Mathematical Logic Project
Implementation of DPLL algorithm

Mina Nayernia

November 2023

1 Introduction

A computer program that searches for a model for a propositional formula is called a SAT Solver. This is a highly developed area of research in computer science because many problems in computer science can be encoded in propositional logic so that a model for a formula is a solution to the problem.

In logic and computer science, the Davis–Putnam–Logemann–Loveland (DPLL) algorithm is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formula in conjunctive normal form, i.e. for solving the CNF-SAT problem.

It was introduced in 1961 by Martin Davis, George Logemann and Donald W. Loveland and is a refinement of the earlier Davis–Putnam algorithm, which is a resolution-based procedure developed by Davis and Hilary Putnam in 1960.

2 fundamentals

2.1 CNF formula

A Conjunctive Normal Form (CNF) formula is a way of representing a logical expression using conjunctions and disjunctions. In CNF, a logical formula is expressed as a conjunction (AND) of one or more clauses, where each clause is a disjunction (OR) of literals. A literal is either a variable or its negation.

2.2 Pure literals

Let S be a set of clauses. A pure literal in S is a literal l that appears in at least one clause of S, but its complement l'c does not appear in any clause of S. In other words, a "pure literal" refers to a variable that always appears with the same polarity (either always positive or always negative) in a given set of clauses. Taking advantage of pure literals can be a helpful strategy in SAT solving. If a pure literal is identified, it can be assigned the value that satisfies its appearance in the clauses, simplifying the formula. The assigned value for a pure literal can then be propagated through the formula, leading to further simplifications and making it easier to determine a satisfying assignment.

2.3 Unit clause and Unit propagation

A unit clause is a type of clause in propositional logic that contains only a single literal. In other words, a unit clause is a clause that has exactly one proposition (variable or its negation) in it. The satisfaction or falsification of the entire clause is then solely determined by the truth value of that single literal.

Unit propagation is a process where the solver identifies and exploits unit clauses in a formula to make new variable assignments, simplifying the problem.

The theorem in this case is: Let { l } $\in$ S, be a unit clause and let S be obtained from S by deleting every clause containing l and by deleting lC from every (remaining) clause.Then S $\approx$ S$'$.

2.4 Resolution

Resolution is a refutation procedure used to check if a formula in clausal form is unsatisfiable by reducing complex problems to simpler ones. The resolution procedure consists of a sequence of applications of the resolution rule to a set of clauses. The rule maintains satisfiability: if a set of clauses is satisfiable, so is the set of clauses produced by an application of the rule. Therefore, if the (unsatisfiable) empty clause is ever obtained, the original set of clauses must have been unsatisfiable.

Algorithm Input: A set of clauses S. Output: S is satisfiable or unsatisfiable. Let S be a set of clauses and define S0 = S. Repeat the following steps to obtain Si+1 from Si until the procedure terminates as defined below:

• Choose a pair of clashing clauses {C1,C2} $\subseteq$ Si that has not been chosen before.

• Compute C = Res(C1,C2) according to the resolution rule.

• If C is not a trivial clause, let Si+1 = Si $\cup$ {C}; otherwise, Si+1 = Si . Terminate the procedure if:

• C is empty clause.

• All pairs of clashing clauses have be resolved.

2.5 Backtracking

Backtracking is a general algorithm used to find all (or some) solutions to a computational problem, particularly in constraint satisfaction problems, combinatorial optimization, and logic programming. It is a systematic, trial-and-error-based search algorithm that explores potential solutions by making choices at each decision point. If a choice leads to an infeasible solution, the algorithm backtracks, undoing the most recent decision and trying a different path.

3 Davis-Putnam Algorithm

Input: A formula A in clausal form. Output: Report that A is satisfiable or unsatisfiable. Perform the following rules repeatedly, but the third rule is used only if the first two do not apply:

• Unit-literal rule: If there is a unit clause l, delete all clauses containing l and delete all occurrences of lc from all other clauses.

• Pure-literal rule: If there is a pure literal l, delete all clauses containing l.

• Eliminate a variable by resolution: Choose an atom p and perform all possible resolutions on clauses that clash on p and $^{-}$p. Add these resolvents to the set of clauses and then delete all clauses containing p or $^{-}$p . Terminate the algorithm under the following conditions:

• If empty clause is produced, report that the formula is unsatisfiable.

• If no more rules are applicable, report that the formula is satisfiable.

4 DPLL Algorithm

Input: A formula A in clausal form. Output: Report that A is unsatisfiable or report that A is satisfiable and return a partial interpretation that satisfies A. The algorithm is expressed as the recursive function DPLL(B,I) which takes two parameters: a formula B in clausal form and a partial interpretation I. It is initially called with the formula A and the empty partial interpretation. DPLL(B,φ)

• Construct the set of clauses B′ by performing unit propagation on B. Construct φ by adding to I all the assignments made during propagation.

• Evaluate B′ under the partial interpretation φ:

– If B′ contains a conflict clause return 'unsatisfiable';

– If B′ is satisfied return φ;

– (otherwise, continue).

• Choose an atom p in B′; choose a truth value val as T or F; I1 is the interpretation φ together with the assignment of val to p.

5 Implementation

The code is written in Python 3, it might not be compatible with previous versions. The problem is defined in CNF form.

- Please write the CNF form problem in a text file in the following way.

For Example: (!B+A+!C)(B+A+!C)(!B+!A+!C)(B)(C) will be written as

‘‘‘

!B A !C

B A !C

!B !A !C

B

C

‘‘‘

- Save it in ”filename” (example problem.txt).
- Run the following command

python3 SATSolver.py –input problem.txt

The DPLL algorithm can be explained by the following pseudocode.

```
solveDpll(cnf):
    while(cnf has a unit clause {X}):
        delete clauses contatining {X}
        delete {!X} from all clauses
    if null clause exists:
        return False
    if CNF is null:
        return True
    select a literal {X}
    cnf1 = cnf + {X}
    cnf2 = cnf + {!X}
    return solveDdpll(cnf1)+solveDpll(cnf2)
```