ECE650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

UNIVERSITY OF WATERLOO

Coding Assignment Bonus

Name:	Weiqiang Yu
Student ID:	21020022
Email:	w8yu@uwaterloo.ca

Date: December 23, 2022

1 DPLL

In this bonus assignment, I implemented a simple sat-solver using DPLL with Pure Literal Propagation as discussed in the lecture. The following is the pseudocode for DPLL.

Algorithm 1 $DPLL(CNF\ \phi, AssignMap\ A)$

```
1: $\phi' = BCP(\phi, A)$;
2: $\phi'' = PLP(\phi')$;
3: **if** ($\phi' = \top$) **then return** SAT;
4: **else if** ($\phi' = \bot$) **then return** UNSAT;
5: $p = choose\_var(\phi')$;
6: **if** ($DPLL(\phi', A[p \mapsto \top])$) **then return** SAT;
7: **else return** ($DPLL(\phi', A[p \mapsto \bot])$);
```

As shown in the above code, DPLL has several steps:

1. Perform Boolean Constraint Propagation, reduce the formula ϕ to ϕ'
2. Perform Pure Literal Propagation, simply the formula ϕ' to ϕ''
3. If the formula ϕ'' is empty, return SAT
4. if the fomula ϕ'' contains empty clauses, return UNSAT
5. choose a decision variable p
6. Set p to true in map A, and recursively call DPLL algorithm with the new map. If the subroutine returns SAT, DPLL returns SAT. Otherwise
7. Set p to false in map A, and recursively call DPLL algorithm with the updated map. If the subroutine returns SAT, DPLL returns SAT. Otherwise, return UNSAT.

The main work in this assignment is to design Boolean Constraint Propagation and Pure Literal Propagation.

2 Boolean Constraint Propagation

Boolean Constraint Propagation, in DPLL, refers to all possible applications of unit resolution.

Algorithm 2 *BCP(CNF ϕ, AssignMap A)*

```
 1: for each clause $C$ in $\phi$ do
 2:     for each literal $l$ in clause $C$ do
 3:         if A contains $l$ or A contains $-l$ then
 4:             if $(l > 0$ && $A[l] == true)$ or $(l < 0$ && $A[-l] == false)$ then
 5:                 remove clause $C$ from $\phi$;
 6:                 break;
 7:             else
 8:                 remove literal $l$ from clause $C$;
 9:             if C is empty clause then
10:                 return
11:     if $C$ is a unit clause then
12:         add it to the map A with appropriate value and remove $C$ from $\phi$;
13:         restart the procedure with reduced formula $\phi'$ and updated map $A'$
14: return modified formula $\phi'$
```

The following are the steps for Boolean Constraint Propagation.

1. Iterate every literal in formula ϕ
2. If current literal l occurs positively in the clause C, and $A[l]$ is true; or l occurs negatively in the clause C, and $A[-l]$ is false: We drop the clause C from formula ϕ
3. If current literal l occurs positively in the clause C, and $A[l]$ is false; or l occurs negatively in the clause C, and $A[-l]$ is true: We remove the literal l from clause C
4. If the clause C becomes empty, BCP terminates (when formula ϕ contains an empty clause, DPLL should return UNSAT)
5. If C is a unit clause. Then we add the variable in C to A with appropriate value and remove unit clause C from formula ϕ. Since in this process, we derive a new unit clause, we need to restart the procedure and apply unit resolution to the modified formula ϕ', with updated map A', until no more unit clause appears in this process.
6. If no unit clause is derived in this procedure, the algorithm returns modified formula ϕ'

3 Pure Literal Propagation

If variable p occurs only positively or negatively in the formula, then p is a pure literal. We can further simplify the formula ϕ by dropping clauses which contain pure literal.

Algorithm 3 *PLP(CNF ϕ, AssignMap A)*

```
1: create a new map **variable_counter**, which records the number of literals in $\phi$
2: **while** true **do**
3:     plp = false;
4:     **for** each (u,v) in variable_counter **do**
5:         **if** variable_counter does not contain $-u$ **then**
6:             plp = true;
7:             add it to the map A with appropriate value and remove it from variable_counter;
8:     **if** plp == false **then**
9:         **return** $\phi'$
10:    **for** each clause $C$ in $\phi$ **do**
11:        **for** each literal $l$ in clause $C$ **do**
12:            **if** A contains $l$ or A contains $-l$ **then**
13:                **for** each literal $l$ in clause $C$ **do**
14:                    variable_counter[l] = variable_counter[l] $-1$;
15:                    **if** variable_counter[l] == 0 **then**
16:                        remove $l$ from variable_counter
17:                drop clause $C$ from $\phi$
```

The above code implements the following procedure:

1. create a new map variable_counter, which records the number of literals in ϕ
2. for each pair of (u, v) in variable_counter, if variable_counter does not contain $\neg u$, then u is a pure literal. We assign the variable with appropriate value in A and remove it from variable_counter.
3. If there is no pure literal in formula ϕ, the algorithm returns ϕ'
4. Iterate every literal l in formula ϕ
5. If A either contains l or A contains $-l$, then we know that l is a pure literal because BCP has already removed other elements in A from formula. Thus, we first decrease variable_counter[l] by 1, and then drop the clause C from formula ϕ.
6. Repeat step 2 to 5, until the formula ϕ has no pure literals.

4 decision variable choice

For simplicity, this function returns the first unassigned variable.

Algorithm 4 $choose\_var(variables\ V)$

```
return V.first();
```