PA3: SAT Solver using DPLL algorithm

The DPLL algorithm is used to determine whether or not the logical proposition may be satisfied. It speeds up the calculations and lowers the overall number of cases that need to be evaluated. The idea is to assign various propositions as true or false in order to test the most likely model for the proposition. If we see that, the proposition is still not possible to satisfy even in that case, we conclude that this proposition is unsatisfiable.

The following is the pseudo code for the modified DPLL algorithm: -

```
int highest_element(assignMap, formula)
{
    //find the most frequent occuring element from the remaining
formula
   1. traverse the formula
   2. store the keys with their count in another map (literal, count)
   3. find the highest value of the count for a literal and return
that literal

}

bool BCP (formula, assignMap)
{
    for all pairs in assignMap
    {
        for all clauses in formula
        {
            for all literals in clauses
            {
                if (literal is positive and value is true || literal
is negative and value is false)
                {
                    remove clause
                }

                else if (literal is positive and value is false ||
literal is negative and value is true)
                {
                  if (unit clause and formula.size()>=1 )  //literal
in unit clause becomes false
                   {
                     return false;
                   }
                   else
                   drop literal from the clause
                }
```

```
                if (unit clause)
                {
                    if (literal is positive)
                    insert (literal, true) in assignMap

                    else
                    insert (literal, false) in assignMap

                    if (literal in positive and the value in map is
false || literal is negative and value in map is true)
                    {
                      return false
                    }
                }

            }
        }
    }
}

bool DPLL (formula, assignMap)
{
    bool bcp_result= bcp(formula, assignMap)

    if (bcp_result is true and formula is empty)
    {
        return true;
    }
    else if (bcp_result is false)
    {
        return false;
    }
    else
    {
        //Assign p such that it occurs the highest number of times in
the formula
        int p= highest_element(assignMap, formula)
        Assign p as true

        Make a copy of assignMap as assign_copy
        make a copy of formula as formula_copy

        dpll_result= dpll(formula, assignMap)

        if(dpll_result is true and formula is empty)
        {
            return true //sat
        }
```

```
        else
        {Assign p as false
          return dpll(formula_copy, assign_copy) //backtrack
        }
    }
}
```

DESCRIPTION: -

The idea behind my modification, is to set the highest occurring proposition true in order to reduce the length of the formula by removing the clause from the formula. If the assignment gives a conflict, it assigns the proposition as false and recursively calls the DPLL() function. If the formula returns true for any of the combinations of the propositions, then the formula is said to be sat and if it is unable to find the combinations of the proposition for which it is true, then the formula is declared as UNSAT.

In the pseudo code above,

1. The dpll() takes in a CNF formula and an assignment map with key as the literal and value as true or false.
2. bcp() is called first and the bcp() checks the edge conditions to return false. If there exists a literal in a unit clause which appears to be false, it returns false since the formula is in CNF format. Otherwise, it returns true.
3. If the bcp () returns true and the formula is empty then the formula is SAT. On the other hand, if the bcp() returns false, the formula is UNSAT.
4. If the bcp() returns true and the formula is not empty(), we need to set a literal from the remaining formula in order to check its satisfiability.
5. To decide which literal to set, select a literal which occurs the maximum number of times in the formula. This helps to efficiently reduce the formula which in turn reduces the total number of cases to check.
6. Set the selected literal as true and call dpll(). Repeat the steps from 2.
7. If, with this literal, the bcp() returns false, we set the literal as false this time and call the dpll() recursively. After checking for all the literals, if the formula still returns false, then the formula is UNSAT.