Fan Zhang (m42zhang)
ECE650
December 22, 2022

Bonus Description

1. Overview

As Fig 1 shows, The SAT Solver first uses a Tokenizer and a Parsers to check the syntax of the Boolean Formula F. Then, it convert the Abstract Syntax Tree into Negative Normal Form, Which c reduces multiple negations. Annihilating multiple negations can generate fewer new variables in the Tseitin Transformation step thus saves time. The Tsetin Transformer output a Boolean formula in CNF. Finally, the DPLL Solver combines unit-resolution propagation and naive truth table method to peform a depth-frist search on the assignment tree.

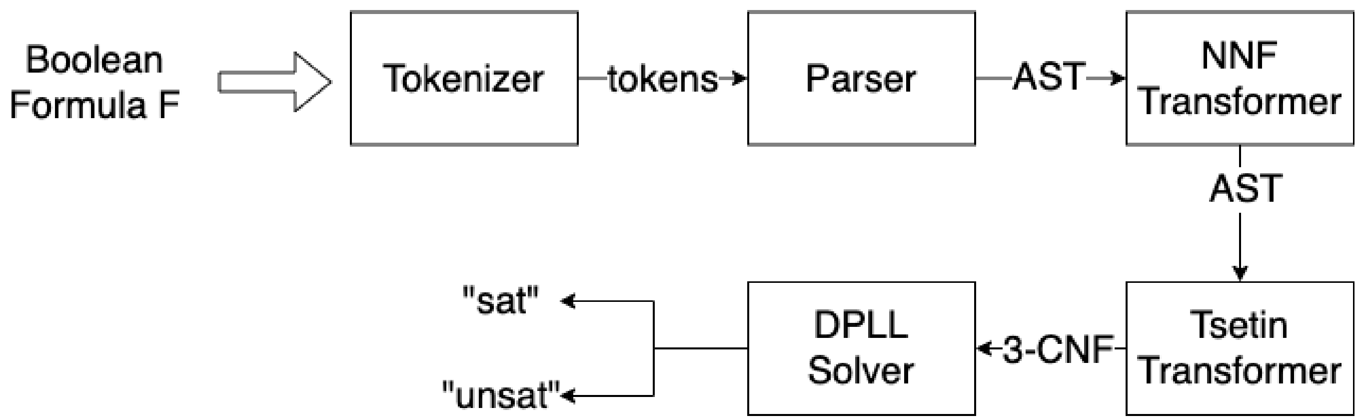

Figure 1. The architecture of the SAT solver. The Boolean Formula F only contains "and", "or" and "negation" logical connectives and variables. Constants and other connectives are illegal.

2. DPLL implementation

My implementation of DPLL is based on the Piazza Pseudo-code [1, 2].

The following loop invariants exists for every recursive BCP call.

- After one call of BCP, the assignment map is empty.
- After one call of BCP, the returned formula contains no variables assigned already assigned, already satisfying formula or unit clauses.

The following loop invariants exists for every recursive PLP call.

- After one call of PLP, the assignment is empty.
- After one call of PLP, the returned fomula contains no clauses with pure literals.

The DPLL recursive function returns

- true, if the CNF forumula is empty.
- false, if the CNF formula contains empty clauses.

I use "$vector < vector < int >>$" as the container for the CNF formula. I also used the vector iterator to implement dropping satisfying clause and dropping unsatisfying literals.

I wish to have constanttime lookup and FIFO access pattern for the assignment map.To implement this feature, I combined the queue and hashmap data structure from STL (Fig 2). Compared to using Queue and Pairs, my method avoids duplicate assignments and guarantees constant-time look-up, which helps to check whether the variable is assigned or not. Compared to using hashmap and a map iterator for looping, my implementation garantees FIFO accessing sequence, which helps to remove the assignments.

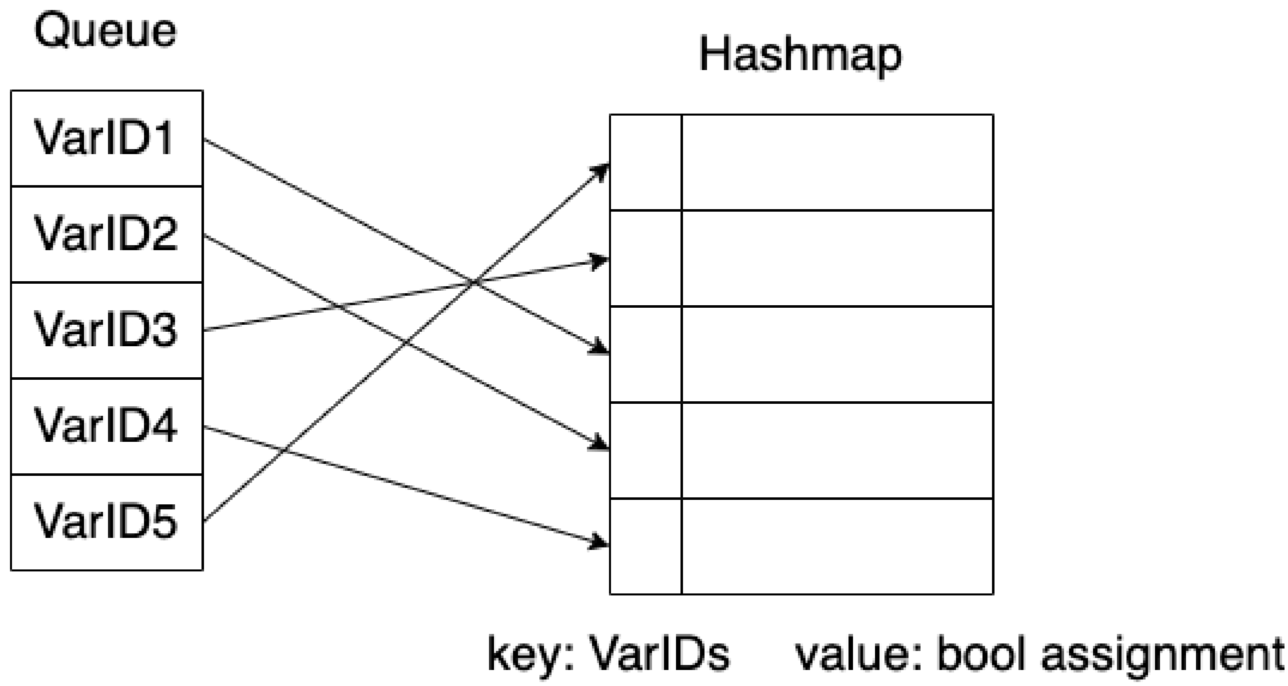

FIGURE 2. Every CURD operations is atomic both for the internal containers queue and map. This container is called "quemap". I used template to implement this class to accomodate other possible types

3. Time and Space Efficiency

At every recursive call, I maintains a new CNF on the stack and a assignment map as a instance variable, both is upper bounded by the number of connectives and variables. The memory at a certain moment of the running is $O(n)$, where n is the number of tokens in the CNF formula. However, on the average case, the bound could much tighter. Since n usually gets smaller during every call.

On the ECE-Tesla Server, it takes 0.03 seconds and 4544 KB run all the test cases in test0.in and test1.in files.

4. Future Work

- My code leaks memeory. I didn't implement all the deconstructors.
- Use a graph instead of nested vector to represent CNFs.
- implement classes for variables, literals, clasues and CNFs.

References

[1] BCP implementation written by the professor in a private post. https://piazza.com/class/l7r8af76c9g6mu/post/372. Accessed: 2022-12-6.
[2] Need pseudocode for plp. https://piazza.com/class/l7r8af76c9g6mu/post/380. Accessed: 2022-12-6.