SAT Solver Notes

Tom

1 可满足（课本 1.4）

判断一个命题逻辑公式是否可满足称为 SAT 问题。

- 可满足的定义
- 可满足和重言式、矛盾式的关系

SAT 问题是著名的 NPC 问题。

- P 问题：能在多项式时间内给出一个问题的答案，那么这个问题是 P 问题。
- NP 问题：如果给定一个问题的输入和答案，能够在多项式时间内判定答案是否正确，那么这个问题就是 NP 问题。P 问题是 NP 问题的子集。
- NPC 问题：NPC 问题是 NP 问题中最难的那部分问题，它本身是 NP 问题，但比所有的其它 NP 问题都要难。

2 SAT Solver

2.1 定义

自动求解 SAT 问题的工具称为 SAT solver，输入是一个命题逻辑公式，输出是 sat 或 unsat。

2.2 WFF 的判定算法

给定一个表达式，首先要判定这个式子是否有意义才能开始求解，也就是判定它是否是一个合式公式（WFF）。合式公式的定义如下：

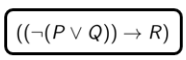
1. 原子命题是合式公式。
2. 如果 A 和 B 是合式公式，那么 $(\neg A),(A\vee B),(A\wedge B),(A\to B),(A\leftrightarrow B)$ 都是合式公式。
3. 当且仅当通过上面两个规则构造的表达式才是合式公式。

这里的定义和课本上的定义在括号的使用上略有不同，但是本质是一样的。

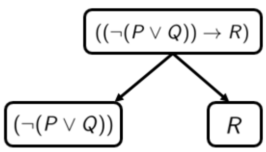
下面介绍自动判定一个表达式是否是合式公式的一个算法。这个算法严格按照上面的定义进行判定，不允许多加括号或者少加括号。输入是一个表达式 P，输出 true 或者 false。

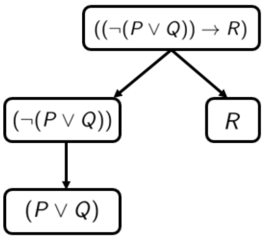
1. 首先构建一棵树，只有一个节点，节点里面是 P。
2. 如果树的所有节点都是原子命题，那么返回 true。
3. 选择一个叶子节点，节点放的不是一个原子命题，是一个表达式 f。
4. 如果 f 开头不是左括号或者结尾不是右括号，那么返回 false。
5. 如果 f=$(\neg Q)$，那么给 f 所在的叶节点新加一个子节点，存放 Q。转步骤 2
6. 现在令 f=(F)，从左到右扫描 F，直到找到第一个非空表达式 A，使得 A 的左括号数量和右括号数量相等。如果找不到这样的 A，那么返回 false。
7. 如果 f=$(A\odot B),\odot\in\{\to,\leftrightarrow,\vee,\wedge\}$，那么给 f 所在的叶子节点新加两个子节点，一个存放 A，一个存放 B。转步骤 2。
8. 返回 false。

下面以 $((\neg(P\vee Q))\to R)$ 为例具体描述算法流程。首先建立一个只有一个节点的树。



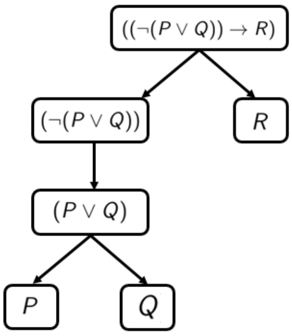
然后我们发现这个节点里并不是原子命题，所以走步骤 3，f=$((\neg(P\vee Q))\to R)$，f 被一对左括号和右括号包围，所以 4 不会返回 false。步骤 5 的条件不满足，现在走步骤 6，F=$(\neg(P\vee Q))\to R$，从左到右扫描直到左右括号数量相等，发现 A=$(\neg(P\vee Q))$。步骤 7，发现 B=Q，$\odot=\to$，那么给当前节点新加两个子节点。





现在从步骤 2 开始，发现左边的叶子节点里不是原子命题，所以现在处理这个叶子节点。步骤 3，f=$(\neg(P \vee Q))$。步骤 4，表达式被左右括号包围。步骤 5，给这个叶子节点新加子节点，放 $(P \vee Q)$。转步骤 2。

现在从步骤 2 开始，发现左边的叶子节点里不是原子命题，所以现在处理这个叶子节点。步骤 3，f=$(P \vee Q)$。步骤 4，表达式被左右括号包围。步骤 5，表达式前面不是 $\neg$，继续往下走。步骤 6，F=$P \vee Q$，A=P。步骤 7，$\odot = \vee$，新加两个子节点，存放 P 和 Q。转步骤 2。



现在所有叶子节点里都是原子命题了，输出 true，这是一个合式公式。

2.3 范式（课本 2.6）

为了方便设计算法和求解，SAT solver 的输入要求是一种标准形式。在命题逻辑中把这种标准形式称为范式，包括合取范式（CNF）、析取范式（DNF）、主范式等。SAT solver 一般采用 CNF 作为输入公式的标准形式，任何公式都可以转化成 CNF。把式子变成 CNF 后就可以用后面的 DPLL 算法求解了。

2.4 DPLL

请看完课本 1.4 和 2.6 的内容后再看本节。

SAT solver 的基本原理是 DPLL 算法，尝试找到一种解释，使得公式的真值为 T。DPLL 算法的基本思想是猜测某个 literal 的真值，然后根据这个真值推导出其它 literal 的真值，如果发现 CNF 的某个子句的真值为 F，说明先前对某个 literal 的真值猜错了，这时进行回溯，重新猜测那个 literal 的真值，如此循环往复，如果发现不论怎么猜，公式的真值都是 F，说明这个公式是不可满足的，否则就是可满足的。

在下面的讲解中，我们把 CNF 的 $\wedge$ 换成”,” 来表示，以 $A \vee \neg B, B \vee \neg C, C \vee A$ 为例讲解 DPLL 算法的流程。DPLL 主要包括下面几条基本规则：

2.4.1 Decide Rule

如果我们从未认定过某个 literal 的真值，那么这个 literal 就叫做 undefined literal。Decide rule 是说，选取一个 undefined literal，猜测它的真值，并标记这个 literal 为 decision literal。例如对于 $A \vee \neg B, B \vee \neg C, C \vee A$，可以按照 decide rule 猜测 A 的真值为 F，并标记 A 为 decision literal。注意，这里如果先选择 B 或者 C 猜测真值也是可以的，另外先猜测 A 的真值是 T 也是可以的。

2.4.2 Unitpropagate Rule

某个 literal 有了真值后，就可以推导出其它一部分 literal 的真值。按照 unitpropagate rule，$A \vee \neg B, B \vee \neg C, C \vee A$ 中 A 的真值是 F，那么可以推导出 B 的真值是 F，因为当且仅当 B 的真值是 F 的时候 $A \vee \neg B$ 的真值才能是 T，整个 CNF 公式的真值才有可能是 T。接着可以发现，我们还可以推导出 C 的真值是 F，因为当前情况下当且仅当 C 的真值是 F 的时候 $B \vee \neg C$ 的真值才能是 T，整个 CNF 公式的真值才有可能是 T。

2.4.3 Backtrack Rule

Backtrack rule 是说如果发现当前 CNF 的某个子句真值是 F，也就是说整个 CNF 的真值已经是 F 的时候，说明对某个 literal 的赋值错了，要进行回溯，找到最近的一个 decision literal 重新赋值，这个真值为 F 的子句称为 conflicting clause。在这个例子中，A 的真值是 F，B 的真值是 F，C 的真值是 F，发现子句 $C \vee A$ 真值是 F，进行回溯，目前最近的一个 decision literal 是 A，A 一开始猜测的真值是 F，按照 backtrack rule，重新猜测 A 的真值是 T，并标记 A 为 non-decision literal，表明 A 的两种真值都考虑过了。由于 B 和 C 的真值都是在猜测 A 真值为 F 的前提下推导出来的，所以现在 B 和 C 的真值无效了，要重新按照 DPLL 的这些规则推导。

2.4.4 Fail Rule

在上面的例子中，由于找到 A 是 decision literal，所以才能使用 backtrack rule 进行回溯。Fail rule 是说如果出现了 conflicting clause 但是又找不到 decision literal，说明这个公式是不可满足的，直接输出 unsat。

2.4.5 Pureliteral Rule

仔细观察例子 $A \vee \neg B, B \vee \neg C, C \vee A$，我们发现其中只有 A 没有 $\neg A$，那么我们可以直接认为 A 的真值是 T，因为 A 是 T 的话那所有包含 A 的子句的真值就都是 T 了，如果在 A 真值是 T 的情况下公式真值还是 F，那么即使把 A 的真值改成 F 公式的真值也还是 F。

类似的，如果发现公式中只有 $\neg A$ 没有 A，那么直接认为 A 的真值是 F。

2.4.6 总结

1. 使用 pureliteral rule 确定某些 literal 的真值，转步骤 2。
2. 用 unitpropagate rule 推导出尽可能多的 literal 的真值，转步骤 3。
3. 如果此时公式真值为 T 了转步骤 4，如果出现了 conflicting clause 就转步骤 5。否则使用 decision rule 猜测某个 undefined literal 的真值，标记其为 decision literal，然后转步骤 2。
4. 输出 sat 以及各个命题变项的真值，结束。
5. 看看当前是否有 decision literal，如果没有，那么按照 fail rule 输出 unsat，结束；否则按照 backtrack rule 进行回溯，找到最近的一个 decision literal，重新猜测它的真值，标记为 non-decision literal，然后转步骤 2。

2.4.7 例子

以下面这个公式为例讲解 DPLL 算法的流程。

$$\emptyset \parallel \quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4}$$

我们用数字表示不同的命题变项，用 $\overline{1}$ 代表 $\neg 1$。把当前认为的各个命题变项的真值写在 $\parallel$ 左边，$\emptyset$ 表示当前还没有判定任何命题变项的真值，$\overline{1}$ 表示当前认为 1 的真值是 F，$\overline{1}^d$ 的意思是当前 1 是 decision literal。

由于不能使用 pureliteral rule，所以使用 decision rule，猜测 1 的真值是 F。

$$\begin{aligned}
\emptyset \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4} \Rightarrow (Decide) \\
\overline{1}^d \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4}
\end{aligned}$$

然后使用 unitpropagate rule 推导出尽可能多的 literal 的真值。

$$\begin{aligned}
\emptyset \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4} \Rightarrow (Decide) \\
\overline{1}^d \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2} \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4}
\end{aligned}$$

$$\begin{aligned}
\emptyset \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4} \Rightarrow (Decide) \\
\overline{1}^d \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2} \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2}\ \overline{4} \parallel &\quad 1 \vee \overline{2},\ \ \overline{1} \vee \overline{2},\ \ 2 \vee 3,\ \ \overline{3} \vee 4,\ \ 1 \vee \overline{4}
\end{aligned}$$

发现当前公式的真值为 T。

$$
\begin{aligned}
\emptyset \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (Decide) \\
\overline{1}^d \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2} \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2}\ \overline{4} \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2}\ \ \overline{4}\ 3 \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4}
\end{aligned}
$$

这时候发现了 conflicting clause 是 $\overline{3} \vee 4$，需要使用 backtrack rule 回溯，最近的 decision literal 是 1，重新猜测 1 的真值是 T。

$$
\begin{aligned}
\emptyset \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (Decide) \\
\overline{1}^d \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2} \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2}\ \overline{4} \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2}\ \ \overline{4}\ 3 \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (Backtrack) \\
1 \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4}
\end{aligned}
$$

然后使用 unitpropagate rule 推导其它 literal 的真值。

$$
\begin{aligned}
\emptyset \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (Decide) \\
\overline{1}^d \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2} \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2}\ \overline{4} \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
\overline{1}^d\ \overline{2}\ \ \overline{4}\ 3 \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (Backtrack) \\
1 \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
1\ \overline{2} \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
1\ \overline{2}\ 3 \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4} \Rightarrow (UnitPropagate) \\
1\ \overline{2}\ 3\ 4 \parallel \quad & 1 \vee \overline{2}, \quad \overline{1} \vee \overline{2}, \quad 2 \vee 3, \quad \overline{3} \vee 4, \quad 1 \vee \overline{4}
\end{aligned}
$$

发现当前公式的真值为 T，输出 sat，若 1、3、4 的真值为 T，2 的真值是 F，那么公式真值为 T。

3 Program Analysis

本节讲解如何把程序转成命题逻辑表达式进行程序分析。程序状态是各个变量的值，程序的执行就是在修改程序状态，分析程序就是判断程序执行的某个时刻是否满足程序员想要的性质。以下面的程序为例，

```
void swap(bool& a, bool& b) {
  a = a^b;
  b = a^b;
  a = a^b;
}
```

程序状态就是 a 和 b 两个 bool 变量，所以初始程序状态 state0 可以用两个命题变项 A0 和 B0 表示，分别代表 a 和 b 当前的值。state0 执行第一行代码变成 state1，state1 用 A1 和 B1 表示 a 和 b 当前的值，满足 $A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0)$ 和 $B1 \leftrightarrow B0$。这样每次走完一条语句，当前 a 和 b 的值就可以用前一个程序状态表示出来。以此类推，我们可以把程序转成下面的命题逻辑公式。

$$
\begin{aligned}
&(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0))\wedge \\
&(B1 \leftrightarrow B0)\wedge \\
&(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1))\wedge \\
&(A2 \leftrightarrow A1)\wedge \\
&(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2))\wedge \\
&(B3 \leftrightarrow B2)
\end{aligned}
$$

现在证明 a 和 b 的值一定互换了，也就是上面的公式可以推导出 a 的值等于 b 的初始值，b 的值等于 a 的初始值，表示成下面的式子。

$$
\begin{aligned}
&(A1 \leftrightarrow (A0 \wedge \neg B0) \vee (\neg A0 \wedge B0))\wedge \\
&(B1 \leftrightarrow B0)\wedge \\
&(B2 \leftrightarrow (A1 \wedge \neg B1) \vee (\neg A1 \wedge B1))\wedge \\
&(A2 \leftrightarrow A1)\wedge \\
&(A3 \leftrightarrow (A2 \wedge \neg B2) \vee (\neg A2 \wedge B2))\wedge \\
&(B3 \leftrightarrow B2)\wedge \\
&\rightarrow \\
&((A3 \leftrightarrow B0) \wedge (B3 \leftrightarrow A0))
\end{aligned}
$$

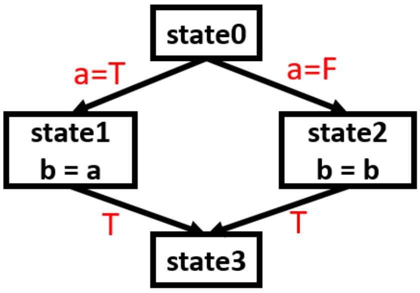
证明 a 和 b 的值一定互换了，就是证明这个公式是永真式，就是证明这个公式的否定是不可满足的。接下来就交给 SAT solver 了。

下面考虑 if 语句。

```
void swap(bool a, bool b) {
  if(a)
```

```
        b = a;
    else
        b = b;
}
```

这个程序可以表示成下图。



如果 state0 的 A 是 T，那么 state3 的 b 等于 state1 的 b，否则 state3 的 b 等于 state2 的 b，所以这个程序转成下面的式子。

$$
\begin{aligned}
&(B1 \leftrightarrow A0)\wedge \\
&(B2 \leftrightarrow B0)\wedge \\
&(B3 \leftrightarrow (A0 \wedge B1) \vee (\neg A0 \wedge B2))
\end{aligned}
$$

如果想证明 b 最后一定是 T，那么证明下面的式子是永真式。

$$
\begin{aligned}
&(B1 \leftrightarrow A0)\wedge \\
&(B2 \leftrightarrow B0)\wedge \\
&(B3 \leftrightarrow (A0 \wedge B1) \vee (\neg A0 \wedge B2)) \\
&\rightarrow \\
&B3
\end{aligned}
$$

也就是证明这个式子的否定是不可满足的，交给 SAT solver，发现 A0=F,B0=F 时 b 最后是 F。

下面考虑循环。如果知道循环最多执行 n 次，那么可以把循环展开 n 次，这样就变成没有循环的程序，用之前的方法解决。如果不知道循环最多执行多少次，那么就直接展开 k 次，k 是自己设定的整数，这样也能分析，但是可能漏掉 bug。

如果有 int 变量，由于计算机用 32 个 bit 存储一个 int，1 个 bit 和 1 个命题变项对应，所以可以用 32 个命题变项表示一个 int 的值，对下面的程序，

```
void swap(int& a, int& b) {
  a = a^b;
  b = a^b;
  a = a^b;
}
```

一个程序状态就包括 64 个命题变项，把程序转成逻辑表达式的方法和之前相同。

程序中的操作符都可以表示成计算机里的数字电路，操作 01 比特，所以操作符自然也能表示成命题逻辑表达式。