形式化方法导引

第 4 章 逻辑问题求解

4.2 理论 - (1) SAT 求解

黄文超

http://staff.ustc.edu.cn/~huangwc/fm.html

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \vDash \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \vDash \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \models \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \models \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \models \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \models \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \models \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \models \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \models \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \models \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \vDash \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \vDash \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \models \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \models \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \models \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \models \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

课前回顾及本章内容

第 4 章: 如何利用 rules 验证 $\mathcal{M} \models \phi$?

- 4.1 应用
  - 将 $\mathcal{M} \models \phi$ 验证问题转化为 validity 问题
  - 将 validity 问题转化为 satifiability 问题
  - 使用 SAT/SMT 工具 Z3 直接求解 satifiability 问题
    - 衍生应用: 软件测试与 Symbolic Execution
- 4.2 本章内容 (理论)
  - 求解 SAT 问题的经典方法?
  - 求解 SMT 问题的经典方法?
  - 其它 SAT 问题的经典方法?

回顾: 定义: Validity

We call ϕ *valid*, if $\vDash \phi$ holds.

回顾: 定义: SAT 问题

SAT is the *decision* problem: given a propositional formula, is it *satisfiable*?

定理:

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.

In other words, ϕ is valid iff $\neg\phi$ is not satisfiable.

总结: Validity 问题可以转化为 SAT 问题

问题: 如何求解 SAT 问题?

回顾：定义：Validity

We call ϕ *valid*, if $\models \phi$ holds.

回顾：定义：SAT 问题

SAT is the *decision* problem: given a propositional formula, is it *satisfiable*?

定理：

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.

In other words, ϕ is valid iff $\neg\phi$ is not satisfiable.

总结：Validity 问题可以转化为 SAT 问题

问题：如何求解 SAT 问题？

回顾: 定义: Validity

We call ϕ *valid*, if $\vDash \phi$ holds.

回顾: 定义: SAT 问题

SAT is the *decision* problem: given a propositional formula, is it *satisfiable*?

定理:

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.

In other words, ϕ is valid iff $\neg\phi$ is not satisfiable.

总结: Validity 问题可以转化为 SAT 问题

问题: 如何求解 SAT 问题?

回顾：定义：Validity

We call ϕ *valid*, if $\models \phi$ holds.

回顾：定义：SAT 问题

SAT is the *decision* problem: given a propositional formula, is it *satisfiable*?

定理：

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.

In other words, ϕ is valid iff $\neg\phi$ is not satisfiable.

总结：Validity 问题可以转化为 SAT 问题

问题：如何求解 SAT 问题？

回顾：定义：Validity

We call ϕ *valid*, if $\models \phi$ holds.

回顾：定义：SAT 问题

SAT is the *decision* problem: given a propositional formula, is it *satisfiable*?

定理：

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.
In other words, ϕ is valid iff $\neg\phi$ is not satisfiable.

总结：Validity 问题可以转化为 SAT 问题
问题：如何求解 SAT 问题？

问题：如何求解 SAT 问题?

回顾：定义：Propositional Logic in *BNF*

$$\phi ::= p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \to \phi)$$

where p stands for any atomic proposition and each occurrence of ϕ to the right of $::=$ stands for any already constructed formula.

Provable equivalence:

$$\begin{array}{ll} \neg(p \wedge q) \dashv\vdash \neg q \vee \neg p & \neg(p \vee q) \dashv\vdash \neg q \wedge \neg p \\ p \to q \dashv\vdash \neg q \to \neg p & p \to q \dashv\vdash \neg p \vee q \\ p \wedge q \to p \dashv\vdash r \vee \neg r & p \wedge q \to r \dashv\vdash p \to (q \to r). \end{array}$$

回顾：rules 太多：推演过于复杂，符号也有冗余

- 减少冗余的符号，设计自动推演算法

问题: 如何减少冗余的符号，设计自动推演算法?

先给部分结果:

- CNF (conjunctive normal form) 合取范式
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \rightarrow\}$

子问题 1: SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

问题：如何减少冗余的符号，设计自动推演算法？
先给部分结果：

- CNF (conjunctive normal form) 合取范式
    - 取如下 (一元、二元) 符号
        - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
    - 取如下 (一元、二元) 符号
        - $\{\wedge, \rightarrow\}$

子问题 1：SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

问题：如何减少冗余的符号，设计自动推演算法?
先给部分结果：

- CNF (conjunctive normal form) 合取范式
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \rightarrow\}$

子问题 1：SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

问题: 如何减少冗余的符号，设计自动推演算法?
先给部分结果:

- CNF (conjunctive normal form) 合取范式
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \rightarrow\}$

子问题 1: SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

问题: 如何减少冗余的符号，设计自动推演算法?
先给部分结果:

- CNF (conjunctive normal form) 合取范式
    - 取如下 (一元、二元) 符号
        - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
    - 取如下 (一元、二元) 符号
        - $\{\wedge, \rightarrow\}$

子问题 1: SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

问题：如何减少冗余的符号，设计自动推演算法？
先给部分结果：

- CNF (conjunctive normal form) 合取范式
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \rightarrow\}$

子问题 1：SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

问题: 如何减少冗余的符号，设计自动推演算法?
先给部分结果:

- CNF (conjunctive normal form) 合取范式
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \rightarrow\}$

子问题 1: SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

问题：如何减少冗余的符号，设计自动推演算法？
先给部分结果：

- CNF (conjunctive normal form) 合取范式
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \vee, \neg\}$
- Horn clauses 霍恩子句
  - 取如下 (一元、二元) 符号
    - $\{\wedge, \rightarrow\}$

子问题 1：SAT 求解

- 使用 CNF
- 如何设计 CNF 的 rules?
- 如何使用 rule 设计算法?

定义: Literal

A *literal* L is either an atom p or the negation of an atom ¬p.

定义: Conjunctive normal form (CNF)

A formula C is in *conjunctive normal form* (*CNF*) if it is a conjunction of *clauses*, where each clause D is a disjunction of literals:

$$L ::= p \mid \neg p$$
$$D ::= L \mid L \vee D$$
$$C ::= D \mid D \wedge C$$

例: Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q
- $(p \vee r) \wedge (\neg p \vee r) \wedge (p \vee \neg r)$

定义: Literal

A *literal* L is either an atom p or the negation of an atom ¬p.

定义: Conjunctive normal form (CNF)

A formula C is in *conjunctive normal form* (*CNF*) if it is a conjunction of *clauses*, where each clause D is a disjunction of literals:

$$L ::= p \mid \neg p$$
$$D ::= L \mid L \vee D$$
$$C ::= D \mid D \wedge C$$

例: Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q
- $(p \vee r) \wedge (\neg p \vee r) \wedge (p \vee \neg r)$

定义: Literal

A *literal* L is either an atom p or the negation of an atom ¬p.

定义: Conjunctive normal form (CNF)

A formula C is in *conjunctive normal form* (*CNF*) if it is a conjunction of *clauses*, where each clause D is a disjunction of literals:

$$L ::= p \mid \neg p$$

$$D ::= L \mid L \vee D$$

$$C ::= D \mid D \wedge C$$

例： Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
    - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q
- $(p \vee r) \wedge (\neg p \vee r) \wedge (p \vee \neg r)$

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).
Step 2: 见下页

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).
Step 2: 见下页

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).
Step 2: 见下页

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).

Step 2: 见下页

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).
Step 2: 见下页

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).
Step 2: 见下页

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).

Step 2: 见下页

Two Problems:

- Problem 1: Checking SAT of a propositional formula
- Problem 2: Checking SAT of a CNF formula

How to solve problem 1?

- Step 1: Transform Problem 1 to Problem 2
- Step 2: Solve Problem 2.

Step 1 (one way by applying the following rules):

- $\neg, \vee, \wedge$: Do nothing
- $\rightarrow$: $p \rightarrow q \equiv \neg p \vee q$
- $\leftrightarrow$: $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

Step 1 (another clever way): Tseitin transformation (见后).
Step 2: 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例： Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \vee r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例： Formulas in CNF

- $(\neg q \lor p \lor r) \land (\neg p \lor r) \land q$
  - clauses: $(\neg q \lor p \lor r)$, $(\neg p \lor r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \lor r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例： Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \vee r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例: Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \vee r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例: Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \vee r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例：Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \vee r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例：Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \vee r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

Idea: Step 2: Checking SAT of a CNF formula

- Design *only one* rule: *resolution rule*

例： Formulas in CNF

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$
  - clauses: $(\neg q \vee p \vee r)$, $(\neg p \vee r)$, q

Is the above formula satisfiable?

- Derive a new clause from the old clauses: $p \vee r$
- Derive another new clause: r
- Answer: sat, $r = \mathbf{T}, p \in \{\mathbf{T}, \mathbf{F}\}, q = \mathbf{T}$

So, how to design the resolution rule? 见下页

定义: Resolution Rule

If there are clauses of the shape $p \vee V$ and $\neg p \vee W$, then the new clause $V \vee W$ may be added.

$$\frac{p \vee V,\ \neg p \vee W}{V \vee W}$$

Discussions:

- Order of literals in a clause does not play a role since $p \vee q \equiv q \vee p$
- Double occurrences of literals may be removed since $p \vee p \equiv p$
- If an *empty clause*, i.e., $\perp$ is derived from a CNF, the CNF is *not satisfiable*.

$$\frac{p,\ \neg p}{\perp}$$

定义: Resolution Rule

If there are clauses of the shape $p \vee V$ and $\neg p \vee W$, then the new clause $V \vee W$ may be added.

$$\frac{p \vee V, \ \neg p \vee W}{V \vee W}$$

Discussions:

- Order of literals in a clause does not play a role since $p \vee q \equiv q \vee p$
- Double occurrences of literals may be removed since $p \vee p \equiv p$
- If an *empty clause*, i.e., $\perp$ is derived from a CNF, the CNF is *not satisfiable*.

$$\frac{p, \ \neg p}{\perp}$$

定义: Resolution Rule

If there are clauses of the shape $p \vee V$ and $\neg p \vee W$, then the new clause $V \vee W$ may be added.

$$\frac{p \vee V, \ \neg p \vee W}{V \vee W}$$

Discussions:

- Order of literals in a clause does not play a role since $p \vee q \equiv q \vee p$
- Double occurrences of literals may be removed since $p \vee p \equiv p$
- If an *empty clause*, i.e., $\perp$ is derived from a CNF, the CNF is *not satisfiable*.

$$\frac{p, \ \neg p}{\perp}$$

定义: Resolution Rule

If there are clauses of the shape $p \vee V$ and $\neg p \vee W$, then the new clause $V \vee W$ may be added.

$$\frac{p \vee V, \ \neg p \vee W}{V \vee W}$$

Discussions:

- Order of literals in a clause does not play a role since $p \vee q \equiv q \vee p$
- Double occurrences of literals may be removed since $p \vee p \equiv p$
- If an *empty clause*, i.e., $\perp$ is derived from a CNF, the CNF is *not satisfiable*.

$$\frac{p, \ \neg p}{\perp}$$

定义: Resolution Rule

If there are clauses of the shape $p \vee V$ and $\neg p \vee W$, then the new clause $V \vee W$ may be added.

$$\frac{p \vee V, \ \neg p \vee W}{V \vee W}$$

Discussions:

- Order of literals in a clause does not play a role since $p \vee q \equiv q \vee p$
- Double occurrences of literals may be removed since $p \vee p \equiv p$
- If an *empty clause*, i.e., $\perp$ is derived from a CNF, the CNF is *not satisfiable*.

$$\frac{p, \ \neg p}{\perp}$$

Example:
We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$
2	$\neg r \vee s$
3	$\neg q \vee r$
4	$\neg r \vee \neg s$
5	$\neg p \vee r$

Example:
We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$	
2	$\neg r \vee s$	
3	$\neg q \vee r$	
4	$\neg r \vee \neg s$	
5	$\neg p \vee r$	
6	$p \vee r$	$(1, 3, q)$

Example:
We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$	
2	$\neg r \vee s$	
3	$\neg q \vee r$	
4	$\neg r \vee \neg s$	
5	$\neg p \vee r$	
6	$p \vee r$	$(1, 3, q)$
7	r	$(5, 6, p)$

Example:
We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$	
2	$\neg r \vee s$	
3	$\neg q \vee r$	
4	$\neg r \vee \neg s$	
5	$\neg p \vee r$	
6	$p \vee r$	$(1, 3, q)$
7	r	$(5, 6, p)$
8	s	$(2, 7, r)$

Example:
We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$	
2	$\neg r \vee s$	
3	$\neg q \vee r$	
4	$\neg r \vee \neg s$	
5	$\neg p \vee r$	
6	$p \vee r$	$(1, 3, q)$
7	r	$(5, 6, p)$
8	s	$(2, 7, r)$
9	$\neg r$	$(4, 8, s)$

Example:
We prove that the CNF consisting of the following clauses 1 to 5 is unsatisfiable

1	$p \vee q$	
2	$\neg r \vee s$	
3	$\neg q \vee r$	
4	$\neg r \vee \neg s$	
5	$\neg p \vee r$	
6	$p \vee r$	$(1, 3, q)$
7	r	$(5, 6, p)$
8	s	$(2, 7, r)$
9	$\neg r$	$(4, 8, s)$
10	$\perp$	$(7, 9, r)$

Remarks for *designing algorithms*:

- A lot of freedom in choice: several other sequences of resolution steps will lead to $\perp$ too.
- Resolution steps on p in which V contains q and W contains $\neg q$ for some q (or conversely) are allowed but useless.
  - In that case the new clause $V \vee W$ is of the shape $q \vee \neg q \vee \cdots$ and hence equivalent to **T**, not containing fruitful information.
- If a clause consists of a single *literal* l (*a unit clause*), then the resolution rule allows to remove the literal $\neg l$ from a clause containing $\neg l$.

Remarks for *designing algorithms*:

- A lot of freedom in choice: several other sequences of resolution steps will lead to $\perp$ too.
- Resolution steps on p in which V contains q and W contains $\neg q$ for some q (or conversely) are allowed but useless.
  - In that case the new clause $V \vee W$ is of the shape $q \vee \neg q \vee \cdots$ and hence equivalent to **T**, not containing fruitful information.
- If a clause consists of a single *literal* l (*a unit clause*), then the resolution rule allows to remove the literal $\neg l$ from a clause containing $\neg l$.

Remarks for *designing algorithms*:

- A lot of freedom in choice: several other sequences of resolution steps will lead to $\perp$ too.
- Resolution steps on p in which V contains q and W contains $\neg q$ for some q (or conversely) are allowed but useless.
    - In that case the new clause $V \vee W$ is of the shape $q \vee \neg q \vee \cdots$ and hence equivalent to **T**, not containing fruitful information.
- If a clause consists of a single *literal* l (*a unit clause*), then the resolution rule allows to remove the literal $\neg l$ from a clause containing $\neg l$.

Remarks for *designing algorithms*:

- A lot of freedom in choice: several other sequences of resolution steps will lead to $\perp$ too.
- Resolution steps on p in which V contains q and W contains $\neg q$ for some q (or conversely) are allowed but useless.
  - In that case the new clause $V \vee W$ is of the shape $q \vee \neg q \vee \cdots$ and hence equivalent to **T**, not containing fruitful information.
- If a clause consists of a single *literal* l (*a unit clause*), then the resolution rule allows to remove the literal $\neg l$ from a clause containing $\neg l$.

Remarks for *designing algorithms*:

- A lot of freedom in choice: several other sequences of resolution steps will lead to $\perp$ too.
- Resolution steps on p in which V contains q and W contains $\neg q$ for some q (or conversely) are allowed but useless.
  - In that case the new clause $V \vee W$ is of the shape $q \vee \neg q \vee \cdots$ and hence equivalent to **T**, not containing fruitful information.
- If a clause consists of a single *literal* l (*a unit clause*), then the resolution rule allows to remove the literal $\neg l$ from a clause containing $\neg l$.

Remarks for *requirements* of the algorithms

- *Soundness*: Correctness of the resolution rule
- *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- *Soundness and Completeness*: A CNF is *unsatisfiable* iff $\perp$ can be derived by *only* using the resolution rule.

Remarks for *requirements* of the algorithms

- *Soundness*: Correctness of the resolution rule
- *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- *Soundness and Completeness*: A CNF is *unsatisfiable* iff $\perp$ can be derived by *only* using the resolution rule.

Remarks for *requirements* of the algorithms

- *Soundness*: Correctness of the resolution rule
- *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- *Soundness and Completeness*: A CNF is *unsatisfiable* iff $\perp$ can be derived by *only* using the resolution rule.

Remarks for *requirements* of the algorithms

- *Soundness*: Correctness of the resolution rule
- *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- *Soundness and Completeness*: A CNF is *unsatisfiable* iff $\perp$ can be derived by *only* using the resolution rule.

Prove using CNF and resolution rules.

定理:

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.

In other words, *ϕ is valid iff $\neg\phi$ is not satisfiable*.

推论 1: How to prove $\psi \vDash \phi$?

Prove $\psi \land \neg\phi$ is unsatisfiable.

- $\neg(\neg\psi \lor \phi) \equiv \psi \land \neg\phi$

推论 2: How to prove $\vDash (\phi \leftrightarrow \psi)$

Prove $(\phi \lor \psi) \land (\neg\phi \lor \psi)$ is unsatisfiable.

- $\neg(\phi \leftrightarrow \psi) \equiv (\phi \lor \psi) \land (\neg\phi \lor \neg\psi)$

Prove using CNF and resolution rules.

定理:

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.

In other words, *ϕ is valid iff $\neg\phi$ is not satisfiable*.

推论 1: How to prove $\psi \vDash \phi$?

Prove $\psi \wedge \neg\phi$ is unsatisfiable.

- $\neg(\neg\psi \vee \phi) \equiv \psi \wedge \neg\phi$

推论 2: How to prove $\vDash (\phi \leftrightarrow \psi)$

Prove $(\phi \vee \psi) \wedge (\neg\phi \vee \psi)$ is unsatisfiable.

- $\neg(\phi \leftrightarrow \psi) \equiv (\phi \vee \psi) \wedge (\neg\phi \vee \neg\psi)$

Prove using CNF and resolution rules.

定理:

Let ϕ be a formula of propositional logic. Then ϕ is *satisfiable* iff $\neg\phi$ is *not valid*.

In other words, *ϕ is valid iff $\neg\phi$ is not satisfiable*.

推论 1: How to prove $\psi \vDash \phi$?

Prove $\psi \wedge \neg\phi$ is unsatisfiable.

- $\neg(\neg\psi \vee \phi) \equiv \psi \wedge \neg\phi$

推论 2: How to prove $\vDash (\phi \leftrightarrow \psi)$

Prove $(\phi \vee \psi) \wedge (\neg\phi \vee \psi)$ is unsatisfiable.

- $\neg(\phi \leftrightarrow \psi) \equiv (\phi \vee \psi) \wedge (\neg\phi \vee \neg\psi)$

Example: A Lewis Carroll Puzzle

1. Good-natured tenured professors are dynamic
2. Grumpy student advisors play slot machines
3. Smokers wearing a cap are phlegmatic
4. Comical student advisors are professors
5. Smoking untenured members are nervous
6. Phlegmatic tenured members wearing caps are comical
7. Student advisors who are not stock market players are scholars
8. Relaxed student advisors are creative
9. Creative scholars who do not play slot machines wear caps
10. Nervous smokers play slot machines
11. Student advisors who play slot machines do not smoke
12. Creative good-natured stock market players wear caps

Then we have to prove that no student advisor is smoking

Example: A Lewis Carroll Puzzle

1. Good-natured tenured professors are dynamic
2. Grumpy student advisors play slot machines
3. Smokers wearing a cap are phlegmatic
4. Comical student advisors are professors
5. Smoking untenured members are nervous
6. Phlegmatic tenured members wearing caps are comical
7. Student advisors who are not stock market players are scholars
8. Relaxed student advisors are creative
9. Creative scholars who do not play slot machines wear caps
10. Nervous smokers play slot machines
11. Student advisors who play slot machines do not smoke
12. Creative good-natured stock market players wear caps

Then we have to prove that no student advisor is smoking

The first step is giving names to every notion to be formalized

name	meaning	opposite
A	good-natured	grumpy
B	tenured	
C	professor	
D	dynamic	phlegmatic
E	wearing a cap	
F	smoke	
G	comical	
H	relaxed	nervous
I	play stock market	
J	scholar	
K	creative	
L	plays slot machine	
M	student advisor	

Example:

1. Good-natured tenured professors are dynamic

$$(A \wedge B \wedge C) \rightarrow D \equiv$$

$$\neg A \vee \neg B \vee \neg C \vee D$$

The first step is giving names to every notion to be formalized

name	meaning	opposite
A	good-natured	grumpy
B	tenured	
C	professor	
D	dynamic	phlegmatic
E	wearing a cap	
F	smoke	
G	comical	
H	relaxed	nervous
I	play stock market	
J	scholar	
K	creative	
L	plays slot machine	
M	student advisor	

Example:

1. Good-natured tenured professors are dynamic

$$(A \wedge B \wedge C) \rightarrow D \equiv$$

$$\neg A \vee \neg B \vee \neg C \vee D$$

The first step is giving names to every notion to be formalized

name	meaning	opposite
A	good-natured	grumpy
B	tenured	
C	professor	
D	dynamic	phlegmatic
E	wearing a cap	
F	smoke	
G	comical	
H	relaxed	nervous
I	play stock market	
J	scholar	
K	creative	
L	plays slot machine	
M	student advisor	

Example:

1. Good-natured tenured professors are dynamic

$$(A \wedge B \wedge C) \rightarrow D \equiv$$

$$\neg A \vee \neg B \vee \neg C \vee D$$

The first step is giving names to every notion to be formalized

name	meaning	opposite
A	good-natured	grumpy
B	tenured	
C	professor	
D	dynamic	phlegmatic
E	wearing a cap	
F	smoke	
G	comical	
H	relaxed	nervous
I	play stock market	
J	scholar	
K	creative	
L	plays slot machine	
M	student advisor	

1. $\neg A \vee \neg B \vee \neg C \vee D$
2. $A \vee \neg M \vee L$
3. $\neg F \vee \neg E \vee \neg D$
4. $\neg G \vee \neg M \vee C$
5. $\neg F \vee B \vee \neg H$
6. $D \vee \neg B \vee \neg E \vee G$
7. $I \vee \neg M \vee J$
8. $\neg H \vee \neg M \vee K$
9. $\neg K \vee \neg J \vee L \vee E$
10. $H \vee \neg F \vee L$
11. $\neg L \vee \neg M \vee \neg F$
12. $\neg K \vee \neg A \vee \neg I \vee E$

So we have to prove that assuming properties 1 to 12, we can conclude $\neg(M \wedge F)$ stating that no student advisor is smoking.
So we have to prove that

$$1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7 \wedge 8 \wedge 9 \wedge 10 \wedge 11 \wedge 12 \wedge M \wedge F$$

is *unsatisfiable*.

回顾：定义：Literal (e.g., unit clause)

A *literal* L is either an atom p or the negation of an atom ¬p.

Method: *Unit resolution* on M and F: *remove* $\neg M$ *and* $\neg F$ *everywhere*

So we have to prove that assuming properties 1 to 12, we can conclude $\neg(M \wedge F)$ stating that no student advisor is smoking.
So we have to prove that

$$1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7 \wedge 8 \wedge 9 \wedge 10 \wedge 11 \wedge 12 \wedge M \wedge F$$

is *unsatisfiable*.

回顾：定义: Literal (e.g., unit clause)

A *literal* L is either an atom p or the negation of an atom ¬p.

Method: *Unit resolution* on M and F: *remove* $\neg M$ *and* $\neg F$ *everywhere*

So we have to prove that assuming properties 1 to 12, we can conclude $\neg(M \wedge F)$ stating that no student advisor is smoking.
So we have to prove that

$$1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7 \wedge 8 \wedge 9 \wedge 10 \wedge 11 \wedge 12 \wedge M \wedge F$$

is *unsatisfiable*.

回顾：定义: Literal (e.g., unit clause)

A *literal* L is either an atom p or the negation of an atom ¬p.

Method: *Unit resolution* on M and F: *remove* $\neg M$ *and* $\neg F$ *everywhere*

Method: *Unit resolution* on M and F: *remove* $\neg M$ *and* $\neg F$ everywhere

1. $\neg A \vee \neg B \vee \neg C \vee D$		1. $\neg A \vee \neg B \vee \neg C \vee D$
2. $A \vee \neg M \vee L$		2. $A \vee L$
3. $\neg F \vee \neg E \vee \neg D$		3. $\neg E \vee \neg D$
4. $\neg G \vee \neg M \vee C$		4. $\neg G \vee C$
5. $\neg F \vee B \vee \neg H$		5. $B \vee \neg H$
6. $D \vee \neg B \vee \neg E \vee G$		6. $D \vee \neg B \vee \neg E \vee G$
7. $I \vee \neg M \vee J$	$\Rightarrow$	7. $I \vee J$
8. $\neg H \vee \neg M \vee K$		8. $\neg H \vee K$
9. $\neg K \vee \neg J \vee L \vee E$		9. $\neg K \vee \neg J \vee L \vee E$
10. $H \vee \neg F \vee L$		10. $H \vee L$
11. $\neg L \vee \neg M \vee \neg F$		11. $\neg L$
12. $\neg K \vee \neg A \vee \neg I \vee E$		12. $\neg K \vee \neg A \vee \neg I \vee E$

Method: *Unit resolution* on $\neg L$: *remove L everywhere*

1. $\neg A \vee \neg B \vee \neg C \vee D$
2. A
3. $\neg E \vee \neg D$
4. $\neg G \vee C$
5. $B \vee \neg H$
6. $D \vee \neg B \vee \neg E \vee G$
7. $I \vee J$
8. $\neg H \vee K$
9. $\neg K \vee \neg J \vee E$
10. H
11. $\neg K \vee \neg A \vee \neg I \vee E$

$\Leftarrow$

1. $\neg A \vee \neg B \vee \neg C \vee D$
2. $A \vee L$
3. $\neg E \vee \neg D$
4. $\neg G \vee C$
5. $B \vee \neg H$
6. $D \vee \neg B \vee \neg E \vee G$
7. $I \vee J$
8. $\neg H \vee K$
9. $\neg K \vee \neg J \vee L \vee E$
10. $H \vee L$
11. $\neg L$
12. $\neg K \vee \neg A \vee \neg I \vee E$

Method: *Unit resolution* on A and H: *remove* $\neg A$ *and* $\neg H$ *everywhere*

1. $\neg A \vee \neg B \vee \neg C \vee D$		
2. A		1. $\neg B \vee \neg C \vee D$
3. $\neg E \vee \neg D$		2. $\neg E \vee \neg D$
4. $\neg G \vee C$		3. $\neg G \vee C$
5. $B \vee \neg H$		4. B
6. $D \vee \neg B \vee \neg E \vee G$	$\Rightarrow$	5. $D \vee \neg B \vee \neg E \vee G$
7. $I \vee J$		6. $I \vee J$
8. $\neg H \vee K$		7. K
9. $\neg K \vee \neg J \vee E$		8. $\neg K \vee \neg J \vee E$
10. H		9. $\neg K \vee \neg I \vee E$
11. $\neg K \vee \neg A \vee \neg I \vee E$		

Method: *Unit resolution* on K and B: *remove* $\neg K$ *and* $\neg B$ *everywhere*

		1. $\neg B \vee \neg C \vee D$
1. $\neg C \vee D$		2. $\neg E \vee \neg D$
2. $\neg E \vee \neg D$		3. $\neg G \vee C$
3. $\neg G \vee C$		4. B
4. $D \vee \neg E \vee G$	$\Leftarrow$	5. $D \vee \neg B \vee \neg E \vee G$
5. $I \vee J$		6. $I \vee J$
6. $\neg J \vee E$		7. K
7. $\neg I \vee E$		8. $\neg K \vee \neg J \vee E$
		9. $\neg K \vee \neg I \vee E$

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Normal Resolution

8. $J \vee E \quad (5, 7, I)$
9. $E \quad (6, 8, J)$

$\Rightarrow$

Unit resolution on E:

1. $\neg C \vee D$
2. $\neg D$
3. $\neg G \vee C$
4. $D \vee G$

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Normal Resolution

8. $J \vee E \quad (5, 7, I)$
9. $E \quad (6, 8, J)$

$\Rightarrow$

Unit resolution on E:

1. $\neg C \vee D$
2. $\neg D$
3. $\neg G \vee C$
4. $D \vee G$

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Normal Resolution

8. $J \vee E \quad (5, 7, I)$
9. $E \quad (6, 8, J)$

$\Rightarrow$

Unit resolution on E:

1. $\neg C \vee D$
2. $\neg D$
3. $\neg G \vee C$
4. $D \vee G$

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Normal Resolution

8. $J \vee E \quad (5, 7, I)$
9. $E \quad (6, 8, J)$

$\Rightarrow$

Unit resolution on E:

1. $\neg C \vee D$
2. $\neg D$
3. $\neg G \vee C$
4. $D \vee G$

1. $\neg C \vee D$
2. $\neg E \vee \neg D$
3. $\neg G \vee C$
4. $D \vee \neg E \vee G$
5. $I \vee J$
6. $\neg J \vee E$
7. $\neg I \vee E$

Normal Resolution

8. $J \vee E \quad (5, 7, I)$
9. $E \quad (6, 8, J)$

$\Rightarrow$

Unit resolution on E:

1. $\neg C \vee D$
2. $\neg D$
3. $\neg G \vee C$
4. $D \vee G$

Unit resolution on $\neg D$:

1. $\neg C$
2. $\neg G \vee C$
3. G

$\Leftarrow$

1. $\neg C \vee D$
2. $\neg D$
3. $\neg G \vee C$
4. $D \vee G$

Unit resolution on $\neg C$ and G:

1. $\neg C$
2. $\neg G \vee C$
3. G

$\Rightarrow$

1. $\perp$

Result: *unsatisfiable*, i.e., it is proved that no student advisor is smoking.
Conclusion: apply *unit resolution* as long as possible.
下一个问题: 如果不能使用 unit resolution, 如何设计算法?

Unit resolution on $\neg C$ and G:

1. $\neg C$
2. $\neg G \vee C$
3. G

$\Rightarrow$

1. $\perp$

Result: *unsatisfiable*, i.e., it is proved that no student advisor is smoking.

Conclusion: apply *unit resolution* as long as possible.

下一个问题: 如果不能使用 unit resolution, 如何设计算法?

Unit resolution on $\neg C$ and G:

1. $\neg C$
2. $\neg G \vee C$
3. G

$\Rightarrow$

1. $\perp$

Result: *unsatisfiable*, i.e., it is proved that no student advisor is smoking.
Conclusion: apply *unit resolution* as long as possible.
下一个问题: 如果不能使用 unit resolution, 如何设计算法?

Unit resolution on $\neg C$ and G:

1. $\neg C$
2. $\neg G \vee C$
3. G

$\Rightarrow$

1. $\perp$

Result: *unsatisfiable*, i.e., it is proved that no student advisor is smoking.
Conclusion: apply *unit resolution* as long as possible.
下一个问题: 如果不能使用 unit resolution, 如何设计算法?

A classical algorithm: *DPLL*

- After more than *50 years* the DPLL procedure still forms the basis for most efficient complete SAT solvers.

Idea of DPLL:

- First apply unit resolution as long as possible
- If you cannot proceed by unit resolution or trivial observations
  - choose a variable p
  - introduce the cases p and $\neg p$
  - and for both cases go on recursively.

A classical algorithm: *DPLL*

- After more than *50 years* the DPLL procedure still forms the basis for most efficient complete SAT solvers.

Idea of DPLL:

- First apply unit resolution as long as possible
- If you cannot proceed by unit resolution or trivial observations
  - choose a variable p
  - introduce the cases p and $\neg p$
  - and for both cases go on recursively.

A classical algorithm: *DPLL*

- After more than *50 years* the DPLL procedure still forms the basis for most efficient complete SAT solvers.

Idea of DPLL:

- First apply unit resolution as long as possible
- If you cannot proceed by unit resolution or trivial observations
    - choose a variable p
    - introduce the cases p and $\neg p$
    - and for both cases go on recursively.

A classical algorithm: *DPLL*

- After more than *50 years* the DPLL procedure still forms the basis for most efficient complete SAT solvers.

Idea of DPLL:

- First apply unit resolution as long as possible
- If you cannot proceed by unit resolution or trivial observations
  - choose a variable p
  - introduce the cases p and $\neg p$
  - and for both cases go on recursively.

A classical algorithm: *DPLL*

- After more than *50 years* the DPLL procedure still forms the basis for most efficient complete SAT solvers.

Idea of DPLL:

- First apply unit resolution as long as possible
- If you cannot proceed by unit resolution or trivial observations
  - choose a variable p
  - introduce the cases p and $\neg p$
  - and for both cases go on recursively.

A classical algorithm: *DPLL*

- After more than *50 years* the DPLL procedure still forms the basis for most efficient complete SAT solvers.

Idea of DPLL:

- First apply unit resolution as long as possible
- If you cannot proceed by unit resolution or trivial observations
  - choose a variable p
  - introduce the cases p and $\neg p$
  - and for both cases go on recursively.

算法: Unit Resolution unit-resol(X)

Input X: a set of clauses.

Algorithm: as long as a clause occurs in X consisting of one literal l (a unit clause):

- remove $\neg l$ from all clauses in X containing $\neg l$
  - i.e., unit resolution
- remove all clauses containing l
  - i.e., remove redundant clauses

算法: Unit Resolution unit-resol(X)

Input X: a set of clauses.
Algorithm: as long as a clause occurs in X consisting of one literal l (a unit clause):

- remove $\neg l$ from all clauses in X containing $\neg l$
  - i.e., unit resolution
- remove all clauses containing l
  - i.e., remove redundant clauses

算法: Unit Resolution unit-resol(X)

Input X: a set of clauses.

Algorithm: as long as a clause occurs in X consisting of one literal l (a unit clause):

- remove $\neg l$ from all clauses in X containing $\neg l$
  - i.e., unit resolution
- remove all clauses containing l
  - i.e., remove redundant clauses

算法: Unit Resolution unit-resol(X)

Input X: a set of clauses.
Algorithm: as long as a clause occurs in X consisting of one literal l (a unit clause):

- remove $\neg l$ from all clauses in X containing $\neg l$
  - i.e., unit resolution
- remove all clauses containing l
  - i.e., remove redundant clauses

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\perp \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\perp \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\perp \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\bot \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\bot$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\bot \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\bot$ for all cases, so unsat

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\perp \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(见右)

- Terminates since every recursive call decreases number of variables

DPLL($X \cup \{p\}$) and DPLL($X \cup \{\neg p\}$)

- If 'satisfiable' is returned from either one, then all involved unit clauses yield a satisfying assignment
- Otherwise, it is a big case analysis yielding $\perp$ for all cases, so unsat

例 1

Consider the CNF consisting of the following nine clauses

$$
\begin{array}{lll}
\neg p \vee \neg s & p \vee r & \neg s \vee t \\
\neg p \vee \neg r & p \vee s & q \vee s \\
\neg q \vee \neg t & r \vee t & q \vee \neg r
\end{array}
$$

No unit resolution possible: choose variable p

Add p, unit resolution:	Add $\neg p$, unit resolution:
$\neg s, \neg r$	r, s
q (use $\neg s$), t (use $\neg r$)	q (use r), t (use s)
$\neg t$ (use q)	$\neg t$ (use q)
$\bot$	$\bot$

Both branches yield $\bot$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
q (use r), t (use s)
$\neg t$ (use q)
$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:

$\neg s, \neg r$

q (use $\neg s$), t (use $\neg r$)

$\neg t$ (use q)

$\perp$

Add $\neg p$, unit resolution:

r, s

q (use r), t (use s)

$\neg t$ (use q)

$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
q (use r), t (use s)
$\neg t$ (use q)
$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:	Add $\neg p$, unit resolution:
$\neg s, \neg r$	r, s
q (use $\neg s$), t (use $\neg r$)	q (use r), t (use s)
$\neg t$ (use q)	$\neg t$ (use q)
$\bot$	$\bot$

Both branches yield $\bot$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:

$\neg s, \neg r$

q (use $\neg s$), t (use $\neg r$)

$\neg t$ (use q)

$\bot$

Add $\neg p$, unit resolution:

r, s

q (use r), t (use s)

$\neg t$ (use q)

$\bot$

Both branches yield $\bot$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:	Add $\neg p$, unit resolution:
$\neg s, \neg r$	r, s
q (use $\neg s$), t (use $\neg r$)	q (use r), t (use s)
$\neg t$ (use q)	$\neg t$ (use q)
$\perp$	$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \lor \neg s & p \lor r & \neg s \lor t \\ \neg p \lor \neg r & p \lor s & q \lor s \\ \neg q \lor \neg t & r \lor t & q \lor \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:

$\neg s, \neg r$

q (use $\neg s$), t (use $\neg r$)

$\neg t$ (use q)

$\perp$

Add $\neg p$, unit resolution:

r, s

q (use r), t (use s)

$\neg t$ (use q)

$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$
\begin{array}{lll}
\neg p \vee \neg s & p \vee r & \neg s \vee t \\
\neg p \vee \neg r & p \vee s & q \vee s \\
\neg q \vee \neg t & r \vee t & q \vee \neg r
\end{array}
$$

No unit resolution possible: choose variable p

Add p, unit resolution:

$\neg s, \neg r$

q (use $\neg s$), t (use $\neg r$)

$\neg t$ (use q)

$\perp$

Add $\neg p$, unit resolution:

r, s

q (use r), t (use s)

$\neg t$ (use q)

$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
q (use r), t (use s)
$\neg t$ (use q)
$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$
\begin{array}{lll}
\neg p \vee \neg s & p \vee r & \neg s \vee t \\
\neg p \vee \neg r & p \vee s & q \vee s \\
\neg q \vee \neg t & r \vee t & q \vee \neg r
\end{array}
$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
q (use r), t (use s)
$\neg t$ (use q)
$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:	Add $\neg p$, unit resolution:
$\neg s, \neg r$	r, s
q (use $\neg s$), t (use $\neg r$)	q (use r), t (use s)
$\neg t$ (use q)	$\neg t$ (use q)
$\perp$	$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
q (use r), t (use s)
$\neg t$ (use q)
$\perp$

Both branches yield $\perp$, so original CNF is unsatisfiable

例 2

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
t (use s)
$\neg q$ (use t)

Yields satisfying assignment $p = q = \mathbf{F}, r = s = t = \mathbf{T}$

例 2

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
t (use s)
$\neg q$ (use t)

Yields satisfying assignment $p = q = \mathbf{F}, r = s = t = \mathbf{T}$

例 2

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
t (use s)
$\neg q$ (use t)

Yields satisfying assignment $p = q = \mathbf{F}, r = s = t = \mathbf{T}$

例 2

Consider the CNF consisting of the following eight clauses

$$
\begin{array}{lll}
\neg p \vee \neg s & p \vee r & \neg s \vee t \\
\neg p \vee \neg r & p \vee s & q \vee s \\
\neg q \vee \neg t & r \vee t &
\end{array}
$$

No unit resolution possible: choose variable p

Add p, unit resolution:

$\neg s, \neg r$

q (use $\neg s$), t (use $\neg r$)

$\neg t$ (use q)

$\perp$

Add $\neg p$, unit resolution:

r, s

t (use s)

$\neg q$ (use t)

Yields satisfying assignment $p = q = \mathbf{F}, r = s = t = \mathbf{T}$

例 2

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
t (use s)
$\neg q$ (use t)

Yields satisfying assignment $p = q = \mathbf{F}, r = s = t = \mathbf{T}$

例 2

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & \end{array}$$

No unit resolution possible: choose variable p

Add p, unit resolution:
$\neg s, \neg r$
q (use $\neg s$), t (use $\neg r$)
$\neg t$ (use q)
$\perp$

Add $\neg p$, unit resolution:
r, s
t (use s)
$\neg q$ (use t)

Yields satisfying assignment $p = q = \mathbf{F}, r = s = t = \mathbf{T}$

Concluding:

- DPLL is a complete method (证明略) for satisfiability, based on unit resolution and case analysis
  - *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- Efficiency strongly depends on the choice of the variable
- Current SAT solvers follow this scheme, combined with good heuristics for variable choice and several optimizations

Concluding:

- DPLL is a complete method (证明略) for satisfiability, based on unit resolution and case analysis
  - *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- Efficiency strongly depends on the choice of the variable
- Current SAT solvers follow this scheme, combined with good heuristics for variable choice and several optimizations

Concluding:

- DPLL is a complete method (证明略) for satisfiability, based on unit resolution and case analysis
  - *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- Efficiency strongly depends on the choice of the variable
- Current SAT solvers follow this scheme, combined with good heuristics for variable choice and several optimizations

Concluding:

- DPLL is a complete method (证明略) for satisfiability, based on unit resolution and case analysis
  - *Completeness*: If a CNF is unsatisfiable, then this can be derived by *only* applying the resolution rule
- Efficiency strongly depends on the choice of the variable
- Current SAT solvers follow this scheme, combined with good heuristics for variable choice and several optimizations

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\bot \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
  - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\bot \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
    - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\bot \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(略)

问题 1: How to choose variable p? (稍等)

问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
  - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\bot \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
  - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\bot \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
    - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\bot \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
  - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\bot \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
  - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
    - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
  - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

CDCL: conflict driven clause learning

- An efficient way to implement DPLL, extended by *optimizations*

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

问题 1: How to choose variable p? (稍等)
问题 2: *How is the computation cost?*

A *naive* implementation

- *cost*: make *copies* of the full CNF X at *every recursive call*

A *better* solution

- *backtracking* instead of recursive call
    - Keep *track* of *a list* M of literals that has been chosen and derived during the execution of DPLL
- *mimic*: unit-resol and case analysis

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \vDash l$, if l occurs in M
- $M \vDash \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \vDash l$, if l occurs in M
- $M \vDash \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

思路:How to keep track of M?

M will be *extended* if

- a case analysis starts: **Decide** or
- a literal is derived by unit resolution: **UnitPropagate**

Part of M will be *removed* if

- case analysis is continued after finding a contradiction: **Backtrack**

定义: list M 的相关定义

For a *literal* l, we write

- $M \models l$, if l occurs in M
- $M \models \neg C$ if $\neg l$ occurs in M for every literal l in C
- l is *undefined* in M if neither l nor $\neg l$ occurs in M

If all literals in M occur as a unit clause, and there is a clause $C \vee l$ satisfying $M \models \neg C$, then by unit resolution all literals in C can be removed

Then the single literal l remains, so the new unit clause l can be derived

This justifies the first rule

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

If all literals in M occur as a unit clause, and there is a clause $C \vee l$ satisfying $M \vDash \neg C$, then by unit resolution all literals in C can be removed

Then the single literal l remains, so the new unit clause l can be derived

This justifies the first rule

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \vDash \neg C$

If all literals in M occur as a unit clause, and there is a clause $C \vee l$ satisfying $M \vDash \neg C$, then by unit resolution all literals in C can be removed

Then the single literal l remains, so the new unit clause l can be derived

This justifies the first rule

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \vDash \neg C$

If all literals in M occur as a unit clause, and there is a clause $C \vee l$ satisfying $M \vDash \neg C$, then by unit resolution all literals in C can be removed

Then the single literal l remains, so the new unit clause l can be derived

This justifies the first rule

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \vDash \neg C$

If no *UnitPropagate* is possible, we have to start a case analysis by **Decide**

Rule 2: Decide

$$M \Longrightarrow M l^d$$

if l is undefined in M

Here the added literal l is marked by $'d'$ (decision literal) in order to be able to do backtracking = go back to last start of case analysis

If no *UnitPropagate* is possible, we have to start a case analysis by **Decide**

Rule 2: Decide

$$M \Longrightarrow M l^d$$

if l is undefined in M

Here the added literal l is marked by 'd' (decision literal) in order to be able to do backtracking = go back to last start of case analysis

If no *UnitPropagate* is possible, we have to start a case analysis by **Decide**

Rule 2: Decide

$$M \Longrightarrow M l^d$$

if l is undefined in M

Here the added literal l is marked by 'd' (decision literal) in order to be able to do backtracking = go back to last start of case analysis

Rule 3: Backtrack

$$Ml^d N \Longrightarrow M \neg l$$

if $Ml^d N \vDash \neg C$ for a clause C in the CNF and N contains no decision literals

So **Backtrack** applies if a contradiction is found, and everything in M behind the last decision literal is removed, and this decision literal is replaced by its negation

Note that this negation is not decision literal anymore: now it has been derived

Rule 3: Backtrack

$$Ml^dN \Longrightarrow M\neg l$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and N contains no decision literals

So **Backtrack** applies if a contradiction is found, and everything in M behind the last decision literal is removed, and this decision literal is replaced by its negation

Note that this negation is not decision literal anymore: now it has been derived

Rule 3: Backtrack

$$Ml^dN \Longrightarrow M\neg l$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and N contains no decision literals

So **Backtrack** applies if a contradiction is found, and everything in M behind the last decision literal is removed, and this decision literal is replaced by its negation

Note that this negation is not decision literal anymore: now it has been derived

In case a contradiction is found, while M does not contain any decision literal, then we have a contradiction for the full formula, so we have derived that the formula is *unsatisfiable*.

This is expressed by the last rule **Fail**

Rule 4: Fail

$$M \Longrightarrow \text{fail}$$

if $M \models \neg C$ for a clause C in the CNF and M contains no decision literals

In case a contradiction is found, while M does not contain any decision literal, then we have a contradiction for the full formula, so we have derived that the formula is *unsatisfiable*.

This is expressed by the last rule **Fail**

Rule 4: Fail

$$M \Longrightarrow \text{fail}$$

if $M \models \neg C$ for a clause C in the CNF and M contains no decision literals

In case a contradiction is found, while M does not contain any decision literal, then we have a contradiction for the full formula, so we have derived that the formula is *unsatisfiable*.

This is expressed by the last rule **Fail**

Rule 4: Fail

$$M \Longrightarrow \text{fail}$$

if $M \models \neg C$ for a clause C in the CNF and M contains no decision literals

算法思路: How to use M instead of recursive call

Start with M being empty and apply the rules as long as possible always ends in either

- *fail*, proving that the CNF is *unsatisfiable*, or
- a *list* M containing p or $\neg p$ for every variable p, yielding a *satisfying* assignment

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$
\begin{array}{lll}
\neg p \vee \neg s & p \vee r & \neg s \vee t \\
\neg p \vee \neg r & p \vee s & q \vee s \\
\neg q \vee \neg t & r \vee t & q \vee \neg r
\end{array}
$$

List M:

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 2: Decide

$$M \Longrightarrow M l^d$$

if l is undefined in M

List M: p^d

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: $p^d \; \neg s$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: $p^d \ \neg s \ \neg r$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: $p^d \; \neg s \; \neg r \; t$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{ccc} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: $p^d \ \neg s \ \neg r \ t \ q$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$
\begin{array}{lll}
\neg p \vee \neg s & p \vee r & \neg s \vee t \\
\neg p \vee \neg r & p \vee s & q \vee s \\
\neg q \vee \neg t & r \vee t & q \vee \neg r
\end{array}
$$

Rule 3: Backtrack

$$M l^d N \Longrightarrow M \neg l$$

if $M l^d N \vDash \neg C$ for a clause C in the CNF and N contains no decision literals

List M: $p^d\ \neg s\ \neg r\ t\ q$

Backtrack:

重新计算：例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 3: Backtrack

$$M l^d N \Longrightarrow M \neg l$$

if $M l^d N \models \neg C$ for a clause C in the CNF and N contains no decision literals

List M: p^d $\neg s$ $\neg r$ t q
Backtrack:
$\neg p$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$
\begin{array}{lll}
\neg p \vee \neg s & p \vee r & \neg s \vee t \\
\neg p \vee \neg r & p \vee s & q \vee s \\
\neg q \vee \neg t & r \vee t & q \vee \neg r
\end{array}
$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: p^d $\neg s$ $\neg r$ t q
Backtrack:
$\neg p$ r

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: $p^d\ \neg s\ \neg r\ t\ q$
Backtrack:
$\neg p\ r\ s$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: $p^d \ \neg s \ \neg r \ t \ q$
Backtrack:
$\neg p \ r \ s \ q$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

if l is undefined in M and the CNF contains a clause $C \vee l$ satisfying $M \models \neg C$

List M: $p^d \ \neg s \ \neg r \ t \ q$
Backtrack:
$\neg p \ r \ s \ q \ t$

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

Rule 4: Fail

$$M \Longrightarrow \text{fail}$$

if $M \vDash \neg C$ for a clause C in the CNF and M contains no decision literals

List M: p^d $\neg s$ $\neg r$ t q
Backtrack:
$\neg p$ r s q t
Fail

重新计算: 例 1

Consider the CNF consisting of the following nine clauses

$$\begin{array}{lll} \neg p \vee \neg s & p \vee r & \neg s \vee t \\ \neg p \vee \neg r & p \vee s & q \vee s \\ \neg q \vee \neg t & r \vee t & q \vee \neg r \end{array}$$

List M: $p^d \ \neg s \ \neg r \ t \ q$
Backtrack:
$\neg p \ r \ s \ q \ t$
Fail
So we have proved that the CNF is *unsatisfiable*

Concluding,

- We saw a way to implement DPLL while only working on the original CNF
- Combined with the optimizations of the *next section*, this is *Conflict Driven Clause Learning*, CDCL, as is used in all current powerful SAT solvers.

Concluding,

- We saw a way to implement DPLL while only working on the original CNF
- Combined with the optimizations of the *next section*, this is *Conflict Driven Clause Learning*, CDCL, as is used in all current powerful SAT solvers.

Concluding,

- We saw a way to implement DPLL while only working on the original CNF
- Combined with the optimizations of the *next section*, this is *Conflict Driven Clause Learning*, CDCL, as is used in all current powerful SAT solvers.

算法思路: DPLL(X)

X:=unit-resol(X)
if $\bot \in X$ then
  return(unsatisfiable)
if $X = \emptyset$ then
  return(satisfiable)
if $\bot \notin X$ then
  choose variable p in X
    DPLL($X \cup \{p\}$)
    DPLL($X \cup \{\neg p\}$)
  return ?(略)

CDCL Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

CDCL Rule 2: Decide

$$M \Longrightarrow Ml^d$$

CDCL Rule 3: Backtrack

$$Ml^dN \Longrightarrow M\neg l$$

CDCL Rule 4: Fail

$$M \Longrightarrow \text{fail}$$

回顾: 问题 1: How to choose variable p?

- 换为另一个问题: How to choose l for case analysis in **Decide**?

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

CDCL Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

CDCL Rule 2: Decide

$$M \Longrightarrow Ml^d$$

CDCL Rule 3: Backtrack

$$Ml^d N \Longrightarrow M\neg l$$

CDCL Rule 4: Fail

$$M \Longrightarrow \text{fail}$$

回顾: 问题 1: How to choose variable p?

- 换为另一个问题: How to choose l for case analysis in **Decide**?

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal

算法思路: DPLL(X)

X:=unit-resol(X)
if $\perp \in X$ then
    return(unsatisfiable)
if $X = \emptyset$ then
    return(satisfiable)
if $\perp \notin X$ then
    choose variable p in X
        DPLL($X \cup \{p\}$)
        DPLL($X \cup \{\neg p\}$)
    return ?(略)

CDCL Rule 1: UnitPropagate

$$M \Longrightarrow Ml$$

CDCL Rule 2: Decide

$$M \Longrightarrow Ml^d$$

CDCL Rule 3: Backtrack

$$Ml^dN \Longrightarrow M\neg l$$

CDCL Rule 4: Fail

$$M \Longrightarrow \text{fail}$$

回顾: 问题 1: How to choose variable p?

- 换为另一个问题: How to choose l for case analysis in **Decide**?

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal

Consider the following example:

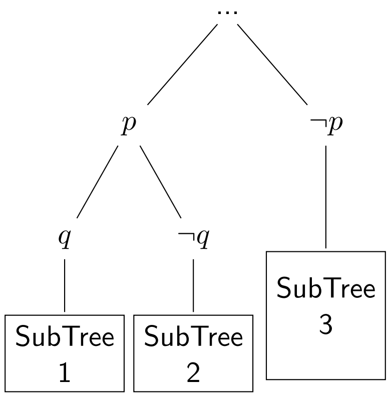
- $Mp^dq^d \ldots$ // explore SubTree 1
- $Mp^d\neg q \ldots$ // explore SubTree 2
- $M\neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \models \neg q \vee t$ and $M \models \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^dq^d \cdots \Longrightarrow M\neg q$
- Instead of $Mp^dq^d \cdots \Longrightarrow Mp^d\neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

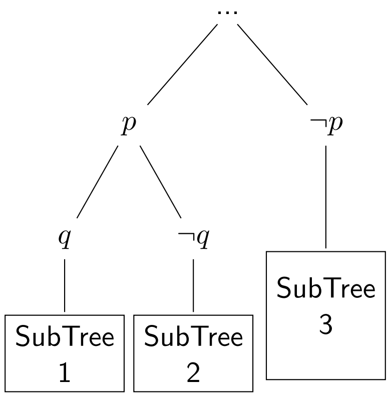
- $Mp^d q^d \ldots$ // explore SubTree 1
- $Mp^d \neg q \ldots$ // explore SubTree 2
- $M \neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \models \neg q \vee t$ and $M \models \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^d q^d \cdots \Longrightarrow M \neg q$
- Instead of $Mp^d q^d \cdots \Longrightarrow Mp^d \neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

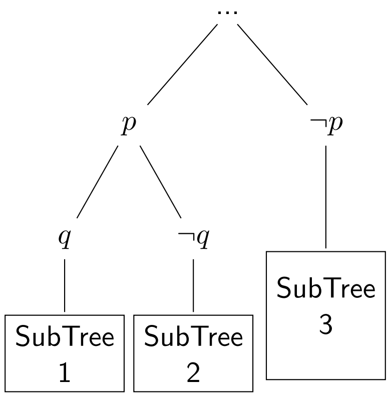
- $Mp^d q^d \ldots$ // explore SubTree 1
- $Mp^d \neg q \ldots$ // explore SubTree 2
- $M \neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \vDash \neg q \vee t$ and $M \vDash \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^d q^d \cdots \Longrightarrow M \neg q$
- Instead of $Mp^d q^d \cdots \Longrightarrow Mp^d \neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

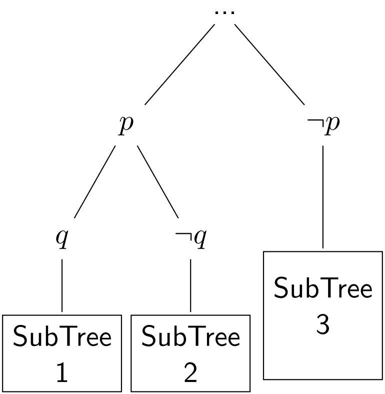
- $Mp^d q^d \ldots$ // explore SubTree 1
- $Mp^d \neg q \ldots$ // explore SubTree 2
- $M \neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \vDash \neg q \vee t$ and $M \vDash \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^d q^d \cdots \Longrightarrow M \neg q$
- Instead of $Mp^d q^d \cdots \Longrightarrow Mp^d \neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

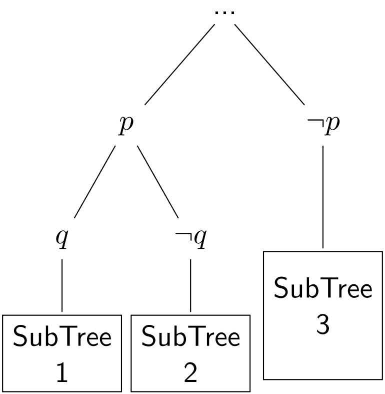
- $Mp^dq^d \ldots$ // explore SubTree 1
- $Mp^d\neg q \ldots$ // explore SubTree 2
- $M\neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \models \neg q \vee t$ and $M \models \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^dq^d \cdots \Longrightarrow M\neg q$
- Instead of $Mp^dq^d \cdots \Longrightarrow Mp^d\neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

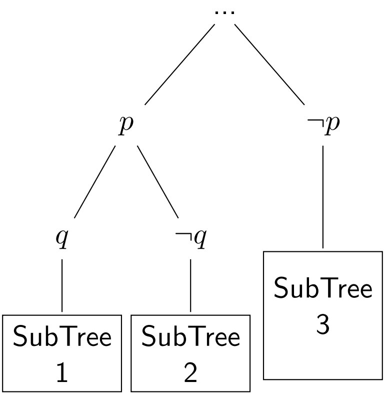
- $Mp^dq^d\ldots$ // explore SubTree 1
- $Mp^d\neg q\ldots$ // explore SubTree 2
- $M\neg p\ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \models \neg q \vee t$ and $M \models \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^dq^d\cdots \Longrightarrow M\neg q$
- Instead of $Mp^dq^d\cdots \Longrightarrow Mp^d\neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

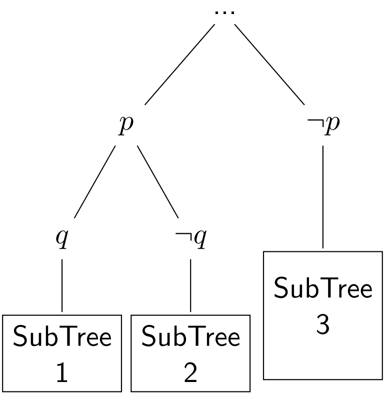
- $Mp^d q^d \ldots$ // explore SubTree 1
- $Mp^d \neg q \ldots$ // explore SubTree 2
- $M \neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \vDash \neg q \vee t$ and $M \vDash \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^d q^d \cdots \Longrightarrow M \neg q$
- Instead of $Mp^d q^d \cdots \Longrightarrow Mp^d \neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

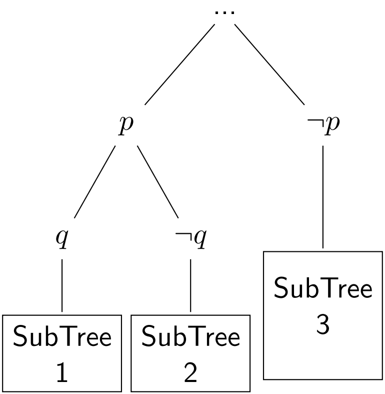
- $Mp^d q^d \ldots$ // explore SubTree 1
- $Mp^d \neg q \ldots$ // explore SubTree 2
- $M\neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \vDash \neg q \vee t$ and $M \vDash \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^d q^d \cdots \Longrightarrow M\neg q$
- Instead of $Mp^d q^d \cdots \Longrightarrow Mp^d \neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

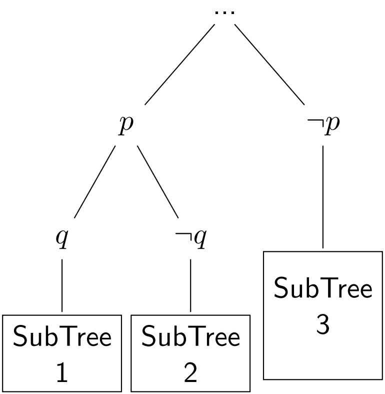
- $Mp^dq^d \dots$ // explore SubTree 1
- $Mp^d\neg q \dots$ // explore SubTree 2
- $M\neg p \dots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \models \neg q \vee t$ and $M \models \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^dq^d \cdots \Longrightarrow M\neg q$
- Instead of $Mp^dq^d \cdots \Longrightarrow Mp^d\neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

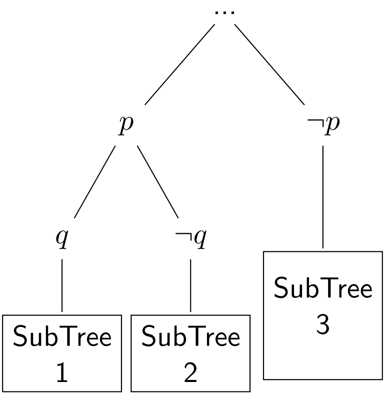
- $Mp^dq^d \ldots$ // explore SubTree 1
- $Mp^d\neg q \ldots$ // explore SubTree 2
- $M\neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \models \neg q \vee t$ and $M \models \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^dq^d \cdots \Longrightarrow M\neg q$
- Instead of $Mp^dq^d \cdots \Longrightarrow Mp^d\neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

还有一个新问题: **Backtrack** *always* goes back to the *last* decision literal



Consider the following example:

- $Mp^d q^d \ldots$ // explore SubTree 1
- $Mp^d \neg q \ldots$ // explore SubTree 2
- $M \neg p \ldots$ // explore SubTree 3

If p does not play a role in contradiction in SubTree 1, e.g.,

- $M \models \neg q \vee t$ and $M \models \neg q \vee \neg t$
- Then $\neg q$ can be derived
- A better way: $Mp^d q^d \cdots \Longrightarrow M \neg q$
- Instead of $Mp^d q^d \cdots \Longrightarrow Mp^d \neg q$

So *jumping back* to an earlier decision literal than the last one (as in *backtrack*) is correct and *increases efficiency*

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \vDash \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \vDash \neg C'$ and l' is undefined in M

Correct by definition: if $C' \vee l'$ would have been in the CNF, then going from M to Ml' is just **UnitPropagate**

问题: How to find the new clause $C' \vee l'$?

- by investigating the literals that play a role in the found contradiction, and mimic this by resolution.

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined in M

Correct by definition: if $C' \vee l'$ would have been in the CNF, then going from M to Ml' is just **UnitPropagate**

问题: How to find the new clause $C' \vee l'$?

- by investigating the literals that play a role in the found contradiction, and mimic this by resolution.

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined in M

Correct by definition: if $C' \vee l'$ would have been in the CNF, then going from M to Ml' is just **UnitPropagate**

问题: How to find the new clause $C' \vee l'$?

- by investigating the literals that play a role in the found contradiction, and mimic this by resolution.

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined in M

Correct by definition: if $C' \vee l'$ would have been in the CNF, then going from M to Ml' is just **UnitPropagate**

问题: How to find the new clause $C' \vee l'$?

- by investigating the literals that play a role in the found contradiction, and mimic this by resolution.

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \vDash \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \vDash \neg C'$ and l' is undefined in M

Apart from doing this **Backjump** step, this new clause $C' \vee l'$ will be *added* to the *CNF*:

- **Learn**: CNF=CNF $\cup\{C' \vee l'\}$

Variants of this idea may also cause **Learn** of new clauses, as long as they can be derived from the original clauses

Original clauses may become *redundant* due to addition of new clauses, and may be removed: **Forget**

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined in M

Apart from doing this **Backjump** step, this new clause $C' \vee l'$ will be *added* to the *CNF*:

- **Learn**: CNF=CNF $\cup\{C' \vee l'\}$

Variants of this idea may also cause **Learn** of new clauses, as long as they can be derived from the original clauses

Original clauses may become *redundant* due to addition of new clauses, and may be removed: **Forget**

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined in M

Apart from doing this **Backjump** step, this new clause $C' \vee l'$ will be *added* to the *CNF*:

- **Learn**: CNF=CNF $\cup\{C' \vee l'\}$

Variants of this idea may also cause **Learn** of new clauses, as long as they can be derived from the original clauses

Original clauses may become *redundant* due to addition of new clauses, and may be removed: **Forget**

Rule: Backjump

$$Ml^dN \Longrightarrow Ml'$$

if $Ml^dN \models \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined in M

Apart from doing this **Backjump** step, this new clause $C' \vee l'$ will be *added* to the *CNF*:

- **Learn**: CNF=CNF $\cup\{C' \vee l'\}$

Variants of this idea may also cause **Learn** of new clauses, as long as they can be derived from the original clauses

Original clauses may become *redundant* due to addition of new clauses, and may be removed: **Forget**

Rule: Backjump

$$Ml^d N \Longrightarrow Ml'$$

if $Ml^d N \models \neg C$ for a clause C in the CNF and there is a clause $C' \vee l'$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined in M

Apart from doing this **Backjump** step, this new clause $C' \vee l'$ will be *added* to the *CNF*:

- **Learn**: CNF=CNF $\cup\{C' \vee l'\}$

Variants of this idea may also cause **Learn** of new clauses, as long as they can be derived from the original clauses

Original clauses may become *redundant* due to addition of new clauses, and may be removed: **Forget**

It often occurs that the process does not make progress, while several new clauses have been learned

- Then it helps to **Restart**: start with empty M using the adjusted CNF

The *new clauses* may influence the *heuristics* of choosing variables and cause better progress

It often occurs that the process does not make progress, while several new clauses have been learned

- Then it helps to **Restart**: start with empty M using the adjusted CNF

The *new clauses* may influence the *heuristics* of choosing variables and cause better progress

It often occurs that the process does not make progress, while several new clauses have been learned

- Then it helps to **Restart**: start with empty M using the adjusted CNF

The *new clauses* may influence the *heuristics* of choosing variables and cause better progress

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_2^d \ x_{11} \ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_7 \ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \ x_4 \ \neg x_3 \ x_8^d \ x_2^d \ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

Decide

$\neg x_1^d$ x_4 x_3^d $\neg x_8$ x_{12} $\neg x_2^d$ x_{11} x_7^d

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d$ x_4 x_3^d $\neg x_8$ x_{12} $\neg x_7$ $\neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d$ x_4 $\neg x_3$ x_8^d x_2^d x_7

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

UnitPropagate

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_2^d\ x_{11}\ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_7\ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d\ x_4\ \neg x_3\ x_8^d\ x_2^d\ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

Decide

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_2^d \ x_{11} \ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_7 \ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \ x_4 \ \neg x_3 \ x_8^d \ x_2^d \ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$$
\begin{array}{lll}
x_1 \vee x_4 & x_1 \vee \neg x_3 \vee \neg x_8 & x_1 \vee x_8 \vee x_{12} \\
x_2 \vee x_{11} & \neg x_7 \vee \neg x_3 \vee x_9 & \neg x_7 \vee x_8 \vee \neg x_9 \\
x_7 \vee x_8 \vee \neg x_{10} & x_7 \vee x_{10} \vee \neg x_{12} &
\end{array}
$$

UnitPropagate

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_2^d \; x_{11} \; x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_7 \; \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \; x_4 \; \neg x_3 \; x_8^d \; x_2^d \; x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

UnitPropagate

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_2^d \ x_{11} \ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_7 \ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \ x_4 \ \neg x_3 \ x_8^d \ x_2^d \ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$$
\begin{array}{lll}
x_1 \vee x_4 & x_1 \vee \neg x_3 \vee \neg x_8 & x_1 \vee x_8 \vee x_{12} \\
x_2 \vee x_{11} & \neg x_7 \vee \neg x_3 \vee x_9 & \neg x_7 \vee x_8 \vee \neg x_9 \\
x_7 \vee x_8 \vee \neg x_{10} & x_7 \vee x_{10} \vee \neg x_{12} &
\end{array}
$$

Decide

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_2^d\ x_{11}\ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF∪$\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_7\ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF∪$\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d\ x_4\ \neg x_3\ x_8^d\ x_2^d\ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} x_1 \vee x_4 & x_1 \vee \neg x_3 \vee \neg x_8 & x_1 \vee x_8 \vee x_{12} \\ x_2 \vee x_{11} & \neg x_7 \vee \neg x_3 \vee x_9 & \neg x_7 \vee x_8 \vee \neg x_9 \\ x_7 \vee x_8 \vee \neg x_{10} & x_7 \vee x_{10} \vee \neg x_{12} & \end{array}$$

UnitPropagate

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_2^d \; x_{11} \; x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_7 \; \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \; x_4 \; \neg x_3 \; x_8^d \; x_2^d \; x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$$
\begin{array}{lll}
x_1 \lor x_4 & x_1 \lor \neg x_3 \lor \neg x_8 & x_1 \lor x_8 \lor x_{12} \\
x_2 \lor x_{11} & \neg x_7 \lor \neg x_3 \lor x_9 & \neg x_7 \lor x_8 \lor \neg x_9 \\
x_7 \lor x_8 \lor \neg x_{10} & x_7 \lor x_{10} \lor \neg x_{12} &
\end{array}
$$

Decide

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_2^d\ x_{11}\ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \lor \neg x_7 \lor x_8\}$

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_7\ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \lor x_7 \lor x_8 \lor x_{10}\}$

$\neg x_1^d\ x_4\ \neg x_3\ x_8^d\ x_2^d\ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

UnitPropagate

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_2^d \; x_{11} \; x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_7 \; \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \; x_4 \; \neg x_3 \; x_8^d \; x_2^d \; x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

Learn

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_2^d \; x_{11} \; x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_7 \; \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \; x_4 \; \neg x_3 \; x_8^d \; x_2^d \; x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

Backjump

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_2^d \; x_{11} \; x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_7 \; \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \; x_4 \; \neg x_3 \; x_8^d \; x_2^d \; x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$$
\begin{array}{lll}
x_1 \vee x_4 & x_1 \vee \neg x_3 \vee \neg x_8 & x_1 \vee x_8 \vee x_{12} \\
x_2 \vee x_{11} & \neg x_7 \vee \neg x_3 \vee x_9 & \neg x_7 \vee x_8 \vee \neg x_9 \\
x_7 \vee x_8 \vee \neg x_{10} & x_7 \vee x_{10} \vee \neg x_{12} &
\end{array}
$$

UnitPropagate

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_2^d \; x_{11} \; x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_7 \; \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \; x_4 \; \neg x_3 \; x_8^d \; x_2^d \; x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

UnitPropagate

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_2^d \ x_{11} \ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_7 \ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \ x_4 \ \neg x_3 \ x_8^d \ x_2^d \ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

Learn

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_2^d\ x_{11}\ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_7\ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d\ x_4\ \neg x_3\ x_8^d\ x_2^d\ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

Backtrack

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_2^d \ x_{11} \ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_7 \ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \ x_4 \ \neg x_3 \ x_8^d \ x_2^d \ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$$\begin{array}{lll} x_1 \vee x_4 & x_1 \vee \neg x_3 \vee \neg x_8 & x_1 \vee x_8 \vee x_{12} \\ x_2 \vee x_{11} & \neg x_7 \vee \neg x_3 \vee x_9 & \neg x_7 \vee x_8 \vee \neg x_9 \\ x_7 \vee x_8 \vee \neg x_{10} & x_7 \vee x_{10} \vee \neg x_{12} & \end{array}$$

UnitPropagate

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_2^d\ x_{11}\ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d\ x_4\ x_3^d\ \neg x_8\ x_{12}\ \neg x_7\ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d\ x_4\ \neg x_3\ x_8^d\ x_2^d\ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

UnitPropagate

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_2^d \; x_{11} \; x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \; x_4 \; x_3^d \; \neg x_8 \; x_{12} \; \neg x_7 \; \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \; x_4 \; \neg x_3 \; x_8^d \; x_2^d \; x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$x_1 \vee x_4$	$x_1 \vee \neg x_3 \vee \neg x_8$	$x_1 \vee x_8 \vee x_{12}$
$x_2 \vee x_{11}$	$\neg x_7 \vee \neg x_3 \vee x_9$	$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$	$x_7 \vee x_{10} \vee \neg x_{12}$	

UnitPropagate

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_2^d \ x_{11} \ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_7 \ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \ x_4 \ \neg x_3 \ x_8^d \ x_2^d \ x_7$

Sat

例 3

Consider the CNF consisting of the following eight clauses

$$
\begin{array}{lll}
x_1 \vee x_4 & x_1 \vee \neg x_3 \vee \neg x_8 & x_1 \vee x_8 \vee x_{12} \\
x_2 \vee x_{11} & \neg x_7 \vee \neg x_3 \vee x_9 & \neg x_7 \vee x_8 \vee \neg x_9 \\
x_7 \vee x_8 \vee \neg x_{10} & x_7 \vee x_{10} \vee \neg x_{12} &
\end{array}
$$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_2^d \ x_{11} \ x_7^d$

Contradiction $(x_9, \neg x_9)$

CNF=CNF$\cup\{\neg x_3 \vee \neg x_7 \vee x_8\}$

$\neg x_1^d \ x_4 \ x_3^d \ \neg x_8 \ x_{12} \ \neg x_7 \ \neg x_{10}$

Contradiction $(x_{12}, \neg x_{12})$

CNF=CNF$\cup\{x_1 \vee x_7 \vee x_8 \vee x_{10}\}$

$\neg x_1^d \ x_4 \ \neg x_3 \ x_8^d \ x_2^d \ x_7$

Sat

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

Concluding, the full *CDCL* Algorithm consists of

- the basic format of **UnitPropagate**, **Decide**, **Backtrack** and **Fail**
- the **Backjump** optimization and variants
- **Learn** new clauses by these optimizations
- **Forget** redundant clauses
- clever heuristics for choosing **Decide** variables
- clever heuristics for when to do **Restart**

This is the heart of *current SAT solvers*.

作业

实验大作业 (可选): 自行设计 CNF 的 SAT 求解算法, 要求:

- 可以使用现有算法 (如 DPLL, CDCL)，也可以自行设计其他算法
- 可以独立设计可执行程序，也可以修改现有开源程序的核心算法 (选取后者分数更高)
- 自己构建测试集（可网上查找测试集）
- 附上详细的文档: 包括实现过程，算法解释，与现有工具 (如 Z3) 等的性能对比