# SMT Solver Notes

Tom

## 1 SMT

SMT solver is a tool that automatically determines whether a predicate logic formula is satisfiable. Because there are already some powerful SAT solvers, the SMT solver will be implemented based on the SAT solver. According to the usage of the SAT solver, it can be divided into eager SMT and lazy SMT.

## 2 Eager SMT Technique

These techniques convert predicate logic formulas into satisfiability-equivalent propositional logic formulas, which are then solved by the SAT solver. For example, the predicate logic formula x = 1 ÿ x = 3, you can use p to represent x = 1, q to represent x = 3, and get the propositional logic formula (¬p ÿ q) ÿ (p ÿ ¬q) (note that x cannot be simultaneously 1 and 3). The satisfiability of these two formulas is equivalent, and then the SAT solver can be used to solve them.

Next, consider the case where the predicate logic formula includes a function. We take the predicate logic formula 1 as an example to illustrate.

$$y = f(z) \ddot{y} x = f(f(z)) \ddot{y} \neg(x = f(y)) \tag{1}$$

Assume that each individual word is of type boolean. Then you can use propositional variables to represent functions, and convert the predicate logic formula 1 to

into propositional logic formula 2.

$$(y \ddot{y} fz)\ddot{y} (x \ddot{y} ff z)\ddot{y} \neg(x \ddot{y} fy) \quad (2)$$

But this does not guarantee the satisfiability of the two formulas is equivalent. In fact, the predicate logic formula 1 is unsatisfiable, and the transformed propositional logic formula 2 is satisfiable. The reason is that the property of the function a = b ÿ f(a) = f(b) is lost in the process of transformation, so this property of the function should also be encoded into propositional logic, as shown in propositional logic formula 3.

$$((y \ddot{y} z) \ddot{y} (fy \ddot{y} fz))\ddot{y} ((y \ddot{y} fz) \ddot{y} (fy \ddot{y} ff z)\ddot{y} ((z \ddot{y} fz) \ddot{y} (fz \ddot{y} ff z))\ddot{y} (y \ddot{y} fz)\ddot{y} (x \ddot{y} ff z)\ddot{y} \neg(x \ddot{y} fy) \quad (3)$$

This way the SAT solver will return unsat.

### 2.1 Summary

The design of this SMT solver is not flexible enough. It is sometimes not easy to convert a predicate logic formula into a propositional logic formula, and it may generate a long propositional logic formula, which makes it difficult for the SAT solver to solve.

# 3 Lazy SMT Technique

The commonly used SMT solvers now use the lazy SMT technique.

## 3.1 DPLL(T)

Consider first the case without quantifiers. Take the predicate logic formula 4 as an example.

$$(x = 1 \ddot{y} x = 3)\ddot{y} (y = 1 \ddot{y} y = 2 \ddot{y} y = 3)\ddot{y} (z = 3)\ddot{y} \neg(x = y)\ddot{y} \neg(x = z)\ddot{y} \neg(y = z) \quad (4)$$

The first step is to replace each atomic proposition in predicate logic formula 4 with propositional variables, and convert it into propositional logic formula 5.

$$(p1 \ddot{y} p2)\ddot{y}$$
$$(p3 \ddot{y} p4 \ddot{y} p5)\ddot{y}$$
$$(p6)\ddot{y}$$
$$\neg p7\ddot{y} \quad (5)$$
$$\neg p8\ddot{y}$$
$$\neg p9$$

Note that this conversion method does not guarantee the satisfiability of the two formulas is equivalent, for example, x > 3 ÿ x < 1 can be replaced by p ÿ q, obviously one is satisfiable and the other is unsatisfiable, and the relationship between them should be predicate logic formula satisfiable ÿ propositional logic formula satisfiable. The second step is to use the SAT solver to solve the propositional logic formula. Because the predicate logic formula can be satisfied ÿ the propositional logic formula can be satisfied, so if the SAT solver returns unsat at this step, then the SMT solver returns unsat directly, otherwise it needs to be checked further. For the above example, the SAT solver returns sat, assuming the following solution is obtained.

$$p1 = T, p2 = F, p3 = T, p4 = F,$$
$$p5 = F, p6 = T, p7 = F, p8 = F,$$
$$p9 = F$$

The third step is to use the theory solver (or T-solver) to judge whether the solution given by the SAT solver is valid. SMT solver contains multiple theory solvers, each of which solves formulas in specific fields such as number theory and strings. The solution given by the SAT solver above actually means that the predicate logic formula 6 is established, and the T-solver is responsible for checking whether the formulas in the predicate logic formula 6 can be established at the same time.

$$x = 1, x \ddot{y}= 3, y = 1, y \ddot{y}= 2,$$
$$y \ddot{y}= 3, z = 3, x \ddot{y}= y, x \ddot{y}= z, \quad (6)$$
$$y\ddot{y}= z$$

If the T-solver finds no problems at this step, the SMT solver returns sat. But here T-solver finds that x = 1, y = 1, x ÿ= y cannot be established at the same time, so it informs SAT solver that the solution given is wrong, and the notification method is to add ¬(p1 ÿ p3 ÿ ¬p7) to the input of SAT solver , at this time the SAT solver solves propositional logic formula 7,

$$(p1 \ddot{y} p2)\ddot{y}$$
$$(p3 \ddot{y} p4 \ddot{y} p5)\ddot{y}$$
$$(p6)\ddot{y}$$
$$\neg p7\ddot{y} \quad (7)$$
$$\neg p8\ddot{y}$$
$$\neg p9\ddot{y}$$
$$(\neg p1 \ddot{y} \neg p3 \ddot{y} p7)$$

Return sat, assuming the following solution is given,

p1 = T, p2 = F, p3 = F, p4 = T,

p5 = F, p6 = T, p7 = F, p8 = F,

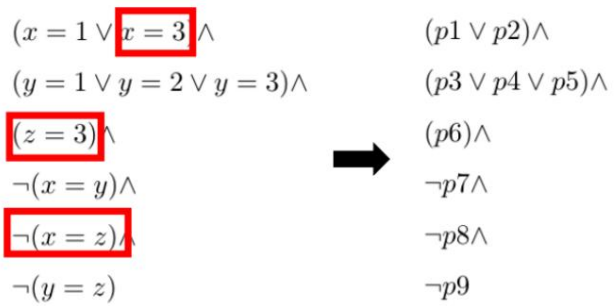
p9 = F

Then repeat this step until the theory solver finds no problem or the SAT solver returns unsat. this time The T-solver found no problems, so the SMT solver returned sat.

## 3.2 Incremental T-solver

The above is the basic principle of SMT solver, and now we talk about some ways to optimize performance.

According to the above algorithm, wait until the SAT solver finds the truth value of all proposition variables and then call T-solver to check. In fact, T-solver can be called in the middle of the execution of the DPLL algorithm to check the existing assignments to some proposition variables. If there is any problem, this can detect the problem as soon as possible, terminate the DPLL algorithm in advance, and reduce the useless work of the SAT solver.




$$\emptyset \parallel 1 \vee 2,\ 3 \vee 4 \vee 5,\ 6,\ \overline{7},\ \overline{8},\ \overline{9} \implies (UnitProp)$$

$$6\ \overline{7}\ \overline{8}\ \overline{9} \parallel 1 \vee 2,\ 3 \vee 4 \vee 5,\ 6,\ \overline{7},\ \overline{8},\ \overline{9} \implies (Decide)$$

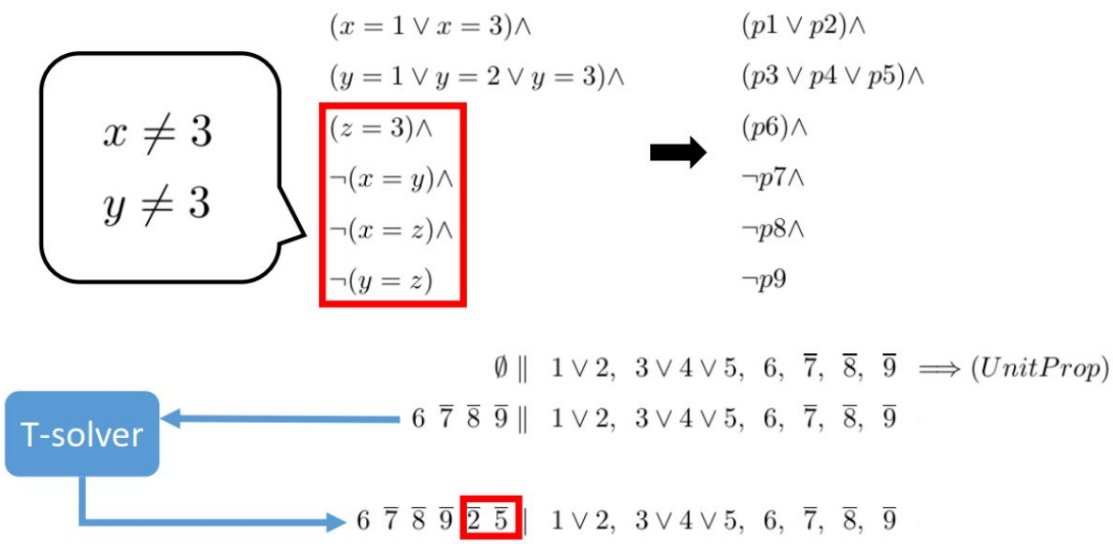
$$6\ \overline{7}\ \overline{8}\ \overline{9}\ \overline{1}^{d} \parallel 1 \vee 2,\ 3 \vee 4 \vee 5,\ 6,\ \overline{7},\ \overline{8},\ \overline{9} \implies (UnitProp)$$

$$6\ \overline{7}\ \overline{8}\ \overline{9}\ \overline{1}^{d}\ 2 \parallel 1 \vee 2,\ 3 \vee 4 \vee 5,\ 6,\ \overline{7},\ \overline{8},\ \overline{9}$$

For example, in the example above, although DPLL has not yet calculated all the assignments, it tells T-solver the intermediate results at this time. T-solver finds that x = 3, z = 3, x ÿ= z cannot be established at the same time, and the DPLL algorithm just There is no need to continue counting, reducing useless calculations. (Note that the pureliteral rule should normally be used first here, but in order to explain how to optimize, it is not used.)

## 3.3 Theory Propagation

The T-solver can not only check whether there is any problem with the assignment of the SAT solver, but also infer the truth value of some literals in the middle of the DPLL algorithm operation. For example, if the SAT solver assigns the proposition variable corresponding to x > 3 to T during the running process, and there is an atomic proposition in the current formula that is x < 0, then the T-solver can infer that the proposition variable corresponding to x < 0 must be assigned a value is F and informs the SAT solver, which can help reduce the workload of the SAT solver.




In the above figure, DPLL tells T-solver the intermediate result, and T-solver can deduce x ÿ= 3, y ÿ= 3 according to z = 3, x ÿ= z, y ÿ= z, that is, p2 and p5 must be assigned For F, help the derivation of the DPLL algorithm. Note that the pureliteral rule should normally be used first, but in order to explain how to optimize, it is not used.

# 4 EUF

EUF (equality and uninterpreted function) is a T-solver among SMT solvers, which is used to process formulas including = and functions. This section describes its rationale.

EUF needs to use the congruence rule when processing functions. This rule means that for the same function, if the input of the function is the same, the output of the function must be the same.

**Theorem 1** (congruence rule). x1 = y1, ..., xn = yn ÿ f(x1, ..., xn) = f(y1, ..., yn)

The algorithm of EUF is explained by taking the predicate logic formula 8 as an example.

$$\begin{aligned} &a = b,\ b = c,\ d = e,\\ &b = s,\ d = t,\\ &f(a, g(d)) \ddot{y}= f(b, g(e)) \end{aligned} \tag{8}$$

The first step is to replace the function in predicate logic formula 8 with individual words and rewrite it as predicate logic formula 9

$$\begin{aligned} &a = b,\ b = c,\ d = e,\\ &b = s,\ d = t,\ v3 \ddot{y}= v4 \end{aligned} \tag{9}$$

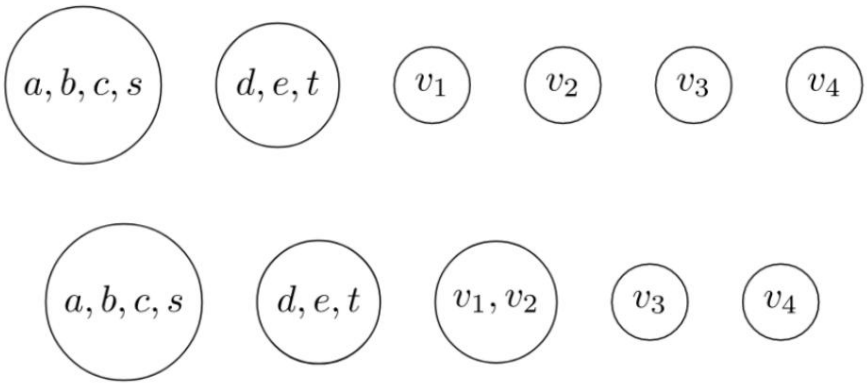
Among them, v1 represents g(e), v2 represents g(d), v3 represents f(a, v2), and v4 represents f(b, v1).

Step 2, draw a picture with a circle for each individual word. Step 3, looking at the equation in Predicate

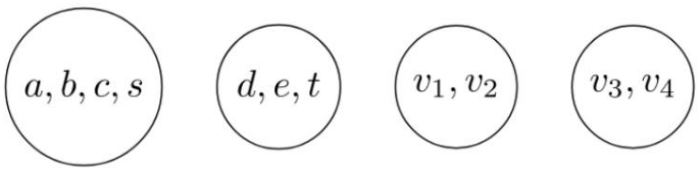
Logic Equation 9, merge the circles where the two equal individual words are. eg root

According to a = b, the circles where a and b are located can be merged into one circle. Finally get the picture below.

In step 4, the circles are further merged according to the congruence rule. Since e and d are in a circle in the figure, g(e) = g(d), merging the circles of v1 and v2.




Now v1 and v2 are in a circle, a and b are in a circle, so f(a, v2) = f(b, v1), merging the circles of v3 and v4.




Finally, because the merge operation is over, now look at the inequalities. If two unequal objects are in a circle, something went wrong. The inequality in the example is v3 ÿ = v4, but now that v3 and v4 are in a circle, the EUF will report an error.