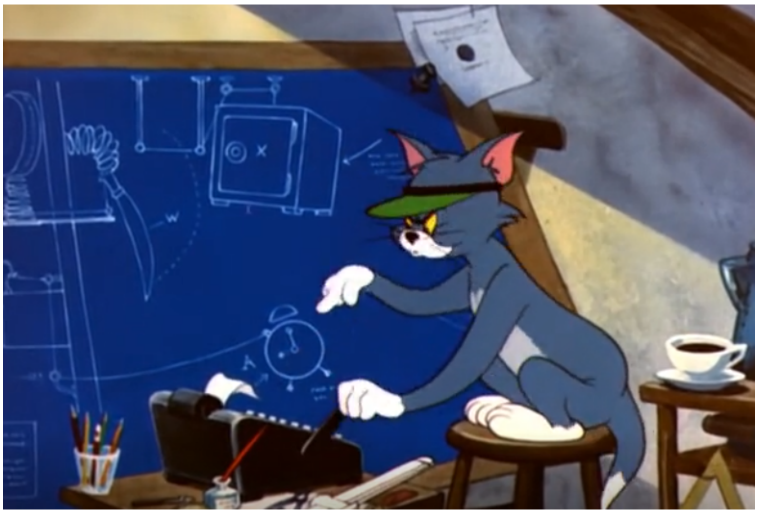
SMT Solver Notes

Tom



1 SMT

SMT solver 是自动判定一个谓词逻辑表达式是否可满足的工具。因为已经有一些能力强大的 SAT solver，所以 SMT solver 会基于 SAT solver 来实现，按照对 SAT solver 的用法不同可以分为 eager SMT 和 lazy SMT。

2 Eager SMT Technique

这类技术把谓词逻辑表达式转成可满足性等价的命题逻辑表达式，然后用 SAT solver 求解。例如 $x = 1 \vee x = 3$，可以用两个命题变项表示 x，得到命题逻辑表达式 $(\neg p \wedge q) \vee (p \wedge q)$，这两个式子的可满足性是等价的，然后用 SAT solver 求解就行了。

下面考虑公式包括函数的情况。例如 $y = f(z) \wedge x = f(f(z)) \wedge \neg(x = f(y))$，假设每个个体词都是布尔类型的。那么可以用命题变项表示函数，把式子转成下面这样。

$$
\begin{aligned}
&(y \leftrightarrow f_z) \wedge \\
&(x \leftrightarrow f_{fz}) \wedge \\
&\neg(x \leftrightarrow f_y)
\end{aligned}
$$

但是这样还不能保证两个式子的可满足性等价，实际上，上面的谓词逻辑公式是不可满足的，而转化成的命题逻辑公式是可满足的。原因是在转化的过程中把函数的性质 $a = b \Rightarrow$

$f(a) = f(b)$ 丢失了，所以应该把函数的这种性质也编码成命题逻辑，如下所示。

$$
\begin{aligned}
&((y \leftrightarrow z) \rightarrow (f_y \leftrightarrow f_z))\wedge \\
&((y \leftrightarrow f_z) \rightarrow (f_y \leftrightarrow f_{fz})\wedge \\
&((z \leftrightarrow f_z) \rightarrow (f_z \leftrightarrow f_{fz}))\wedge \\
&(y \leftrightarrow f_z)\wedge \\
&(x \leftrightarrow f_{fz})\wedge \\
&\neg(x \leftrightarrow f_y)
\end{aligned}
$$

这样 SAT solver 会返回 unsat。

2.1 总结

这种 SMT solver 的设计不够灵活，把一个谓词逻辑表达式转成命题逻辑表达式有时并不简单，而且可能产生很长的命题逻辑公式，导致 SAT solver 求解困难。

3 Lazy SMT Technique

现在常用的 SMT solver 采用 lazy SMT technique。

3.1 DPLL(T)

先考虑没有量词的情况。以下面这个式子为例。

$$
\begin{aligned}
&(x = 1 \vee x = 3)\wedge \\
&(y = 1 \vee y = 2 \vee y = 3)\wedge \\
&(z = 3)\wedge \\
&\neg(x = y)\wedge \\
&\neg(x = z)\wedge \\
&\neg(y = z)
\end{aligned}
$$

第一步把公式中的每个原子命题替换成命题变项，上面的式子可以转成下面的公式。

$$
\begin{aligned}
&(p1 \vee p2)\wedge \\
&(p3 \vee p4 \vee p5)\wedge \\
&(p6)\wedge \\
&\neg p7\wedge \\
&\neg p8\wedge \\
&\neg p9
\end{aligned}
$$

注意，这种转换方法不保证两个式子的可满足性等价，例如 $x > 3 \wedge x < 1$ 可以替换成 $p \wedge q$，明显一个可满足一个不可满足，它们之间的关系应该是谓词逻辑公式可满足 $\Rightarrow$ 命题逻辑公式可满足。

第二步是用 SAT solver 求解命题逻辑表达式。因为谓词逻辑公式可满足 $\Rightarrow$ 命题逻辑公式可满足，所以如果这一步 SAT solver 返回 unsat，那么 SMT solver 直接返回 unsat，否则要进一步检查。对于上面的例子，SAT solver 返回 sat，假设得到下面的解。

$$
\begin{aligned}
&p1 = T, p2 = F, p3 = T, p4 = F, \\
&p5 = F, p6 = T, p7 = F, p8 = F, \\
&p9 = F
\end{aligned}
$$

第三步是利用 theory solver（或者叫 T-solver）去判断 SAT solver 给出的解能不能成立。SMT solver 中包含多个 theory solver，各自求解数论、字符串等特定领域的公式。上面 SAT solver 给出的解实际上是说下面的这些公式同时成立，T-solver 负责检查这些公式能不能同时成立。

$$
\begin{aligned}
&x = 1, x \neq 3, y = 1, y \neq 2, \\
&y \neq 3, z = 3, x \neq y, x \neq z, \\
&y \neq z
\end{aligned}
$$

假如这一步 T-solver 没有发现问题，那么 SMT solver 就返回 sat。但是这里 T-solver 发现 $x = 1, y = 1, x \neq y$ 不能同时成立，于是通知 SAT solver 给的解不对，通知方法是把 $\neg(p1 \wedge p3 \wedge \neg p7)$ 加入到 SAT solver 的输入中，这时候 SAT solver 求解下面的式子，

$$
\begin{aligned}
&(p1 \vee p2) \wedge \\
&(p3 \vee p4 \vee p5) \wedge \\
&(p6) \wedge \\
&\neg p7 \wedge \\
&\neg p8 \wedge \\
&\neg p9 \wedge \\
&(\neg p1 \vee \neg p3 \vee p7)
\end{aligned}
$$

返回 sat，假设给出下面的解，

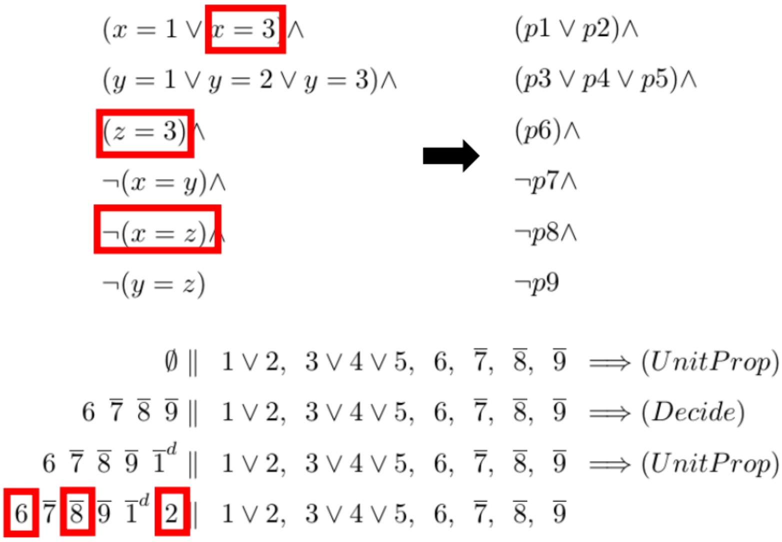
$$
\begin{aligned}
&p1 = T, p2 = F, p3 = F, p4 = T, \\
&p5 = F, p6 = T, p7 = F, p8 = F, \\
&p9 = F
\end{aligned}
$$

接下来重复本步骤，直到 theory solver 查不出问题或者 SAT solver 返回 unsat。这次 T-solver 没有发现问题，所以 SMT solver 返回 sat。

3.2 Incremental T-solver

上面讲的是 SMT solver 的基本原理，现在讲一些优化性能的方式。

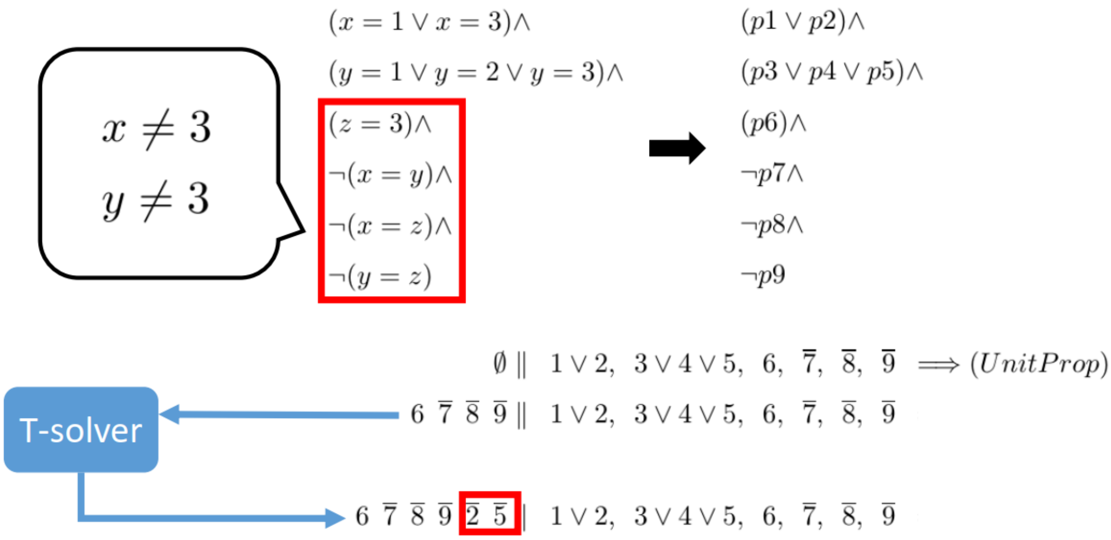
按照上面的算法，等到 SAT solver 求出所有命题变项的真值后再调用 T-solver 检查，实际上可以在 DPLL 算法执行中间就调用 T-solver，检查已有的对部分命题变项的赋值有没有问题，这样可以尽早发现问题，提前终止 DPLL 算法，减少 SAT solver 的无用功。



例如上图的例子中，虽然 DPLL 还没有算出所有的赋值，但是这时候把中间结果告诉 T-solver，T-solver 发现 $x = 3, z = 3, x \neq z$ 不能同时成立，DPLL 算法就不需要再继续算下去了，减少无用计算。注意，这里正常应该先使用 pureliteral rule，但是为了讲解如何优化，所以没有使用。

3.3 Theory Propagation

T-solver 不仅可以检查 SAT solver 的赋值有没有问题，而且还可以在 DPLL 算法运行中间推测出某些 literal 的真值。例如如果 SAT solver 运行过程中把 $x > 3$ 对应的命题变项赋值为 T，当前公式中还有一个原子命题是 $x < 0$，那么 T-solver 可以推测出 $x < 0$ 对应的命题变元必须赋值为 F，并通知 SAT solver，这样可以帮助减少 SAT solver 的工作量。



在上图中，DPLL 把中间结果告诉 T-solver，T-solver 可以根据 $z = 3, x \neq z, y \neq z$ 推出 $x \neq 3, y \neq 3$，也就是 p2 和 p5 必须赋值为 F，帮助 DPLL 算法做推导。注意，这里正

常应该先使用 pureliteral rule，但是为了讲解如何优化，所以没有使用。

4 EUF

EUF 是 SMT solver 中的一种 T-solver，本节描述它的基本原理。

EUF 全称是 equality and uninterpreted function，用来处理包括 = 和函数的公式。处理函数时需要用到 congruence rule。

Theorem 1 (congruence rule)**.** $x_1 = y_1, ..., x_n = y_n \Rightarrow f(x_1, ..., x_n) = f(y_1, ..., y_n)$

以下面的公式为例讲解 EUF 的算法。

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

第一步，把公式中的函数换成个体词。上面的式子改写为

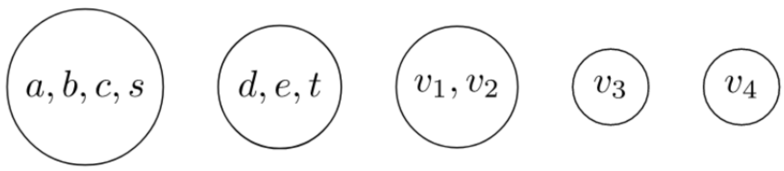
$$\begin{aligned}
&a = b, b = c, d = e, \\
&b = s, d = t, v3 \neq v4
\end{aligned}$$

其中，v3 表示 f(a,v2)，v4 表示 f(b,v1)，v1 表示 g(e)，v2 表示 g(d)。

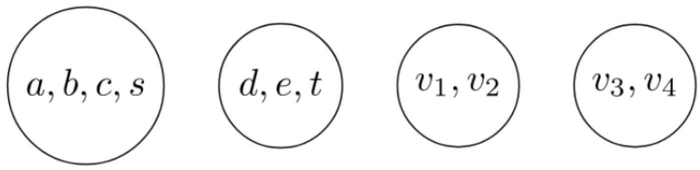
第 2 步，画一张图，每个个体词对应一个圆圈。

第 3 步，查看公式中的等式，把两个相等的个体词所在的圆圈合并。例如根据 $a = b$ 可以把 a 和 b 所在的圆圈合并成一个圆圈。最终得到下面的图。

第 4 步，根据 congruence rule 进一步合并圆圈。因为图中 e 和 d 在一个圆圈，所以 $g(e) = g(d)$，合并 v1 和 v2 的圆圈。



现在 v1 和 v2 在一个圆圈，a 和 b 在一个圆圈，所以 $f(a, v2) = f(b, v1)$，合并 v3 和 v4 的圆圈。



最后，因为合并操作都结束了，现在查看不等式，如果两个不相等的对象在一个圆圈中，说明出错了。例子中的不等式是 $v3 \neq v4$，但是现在 v3 和 v4 在一个圆圈中，EUF 会报错。