**Undecidability:**  
- ce este undecidability?(ar parea ca avem computere suficient de puternice sa rezolve orice problema folosing algritmi, dar exista totusi probleme, chiar din viata de zi cu zi, care nu pot fi rezolvate computational. Aceste probleme se numesc probleme nerezolvabile. O problema relevanta ar fi problema A\_TM = pentru o TM M si un string w, putem decide daca M accepta inputul w? Vom arata mai tarziu ca aceasta problema este nedecidabila. O TM universala, propusa de Alan Turing in 1936, avea ca scop fix rezolvarea acestei probleme si arata cam asa: TM U primeste ca input reprezentarea sub forma de string a TM M si stringului w, <M,w> si simuleaza M pe inputul w, daca M intra in stare de accept, accepta si U, iar daca M intra in stare de reject, rejecta si U. Totusi, nu putem spune daca aceasta masina se va opri. Poate, pe unele inputuri, va merge la nesfarsit si un algoritm nu poate afla raspunsul la aceasta intrebare)

- metoda diagonalizarii + multimi numarabile(Q)/nenumarabile(R)(Georg Cantor si-a pus problema compararii dimensiunii a 2 multimi. La cele finite este usor, aflam cardinalul acestora si gata, dar la cele infinite nu putem numara cate elemente avem. Totusi, exista o tehnica prin care elementele unei multimi pot fi asociate cu elementele unei alte multimi, lucru, care, la multimile finite, functioneaza si ajuta la compararea dimensiunilor prin functia de asociere f. Putem extinde aceasta idee si la multimi infinite. De exemplu intre N si 2N, functia de asociere va fi f(n)=2n, iar astfel putem spune ca N si 2N au aceeasi dimensiune, desi pare contraintuitiv. Numim o multime ca fiind numarabila daca este finita sau are aceeasi dimensiune cu N. Aratam ca Q={i/j | i,j nr naturale} este numarabila folosing metoda diagonalizarii: cream o matrice infinita unde, pe linia i se afla fractiile cu numaratorul i, iar pe coloana j se afla fractiile cu numitorul j. Asociem N cu Q, luand primul element din Q cu primul din N, al doilea din Q cu al doilea din N samd. Convertim matricea in lista pentru a face acest lucru. O idee rea de a converti matricea in lista ar fi sa punem liniile cap la cap, doar ca, matricea fiind infinita si liniile sunt infinite si deci nu vom ajunge niciodata pe a doua linie a matricei. Ideea propusa de Georg Cantor in 1871 este sa luam pe diagonala incepand din coltul din stanga sus. Prima diagonala contine doar pe 1/1. A doua diagonala contine 2/1 si 1/2. Apare o problema la a treia diagonala unde apar 3/1, 2/2 si 1/3. 2/2 este aceeasi cu 1/1 asa ca nu o alegem, ramanand doar 3/1 si 1/3 samd, obtinand astfel toate numerele din Q si punand Q in corespondenta cu N. Desi pare posibil sa gasim o corespondenta intre orice 2 multimi, vom arata ca nu exista o asemenea legatura intre N si R. Vom face asta, construind un numar real X care este diferit de orice valoare f(n) reala pentru n natural. Cum difera numarul nostru X de toate celelalte? Este un numar intre 0 si 1, care, are cifra zecimala de pe pozitia i diferita de cifra zecimala de pe pozitia i a numarului f(i), adica, pentru f(1)=0.12345 alegem arbitrar 4!=1, pentru f(2)=43.3765 alegem arbitrar 5!=7 samd. Exista o problema care poate aparea, de exemplu numerele 0.29999999 si 0.30000000 sunt egale desi au scriere diferita, asa ca atunci cand alegem zecimala la constructia numarului, evitam valorile de 0 si 9. Astfel am aratat ca R nu poate fi pusa in asociere cu N, deci R este nenumarabila)

- limbaje non Turing recognizable(multimea masinilor turing e numarabila, masinile turing avand reprezentarea sub forma de string, iar multimea stringurilor peste alfabetul Sigma este numarabila, putand avea un numar finit de string-uri de dimensiune 0,1,2,… Pe de alta parte multimea limbajelor este nenumarabila putand fi pusa in asociere cu multimea secventelor binare infinite B. Avem limbajul A din multimea limbajelor L. A are o configuratie unica in B, unde, pentru Sigma\* = {s0,s1,s2,s3,…} pe bitul i avem valoarea 1 daca si este in A si 0 daca si nu este in A. Aceasta asociere se numeste secventa caractersitica a lui A in B si se noteaza cu XA. Astfel, am aratat ca exista un numar nenumarabil de limbaje pentru un numar numarabil de TM, deci unele limbaje nu sunt nici macar TR, deci sunt non TR)

- A\_TM este nedecidabila(Alan Turing s-a folosit de metoda diagonalizarii a lui Cantor si a aplicat-o asupra TM. Sa presupunem ca A\_TM este decidabila si ca avem un decider H care decide problema. H functioneaza in felul urmator. H – input <M,w> - accepta daca M accepta w, rejecta daca M nu accepta w. Construim o noua TM D care foloseste H ca subrutina si afiseaza opusul a ceea se raspunde H. Pe scurt, D – input <M> - Ruleaza H pe <M,<M>> si accepta daca M nu accepta <M>, rejecta daca M accepta <M>. Acum rulam D<D>(adica TM D primeste ca input configuratia proprie). Practic asta ne spune ca D accepta inputul <D> daca D nu accepta inputul <D> si rejecta inputul <D> daca D accepta inputul <D> => contradictie, deci A\_TM este nedecidabila. Diagonalizarea apare daca construim o matrice unde punem pe linii toate TM-urile, M1,M2,M3,... si pe coloane reprezentarea acestora sub forma de string <M1>,<M2>,<M3>,... . Daca Mi accepta <Mj> atunci in celula [i,j] punem accept, daca nu reject. Contradictia apare mai jos pe linia lui D si coloana <D>, nestiind ce sa punem, accept sau reject)

**NP-complete:**

- SAT formula(In logica booleana avem valori de TRUE/FALSE. Pentru a reprezenta o formula in aceasta logica folosim variabile si operatii precum ∨, ∧, ¬. Se pune problema, avand o expresie booleana, daca aceasta este satisfiabila sau nu. O expresie booleana poate arata ceva de genul (¬x1 ∨ x2) ∧ x3. Deci, problema satisfiabilitatii unei expresii vrea sa determine SAT = {Φ | Φ este o expresie logica satisfiabila})

- Reductibilitate in timp polinomial(Stim despre notiunea de a reduce eficient o problema la alta. Trebuie sa tinem cont si de complexitatea de timp, asa ca acum vorbim despre reducerea unei probleme la alta in timp polinomial. O functie f:Sigma\*->Sigma\* se numeste polinomial computabila daca exista o TM M care se opreste doar cu f(w) pe banda primind ca input w. Spunem ca limbajul A se reduce in timp polinomial la limbajul B daca exista o functie polinomial computabila f unde pentru orice w din A ⬄ f(w) este in B. Daca avem o problema A care se reduce in timp polinomial la o alta problema B, daca gasim o solutie in timp polinomial pentru a rezolva problema B, atunci, putem folosi solutia de la B ca sa gasim o solutie in timp polinomial si pentru A, pentru ca o compunere de polinomiale este si ea tot o polinomiala, deci nu afecteaza eficienta. Deci, un algoritm eficent H pentru A arata in felul urmator: H – input w – calculeaza f(w), folosind algoritmul polinomial G al lui B, afisam raspunsul lui G pentru f(w).)

- 3CNF formula(O formula boolean de tip CNF are in compozitia sa literali(variabile) legate intre ele prin ∨ care formeaza o clauza, iar clauzele sunt legate intre ele prin operatorul ∧. O formula se afla in 3CNF(3 conjunctive normal form), daca, fiecare clauza are exact 3 literali. Astfel putem considera urmatoarea teorema: problema 3SAT se reduce la problema CLIQUE. Adica, a verifica daca 3SAT = {Φ | Φ este o formula satisfiabila booleana de forma 3CNF} se poate reduce in timp polinomial la problema clicii. Daca un graf are o k-clica este sinonim cu a avea un subgraf complet cu k noduri. Reducerea in timp polinomial de la 3SAT la CLIQUE genereaza un graf neorientat in care nodurile(literalii din 3SAT) sunt impartite in k triplete(clauzele din 3SAT) si exista muchii intre fiecare nod, mai putin intre nodurile din acelasi triplet si nodurile cu valori complementare(x1 si ¬x2). Aceasta reductie merge si vom arata ambele sesuri ale relatiei. Aratam ca Φ este satisfiabila daca si numai daca G are o k-clica. Daca Φ este satisfiabila, atunci in fiecare dintre cele k clauze avem cel putin un literal evaluat la true. Daca avem mai multi, alegem unul arbitrar. Prin operatia ∧ care leaga cele k clauze si, implicit, cei k literali alesi de noi, putem spune cu siguranta ca avem un subgraf complet cu k noduri in G. Mai mult decat atat, doua noduri complementare nu pot fi legate intre ele, pentru ca, nu am fi putut alege atat x cat si ¬x ca variabile evaluate la TRUE din clauzele noastre, iar numarul nodurilor este exact k, pentru ca am ales exact un literal din fiecare clauza, neputand avea in subgraful nostru complet, astfel mai multe noduri corespunzand aceluiasi triplet al grafului, deci, daca Φ e satisfiabila, atunci G contine o k-clica. Pe cealalta parte a demonstratiei, daca avem un G care are o k-clica, atunci avem fix cate un nod din fiecare triplet, corespunzand unui literal evaluat la TRUE din fiecare din cele k clauze. Doua noduri ale k-clicii nu pot fi in acelasi triplet pentru ca nu am avea muchii intre ele daca ar fi. Fiecare nod al grafului corespunde unui literal, iar fiecare triplet unei clauze a lui Φ. Nici nu putem avea doua noduri cu valori complementare legate intre ele din constructia grafului, pentru ca am specificat ca prin constructie, graful nu poate avea muchii intre noduri complementare. Asadar fiecarui nod din k-clica ii corespunde un singur literal din cele k clauze. In concluzie, problema 3SAT se poate reduce la problema CLIQUE.)

- NP-complete(Exista anumite probleme in clasa de probleme NP de complexitatea carora depinde complexitatea tuturor problemelor din NP. Aceste probleme se numesc probleme NP-complete si au o importanta practica si teoretica in stiinta. La nivel teoretic, o persoana care vrea sa demonstreze ca P!=NP poate arata ca orice problema din NP are nevoie de un algoritm mai mult decat polinomial ⬄ o problema din NP nu are solutie in timp polinomial. Cineva care vrea sa arate P=NP poate incerca sa gaseasca o solutie in timp polinomial a unei probleme NP-complete ⬄ orice problema din NP are o solutie polinomiala. La nivel practic, clasa de probleme NP-complete ofera un indiciu destul de important ca poate e inutil sa ne batem capul in a gasi algoritmi polinomiali inexistenti si ca faptul ca o problema se afla in clasa problemelor NP-complete inseamna ca cel mai probabil nu exista o solutie in timp polinomial pentru ea. Spunem ca un limbaj B este NP-complete daca este in NP si daca toate celelalte limbaje din NP se reduc in timp polinomial la B. Daca avem un limbaj B NP-complete si B este in P, atunci P=NP. Daca avem un limbaj A care se reduce in timp polinomial la B, iar A este NP-complete, atunci si B este NP-complete. Suna foarte bine, dar partea grea acum ar fi sa gasim prima problema NP-complete pentru a putea sa ne folosim de ea mai departe.)

- Teorema Cook-Levin(SAT e NP-complete)(In anii 1970, Stephen Cook si Leonid Levin au gasit in aproximativ acelasi moment de timp, dar in parti diferite ale lumii, prima problema NP-complete si anume SAT problem. Pentru a arata ca SAT este in clasa NP-complete trebuie mai intai sa aratam ca este in NP. Acest lucru este evident, deoarece o masina turing nedeterminista poate ghici assignment-ul unei formule si sa-l accepte daca satisface Φ. Apoi, trebuie sa aratam ca orice limbaj A din NP se reduce la SAT. Fie N o masina turing nondeterminista care decide A in timp n^k. Fie un tablou de dimensiuni n^k x n^k unde fiecare linie a tabloului reprezinta configuratia benzii masinii turing N pe un anumit branch al computatiei pe inputul w. Un tablou este in stare de accept daca pe cel putin una dintre liniile sale se gaseste o stare de accept. Astfel, problema de a decide daca N accepta w este echivalenta cu a decide daca exista un tablou in stare de accept al lui N pe w. Trebuie sa descriem reducerea in timp polinomial f a lui A la SAT. Descriem variabilele lui Φ. Fiecare celula poate contine unul din simbolurile din C=Q∪Γ∪{#}, unde Q este multimea starilor si este Γ alfabetul benzii. De asemenea, trebuie sa descriem si functia Φ = Φcell ∧ Φstart ∧ Φmove ∧ Φaccept. Φcell = Si pentru i si j de la 1 la n^k ((Sau pentru s din C Xijs) si (Si pentru s,t din C cu s!=t ¬Xijs sau ¬Xijt)) se refera la ce se poate alfa in fiecare celula, Φstart = X11# ∧ X12q0 ∧ X13w1 ∧ ... ∧ X1n+2w ∧ X1n+3\_ ∧ ... ∧ X1n^k-1\_ ∧ X1n^k#) se refera la prima configuratie, cea de start, a TM N, Φaccept = SAU pentru i,j de la 1 la n^k Xijq\_accept se refera la faptul ca in macar o celula din tablou ar trebui sa fie o stare e accept si Φmove = SI pentru i,j de la 1 la n^k fereastra (i,j) este legala. O fereastra (i,j) este legala daca respecta functia de tranzitie a NTM N. Putem rescrie Φmove = SAU pentru a1,...,a6 fereastra legala (Xij-1a1 ∧ Xija2 ∧ Xij+1a3 ∧ Xi+1j-1a4 ∧ Xi+1ja5 ∧ Xi+1j+1a6) si se refera ca orice fereastra(o portiune de 2x3 din tablou trebuie sa respecte funtia de tranzitie a NTM N. Analizam complexitatea fiecarei bucati din Φ. Mai intai observam ca avem n^2k \* l variabile, unde l nu depinde de n ci doar de masina turing nondeterminista N, deci putem spune ca avem O(n^2k) varibile pentru Φcell. Φstart contine doar prima linie din tablou deci putem spune ca are complexitatea O(n^k). Φmove si Φaccept se refera la intreg tabloul, deci si ele au complexitatea O(n^2k). In concluzie, Φ are complexitatea O(n^2k), adica polinomiala, suficienta pentru demonstratia noastra. Deci, putem spune ca avem o reductie polinomiala de la inputul w la Φ, asadar orice limbaj A din NP se reduce la SAT => SAT este NP-complete.)

- 3SAT e NP-complete(Pentru a arata ca alte probleme sunt NP-complete ne putem folosi de ce am demonstrat mai sus, dar, de obicei, este preferata forma particulara a problemei SAT si anume 3SAT. Pentru a folosi 3SAT in demonstratia altor probleme ca fiind NP-complete trebuie sa aratam ca si 3SAT este NP-complete. Acest lucru este destul de usor, ne uitam la demonstratia lui SAT si observam ca Φ este aproape intr-o forma 3CNF. Contine literali legati prin ∨ care formeaza clauze, clauze legate prin ∧ intre ele. Ca sa o aducem 100% in 3CNF-form, luam fiecare componenta a lui Φ si, daca contin clauze cu un literal sau doi, duplicam unul dintre ei pana ajungem la 3 literali pe clauza, iar in cazul in care o clauza are mai mult de 3 literali, o spargem in mai multe clauze, folosind literali de legatura – ex: (x1 ∨ x2 ∨ x3 ∨ x4) = (x1 ∨ x2 ∨ z) ∧ (¬z ∨ x3 ∨ x4). Astfel, demonstratia ca 3SAT este NP-complete este evidenta.)