondență f între N și R

- Construim un număr x care diferă de fiecare număr din corespondență

- Fie f(1) = 3.14159..., f(2) = 55.555555..., f(3) = 1.1111..., etc.

- Construim x între 0 și 1

- Alegem fiecare cifră a lui x să difere de cifra corespunzătoare din f(n)

- Prima cifră ≠ prima cifră din f(1)

- A doua cifră ≠ a doua cifră din f(2)

- Și așa mai departe

- Acest x este în R dar nu este în listă

- Notă: Nu selectăm niciodată 0 sau 9 când construim x pentru a evita probleme cu reprezentări echivalente (0.1999... = 0.2000...)

6. Unele Limbaje nu sunt Turing-Recognoscibile

- Fie L mulțimea tuturor limbajelor peste alfabetul Σ

- Arătăm că L este nenumărabilă prin stabilirea unei corespondențe cu B (mulțimea secvențelor binare infinite)

- Fie Σ\* = {s1, s2, s3, ...}

- Fiecare limbaj A din L are o secvență unică în B

- Al i-lea bit din secvență este 1 dacă si este în A, 0 în caz contrar

- Exemplu:

- Dacă A ar fi limbajul tuturor șirurilor care încep cu 0 peste alfabetul {0,1}:

- Σ\* = {ε, 0, 1, 00, 01, 10, 11, 000, 001, ...}

- A = {ε, 0, 00, 01, 000, 001, ...}

- Secvența = 0 1 0 1 1 0 0 1 1 ...

- Funcția f: L → B, unde f(A) este secvența caracteristică a lui A, este o corespondență

- Prin urmare, deoarece B este nenumărabilă, L este nenumărabilă

- Concluzie: Există un număr nenumărabil de limbaje dar doar un număr numărabil de TM, deci unele limbaje nu pot fi recunoscute de nicio TM

7. ATM este Nedecidabilă

- ATM = {⟨M,w⟩| M este o TM și M acceptă w}

- Demonstrație prin contradicție:

- Presupunem că ATM este decidabilă

- Fie H un decider pentru ATM unde:

H(⟨M,w⟩) = acceptă dacă M acceptă w

rejectă dacă M nu acceptă w

- Construim D folosind H ca subrutină:

D(⟨M⟩) = acceptă dacă M nu acceptă ⟨M⟩

rejectă dacă M acceptă ⟨M⟩

- Când rulăm D cu propria sa descriere ⟨D⟩ ca input:

D(⟨D⟩) = acceptă dacă D nu acceptă ⟨D⟩

rejectă dacă D acceptă ⟨D⟩

- Aceasta creează un paradox - D trebuie să facă opusul a ceea ce face pe propriul input

- Contradicție → ATM este nedecidabilă

- Diagonalizarea apare dacă construim o matrice unde:

- Linii: toate TM-urile (M1, M2, M3, ...)

- Coloane: reprezentările string (⟨M1⟩, ⟨M2⟩, ⟨M3⟩, ...)

- Celula [i,j]: "accept" dacă Mi acceptă ⟨Mj⟩, "reject" dacă nu

- Contradicția apare în celula pentru D și ⟨D⟩

**NP-Completitudine**

1. Formulă Booleană Satisfiabilă

- Variabilele care pot lua valorile TRUE și FALSE se numesc variabile booleene

- Operațiile booleene: AND (∧), OR (∨), NOT (¬)

- O formulă booleană este o expresie care implică variabile și operații booleene

Exemplu: φ = (x ∧ y) ∨ (x ∧ z)

- O formulă booleană este satisfiabilă dacă există o atribuire de 0 și 1 variabilelor care face formula să evalueze la 1

Exemplu: Formula de mai sus este satisfiabilă cu x=0, y=1, z=0

- Definim SAT = {⟨φ⟩| φ este o formulă booleană satisfiabilă}

2. Reductibilitate în Timp Polinomial

- O funcție f: Σ\*→Σ\* este calculabilă în timp polinomial dacă există o mașină Turing M care se oprește cu doar f(w) pe bandă când pornește cu orice input w

- Limbajul A este reductibil în timp polinomial la limbajul B (notat A ≤P B) dacă:

- Există o funcție f: Σ\*→Σ\* calculabilă în timp polinomial

- Pentru orice w, w ∈ A ⇔ f(w) ∈ B

- f se numește reducerea în timp polinomial a lui A la B

- Dacă A ≤P B și B este în P, atunci A este în P

- Demonstrație:

- Fie M algoritm în timp polinomial care decide B

- Fie f reducerea în timp polinomial de la A la B

- Descriem algoritm în timp polinomial N care decide A:

N = "Pe input w:

1. Calculează f(w)

2. Rulează M pe f(w) și returnează ce returnează M"

- Deoarece f este reducere de la A la B → w ∈ A oricând f(w) ∈ B

- Astfel M acceptă f(w) oricând w ∈ A

3. Formula (3)CNF

- Un literal este o variabilă booleană sau negația unei variabile booleene

- O clauză este formată din mai mulți literali conectați prin ∨

- O formulă booleană este în formă normală conjunctivă (CNF) dacă este compusă din mai multe clauze conectate prin ∧

- Este o formulă 3CNF dacă toate clauzele au exact trei literali

- Definim 3SAT = {⟨φ⟩| φ este o formulă 3CNF satisfiabilă}

- Exemplu de reducere: 3SAT se reduce la CLIQUE

- Pentru o formulă 3CNF, generăm un graf neorientat unde:

- Nodurile sunt literali din 3SAT

- Nodurile sunt împărțite în k triplete (clauzele din 3SAT)

- Există muchii între toate nodurile exceptând:

- Nodurile din același triplet

- Nodurile cu valori complementare (x1 și ¬x1)

- Formula este satisfiabilă dacă și numai dacă G are o k-clică

4. NP-Complete

- Un limbaj B este NP-Complete dacă:

a. B este în NP

b. Orice A din NP este reductibil în timp polinomial la B

- Importanță teoretică:

- Pentru a demonstra P≠NP: arată că orice problemă NP necesită mai mult decât timp polinomial

- Pentru a demonstra P=NP: găsește soluție în timp polinomial pentru orice problemă NP-completă

- Importanță practică:

- Indică futilitatea căutării algoritmilor polinomiali pentru probleme NP-complete

- NP-completitudinea sugerează că nu există soluție în timp polinomial

- Ajută cercetătorii să evite investirea timpului în căutarea algoritmilor în timp polinomial care probabil nu există

- Ghidează focusul spre algoritmi de aproximare și soluții euristice

- Teoreme:

- Dacă B este NP-Complete și B este în P, atunci P=NP

- Dacă B este NP-Complete și B ≤P C pentru C în NP, atunci C este NP-Complete

5. Teorema Cook-Levin

- Afirmă că SAT este NP-Complete

- Descoperită independent de Stephen Cook și Leonid Levin în anii 1970

- Demonstrație:

1. Arată că SAT ∈ NP:

- O TM nedeterministă poate ghici atribuirea și verifica în timp polinomial

2. Arată că orice A ∈ NP se reduce la SAT:

- Fie N o NTM care decide A în timp nk

- Pentru input w, construiește formula φ care simulează N pe w

- Construiește un tablou (tabel de calcul) de dimensiune nk × nk

- φ = φcell ∧ φstart ∧ φmove ∧ φaccept unde:

- Variabile: xi,j,s pentru poziția (i,j) și simbolul s

- φcell: asigură un singur simbol per celulă

Formula: ∧i,j(∨s∈C xi,j,s) ∧ ∧i,j,s,t∈C,s≠t(¬xi,j,s ∨ ¬xi,j,t)

- φstart: codifică configurația inițială pe w

Formula: x1,1,# ∧ x1,2,q0 ∧ x1,3,w1 ∧ ... ∧ x1,n+2,wn ∧ x1,n+3,\_ ∧ ... ∧ x1,nk,#

- φmove: asigură tranziții valide între configurații

Formula: Verifică toate ferestrele 2×3 pentru tranziții valide conform funcției de tranziție a lui N

- φaccept: asigură apariția stării de acceptare

Formula: ∨i,j xi,j,qaccept

3. Proprietăți cheie:

- φ este satisfiabilă ⇔ N acceptă w

- Dimensiunea lui φ este polinomială în |w|

- Construcția durează timp polinomial

4. Structura tabloului:

- Fiecare rând reprezintă o configurație

- Primul rând este configurația inițială

-Fiecare rând următor rezultă din cel anterior prin funcția de tranziție a lui N

* Un tablou de acceptare conține o configurație de acceptare

1. 3SAT este NP-Completă

* Demonstrație:
  1. Arată că 3SAT ∈ NP (evident)
  2. Arată NP-Completitudinea modificând demonstrația Cook-Levin:
     + Începe cu formula φ = φcell ∧ φstart ∧ φmove ∧ φaccept
     + Convertește fiecare parte în CNF:
       - φcell: deja în CNF (AND de clauze)
       - φstart: deja CNF (AND de variabile simple)
       - φaccept: clauză simplă (OR de variabile)
       - φmove: convertește folosind legea distributivității (OR de ANDs → AND de ORs)
  3. Convertește în 3CNF:
     + Pentru clauze < 3 literali:
       - Replică literali până la 3
     + Pentru clauze > 3 literali: (a₁ ∨ a₂ ∨ a₃ ∨ a₄) → (a₁ ∨ a₂ ∨ z) ∧ (¬z ∨ a₃ ∨ a₄)
     + Cazul general pentru l literali: (a₁ ∨ ... ∨ aₗ) → (a₁ ∨ a₂ ∨ z₁) ∧ (z₁ ∨ a₃ ∨ z₂) ∧ ... ∧ (zₗ₋₃ ∨ aₗ₋₁ ∨ aₗ)
  4. Puncte cheie:
     + Transformarea păstrează satisfiabilitatea
     + Dimensiunea crește doar cu un factor constant
     + Procesul durează timp polinomial