Verificare Formala, WS 2023/2024. Lab 1[1]

Obiective

Folosirea unui SAT *solver* pentru a decide satisfiabilitatea (SAT) unor probleme.

1 Preliminarii

Un SAT *solver* este un program care rezolva in mod automat problema de satisfiabilitate a unui formule booleene (*Boolean satisfiabiliy problem*).

Problema de satisfacere (SAT problem) poate fi rezolvata folosind tabelul de adevar (truth table):

1. *Cazul 1*: Pentru a verifica daca formula F este satifiabila, realizam tabelul de adevar pentru F. Daca exista un rand in care *true* apare ca valoare pentru F, atunci F este satifiabila. In caz contrar, F este nesatifiabila.

2. *Cazul 2*: Pentru a verifica daca $F_1, ..., F_n \models G$ [2], se verifica satisfiabilitatea lui $F_1 \wedge .... \wedge F_n \wedge \neg G$. Daca este nesatisfiabila, atunci $F_1, ..., F_n \models G$, altfel $F_1, ..., F_n \not\models G$.

Care este complexitatea unui astfel de algoritm? Putem face mai bine decat $\mathcal{O}(2^n)$? (n este numarul de atomi din formula propozitionala).

SAT a fost prima problema aratata a fi NP-complete [1]: toate problemele din clasa NP pot fi rezolvate prin traducerea lor (in timp polinomial) in SAT. Prin urmare, dac am putea cumva sa construim un rezolvator (solver) rapid pentru SAT, acesta ar putea fi folosit pentru a rezolva multe alte probleme. In teorie, acest lucru pare dubios, deoarece se stie ca problemele din NP iau timp exponenial in cel mai rau caz. In mod remarcabil, solverele moderne SAT sunt foarte rapide de cele mai multe ori!

O formula trebuie sa fie n forma normala conjunctiv (CNF) pentru a putea fi tratata de un solver SAT.

Example 1 *Fie formula urmatoare (in CNF):*

$$\begin{aligned}(\neg A \vee \neg B \vee E) \wedge (\neg E \vee A) \wedge (\neg E \vee B) \wedge \\ (\neg C \vee F) \wedge (\neg F \vee C) \wedge \\ (\neg D \vee \neg E \vee G) \wedge (\neg G \vee D) \wedge (\neg G \vee E) \wedge \\ (\neg E \vee \neg F \vee H) \wedge (\neg H \vee E) \wedge (\neg H \vee F) \wedge \\ (G \vee H \vee \neg I) \wedge (\neg H \vee E) \wedge (\neg H \vee F) \wedge \\ I\end{aligned}$$

Notatii alternative sunt:

1. *Alternativa* 1*:*

[1] Bazat pe: Lecture notes SAT: Theory and Practice by Clark Barrett from VSTA 2008 Summer School

[2] $A \models B$ inseamna ca pentru toate interpretarile $I \in \mathcal{I}$, daca A este adevarata in I, atunci si B este adevarata in I

Example 2 *Pentru exemplul de mai sus avem:*

$$\begin{aligned}(A' + B' + E)(E' + A)(E' + B)\\ (C' + F)(F' + C)\\ (D' + E' + G)(G' + D)(G' + E)\\ (E' + F' + H)(H' + E)(H' + F)\\ (G + H + I')(H' + E)(H' + F)\\ I\end{aligned}$$

2. *Alternativa* 2: Standardul DIMACS: Fiecare variabila este reprezentata de un numar intreg pozitiv. Un numar intreg negativ inseamna variabila negata. Clauzele sunt notate ca secvente de numere intregi separate prin spatiu. Numarul 0 (zero) termina clauza.

Example 3 *Pentru exemplul de mai sus avem:*

$$\begin{array}{llll} -1\ -2\ 5\ 0 & -5\ 1\ 0 & -5\ 2\ 0\\ & -3\ 6\ 0 & -6\ 3\ 0\\ -4\ -5\ 7\ 0 & -7\ 4\ 0 & -7\ 5\ 0\\ -5\ -6\ 8\ 0 & -8\ 5\ 0 & -8\ 6\ 0\\ 7\ 8\ -9\ 0 & -8\ 5\ 0 & -8\ 6\ 0\\ & & 9\ 0 \end{array}$$

Primele solvere SAT s-au bazat pe metoda Davis-Putnam (vezi cursuri). In prezent, solverele SAT, care sunt extrem de rapide, fiind capabile sa rezolve formule cu mai mult de 1000K clauze si zeci de mii de variabile, se bazeaza pe:

- Algoritmul Conflict-Driven Clause Learning, care poate fi vazut ca o varianta moderna a algoritmului DPLL (de exemplu, solverul SAT Chaff)
- algoritmi stochastici de cautare locala (de ex. SAT solver WalkSAT).

Vom folosi un rezolvator online SAT, de ex. `https://www.msoos.org/2013/09/minisat-in-your-browser/`. Experimentarea cu soluii SAT mai avansate (consultai http://www.satcompetition.org/ pentru cei mai competitivi rezolvatori SAT) este puternic incurajata.

Ar putea solverele SAT sa ajute la rezolvarea problemei de lunga durata n informatica teoretica textsc P = NP? Mai multe detalii aici:
`http://www.se-radio.net/2017/07/se-radio-episode-298-moshe-vardi-on-p-versus-np/`
.

2 Exercitii

2 p Problema de satisfacere a circuitului (circuit satisfiability problem – CSP) este problema de a determina daca un anumit circuit boolean are o atribuire a intrarilor sale care face iesirea adevarata. Rezolvati CSP pentru circuitul din Figura 2 folosind un solver SAT.

5 p. Folosind ideile din lucrarea [2] (codificarea minimala - minimal encoding, pag. 4) sau din [3], utilizati un solver SAT pentru a rezolva problema Sudoku.

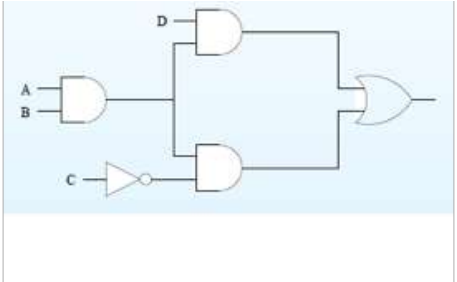

Figure 1: Exemplu de circuit

3 p Folosind solverul CaDiCaL (`https://github.com/arminbiere/cadical`, stable release 1.8.0 in data de 10.10.2023) rulati o formula creata de voi cu 5 variabile care au mai mult de 2 literali. Formula trebuie sa fie nesatisfiabilila. Salvati rezultatul intr-un fisier. Apoi, generati si certificatul prin care se arata nesatisfiabilitatea (unsatisfiability of the problem) (vz. `https://satcompetition.github.io/2023/certificates.html`). Certificatul arata formulele intermediare, pana la rezultat, ce au fost generate de catre SAT solver. Certificatul salvat intr-un fisier `proof.out` se poate apoi verifica daca a fost corect generat (optional).

References

[1] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ACM, p. 151158.

[2] Lynce, I., and Ouaknine, J. Sudoku as a SAT Problem. In *PROCEEDINGS OF THE 9 TH INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE AND MATHEMATICS, AIMATH 2006, FORT LAUDERDALE* (2006), Springer.

[3] Weber, T. A SAT-based Sudoku solver. In *LPAR* (2005), pp. 11–15.