Bonus Description

Yichu Li

December 2022

In this bonus part, I implement my own SAT solver by using method of the Davis–Putnam–Logemann–Loveland (DPLL) algorithm instead of directly calling the MiniSat solver. Given the original formula F, first I use the Tseitin's transformation to convert F to equisatisfiable formula F' in CNF, which I use the same way in my pa2, then I write a function called DPLL in the satSover.cc file, which uses the Boolean Constraint Propagation (BCP) and Pure Literal Propagation (PLP) to do the deduction function. I wrote the following pseudocode to implement the DPLL:

$\boldsymbol{DPLL}(\boldsymbol{cnf}, \boldsymbol{assignment})$

1. $cnf' \leftarrow \boldsymbol{BCP}(cnf, assignment)$
2. $assignment \leftarrow \boldsymbol{PLP}(cnf', assignment)$
3. $\boldsymbol{if}\ cnf'\ is\ empty\ \boldsymbol{then}$:
4. $\quad \boldsymbol{return}\ sat$
5. $\boldsymbol{if}\ size\ of\ cnf'\ is\ 1\ and\ the\ only\ formula\ in\ cnf'\ is\ empty\ \boldsymbol{then}$:
6. $\quad \boldsymbol{return}\ unsat$
7. $num \leftarrow -1$
8. $S \leftarrow assignment$
9. $\boldsymbol{for}\ each\ atom\ a\ in\ cnf'\ \boldsymbol{do}$:
10. $\quad \boldsymbol{if}\ assignment[a]\ is\ undecided\ \boldsymbol{then}$:
11. $\quad\quad num \leftarrow a$
12. $\quad\quad break$
13. $\boldsymbol{if}\ num = -1\ \boldsymbol{then}$:
14. $\quad \boldsymbol{return}\ DPLL(cnf', assignment)$
15. $assignment[num] \leftarrow T$
16. $\boldsymbol{if}\ DPLL(cnf', assignment) = sat\ \boldsymbol{then}$:
17. $\quad \boldsymbol{return}\ sat$
18. $\boldsymbol{else\ then}$:
19. $\quad assignment \leftarrow S$
20. $\quad assignment[num] \leftarrow \perp$
21. $\quad \boldsymbol{return}\ DPLL(cnf', assignment)$

DPLL updates the CNF and assignment by BCP and PLP, which can help to deduce the formula and make it easier to compute the satisfiability. After the reduction, if the new CNF' is empty then it means by this assignment, every atom can be reduced, which make it satisfiable. Otherwise, if the new CNF' has contradiction, then this assignment will make it unsatisfiable. Then, we go to the choose-variable part. We choose the first variable v in the CNF' that has not been assigned. If all the variable has been assigned, it means that we need to use BCP to deduce the formula, then we return DPLL, input the updated CNF' and assignment. Otherwise, we first assume v is true, then we return DPLL, input the updated CNF' and assignment. If it finally

returns sat, then it means this formula is satisfiable by this assignment. If it is not, it will backtrack and recover all the assignment back to before the assign step. Then, we assign v as false and return DPLL by the updated CNF' and assignment. Because it is binary, if it also returns unsat, then, the formula is unsatisfiable.

Since I also use the BCP and PLP to implement the DPLL, I will also introduce the pseudocode of BCP and PLP:

BCP(**cnf**, **a**)

1. $idx \leftarrow -1$
2. **for** *each variable v in a* **do**:
3. **if** *v is assigned and cnf exists v or* $-v$ **then**:
4. $idx \leftarrow i$
5. *break*
6. **if** $idx = -1$ **then**:
7. $cnf' \leftarrow cnf$
8. **else then**:
9. **for** *each clause c in cnf* **do**:
10. $code \leftarrow -1$
11. **if** $(idx \in c \text{ and } a[idx] = 1)$ **or** $(-idx \in c \text{ and } a[idx] = 0)$ **then**:
12. $code \leftarrow 0$
13. **else if** $(idx \in c \text{ and } a[idx] = 1)$ *or* $(-idx \in c \text{ and } a[idx] = 0)$ **then**:
14. $code \leftarrow 1$
15. **if** $code = -1$ **then**:
16. cnf' *append* c
17. **else if** $code = 0$ **then**:
18. cnf' *append* c *expect for* idx *and* $-idx$
19. **else if** $code = 1$ **then**:
20. cnf'*do nothing to drop this clause*
21. **for** *each clause* c'*in* cnf' **do**:
22. **if** c'*is a unit clause with the only variable* v' **then**:
23. **if** $v' > 0$ *and* $a[v']$ *is not assigned* **then**:
24. $a[v'] \leftarrow T$
25. **else if** $v' < 0$ *and* $a[-v']$ *is not assigned* **then**:
26. $a[v'] \leftarrow \perp$
27. **else if** $v' > 0$ *and* $a[v'] \leftarrow \perp$ **then**:
28. *return* {{}}
29. **else if** $v' < 0$ *and* $a[-v'] \leftarrow T$ **then**:
30. **return** {{}}
31. **else then**:
32. $cnf'' \leftarrow c'$
33. **return** cnf''

In BCP function, I can detect all the assigned variables and choose to drop the variable or the whole clause to shrink the formula. After that, if one clause becomes a unit clause, it will be assigned properly.

PLP$(\boldsymbol{cnf}, \boldsymbol{assignment})$

1. ***for*** *each variable* v *from assignment* ***do***:
2. ***if*** v *is undecided* ***and*** *only positively occur in cnf* ***then***:
3. $assignment[v] \leftarrow T$
4. ***else if*** v *is undecided* ***and*** *only positively occur in cnf* ***then***:
5. $assignment[v] \leftarrow \perp$
6. ***return*** *assignment*

In PLP function, if variable p occurs only positively in the formula, then p must be set to T, else if variable p occurs only negatively in the formula, then p must be set to $\perp$. The newly assigned variable will shrink the formula in the next BCP step.

In conclusion, my SAT Solver leverages the method of DPLL, combined with BCP and PLP to make the computing process more efficient.