Serial SAT Solver

Abraham White

March 20, 2020

Description

The Boolean Satisfiability Problem (SAT) is an NP-complete decision problem, where the goal is to find whether there is an assignment of values to boolean variables in a propositional formula such that the formula evaluates to true. For example, determining whether

$$a \wedge \neg a$$

can be true for some value of a is an SAT problem.

Most SAT solvers take their input as a propositional logic formula in conjunctive normal form (CNF), involving variables and the operators *negation* ($\neg$), *disjunction* ($\vee$), and *conjunction* ($\wedge$). A propositional logic formula in CNF is the conjunction of a set of clauses. A *clause* is a disjunction of literals, and a *literal* is a boolean variable A which can be either positive (A), or negative ($\neg A$).

An *interpretation* is a mapping from a CNF formula to the set of truth values $\{\top, \bot\}$ through assignment of truth values to the literals in the formul. SAT solvers use *partial interpretations*, where only some of the literals in the formula are assigned truth values. These variables are replaced with their truth values in the formula and the formula is then simplified using the rules of propositional logic.

The Boolean Satisfiability Problem is the question of whether there exists an interpretation for a formula such that the formula evaluates to $\top$ under this interpretation.

There are two types of SAT solvers: complete, and stochastic. Complete solvers attempt to either find a solution, or show that no solutions exist. Stochastic solvers cannot prove that a formula is unsolvable, but can find solutions for specific kinds of problems very quickly. We are attempting to build a complete SAT solver.

The majority of modern complete SAT solvers are based on a branch and backtracking algorithm called Davis-Putnam-Logemann-Loveland (DPLL), a refinement of the earlier Davis-Putnam algorithm and introduced in 1962 by Martin Davis, George Logemann, and Donald W. Loveland. Many of these solvers add additional heuristics on top of the DPLL algorithm, which can increase efficiency, but adds significant complexity to the implementation. Because of this, we will only focus on the DPLL algorithm and refrain from any heuristical approaches. We use Formalization and Implementation of Modern SAT Solvers [1] as a guide in the implementation of our serial SAT solver.

Implementation

The basic DPLL algorithm can be defined recursively as in Algorithm 1. In the algorithm, $F[l \to \top]$ denotes the formula obtained by replacing the literal l with $\top$ and $\neg l$ with $\bot$ in F. A literal is pure if it occurs in F but its opposite does not. A clause is unit if it contains only one literal. Unfortunately, this recursive algorithm becomes unusable with larger formulae.

The DPLL algorithm consists of two key steps:

1. **Literal Elimination**: If some literal is only seen in pure form, we can immediately determine the truth value for that literal. For instance, if the literal is in the form A, we know that A must be $\top$, and if the literal is in the form $\neg A$, A must be $\bot$. This step occurs on line 6 of the recursive algorithm .

2. **Unit Propogation**: If there is a unit clause then we can immediately assign a truth value in the same way we do for literal elimination. This is done on line 8 of the recursive algorithm.

Algorithm 1 The recursive DPLL algorithm

```
1: function DPLL(F : Formula)
2:     if F is empty then
3:         return SAT
4:     else if F contains an empty clause then
5:         return UNSAT
6:     else if F contains a pure literal l then
7:         return DPLL(F[l → ⊤])
8:     else if F contains a unit clause [l] then
9:         return DPLL(F[l → ⊤])
10:    else
11:        let l be a literal in F
12:        if DPLL(F[l → ⊤]) = SAT then
13:            return SAT
14:        else
15:            return DPLL(F[l → ⊥])
16:        end if
17:    end if
18: end function
```

References

[1] Filip Marić. Formalization and implementation of modern sat solvers. *Journal of Automated Reasoning*, 43(1):81–119, 2009.