SAT 问题简介

吉建民

USTC
jianmin@ustc.edu.cn

2023 年 3 月 26 日

Used Materials

Disclaimer: 本课件采用部分网络博客内容

Table of Contents

SAT 问题基本定义

- SAT 问题 (Boolean satisfiability problem, 布尔可满足性问题, SAT):
  - 给定布尔 (Boolean) 表达式 (由 AND, OR, NOT, 变量, 和括号构成), 是否存在对变量 TRUE 或 FALSE 的赋值, 使得整个表达式真。

$$(x + y\bar{z})(x + y + z)(\bar{y} + \bar{z})$$

- SAT 问题 (Propositional satisfiability problem, 命题可满足问题, SAT):
  - 给定命题逻辑公式 (propositional formula), 是否可满足 (存在一个模型)。

$$(\neg(x_1 \to \neg x_2) \to x_3) \to (x_1 \to (x_2 \to x_3))$$

- 任意布尔表达式和命题逻辑公式都可以等价转换为合取范式。

SAT 问题基本定义 (cont'd)

- SAT 问题: 合取范式 (conjunctive normal form, CNF) 的可满足问题。
  - 一个合取范式形如: $C_1 \wedge C_2 \wedge \cdots \wedge C_n$, 子句 C_i $(1 \leq i \leq n)$ 形如: $l_1 \vee l_2 \vee \cdots \vee l_k$, 其中, l_i 称为文字, 为某一布尔变量或该布尔变量的非。

$$(\neg a_1 \vee a_2) \wedge (\neg a_1 \vee a_3 \vee a_4) \wedge (a_1 \vee \neg a_2 \vee a_3 \vee \neg a_4)$$

  - SAT 问题是指, 是否存在一组对所有布尔变量 TRUE 或 FALSE 的赋值, 使得整个合取范式取值为真。
- k-SAT 问题: 若合取范式 F 中每个子句 C 中所包含的文字数目均为 k, 则称为 k-SAT 问题。
  - 2-SAT 问题: $k = 2$。
  - 3-SAT 问题: $k = 3$。任何 k-CNF 公式都可以转换为 3-CNF。

$$(\neg a_1 \vee a_2 \vee x_1) \wedge (\neg a_1 \vee a_2 \vee \neg x_1) \wedge (\neg a_1 \vee a_3 \vee a_4) \wedge (a_1 \vee \neg a_2 \vee x_2) \wedge (a_3 \vee \neg a_4 \vee \neg x_2)$$

- SAT 问题从 1960 年就开始不断研究。

问题复杂性

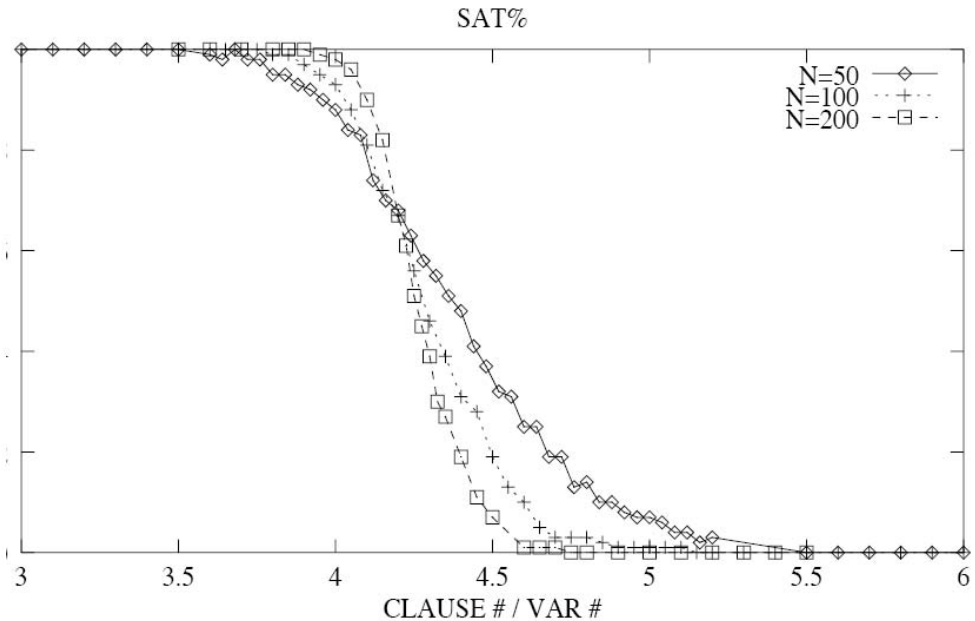
- SAT 问题是第一个被证明为 NP-complete 的问题：库克定理 (Cook's theorem, Cook–Levin theorem)。 details
- 3-SAT 问题是 NP-complete。
- 2-SAT 问题可以在多项式时间解决，是 NL-complete (非确定图灵机对数空间可解问题的完全类)。
- Horn-SAT 问题，每个子句必须是 Horn 子句 (每个子句至多有一个真文字)，是 P-complete。

SAT 相变 (Phase transition)

- 对一个 k-SAT 问题, 基于 N 个变量, 有 L 个子句。增加 L/N, 从 100% 可满足, 转变到 100% 不可满足。
- 给定 L 和 N 随机生成 k-SAT 问题, 记为 $R_k(N, L)$。当 N 趋近于无穷时, $R_k(N, L)$ 可满足的概率有如下性质:

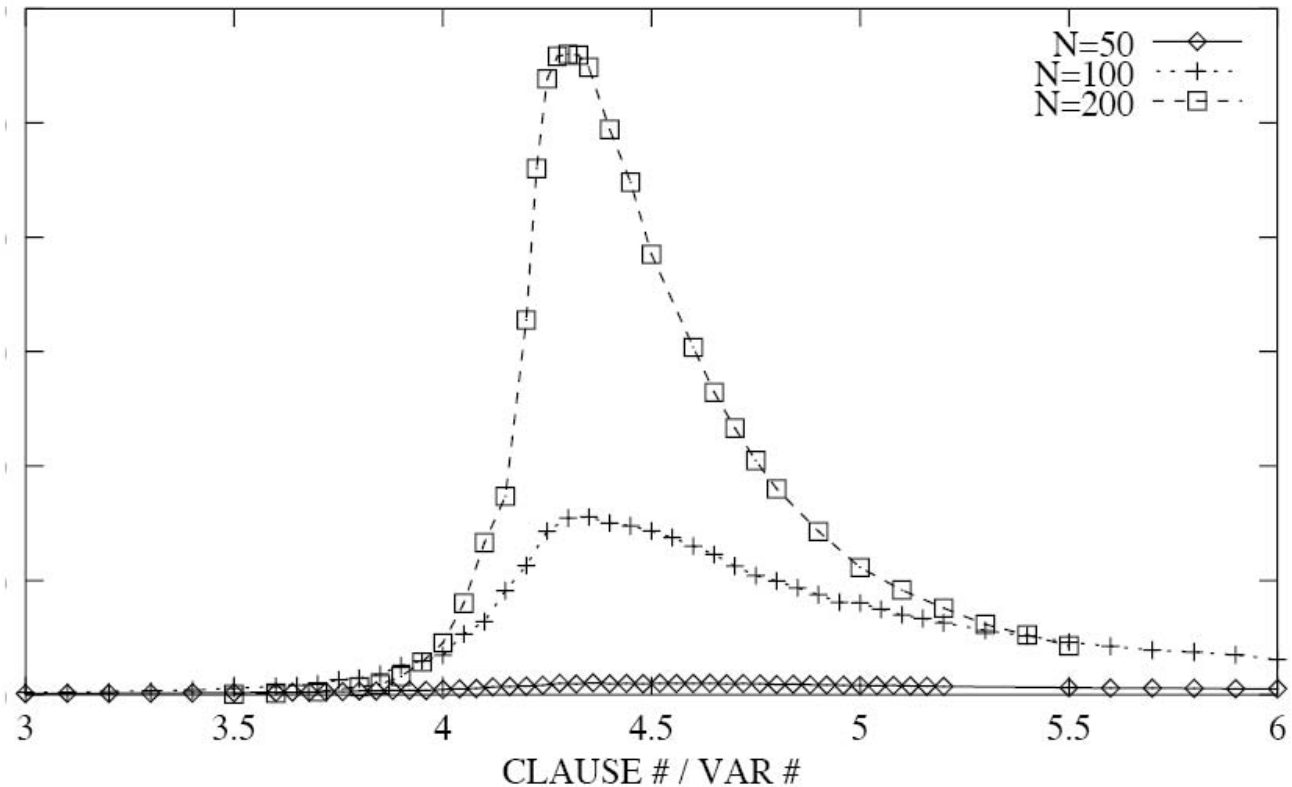
$$\lim_{N\to\infty} Prob(sat, R_k(N, c \cdot N)) = \begin{cases} 0 & \text{for } c > c_k \\ 1 & \text{for } c < c_k \end{cases}$$

可以证明 $c_2 = 1$ 且 $3.003 < c_3 < 4.81$, 实验表明 $c_3 \approx 4.28$。



SAT 相变 (Phase transition) (cont'd)

- 随着 L/N 的增长, SAT 问题呈现 "easy-hard-easy" 的样式。一般的 NP 问题都有类似的情况。
- 用 DPLL 来求解 3-SAT 问题, 问题在 $L/N \approx 4.28$ 时显得困难。



SAT 问题应用

- 在很多应用中，把 SAT 作为中间域建立通用问题求解框架。即把待求解的问题转换为 SAT 问题进行求解，将求得的解转换到原问题域从而得到原问题的解。
  - SAT 问题是 NP-complete 的，每个 NP 问题可以在多项式时间内归约到 SAT 问题，SAT 作为中间语言具有足够的表达能力；
  - SAT 的语法和语义简单，设计和评价 SAT 算法比较方便。
- 对 SAT 的研究不仅具有重要的理论意义，而且在
  - EDA (Electronic Design Automation): Combinational Equivalence Checking、Automatic Test Pattern Generation、计算机辅助设计、计算机辅助制造、集成电路设计自动化、计算机结构设计、以及计算机网络设计等；
  - AI (Artificial Intelligence): 规划、逻辑程序、自动推理等；
  - 有界模型检测 (Bounded Model Checking)

  都有着直接的应用。

Table of Contents

An overview

主要有两类求解 SAT 问题的高效算法:

- Davis-Putnam-Logemann-Loveland (DPLL or DLL, 1962) 算法的各种变形。
  - 基于回溯的搜索算法,
  - 当代大多数 SAT solver 的基础,
  - 在此基础上可以分为: “conflict-driven” 和 “look-ahead” 形式。
- 随机局部搜索算法 (GSAT 1992, WalkSAT 1993)。
  - 无法证明不可满足性，但很多时候可以很快找到满足的解,
  - 主要基于局部搜索和爬山算法。

Unit Clauses 和 Pure Literals

- 只含有一个文字的子句是 unit clause
  - Unit propagation: unit clauses $\{l\}$
    1. every clause containing l is removed;
    2. in every clause that contains $\neg l$ this literal is deleted.
- 只在公式中正出现或负出现的文字是 pure literal
  - Pure literal elimination: delete all clauses containing pure literals.

例

考虑 CNF: $(p \vee \neg q \vee \neg s) \wedge p \wedge (q \vee \neg r \vee \neg s \vee \neg t) \wedge (r \vee p) \wedge t$

- p 和 $\neg s$ 是 pure literals;
- 子句 $\{p\}$ 和 $\{t\}$ 是 unit clauses。

DPLL 算法

function DPLL(ϕ, μ)

if ϕ is empty

then return TRUE;

if ϕ contains an empty clause

then return FALSE;

if l is a unit clause in ϕ

then return DPLL(assign(l, ϕ), $\mu \wedge l$);

if l is a pure literal in ϕ

then return DPLL(assign(l, ϕ), $\mu \wedge l$);

$l := \mathit{choose\text{-}literal}(\phi)$;

return DPLL(assign(l, ϕ), $\mu \wedge l$) or

DPLL(assign($\neg l, \phi$), $\mu \wedge \neg l$).

DPLL 算法特点

- 二叉分支搜索 (某个变量的 t/f);
- 在 unit propagation 和 pure literal elimination 基础上, 尽可能晚的推迟分支;
- 只需要多项式空间;
- *choose-literal*() 是算法的关键;
- 几乎是所有精确求解 SAT Solver 的基础。

提高 DPLL 的技术

- Preprocessing (预处理): 对输入的 CNF 公式作预处理, 使其便于计算。
- Look-ahead: 更多的利用未知搜索空间的信息。
  - 加强 unit propagation 和 pure literal elimination;
  - 用启发式函数 *choose-literal*()。
  - 对 "small but hard" 的问题比较有效
- Look-back (Conflict-driven): 利用搜索过程中已知的信息, 尤其是 conflict。
  - Non-chronological backtracking (backjumping): 通过调整搜索顺序来避免无用的搜索。
  - Conflict driven clause learning: 通过学习得到新的子句来避免无用的搜索。
  - 其他技术: "two-watched-literals" unit propagation, adaptive branching。
  - 对 "big but simple" 的问题比较有效

预处理

- Sorting+subsumption:

$$\varphi_1 \wedge (l_2 \vee l_1) \wedge \varphi_2 \wedge (l_2 \vee l_3 \vee l_1) \wedge \varphi_3$$
$$\Downarrow$$
$$\varphi_1 \wedge (l_1 \vee l_2) \wedge \varphi_2 \wedge \varphi_3$$

- 2-simplifying: 利用 binary clauses 化简 CNF。

  Repeat:

  1. 根据 CNF 中 binary clauses 构造 implication graph。CNF 中有一个 binary clause $\{\neg l, l'\}$, 则在 implication graph 中存在一条 l 到 l' 的边。
  2. 找出 implication graph 中的强连通分量 (SCC), 将 SCC 中所有文字等价为一个新的文字。
  3. 将 CNF 中所有 SCC 中文字和他们的负, 分别用新文字和它的负替换。
  4. 应用 unit propagation 和 pure literal elimination。

  Until 没有新的化简。

“Look-ahead” DPLL

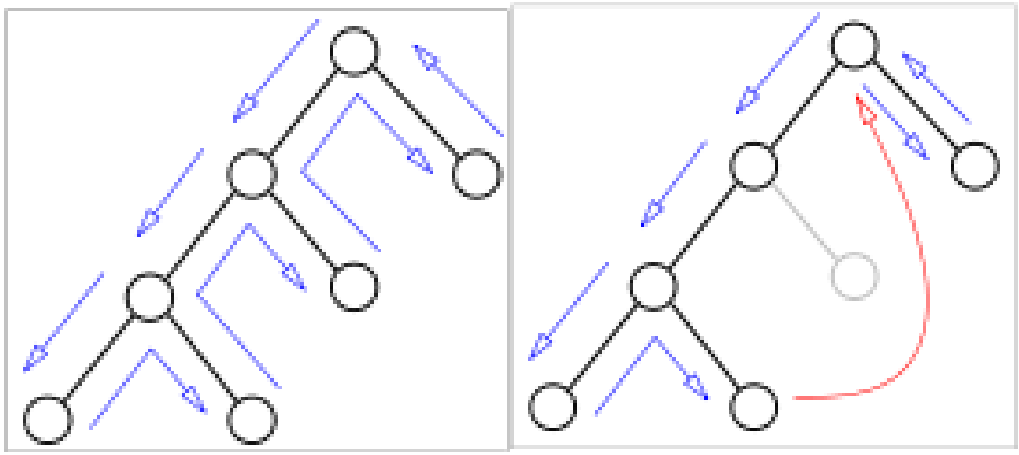
- *Lookahead*() 函数: 如果加入 l 后, 用 unit propagation 得到矛盾, 则加入 $\neg l$。
  - 比 unit propagation 更强大, 推出更多的东西;
  - 但效率损失更大, 不断在实践中被证明, 用 *Lookahead*() 不如简单用 unit propagation。
- 各种不同的启发函数 *choose-literal*():
  - MOM heuristics: 选择在极小子句中最常出现的文字。
    - 简单高效
  - Jeroslow-Wang: 选择有极大 score 值的文字

$$score(l) := \sum_{l \in c \wedge c \in \varphi} 2^{-|c|}$$

    - 评估 l 对于 CNF φ 满足的贡献
  - Satz: 从待选文字集中, 选择通过 unit propagation 得到最小子句集的文字。
    - 充分利用 unit propagation 得到的结果

“Conflict-driven” DPLL

- Non-chronological backtracking (backjumping):



  - Idea: 当一个分支失败时
    1. 找出导致失败的部分赋值 (conflict set);
    2. 回溯到 conflict set 中的 most recent branching point。
  - Conflict set:
    - 在已知的部分赋值下, 某个文字成真是由于在某个子句上用 unit propagation, 将此子句称为此文字的原因。例, 在 $\{\neg a, \neg b\}$ 下, 子句 $(a \vee b \vee \neg x)$ 就是文字 $\neg x$ 的原因。
    - 当出现矛盾时, 存在两个互补的文字, 将这两个文字的原因 (子句) 归结为一个子句, 此子句对应的补文字集为当前矛盾的 conflict set。

"Conflict-driven" DPLL (cont'd)

- Conflict driven clause learning:
  - Idea: C 是一个 conflict set, 则 $\neg C$ 可以加到原始子句集里面。
    - 保证包含 C 的赋值不成立, 在 DPLL 的搜索过程中避免了类似的赋值。
    - 例, 当前部分赋值 $\{\neg p, q, r, \neg t, s\}$, 子句 $(x \vee p \vee \neg q)$ 是 x 成立的原因, 子句 $(\neg x \vee \neg r \vee t)$ 是 $\neg x$ 成立的原因。
      则 $C = \{\neg p, q, r, \neg t\}$ 是相应的 conflict set。我们可以把子句 $(p \vee \neg q \vee \neg r \vee t)$ 加入原始子句集中。
  - Problem: 可能导致空间膨胀。
    - 用其他技术来控制 learning, 并且在需要时丢掉学到的子句。

局部搜索算法

- 不是构造解，而是通过修改完整赋值来不断满足更多的子句。
- 不能证明公式的不可满足性。
- 主要包括 GSAT 和 WalkSAT，也有其他的局部搜索方法:
  - Simulated annealing, 模拟退火法
  - Tabu search, 禁忌搜索
  - Genetic algorithms, 遗传算法
  - Hybrid methods, 杂合方法: 结合 DPLL 方法和局部搜索的优势。
- 一般需要多项式空间。
- 对一些 (可满足的) 问题非常有效。

GSAT

- GSAT **算法结构**:

  Repeat MAX-TRIES times or until clauses satisfied

  $T :=$ random truth assignment

  Repeat MAX-FLIPS times or until clauses satisfied

  $v :=$ variable which flipping maximizes number of SAT clauses

  $T := T$ with v's value flipped

- **可以加入** restarts **和** greediness。
- **贪心搜索, 寻找较好的** "neighbor"。

WalkSAT

- WalkSAT 算法结构:

  Repeat MAX-TRIES times or until clauses satisfied

  $T :=$ random truth assignment

  Repeat MAX-FLIPS times or until clauses satisfied

  $c :=$ unsat clause chosen at random

  $v :=$ var in c chosen either greedily or at random

  $T :=$ T with v's value flipped

- 关注于没有满足的子句。

Table of Contents

An overview

- GRAPS
  - João Marques Silva (葡萄牙人), 1996.
  - 最早的 "conflict-driven" DPLL SAT solver, 最早使用 backjumping 和 clause learning。
- zChaff
  - Lintao Zhang (中国人, 微软研究院), 2001–2007.
  - "conflict-driven" DPLL SAT solver, 同时使用了很多高效技术: 2-literal watching, restarts, VSIDS 作为启发函数, ...
  - 公认较快的 SAT solver, 在 "SAT Competition" 2003, 2004, 2005 表现优异。
- MiniSAT
  - Niklas Eén, Niklas Sörensson, 2003–2007.
  - "conflict-driven" DPLL SAT solver, 类似 zChaff, 同时增加 Conflict clause minimization。
  - 公认较快的 SAT solver, 在 "SAT Competition" 2005, 2007 表现优异。
- mach_dl
  - Marijn Heule, Hans van Maaren, 1999.
  - "look-ahead" DPLL SAT solver, 同时使用其他高效技术。
  - 在 "SAT Competition" 2004, 2005, 2007 表现不错。

The Timeline

- 从 2002 年开始，持续举办 SAT 竞赛，每年一届。
  `http://www.satcompetition.org/`
- 较强的 SAT solvers: Chaff (2001), MiniSAT (2003), zChaff (2004)

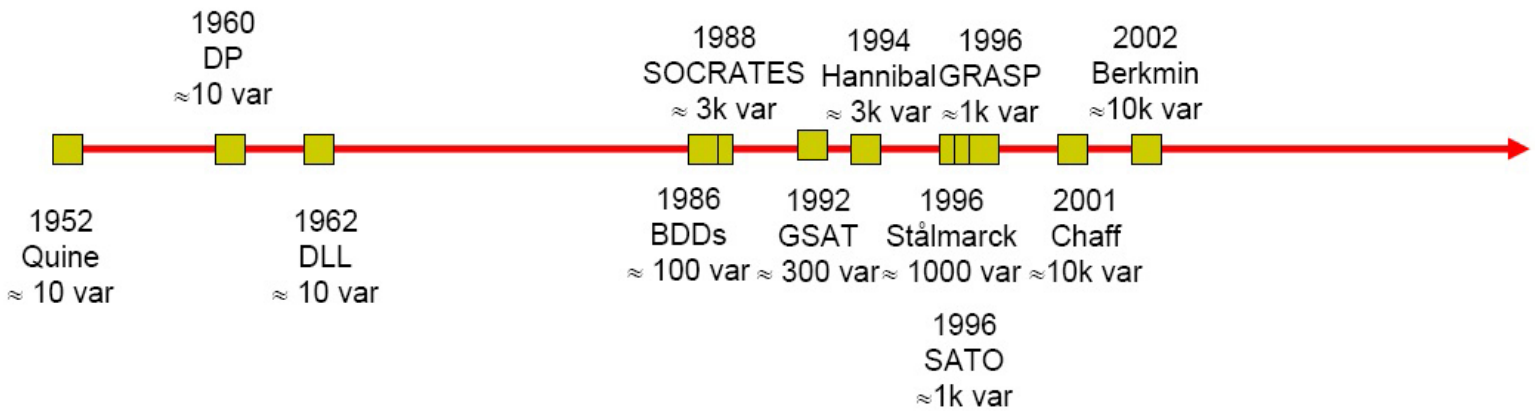


Table of Contents

Hardware Verification

- 如何验证硬件设计的正确性? Model Checking
  1. 将硬件设计描述为一个有限状态机 M (初始状态 + 状态转移关系);
  2. 将需要满足的性质 P 用时序逻辑描述出来;
  3. 判断 M 是否是 P 的一个模型, 即, $M \models P$。
- 一般需要满足的性质有如下形式:
  - 安全性性质: "x always holds", 从初始状态出发, 任何可达的状态上都性质 x 成立。x 一般是命题公式。
  - 活性性质: "there will be a point in time when x holds"。例如, 请求被回答。
- Bounded Model Checking (BMC): 判断性质 "always p" 是否在长度小于等于 k 的运行周期内成立。
  - BMC 问题, 可以很容易转化为一个 SAT 问题: 是否存在一条长度小于等于 k 的路径, 从初始状态到一个 p 不成立的状态。

BMC as SAT

- 用谓词 $I(s)$ 来刻画初始状态，每个状态 $s = (x_1, \ldots, x_n)$ 是变量的集合；用谓词 $\tau(s, s')$ 表达状态 s 的后继状态是 s'。
- BMC 问题等价为下面公式是否可满足：

$$I(s_0) \wedge \bigwedge_{i=0}^{k-1} \tau(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i).$$

- 如果此公式可满足，则存在一条这样的路径，此时我们找到一个 “always p” 的反例。
- 如果此公式不可满足，则不存在长度小于等于 k 的这样的路径，我们可以增加 k 再验证。

BMC Example

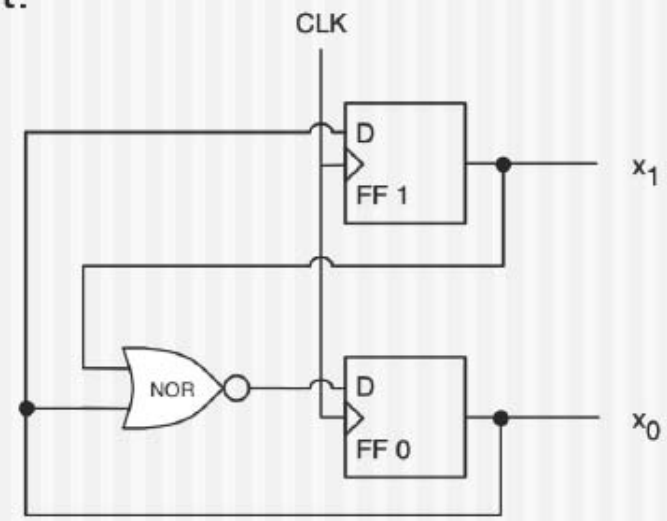
- 考虑一个二进制加法器，不断循环地从 $c = 0$ 加到 $c = 2$。证明如果 initially $c \neq 3$ 则 always $c \neq 3$。
- 状态用二进制数位表示：$s_i = (x_0^i, x_1^i)$，计数器 $c = 2 \cdot x_1^i + x_0^i$
- 初始状态条件：$I(s_0) = \neg(x_0^0 \wedge x_1^0)$ (i.e., $c_0 \neq 3$)

Transition relation:

S_i		S_{i+1}	
x_1	x_0	x'_1	x'_0
0	0	0	1
0	1	1	0
1	0	0	0
1	1	DC	DC

$$x'_1 = x_0,\ x'_0 = \neg(x_0 \vee x_1)$$

Circuit:



BMC Example (cont'd)

- **状态转移关系**, $\tau(s, s')$:

$$\tau(s_i, s_{i+1}) = (x_1^{i+1} \equiv x_0^i) \wedge (x_0^{i+1} \equiv \neg(x_1^i \vee x_0^i))$$

- **性质**, $p(s)$:

$$p(s_i) = \neg(x_1^i \wedge x_0^i)$$

- SAT solver **可以证明对任意** k, BMC **对应公式都不可满足**, **所以** always p **总成立**。

Table of Contents

SMT

- Satisfiability Modulo Theories (SMT): 对函数词和谓词有特殊解释的一阶公式是否可满足的判定问题。例如: 判断下面公式是否可满足

$$(\sin(x)^3 = \cos(\log(y) \cdot x) \vee b \vee -x^2 \geq 2.3y) \\ \wedge (\neg b \vee y < -34.4 \vee \exp(x) > \frac{y}{x}).$$

- SAT solver 效率很高, 但 SAT 不能处理数字运算, 线形约束等各种不同的约束。SMT 可以处理这类问题。
- SMT solver 本质上是一个高效的 SAT solver 不断调用一个处理特殊约束的求解器。将子句中的原子看作普通的命题原子调用普通的 SAT solver 来处理, 需要时再调用特殊约束的求解器来判断原子的真假。
- SMT 问题一般是 NP-complete。

QBF

- Quantified Boolean Formula problem (QBF): 对于容许 "for all" 和 "there exists" 出现的布尔表达式是否可满足的判定问题。例如: 判断下面公式是否可满足

$$\forall x \exists y \exists z. (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z).$$

- QBF 问题是 PSPACE-complete, 比 SAT 问题更复杂, 还没有可以实用的 QBF sovler。

Table of Contents

小结

- SAT 问题是布尔表达式可满足性问题，一般以 CNF 公式作为其标准输入。
- SAT 问题是 NP-complete。
- SAT 在设计自动化，AI 等领域有广泛应用。
- 求解 SAT 问题的方法，一般是基于 DPLL 的搜索算法，或者是随机局部搜索算法。
- 现代的 SAT solver 已经达到实用程度，主要基于 DPLL 算法。

Table of Contents

“容易”与“困难”的问题

- 一个问题需要指数时间才可以计算出结果，称为“困难”的。

例 (货郎担问题, Traveling Sales Person problem, TSP)

某售货员要到若干个村庄售货，各村庄之间的路程是已知的，为了提高效率，售货员决定从所在商店出发，到每个村庄售一次货然后返回商店，问他应选择一条什么路线才能使所走的总路程最短?

求取具有最小成本的周游路线。

- 一个问题用多项式时间就可以计算出结果，称为“容易”的。
  - 多项式时间是 $O(n^k)$，其中 k 是常数，n 是问题输入的规模。
  - 这里的“容易”是相对的，实际计算中 $O(n^{100})$ 就可以很困难。

计算复杂性理论基本定义

- 什么是计算？计算模型。
- 图灵机模型:

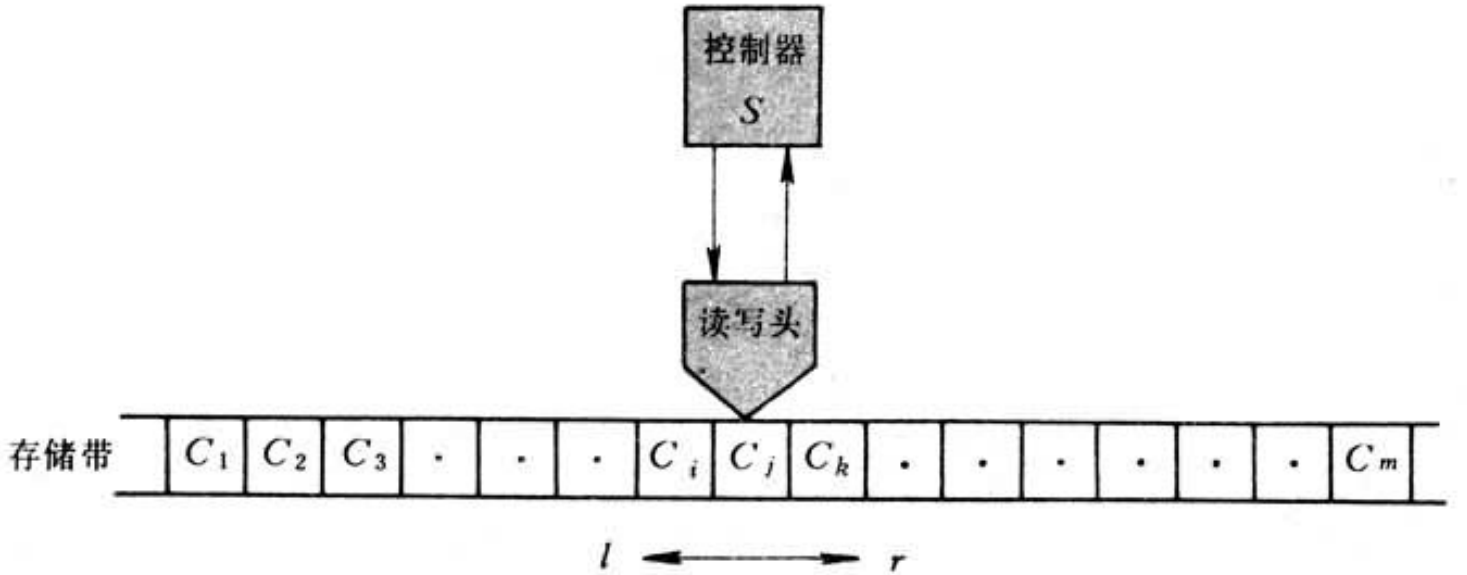


图 灵 机 的 构 造

- 图灵机模型是计算机的计算能力的极限，目前还没有计算模型超越图灵机模型。

确定性与非确定性图灵机

- **确定性图灵机** (Deterministic Turing Machine): 图灵机状态转移函数是单值的, 每种格局有确定的转移方式。

$$\delta : Q \times T \to Q \times T \times \{L, R\}$$

- **非确定图灵机** (Nondeterministic Turing Machine): 图灵机状态转移函数有不确定性, 有多种转移方式进行选择。

$$\delta : Q \times T \to \mathcal{P}(Q \times T \times \{L, R\})$$

  - 猜想阶段
  - 验证阶段

P 类和 NP 类问题

- P 类问题:
  - 在确定性图灵机上多项式时间可解的问题。
- NP 类问题:
  - 在非确定性图灵机上多项式时间可解的问题;
  - 在确定性图灵机上多项式时间可验证的问题。
- $P \subseteq NP$
- $P \stackrel{?}{=} NP$: 大多数计算机科学家认为 NP 类包含了不属于 P 类的问题, 即 $P \neq NP$。
- 直观上看, P 类问题是确定性计算模型下的易解问题类, 而 NP 类问题是非确定性计算模型下的易验证问题类。

NP-hard 和 NP-complete 问题

- 问题 Q 是 NP-hard 问题，如果：
  - 每个 NP 问题都可以在多项式时间归约为问题 Q。
- 问题 Q 是 NP-complete 问题，如果：
  - 它是 NP 问题，同时它还是 NP-hard 问题。
- NP-complete 问题的性质：
  - 所有 NP-complete 问题，相对于多项式时间归约关系，是自反，对称和传递的，即构成一个等价类。
  - 如果能找到一个 NP-complete 问题的多项式算法，则 $P = NP$。

back