Implementarea Algoritmilor de rezolvare SAT: Rezolutie, DP, DPLL In Limbajul C, compararea si analiza acestora

Robert Bosca
Facultatea de Matematica si Informatica
Universitatea de Vest din Timisoara
robert.bosca05@e-uvt.ro

12 mai 2025

Rezumat

In aceasta lucrare, voi prezenta implementarea algoritmiilor de Rezolutie, Davis-Putman si Davis-Putman-Logemann-Loveland in limbajul de programare C, si voi testa fiecare algoritm folosind formule logice de diferite dimensiune.

Cuprins

1 Introducere

Problema satisfiabilitatii (SAT) este una dintre cele mai importante probleme in informatica. Pe scurt, SAT verifica daca pentru o formula logica exista macar o combinatie de asignari de literali cu Adevarat sau Fals astfel incat formula sa fie adevarata.

Problema SAT a fost automatizata pentru prima data in anii 60' si este folosita in numeroase domenii in ziua de astazi.

Pentru automatizare sunt folositi diferiti algotimi de rezolvare SAT. Printre primii si cei mai popular algoritmi, sunt cei discutati in aceasta lucrare, Rezolutie, Davis-Putman, Davis-Putman-Logemann-Loveland.

Voi implementa algoritmii in limbajul C, ii voi testa corectitudinea implementarii, performanta, si ii voi compara.

Intregul cod al implementari este scris si planifica de catre mine.

2 Prezentarea Algoritmilor

Algoritmii de Rezolutie, D.P. si D.P.L.L. performa operatii asupra unei functii logice in forma normala conjunctiva, adica, o conjunctie de functii (clauze), care reprezinta o disjunctie de literali, (exemplu: $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_4)$ unde $\wedge$ este chivalent cu si, $\vee$ echivalent cu sau, si $\neg$ reprezinta negatia).

2.1 Rezolutia

Rezolutia este o regula in logica care permite simplificarea a doua clauze care contine literali complementari (un literal este negat si celalalt nu este).

In situatia in care doua clauze contin literali complementari x si $\neg x$, acestea se pot rescrie ca clauza tuturor literalilor din cele doua clauze precedente cu exceptia literalilor x si $\neg x$. De exemplu, daca avem clauzele $(A \vee B \vee C) \wedge (\neg A \vee D)$ dupa rezolutie vom avea $(B \vee C \vee D)$.

Rezolutia poate fi folosita pentru a gasi satisfiabilitatea unei formule, prinal alegerea unui literal x si rezolutionarea tuturor clauzelor care contin x sau $\neg x$ din formula pana cand exista o clauza fara literal, insemnand ca formula e nesatisfiabila, sau nu mai exista nicio clauza, insemnand ca formula e satisfiabila.

2.1.1 Pseudocod Rezolutie

Mai jos este pseudocodul algoritmului de rezolutie pentru rezolvarea SAT:

1: Rezolutie

```
While True
    For each c1 ∈ Functie If x ∈ c1 then
        For each c2 ∈ Functie If ¬x ∈ c2 then
            Add Rezolutie(c1, c2) to Functie;
    For each c ∈ Functie If x ∈ c or ¬x ∈ c then
        Remove c from Functie;
    If gol(c) then
        return 0
    If dimensiune(Functie) = 0 then
        return 1
```

2.1.2 Exemplu Functionare

Pentru clauzele:

1. $(A \vee B \vee \neg C)$
2. $(D \vee B \vee C)$
3. $(E \vee \neg B \vee C)$

Algoritmul va proceda astfel:

1. Alegem literalul B
2. $Rezolutie(1, 3) \rightarrow (A \vee E \vee \neg C \vee C)$
3. $Rezolutie(2, 3) \rightarrow (D \vee E \vee C)$
4. $Sterge(1); Sterge(2); Sterge(3)$

Noua functie va fi:

1. $(A \vee E \vee \neg C \vee C)$
2. $(D \vee E \vee C)$

Alegem literalul C

1. $Rezolutie(1, 2) \rightarrow (A \vee E \vee D)$

Noua functie va fi:

1. $(A \vee E \vee D)$

Alegem literalul A

1. $Rezolutie(1, 1) \rightarrow GOL$

Ceea ce inseamna ca functia e satisfiabila.

2.2 Algoritmul Davis-Putman

Algoritmul Davis-Putman a fost dezvoltat in 1960 de catre Martin Davis si Hilary Putman. Acest algoritm are la baza tot rezolutia, insa adauga 3 pasi care reduce significativ numarul de iteratii pentru majoritatea problemelor, acesti pasi sunt:

1. Propagare: In cazul in care exista o clauza cu un singur literal x, se sterg toate clauzele care contin x si se sterge literalul $\neg x$ din toate clauzele care il contin.
2. Sterge Literalul Pur: Daca exista un literal x care are o singura polaritate in formula, fie x ori $\neg x$, se sterg toate clauzele din functie care il contin.
3. Eliminare Tautologie: In cazul in care o clauza contine o tautologie, adica contine ambele polaritati ale unui literal, se sterge clauza respectiva.

acesti pasi se executa la fiecare iteratie, deci, pseudocodul devine:

2.2.1 Pseudocod D.P.

2: Davis-Putman

```
While True
    Propagare;
    While ∃ clauza unitara
        literal unitar ← clauza unitara;
        For each c ∈ Functie If literal unitar ∈ c then
            Remove c from Functie;
    Eliminare Literali Puri;
    While ∃ literal pur
        For each c ∈ Functie If literal pur ∈ c then
            Remove c from Functie;
    Eliminare Tautologii;
    For each c ∈ Functie If x ∧ ¬x ∈ c then
        Remove c from Functie;
    Rezolutie;
    For each c1 ∈ Functie If x ∈ c1 then
        For each c2 ∈ Functie If ¬x ∈ c2 then
            Add Rezolutie(c1, c2) to Functie;
    For each c ∈ Functie If x ∈ c or ¬x ∈ c then
        Remove c from Functie;
    If gol(c) then
        return 0
    If dimensiune(Functie) = 0 then
        return 1
```

2.2.2 Exemplu Functionare

Problema de la Rezolutie,

1. $(A \vee B \vee \neg C)$
2. $(D \vee B \vee C)$
3. $(E \vee \neg B \vee C)$

Este rezolvata mult mai rapid de catre D.P. datorita pasilor adaugati.

Algoritmul va proceda astfel:

1. nu gaseste nicio clauza unitara, deci trece la identificarea literalilor puri
2. identifica literalul pur A
3. Remove $(A \vee B \vee \neg C)$ from Functie

4. identifica literalul pur D
5. Remove $(D \vee B \vee C)$ from Functie
6. identifica literalul pur E
7. Remove $(E \vee \neg B \vee C)$ from Functie
8. Functia ramane fara clauze, deci trece peste eliminarea tautologiei si rezolutie, si returneaza satisfiabil

2.3 Algoritmul Davis-Putman-Logemann–Loveland

Algoritmul Davis-Putman-Logemann–Loveland a fost dezvoltat in 1961-1962 de catre Davis Putman, George Logemann si Donald W. Loveland. Acesta inlocuieste Rezolutia cu o cautare recursiva, pentru a evita problema principala a algoritmului D.P., adica, cresterea exponentiala a clauzelor, despre care voi vorbi mai in detaliu la partea de experimentare.

D.P.L.L. face propagarea, eliminarea de literali unitari si eliminarae tautologiilor, la fel ca D.P., dar in loc sa faca rezolutia dupa, acesta alege un literal x, ii atribuie valorea de adevarat, sterge toate clauzele care il contin si toti literali $\neg x$ din clauze. In cazul in care acest proces nu rezulta in satisfiabilitate/nesatisfiabilitate, se alege un alt literal si se apeleaza recursiv D.P.L.L de doua asupra functiei rezultate, o data cu literalul nou ales, si o data cu literalul negat.

2.3.1 Pseudocod D.P.L.L

3: Davis-Putman-Logemann–Loveland

```
Function DPLL(Functie):
    Propagare;
    While ∃ clauza unitara
        literal unitar ← clauza unitara;
        For each c ∈ Functie If literal unitar ∈ c then
            Remove c from Functie;
    Eliminare Literali Puri;
    While ∃ literal pur
        For each c ∈ Functie If literal pur ∈ c then
            Remove c from Functie;
    Eliminare Tautologii;
    For each c ∈ Functie If x ∧ ¬x ∈ c then
        Remove c from Functie;
    If Functie = ∅ then
        return 1
    If ∃c ∈ Functie : gol(c) then
        return 0
    Choose literal l from Functie;
    For each c ∈ Functie If l ∈ c then
        Remove c from Functie;
    If ¬l ∈ c then
        Remove ¬l from c;
    return DPLL(Functie) ∨ DPLL(Functie ∪¬l)
```

2.3.2 Exemplu Functionare

Pentru Urmatoarea functie logica:

1. $(A \vee B)$
2. $(\neg A \vee C)$
3. $(\neg B \vee \neg A)$

Algoritmul DPLL va proceda astfel:

1. Se gaseste literalul pur C si se sterge $(\neg A \vee C)$
2. Se alege literalul A si se incearca ca pozitiv. Functia devine $(\neg B)$
3. $(\neg B)$ e literal unitar, deci se sterge
4. functia are dimensiunea 0, deci se returneaza Satisfiabil.

3 Implementarea algoritmiilor in limbajul C

3.1 De ce C?

Limbajul C a fost ales pentru performanta. Intrucat C este proiectat pentru inteligenta la nivel redus, operatiile se vor realiza la o viteza significativ mai mare decat ar fi pentru alte limbaje cu mai multe abstractizari precum Python sau Java. Acest aspect va fi crucial cand voi testa functii mari.

3.2 Structura Generala a implementarii

Pentru implementare, am impartit algoritmii in mai multe parti:

1. structura functiei si a clauzelor
2. Pentru Algoritmul de rezolutie:
   (a) functia de stergere de literali
   (b) functia de stergere de clauza
   (c) functia de adaugare a unei caluze in functie;
   (d) functia de rezolutie
3. Pentru D.P.
   (a) functia de propagare
   (b) functia de eliminare a literalului pur
   (c) functia de eliminare a tautologiilor.
4. Pentru D.P.L.L
   (a) functia recursiva de atribuire de valori, D.P.L.L.

Librariile folosite sun stdio.h, string.h, stdlib.h si stdbool.h.
Mediul de dezvoltare este Visual Studio Code si compilatorul GCC.
Sistemul de operare este Debian Linux/GNU

3.3 Structura Functiei logice si a Clauzelor

In program, literalii vor fi reprezentati de numere intregi, spre exemplu, A = 1, B = 2, etc.. Daca literalul este negat, acesta va fi reprezentat de catre un numar negativ, $\neg A$ =-1, $\neg B = -2$, etc.

O clauza este reprezentata de o structura cu doua componente, un numar marime care arata numarul de literali din clauza, si un tablou de numere intregi (literali). De asemenea exista si un numar pentru dimensiunea functiei, care se initiaza dupa initializarea acesteia.

Functia logica este reprezentata de un tablou de clauze.

```
struct clauza
{
    int marime;
    int *literali;
};
```

3.4 Functia de stergere a unui literal intr-o clauza

Functia de stergere pentru literal primeste parametrii:

1. tabloul functiei, lista clauze
2. Clauza din care se vrea sters literalul, clauza
3. Literalul care se doreste sters, literal

si nu returneaza nimic.

Functia creaza un tablou de numere, si parcurge in totalitate clauza. Pentru fiecare element, daca este diferit de literalul, se adauga la noul tablou

Dupa terminarea clauzei, fiecare element al acesteia este inlocuit cu cel echivalent din noul tablou, si marimea acesteia devine cea a tabloului.

La final literalul este considerat sters din clauza.

Complexitate O(marime-clauza)

3.5 Functia de Stergere a unei clauze din functie

Functia de stergere pentru clauza primeste parametrii:

1. tabloul functiei, lista clauze
2. Clauza din care se vrea sters literalul, clauza
3. Dimensiunea , dimensiune

Intrucat ordinea clauzelor nu este importanta in acesti algoritmi, in loc de a sterge clauza dupa modul traditional de stergere dintr-un tabou, adica mutarea fiecarui element cu o pozitie in fata, Functia de stergere pur si simplu muta ultima clauza din lista pe locul clauzei care se doreste stearsa, si scade numarul dimensiune cu 1.

Complexitate O(1)

3.6 Functia de Adaugare

Functia de adaugare de clauza primeste parametrii:

1. Tabloul functiei, lista clauze
2. Dimensiunea , dimensiune
3. Un tablou de numere intregi, clauza

Functia realoca multimea de clauze si o mareste cu 1, dupa muta clauza care se doreste adaugata pe ultimul loc.

Complexitate O(1)

3.7 Functia de rezolutie

Functia de rezolutie primeste parametrii:

1. Tabloul functiei, lista clauze
2. Dimensiunea , dimensiune
3. literalul asupra caruia se va face rezolutia, literal

Functia cauta in lista de clauze fiecare clauza care contine literalul, si pentru fiecare clauza gasita cauta in lista o clauza ce contine literalul cu polaritatea opusa

Complexitatea este O(dimensiune * marime clauza). Este cea mai costisitoare functie.

3.8 Functia de Propagare

Functia de rezolutie primeste parametrii:

1. Tabloul functiei, lista clauze
2. Dimensiunea , dimensiune

Functia parcurge lista de clauze si cauta o clauza cu dimensiunea 1. In cazul in care aceasta se gaseste, se parcurge iarasi toata lista si se elimina toate clauzele care contin unicul literal din clauza unitara, si sterge toti literali de polaritate opusa din lista.

Complexitatea O($dimensiune/2$)

3.9 Functia de eliminare a literalilor puri

Pentru aceasta functie am implementat o alta functie pentru gasirea literalilor puri in lista:

1. Tista de clauze, lista clauze
2. Dimensiune, dimensiune
3. Tabloul literalilor puri, puri
4. Dimensiunea tabloului de literali puri

Aceasta functie se foloseste de o structura de date literal

```
    struct literal
{
    unsigned char POZ;
    unsigned char NEG;
};
```

Se creaza o lista de structuri

Functia parcurge toti literalii din fiecare clauza, si in cazul in care este pozitiv, structura de pe lista de structuri cu pozitia literalului devine va avea POZ=1, in caz contrar NEG=1;

Se parcurge toata lista de structuri si daca un element are doar POZ=1 sau doar NEG=1, numarul pozitiei acelui element este adaugat in tabloul puri.

Dupa ce avem tabloul de literali puri, trecem la functia de stergere de literali puri, care pentru fiecare literal pur din lista cauta si sterge clauzele care il contin.

Complexitate O(dimensiune * numar clauze)

3.10 Algoritmul de Stergere al Tautologiilor

Algoritmul de Stergere al Tautologiilor are

1. Tista de clauze, lista clauze
2. Dimensiune, dimensiune

Aceasta functie, la fel ca cea precedenta, se foloseste de stuctura

```
    struct literal
{
    unsigned char POZ;
    unsigned char NEG;
};
```

Functia la fiecare literal reseteaza lista de structuri, si in cazul in care intr-o clauza o structura are POZ=1 si NEG=1, sterge clauza, intrucat asta inseamna ca e o tautologie.

Complexitate O(dimensiune * marime clauza)

3.11 Functia D.P.L.L.

Intrucat D.P.L.L este recursiv, functiile de propagare, eliminare literali puri si eliminare tautologie.

Functia primeste parametrii:

1. lista de clauze, lista clauze
2. dimensiunea, dimensiune
3. literalul pentru care se face procedura DPLL, literal

Functia executa propagarea, eliminarea de literali puri si de tautologi. Daca este satisfiabila (dimensiune este 0) sau nesatisfiabila (exista o clauza goala), se returneaza 1 respectiv 0

Daca nu se returneaza niciuna, se sterg toate clauzele care contin literalul si toti literalii cu polaritate opusa din lista de clauze.

Dupa, se alege un alt literal, si se apeleaza recursiv functia cu o copie a listei de clauze si cu noul literal.

Daca acea functie returneaza 0, o mai apelam o data cu literalul de polaritate opusa.

Functia curenta returneaza rezultatul ultimei functii.

Complexitate O($2^{dimensiune}$)

4 Implementarea Functiilor in algoritmi

Algoritmul de Rezolutie foloseste doar functia Rezolutie, si verifica pentru fiecare iteratie daca lista de clauze este satisfiabila/nesatisfiabila

```
while(true)
   {
       if(dimensiune==0)
       {
           goto Satisfiabil;
       }
       for(int i=0;i<dimensiune;i++)
       {
           if(lista_clauze[i].marime==0)
           {
               goto Nesatisfiabil;
           }
       }
       rezolutie(lista_clauze,&dimensiune,cel_mai_frecvent(lista_clauze,dimensiune));
       afisare_clauze(lista_clauze,dimensiune);
   }
```

Algoritmul D.P. apeleaza propagarea, stergerea de literali puri si stergerea de tautologi pe langa rezolutie.

```
while(true)
    {
        while(propagare(lista_clauze,&dimensiune));
        sterge_tautologiile(lista_clauze,&dimensiune);
        while(stergere_literali_puri(lista_clauze,&dimensiune));
        if(dimensiune==0)
        {
            goto Satisfiabil;
        }
        for(int i=0;i<dimensiune;i++)
        {
```

```
            if(lista_clauze[i].marime==0)
            {
                goto Nesatisfiabil;
            }
        }
        rezolutie(lista_clauze,&dimensiune,cel_mai_frecvent(lista_clauze,dimensiune));
        afisare_clauze(lista_clauze,dimensiune);
    }
```

Algoritmul D.P.L.L. apeleaza doar functia DPLL cu literalul ales fiind primul din prima clauza din lista de clauze.

```
if(DPLL(lista_clauze,dimensiune,lista_clauze[0].literali[0]))
    {
        printf("satisfiabil\n");
    }
    else
    {
        printf("nesatisfiabil\n");
    }
```

4.1 Utilizare

Algoritmii sunt impartiti intree ei, fiecare fiind un program separat, dar avand aceasi lista de clauza, un fisier txt "$Clauze_DP.txt$".$IntimpulrulariiprogramulreturneazatimpulsispatiulRAM$

4.2 Testarea, Analiza si Compararea Algoritmiilor

Pentru testarea voi folosi acelasi lista de valori si voi monitoriza timpul de executie si memoria RAM utilizata. Sistemul pe care voi rula aceste teste este un laptop Lenovo V15 G4 cu 6 Gb RAM si procesor i5-13420H.

4.2.1 Primul test

Pentru primul test am clauza :

C1	1	2	-3
C2	-2	3	4
C3	1	-4	5

Toti algoritmii au returnat acelasi rezultat, satisfiabil.

Cand vine vorba de timpul utilizat, Algoritmul D.P a fost cel mai incet.

D.P si D.P.L.L. au avut constant in jur de 150-190 de tick-uri, pe cand Rezolutia a avut intre 3000-4000.

D.P		Rezolutia		D.P.L.L.	
Memorie	Timp	Memorie	Timp	Memorie	Timp
34632	174	34368	3018	34344	172

Tabela 1: Memorie Utilizata si Timp de Executie Test 1

4.2.2 Al 2-lea test

Test 2):

(1)	$1 \vee 2 \vee 3$	(13)	$1 \vee 4 \vee 7$
(25)	$1 \vee 5 \vee 9$	(37)	$10 \vee 1 \vee 2$
(2)	$\neg 1 \vee \neg 2 \vee 3$	(14)	$\neg 1 \vee \neg 4 \vee 7$
(26)	$\neg 1 \vee \neg 5 \vee 9$	(38)	$\neg 10 \vee \neg 1 \vee 2$
(3)	$\neg 1 \vee 2 \vee \neg 3$	(15)	$\neg 1 \vee 4 \vee \neg 7$
(27)	$\neg 1 \vee 5 \vee \neg 9$	(39)	$\neg 10 \vee 1 \vee \neg 2$
(4)	$1 \vee \neg 2 \vee \neg 3$	(16)	$1 \vee \neg 4 \vee \neg 7$
(28)	$1 \vee \neg 5 \vee \neg 9$	(40)	$10 \vee \neg 1 \vee \neg 2$
(5)	$4 \vee 5 \vee 6$	(17)	$2 \vee 5 \vee 8$
(29)	$2 \vee 6 \vee 7$	(41)	$10 \vee 4 \vee 5$
(6)	$\neg 4 \vee \neg 5 \vee 6$	(18)	$\neg 2 \vee \neg 5 \vee 8$
(30)	$\neg 2 \vee \neg 6 \vee 7$	(42)	$\neg 10 \vee \neg 4 \vee 5$
(7)	$\neg 4 \vee 5 \vee \neg 6$	(19)	$\neg 2 \vee 5 \vee \neg 8$
(31)	$\neg 2 \vee 6 \vee \neg 7$	(43)	$\neg 10 \vee 4 \vee \neg 5$
(8)	$4 \vee \neg 5 \vee \neg 6$	(20)	$2 \vee \neg 5 \vee \neg 8$
(32)	$2 \vee \neg 6 \vee \neg 7$	(44)	$10 \vee \neg 4 \vee \neg 5$
(9)	$7 \vee 8 \vee 9$	(21)	$3 \vee 6 \vee 9$
(33)	$3 \vee 4 \vee 8$	(45)	$10 \vee 7 \vee 8$
(10)	$\neg 7 \vee \neg 8 \vee 9$	(22)	$\neg 3 \vee \neg 6 \vee 9$
(34)	$\neg 3 \vee \neg 4 \vee 8$	(46)	$\neg 10 \vee \neg 7 \vee 8$
(11)	$\neg 7 \vee 8 \vee \neg 9$	(23)	$\neg 3 \vee 6 \vee \neg 9$
(35)	$\neg 3 \vee 4 \vee \neg 8$	(47)	$\neg 10 \vee 7 \vee \neg 8$
(12)	$7 \vee \neg 8 \vee \neg 9$	(24)	$3 \vee \neg 6 \vee \neg 9$
(36)	$3 \vee \neg 4 \vee \neg 8$	(48)	$10 \vee \neg 7 \vee \neg 8$
(49)	$\neg 10 \vee \neg 10 \vee \neg 10$	(50)	$10 \vee 10 \vee 10$

La testul 2, iarasi toti algoritmii au returnat rezultatul corect, nesatisfiabil.

Rezolutia a durat cel mai mult, 3.4 milioane de ticks, pe cand DPLL si D.P au durat 423 tiks respectiv 490 ticks.

D.P	Rezolutia	D.P.L.L.
Timp	Timp	Timp
490	3.4 mil	423

Tabela 2: Memorie Utilizata si Timp de Executie Test 1

4.3 Al 3-lea test

Pentru al 3-lea test voi folosi o lista de clauze de pe site-ul https://www.cs.ubc.ca/ hoos/SATLIB/benchm.html mai precis uuf50-05.cnf.

La aceasta lista, Rezolutia si D.P. nu au putut afla daca functia este satisfiabila, intrucat dimensiunea a sarit peste pragul maxim. pentru Rezolutie a durat 34 mil ticks si pentru D.P 5 mil.

D.P.L.L. in schimb a reusit sa returneze nesatisfiabil in 59983 ticks.

D.P	Rezolutia	D.P.L.L.
Timp	Timp	Timp
5 mil.	34 mil.	59980

Tabela 3: Memorie Utilizata si Timp de Executie Test 1

4.4 Test 4, doar D.P.L.L.

pentru testul 4 am utilizat uuf250-01.cnf

la inceput memoria utilizata nu a crescut, insa dupa 3 milioane de ticks memoria utilizata a inceput sa creasca significativ.

Memoria a continuat sa creasca pana la 13 milione kb, unde s-a stabilizat.

Algoritmul a ramas intre 13-15 milioane kb timp de 3 minute pana cand l-am oprit.

5 Comparatie cu Literatura

Comportamentul mentionat anterior nu este surprinzator, asa cum au mentionat Putman, loveland si Logemann in lucrarea in care au introdus D.P.L.L., Principala problema a rezolutiei este "explodarea de clauze", adica, aplicand rezolutia, nr de clauze creste exponential dimensiunii problemei.

De asemenea, nu este surprinzator faptul ca D.P.L.L. nu are aceasta problema, deoarece acesta a fost creat cu scopul de a scapa de aceasta problema.

6 Concluzie

Ca urmare a implementarii, putem observa faptul ca dintre acesti 3 algoritmi, D.P.L.L este singurul care este folositor pentru rezolvarea problemelor mai mari SAT, intrucat dupa ce isi aloca memoria necesara, mai are nevoie doar de timp si eventual v-a ajunge la rezultatul corect.

6.1 Direcii Viitoare

Algoritmii de Rezolutie si D.P. pot fi in continuare imbunatatiti. Desi acestia prin natura lor nu pot trece peste explozia de clauze, se poate reduce efectul acesteia prin selectarea literalilor mai bine.

7 Bibliografie

codul sursa poate fi gasit pe: https://github.com/BoscaRobert/Satisfiabilitatea

Davis, Martin; Putnam, Hilary (1960). "A Computing Procedure for Quantification Theory" https://dl.acm.org/doi/10.1145/321033.321034

Davis, Martin; Logemann, George; Loveland, Donald (1962). "A Machine Program for Theorem Proving" https://archive.org/details/machineprogramfo00davi