CS 264A - Project Report

Yuan He and Tianyi Zhang

1 Data Structure

The data structure that we used in the the SAT primitive includes var, literal, clause and sat_state_t.

- **BOOLEAN**: represents the value of the boolean variable
- **c2dSize**: Index for boolean variables
- **c2dListeral**: Index for boolean variables
- **c2dWmc**: for (weighted) model count
- **var**: It describes the variables that the input CNF uses. Other that "index" denote the its index in all variables that used in the CNF, two pointers "pos" and "neg" points the positive literal and negative literal for the current variable. The "clauses" points to all clauses that mentioned the current variable and "clauses_num" counts the total number of such clauses. "clause_capacity" is used for memory allocation efficiency. The "value" represents the actually value of the variable.
- **literal**: It describes the literal that appear in the CNFs. "index" denotes the literal index among all CNF. A positive index denote a positive literal and vice versa. Variable pointer "var" points to the variable of the current literal. "decision_level" is used in the SAT algorithm. If a literal is decided by the algorithm or implied by the unit resolution, "decision_level" is set to be the current decision level of the SAT algorithm. The clause pointer "reason" is used in generating the asserting clause. If the current literal is implied in the unit resolution, the "reason" points to the clause that implies the literal during unite resolution.
- **clause**: "index" represents clause index in the input CNF. "lits" points to the literal array that in the current clause. "size" is the total number of literals in the current clause. In the unit resolution, if the current clause is subsumed, we set "subsume" to be true. If the current clause is a assertion clause, "assertion_level" is set to be the assertion level.
- **sat_state_t**: "vars" points to all variables that are used in the CNF. "var_num" is the total number of variables in the CNF. "lits" points to all literals in the CNF. "lit_num" is the total number of literals. "cnf" points to the all clauses in the original CNF. "learns" saves all learn clauses in the unit resolution. "learn_num" is the total number of clauses. "learn_capacity" is used in allocating and reallocating the clauses. "decisions" points to the decided literal array. "decision_level" represents d the current decision level. "implies" and "implies_num" represents the literals and the number of literals that are implied in the unit resolution. "asserting" is the asserting clause that are generated when unit resolution found a conflict.

```
typedef struct var {
   c2dSize index;
   struct literal * pos;
   struct literal * neg;
   struct clause ** clauses;
   int clause_num;
   int clause_capacity;
   int value;  // 1 —> true , 0 —> false , −1 —> unset
   BOOLEAN mark;
```

```c
} Var;

typedef struct literal {
   c2dLiteral index;
   Var * var;
   int decision_level;
   struct clause * reason;
} Lit;

typedef struct clause {
   c2dSize index;
   Lit** lits;
   int size;
   BOOLEAN subsume;
   int assertion_level;
   BOOLEAN mark;
} Clause;

typedef struct sat_state_t {
   Var ** vars;
   int var_num;
   Lit ** lits;
   int lit_num;
   Clause ** cnf;
   int clause_num;
   Clause ** learns;
   int learn_num;
   int learn_capacity;
   Lit ** decisions;
   int decision_level;
   int decision_capacity;
   Lit ** implies;
   int implies_num;
   int implies_capacity;
   Clause * asserting;
} SatState;
```

2 Implementation

2.1 Unit Resolution

The algorithm of the unit resolution is depicted as Algorithm 1. Here, we briefly describe our algorithm. The input is a SAT state Δ. I is the set of implied literals, it is actually a part in SAT state Δ. The algorithm body is a do while loop. For every clauses (input CNF and learned clauses), it first checks whether the clause is subsumed. If the clause is not subsumed, it check whether there exists only one literal is not set. If so, it simply implies that literal and then adds it in the implied literals. If all literals in the clause are falsified, then this clause is a empty clause, which is a contradiction. Therefore, we generate the learning clause using Algorithm 2 and then return false. If no new literal is implied in previous iteration, the algorithm return true,other wise the algorithm recheck all clauses again until it finds a contraction or no more literal is implied.

Note that, in unit resolution, we track the reason clause of every implied literal and decided literal. The reason clause is used for conveniently generate the UIP learned clause in Algorithm 2.

Algorithm 1: Unit Resolution (Δ)

```
Input  : $\Delta$: a SAT state, $I$: implied literals
Output: true if unit resolution succeeds; false if it finds a contradiction
do
    for all clauses in $\Delta$ do
        if the clause is not subsumed then
            if there is only one literal $l$ in the clause is not set then
                $I \leftarrow I \cup \{l\}$                              /* $l$ is implied */
            else if the clause is empty                   /* find a contradiction */
            then
                generate the assertion clause             /* detail see Algorithm 2 */
                return false
while a new literal is added in $I$
return true
```

2.2 Clause Learning

Algorithm 2 is used for generating the first UIP learned clause when unit resolution finds a contraction. The algorithm is originally discussed in [1]. The input is the SAT state Δ and the empty clause that are found by unit resolution. At beginning, we set variable *learned clause* to be the empty clause. Next, we check whether the *learned clause* is asserting. Recall that a clause is asserting if only one variable is set at the last decision level in SAT state Δ. If *learned clause* is not a asserting clause, then we find out the literal l that was falsified last in the unit resolution. Next, we find out the *reason clause* of the $\neg l$' that saved during the unit resolution. After that, we apply linear resolution on the *learned clause* and *reason clause* and then updat the *learned clause* to be the resolvent. We repeat the process unit the *learned clause* is assertion clause.

Whenever unit resolution discovers an empty clause, it applies the Algorithm 2 to derive an asserting clause. The algorithm keeps resolving the empty clause with reasons of some unit implications until the clause becomes asserting. Moreover, Algorithm 2 is guaranteed to generate the first UIP asserting clause from the empty clause.

Algorithm 2: The First UIP Clause Learning

```
Input  : $\Delta$: the SAT state, $c$: the empty clause
Output: learned clause
learned clause $\leftarrow c$
while learned clause is not asserting do
    $l \leftarrow$ literal in learned clause that was falsified last
    learned clause $\leftarrow$ resolvent of learned clause and $\neg l$'s reason
return learned clause
```

3 Evaluation

References

1. Adnan Darwiche. Chapter 3: Satisfiability. In *Lecture Notes of CS 264A*, pages 18–21.