SAT Solvers

Ali Ahmadi

Contents

- What is a SAT Solver
- Algorithms
- Input Integration
- DPLL
- CDCL

What is a SAT Solver

- A **SAT solver** is a computer program that aims to solve the **Boolean satisfiability problem.**
- Input: A Boolean expression (We can use a CNF)
- Output: A Boolean to show the selected Boolean expression is satisfiable or not. Also, if the expression is satisfiable we can output an assignment that satisfies the expression.

Why Not A True Table

- In large examples, we are unable to try a million random guesses, to return the correct output.

- Algorithms:
  1) DPLL
  2) CDLL

Input Integration

- We can convert any Boolean expression to CNF using the CNF Algorithm.
- So we can have a CNF as input. (If not, we can convert it to CNF)
- A CNF:

```
AND((OR x1 x2) (OR NOT(x3) x4) (OR NOT(x2) x3))
=> [[1, 2], [-3, 4], [-2, 3]]
```

DPLL

1. "Guess" a variable
2. Find all unit clauses created from the last assignment and assign the needed value
3. Iteratively retry step 2 until there is no change (found transitive closure)
4. If the current assignment cannot yield true for all clauses - fold back from recursion and retry a different assignment
5. If it can - "guess" another variable (recursively invoke and return to 1)

```python
def DPLL(cnf):
    if not cnf:
        return True
    if any(not clause for clause in cnf):
        return False
    for clause in cnf:
        if len(clause) == 1:
            literal = clause[0]
            cnf = unitPropagate(cnf, literal)
            return DPLL(cnf)
    literal = cnf[0][0]
    cnf = unitPropagate(cnf, literal)
    if DPLL(cnf):
        return True
    cnf = unitPropagate(cnf, -literal)
    return DPLL(cnf)

def unitPropagate(cnf, literal):
    newCnf = [clause for clause in cnf if literal not in clause]
    newCnf = [list(filter(lambda l: l != -literal, clause)) for clause in newCnf]
    return newCnf
```

```python
cnf = [
    [-2, 1, 3],
    [-2, 1, 3],
    [2, 1, 3],
]
result = DPLL(cnf)
print(result)
```

CDCL

.The Conflict-Driven Clause Learning (CDCL) algorithm is a method used to solve the Boolean satisfiability problem (SAT). Here are the main steps of the CDCL algorithm:

1. **Select a variable and assign True or False**: This is referred to as the decision state.
2. **Apply Boolean constraint propagation (unit propagation)**: If an unsatisfied clause has all but one of its literals or variables evaluated at False, then the free literal must be True in order for the clause to be True.

CDCL

3.Build the implication graph: This graph represents the implications of the assignments made.

4.Check for conflicts: If there is any conflict, find the cut in the implication graph that led to the conflict.

5.Derive a new clause and backtrack: Create a new clause which is the negation of the assignments that led to the conflict. Then, non-chronologically backtrack (also known as “back jump”) to the appropriate decision level, where the first-assigned variable involved in the conflict was assigned.

6.Repeat the process: Continue from step 1 until all variable values are assigned.

.This algorithm is widely used in various SAT solvers due to its efficiency in handling complex CNF formulas.

CDCL

Sample codes for CDCL are too long thus it was decided to put a reference link:

- CDCL Code :

  [sat-solver/implementation/CDCL.py at master · Mr-Ahmadi/sat-solver (github.com)]

- Also DPLL Code (Too easy) :

  [sat-solver/implementation/DPLL.py at master · Mr-Ahmadi/sat-solver (github.com)]

Bye :)