科学出版社

逻辑公式的可满足性判定

——方法、工具及应用

张健 著

博士丛书 · 博士丛书 · 博士丛书 · 博士丛书 ·

博 士 丛 书

逻辑公式的可满足性判定
——方法、工具及应用

张 健 著

科 学 出 版 社
2000

内 容 简 介

逻辑公式的可满足性问题是计算机科学和人工智能中的著名问题.本书前三章主要介绍经典的命题逻辑和一阶谓词逻辑公式以及模态逻辑公式的可满足性判定算法,也介绍了有关的软件工具.第四章则介绍它们在离散数学研究、软件和硬件的形式验证与测试等方面的应用.

本书可供从事计算机科学和人工智能研究的有关人员阅读,也可供高等院校计算机专业的本科生和研究生参考.

图书在版编目(CIP)数据

逻辑公式的可满足性判定——方法、工具及应用/张健著 . -北京:科学出版社,2000.10

(博士丛书/卢嘉锡,钱伟长主编)

ISBN 7-03-008364-4

Ⅰ.逻… Ⅱ.张… Ⅲ.人工智能 Ⅳ.O141

中国版本图书馆 CIP 数据核字(2000)第 05378 号

科学出版社 出版

北京东黄城根北街 16 号

邮政编码:100717

科地亚印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2000 年 10 月第 一 版 开本:850×1168 1/32

2000 年 10 月第一次印刷 印张:5 3/4

印数:1—3 500 字数:145 000

定价:18.00 元

(如有印装质量问题,我社负责调换〈新欣〉)

序

环顾当今世界，国家的发达，民族的振兴，无一例外地离不开科学技术的推动作用．年轻博士们历来是科技队伍中最活跃、最富创造性的生力军．他们的科研成果是学科发展强有力的推动力量，是体现一个国家高层次教育水平和科研水平的窗口．为了系统地反映年轻博士们的科研成果，促使他们的快速成长，加强国际国内的学术交流，在老一辈科学家的热心支持下，科学出版社决定出版一套《博士丛书》．

我们指导思想是突出本丛书的学术性、创造性、新颖性、先进性和代表性，使之成为所有青年博士平等竞争的学术舞台和优秀科研成果的缩影．

这套丛书以专著为主，并适时组织编写介绍学科最新进展的综述性著作．它将覆盖自然科学各个领域，是一套充分体现我国青年学者科研成果和特色的丛书．

丛书编委会将在由著名科学家组成的专家委员会指导下开展编辑工作．本丛书得到了国家自然科学基金委员会和全国博士后管理协调委员会的特别资助．在此我们深表谢意．

《博士丛书》编委会

1993 年 10 月

《博士丛书》专家委员会

前　言

数理逻辑是数学和计算机科学的基础. 逻辑公式的可满足性判定是计算机领域的一个重要问题. 过去几十年, 很多学者在这方面做了大量的工作, 取得了很大的进展. 本书将重点介绍实际可用的判定算法而不是可判定性、复杂性等理论结果.

本书的主体分四章. 前三章分别讨论经典的命题逻辑和一阶谓词逻辑公式以及命题模态逻辑公式的可满足性判定算法, 也提到有关的软件工具. 最后一章则介绍它们在离散数学研究、软件和硬件的形式验证与测试等方面的应用.

本书适合于从事计算机科学和人工智能研究的有关人员阅读, 也可供高等院校计算机软件和理论等专业的高年级本科生和研究生参考.

由于篇幅和作者知识面的限制, 我们不可能详细介绍逻辑公式可满足性判定方面的所有重要工作. 书后附有较多的参考文献, 以便有兴趣的读者对有关问题作深入研究. 这些文献主要是近 20 年在国际会议和刊物上发表的重要论文.

作者近几年的研究工作先后得到国家高技术研究发展计划 (863) 、国家自然科学基金和国家重点基础研究发展规划 (973) 项目的资助.

作者于 1988 年到 1994 年期间先后得到陈国良教授、董韫美院士、唐稚松院士的指导, 学习了 NP 难问题求解、形式化方法、等式逻辑和模态逻辑的有关知识, 并认识到自动推理的重要性. Larry Wos 博士提供的材料使作者能及时了解自动推理领域的最新发展和问题. 在与张瀚涛教授的合作中, 作者学到了很

多知识. 作者向他们表示衷心的感谢.

清华大学李大法教授以及中国科学院软件研究所的官荷卿、李广元、薛锐等先后阅读了书稿的部分章节, 并提出了宝贵的意见, 作者深表谢意. 最后, 感谢朱亚军的大力支持.

由于作者水平有限, 错误在所难免, 敬请读者批评指正.

张　健

2000 年 1 月

目　录

引　言

如果你用高级语言（如 Pascal, C ）编过程序，或者学过数字电路的基本知识，那么你对布尔表达式不会陌生. 这样的表达式由布尔变量通过“与”、“或”、“非”等组合而成. 给每个布尔变量一个真假值，我们就能很容易判断出整个表达式是否为真. 但是反过来，给定一个布尔表达式，是否能为每个变量找到值，使得整个表达式成真呢？这就是可满足性问题的一种形式.

逻辑公式的可满足性问题是理论计算机科学和人工智能中的著名问题. 世界各国学者在这方面做了大量的研究工作，特别是关于命题逻辑中合取范式的可满足性问题（即 SAT ）. 最近 10 年，实现了一些高效可用的软件工具. 国内也有不少人研究 SAT[89,112,113,114] ，特别是解决它的局部搜索法. 近几年国内外还有多篇硕士和博士论文与这一问题相关 [22,54,80]. 我国 863 计划智能机专家组于 1996 年主办了国际 SAT 比赛. 美国 DIMACS（离散数学和计算机科学研究中心）也于 1996 年主办了 SAT 研讨会.

逻辑是计算机科学的基础之一. 因此很多不同背景的学者都研究可满足性问题. SAT 复杂度的结果最早发表于 ACM 的计算理论研讨会（ STOC ）上 [35]. 后来的学术论文出现在各个领域的著名杂志和会议录中. 这些领域包括：人工智能（逻辑程序设计和自动推理）、运筹学、算法、电路辅助设计、离散数学，等等. 本书主要介绍可满足性判定算法及其应用. 其重点在于怎样自动地、高效地判断公式的可满足性. 我们讲的公式既可以是命

题逻辑公式，也可以是一阶公式，还包括命题模态公式.

过去 20 多年来，国内外出版过不少自动定理证明方面的著作 [31,115,116,117,120,196]，但是其中只有少数章节提到可满足性问题，而且只是在命题逻辑的层次. 人工智能的有关教科书 [55,119,165,174] 也基本如此. 它们重点介绍像消解法这样的自动定理证明技术. 实际上，可满足性判定与我们常说的定理证明是密切相关的. 后面将有一节（ §2.4 ）专门讨论它们之间的关系.

本书主要面向计算机学科的研究人员和大学生、研究生. 假定读者学习过离散数学（包括集合论、图论和数理逻辑）. 我们将尽量采用比较通用的术语和记号. 在命题逻辑、一阶谓词逻辑和模态逻辑这三章的第一节，我们简要介绍一些基本知识. 但书中不介绍公理系统以及可靠性、完备性等定理. 关于这些内容，可参看数理逻辑教科书 [50,121,190,191]. 在少数地方，我们还提到问题的复杂度. 关于这方面的背景知识，可参看文献 [63, 41].

对书中提到的定理，一般不给出详细证明，读者可参考有关文献. 除非特别声明，我们用类似于 C 语言的语法来描述算法. `/*  */` 表示注释，`{  }` 表示 `begin … end`. 在提到程序运行时间时，我们一般假设所用的机器是 SUN SPARC station 2.

第一章　命题逻辑

§1.1　命题逻辑简介

命题逻辑（又称命题演算、布尔逻辑）是最简单的一种形式逻辑系统. 命题是其主要研究对象. 每个命题可能为真，也可能为假. 例如，“1 加 2 等于 3”是真的，而“8 除以 3 余数为 1”则是假的. 通常我们用 $p, q, r, \cdots$ 这些符号代表任意的命题，称为 **命题变元**. 我们用 1, 0 分别表示命题的真假值. 有时也可以用 $\mathbf{T}, \mathbf{F}$ 或者 $\top, \bot$ 分别表示命题的真假值.

在真假值集合上可以定义一些运算或函数，称为 **真值函数**. 一个 n 元真值函数是从 $\{0,1\}^n$ 到 $\{0,1\}$ 的映射. 常见的函数包括：“与”（又称“合取”，记为 $\wedge$），“或”（又称“析取”，记为 $\vee$），“非”（又称“否定”，记为 $\neg$），“蕴涵”（记为 $\rightarrow$），“等价”（记为 $\leftrightarrow$），“异或”（记为 $\oplus$）.

和小学算术中的加法表、乘法表类似，我们可以用下面这样的表来定义上述运算.

$\neg$	
0	1
1	0

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

$\rightarrow$	0	1
0	1	1
1	0	1

$\leftrightarrow$	0	1
0	1	0
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

当 $n > 2$ 时，不能用上面这种方式表示 n 元真值函数. 一般情况下，我们使用下述形式的表：

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

这样的表称为 **真值表**（ truth table ），其中 p 和 q 是命题变元. 除了表头外，还有 2^n 行. 每一行的左部是变元的各种可能取值组合，右部是运算结果.

真值函数的名字也被称为 **连接符**（ connectives ）. 由命题变元和连接符可以按以下规则形成**命题逻辑公式**：

1. 命题变元是公式（称为 **原子公式** ）.

2. 如果 φ 是公式，则 $(\neg\varphi)$ 也是公式.

3. 如果 φ_1 和 φ_2 是公式，则 $(\varphi_1 * \varphi_2)$ 也是公式. 这里的 $*$ 表示任何一个二元连接符，常用的包括 $\vee, \wedge, \rightarrow, \leftrightarrow$.

4. 只有由上面三条规则生成的表达式是公式.

在不引起混淆的情况下，我们可以省略公式中的一些括号. 一般认为，$\neg$ 的优先级最高，$\wedge$ 和 $\vee$ 次之，而 $\rightarrow$ 和 $\leftrightarrow$ 最低. 例如，$p \rightarrow \neg q \wedge r$ 代表公式 $(p \rightarrow ((\neg q) \wedge r))$.

有时候我们也将 $\neg p$, $p \wedge q$, $p \vee q$ 分别写为 $\overline{p}$, $p \cdot q$（或者 pq ），$p + q$. 如果采用这种写法，“与”的优先级比“或”高. 例如，公式 $p \vee (\neg q \wedge r)$ 也可写成 $p + \overline{q}r$.

定义 从命题变元集合到真假值集合（即 $\{0, 1\}$ ）的函数叫做 **真值赋值**（ truth assignment ），简称 **赋值** ，又叫 **解释**. 如果这个函数没有完全定义（即只有一部分变元具有真假值），那么

称之为 **部分赋值**.

给定一个赋值，根据前面所定义的各种布尔运算的真值表，我们可以确定公式的值. 例如，如果 p,q,r 的值分别是 $1,0,1$，那么公式 $(p \to q) \vee r$ 的值就是 $(1 \to 0) \vee 1 = 0 \vee 1 = 1$.

定义 如果有一个赋值，使公式 φ 的值为 1，那么 φ 是 **可满足的**（ satisfiable ）. 使它得到满足的这个赋值就是 φ 的一个 **模型**（ model ）.

例如，公式 $(p \to q) \vee r$ 的一个模型是 $\{ p = 1, q = 0, r = 1\}$. 它也可表示为 $\{ p, \neg q, r \}$.

定义 如果对于任何赋值，公式 φ 的值都为 1，那么 φ 是 **重言式**（ tautology ）或 **永真公式**.

例 φ_1： $(p \to q)$ 不是永真公式，因为当 $p = 1, q = 0$ 时，公式的值为 0. 但 φ_1 是可满足的，因为当 $p = 0, q = 0$ 时，它的值是 1. 我们再看公式 φ_2： $(p \to p)$. 不管 p 的值是 0 还是 1，它的值都是 1. 因此 φ_2 是永真公式.

定义 如果对于任何赋值，公式 φ_1 和 φ_2 都具有相同的真假值，那么这两个公式被称为 **等值**，记为 $\varphi_1 = \varphi_2$.

下面是一些等值公式的例子：

(1) $(p \leftrightarrow q) = (p \to q) \wedge (q \to p)$

(2) $(p \to q) = (\neg p \vee q)$

(3) $\neg(\neg p) = p$

(4) $(p \vee q) = (q \vee p)$

(5) $(p \wedge q) = (q \wedge p)$

(6) $(p \vee (q \vee r)) = ((p \vee q) \vee r)$

(7) $(p \wedge (q \wedge r)) = ((p \wedge q) \wedge r)$

(8) $(p \vee (q \wedge r)) = (p \vee q) \wedge (p \vee r)$

(9) $(p \wedge (q \vee r)) = (p \wedge q) \vee (p \wedge r)$

(10) $\neg(p \vee q) = (\neg p \wedge \neg q)$

(11) $\neg(p \wedge q) = (\neg p \vee \neg q)$

这里，(4) 和 (5)、(6) 和 (7)、(8) 和 (9) 分别是交换律、结合律、分配律，而 (10) 和 (11) 被称作 De Morgan 律. 从 (1) 和 (2) 我们可以看出，有些连接符可以通过其他连接符来定义.

定义 连接符集合 M 是 **完备的**，如果任何 n 元连接符都能由 M 中的连接符来定义. 这里 n 是正整数.

常见的完备集包括，$\{\neg, \vee\}$，$\{\neg, \wedge\}$，$\{\neg, \rightarrow\}$ 和 $\{\neg, \vee, \wedge\}$. 在后面的讨论中，我们将主要关心 $\neg, \vee, \wedge$ 这几个连接符. 由于 $\vee$ 和 $\wedge$ 都具有交换律和结合律，我们可以将 $(p \vee (q \vee r))$ 写成 $(p \vee q \vee r)$，将 $((p \wedge r) \wedge (q \wedge s))$ 写成 $(p \wedge q \wedge r \wedge s)$，等等.

定义 我们称原子公式或其否定为 **文字**（literal）. 原子公式本身称为 **正文字**，而原子公式的否定称为 **负文字**. 若干个文字的析取构成 **子句**（clause），其 **长度** 是指所含文字的个数. 只有一个文字的子句称为 **单子句**（unit clause）. 没有文字的子句称为 **空子句**（empty clause），记为 $\square$.

例如，p 和 $\neg q$ 都是文字，同时也可被看成是单子句，而公式 $(p \vee \neg q \vee \neg r)$ 是长度为 3 的子句.

原子公式和它的否定式 **互补**. 例如，p 与 $\neg p$ 互补. 我们又称 p 的补是 $\neg p$，而 $\neg p$ 的补是 p. 我们用 $\neg L$ 表示文字 L 的补.

一般说来，子句中的文字越多，该子句就越容易被满足. 举一个简单的例子. 单子句 p 只有在 p 为真时才被满足. 而对于子句 $p \vee q$，不仅在 p 为真时被满足，而且在 q 为真时也被满足. 所以，文字越少的子句越难满足. 一个极端的情形是，空子句不可满足.

定义 若干个子句的合取是一种特殊形式的公式，称为 **合**

取范式（conjunctive normal form，简写为 CNF），其一般形式为

$$(L_{11} \vee \ldots \vee L_{1n_1}) \wedge \ldots \wedge (L_{m1} \vee \ldots \vee L_{mn_m})$$

这里的每个 L_{ij} 是文字. 一个公式被称为 **析取范式**（disjunctive normal form 即 DNF），如果它的形式为 $\varphi_1 \vee \ldots \vee \varphi_n$. 这里每个 φ_i 是若干个文字的合取.

例如，$(p \vee \neg q \vee \neg r) \wedge (\neg p \vee r)$ 是合取范式，而 $(\bar{p}q + p\bar{q} + \dot{q}r)$ 是析取范式.

定理 对任意公式，都有与之等值的合取范式和析取范式.

将一般形式的公式化成范式有两种简单办法. 一种是利用上面所列出的那些等值公式进行变换，另一种是通过真值表.

例 将公式 $(p \to q) \to (r \vee s)$ 转换成合取范式.

$$\begin{aligned} & (p \to q) \to (r \vee s) \\ = \ & \neg(p \to q) \vee (r \vee s) \\ = \ & (p \wedge \neg q) \vee (r \vee s) \\ = \ & (p \vee r \vee s) \wedge (\neg q \vee r \vee s) \end{aligned}$$

对于变元个数较少的公式，一种简单的构造范式的办法是通过真值表. 在构造析取范式时，考虑那些使公式取真值的行；在构造合取范式时，考虑那些使公式取假值的行.

例 公式 φ：$p \leftrightarrow q$ 的真值表如下：

p	q	$p \leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

根据第一行和最后一行，我们得到 φ 的析取范式：

$$\varphi \;=\; (\neg p \wedge \neg q) \vee (p \wedge q)$$

而根据第二行和第三行，我们知道，

$$\neg\varphi \;=\; (\neg p \wedge q) \vee (p \wedge \neg q)$$

将等式的两边取反，就得到 φ 的合取范式：

$$\varphi \;=\; (p \vee \neg q) \wedge (\neg p \vee q)$$

对于复杂的公式，可以采用专门的办法．事实上，有一个线性时间复杂度的算法 [193] 将命题逻辑公式转换成一组子句．

需要指出的是，一个比较简单的布尔表达式可能对应于很复杂的范式．例如，如果将下面的公式变成 DNF，其中文字的个数将会是指数级的：

$$(p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \ldots \wedge (p_n \vee q_n)$$

而如果将下面的公式变成 CNF，也会得到非常多的子句．

$$\begin{aligned} &(p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n) \\ \vee\ &(\neg p_1 \wedge p_2 \wedge \ldots \wedge \neg p_n) \\ \vee\ &\ldots \\ \vee\ &(\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge p_n) \end{aligned}$$

该公式在只有一个变元取真值的情况下为真．在 §1.5 我们会提到另一种范式．

通常也将 CNF 公式写成子句的集合或者一组子句．例如，公式 $(p \vee \neg q \vee \neg r) \wedge (\neg p \vee r)$ 也可以写为 $\{p \vee \neg q \vee \neg r,\ \neg p \vee r\}$，或者如下表示：

$$\begin{aligned} &p \vee \neg q \vee \neg r \\ &\neg p \vee r \end{aligned}$$

注意，子句之间是合取关系.

合取范式 S 被一个赋值所满足，当且仅当 S 中每个子句都被这个赋值满足. 如果 S 是空集（ $\emptyset$ ），那么它是可满足的. 这时我们可以给变量赋任何真假值.

子句 $(\neg p_1 \vee \ldots \vee \neg p_n \vee q_1 \vee \ldots \vee q_m)$ 等价于

$$(p_1 \wedge \ldots \wedge p_n) \rightarrow (q_1 \vee \ldots \vee q_m)$$

有些作者也将它写成这种形式： $p_1, \ldots, p_n \rightarrow q_1, \ldots, q_m$. 这里 p_i 和 q_j 都是原子公式. 蕴涵符的左侧是 **前件**(antecedent)，其右侧称为 **后件** (consequent). 前件中的原子之间是合取关系，而后件中的原子之间是析取关系. 子句为真的意思是，当所有 p_i 取真值时，至少有一个 q_j 取真值. .

采用上述形式表示子句 C 时，如果 $n = 0$，即前件为真，则称 C 为 **正子句**；如果 $m = 0$，即后件为假，则称 C 为 **负子句**；如果 m 和 n 都是正整数，则称 C 为 **混合子句**. 当 $m \leq 1$ 时，称 C 为 Horn **子句**，否则称为 **非** Horn **子句**. 例如，$(p \vee \neg q \vee \neg r)$ 和 $(\neg p \vee \neg q)$ 都是 Horn 子句.

如果变元的个数很多，我们也可以用正整数而不是字母来表示它们，而子句就表示成一串数字（通常加上括弧）. 例如，(1 -2 3) 代表 $(p_1 \vee \neg p_2 \vee p_3)$ 或者 $(x_1 \vee \neg x_2 \vee x_3)$. 这种表示的好处是便于计算机处理.

§1.2 可满足性问题

§1.2.1 合取范式的可满足性问题

永真性和可满足性是逻辑公式的重要性质. 理论计算机科学中的一个重要问题是命题逻辑公式的可满足性问题（ satisfiability problem，简称 SAT ）. 通常我们假定公式已经被转换

为合取范式. 一个 SAT 算法能在有限的时间内, 判定任意给定的 CNF 形式的命题逻辑公式 (或者说任意的子句集合) 是否可满足. 而且在可满足的情形, 算法往往会给出使子句集满足的一个赋值.

判断命题公式可满足性的一个直接办法是穷举法, 或者说构造出它的真值表. 因为变量的个数是有限的, 而每个变量只有两种可能取值, 所以这种办法从理论上讲是可行的. 但在实际使用时, 它显然难以对付变量比较多的公式.

从算法复杂度上讲, SAT 是第一个被证明为 NP 难的问题 [35]. (关于复杂度方面的详细介绍, 可参看文献 [63, 41].) 根据目前计算机界的研究成果, 人们通常认为, 不太可能有多项式时间复杂度的算法. 但是人们后来发现, 大多数随机生成的问题实例并不是那么难解决 (这里我们所说的问题实例指的是, 给定一个具体的子句集合, 判断它是不是可满足).

对一些特定形式的公式, 存在多项式复杂度的可满足性判定算法. 下面是两类比较有名的特殊问题.

• 2SAT 问题. 它要求每个子句中最多只有两个文字. Aspvall 等人 [5] 提出了一个具有线性时间复杂度的 2SAT 算法.

• Horn 子句集的可满足性问题. Dowling 和 Gallier[46] 给出了两个解决 Horn 子句集可满足性问题的算法. 它们都具有线性的时间复杂度. (Scutellà[168] 后来指出其中一个算法是错误的, 并给出了正确的判定算法.)

可满足性问题还有不少变种, 其中一些是 NP 难问题 [63]. 下面列出的是比较常见的几类问题:

• 任意形式的布尔表达式是否可满足? 它们不一定是合取范式或析取范式.

• 带量词布尔表达式 (Quantified Boolean Formula 即

QBF ）的正确性．给定一组布尔变元 $\{ p_1, p_2, \ldots, p_n \}$ 以及由它们构成的布尔表达式 E，判断 $(Q_1p_1)(Q_2p_2)\ldots(Q_np_n)E$ 是否成立．这里的每个 Q_i 是 $\exists$（存在）或者 $\forall$（任意）．例如，$\forall p \exists q(p \to q)$ 是正确的，而 $\forall p(p \to \neg p)$ 是错误的．换句话说，前者的值为真，而后者的值为假．QBF 的求值问题是 PSPACE 完全的．如果每个 Q_i 都是 $\exists$，它就相当于可满足性问题．

• 唯一可满足性问题（ UNIQUE SAT ）：是否有唯一的赋值，使得给定的子句集合得到满足？从复杂度上讲，这个问题是 co-NP 难的 [12]．不过 Hansen 和 Jaumard[76] 以及 Pretolani[156] 分别得出结论，如果每个子句的长度不超过 2 或者所有的子句都是 Horn 子句，那么可用线性时间复杂度的算法来解决 UNIQUE SAT．

• 最大可满足性问题（ Max-SAT ）：找出一个赋值，使得被满足的子句数达到最多．即使限定所有的子句都是 Horn 子句，这个问题也是 NP 难的 [96]．

§1.2.2 约束满足问题

约束满足问题 (constraint satisfaction problem 简称 CSP)[123] 是人工智能领域被广泛研究的一类问题．其基本形式是这样的：给定有限个变量（或者叫未知量）以及每个变量的取值范围，要求找出所有变量的值，使得一些给定的约束条件被满足．如果每个变量的取值集合都是有限的，这样的问题被称为有限 CSP．

在 CSP 研究中，通常不考虑约束条件的具体形式．只要求在知道变量的值时，能判断约束条件是否被满足．有时可以这样来定义一个约束条件：列出所有满足该条件的变量取值组合．例如，设 x 和 y 是两个变量，其取值范围是整数集合．它们之间要

满足的一个约束条件是 $x = 2 * y$, 那么该条件可以表示为集合

$$\{ \ldots, \langle x = -2, y = -1\rangle, \langle x = 0, y = 0\rangle, \langle x = 2, y = 1\rangle, \ \ldots \}$$

当然，我们也可以列出不满足条件的所有取值组合. 之所以不考虑各种具体的约束条件形式，是为了避免问题的复杂化.

显然，很多具体的问题都可看成是 CSP 的特例. SAT 也是如此. 其中每个变量的取值范围都是真假值集合，而要满足的约束条件就是所给的命题逻辑公式.

另一方面，有限 CSP 也可被转换成 SAT. 我们介绍一种简单的转换方法 [42]. 假设 CSP 中的变量为: $x_1, x_2, \ldots, x_n$. 变量 x_i 的取值范围是 $D_i = \{a_{i1}, \ldots, a_{in_i}\}$. 对 CSP 的每个变量 x 和它的每个可能取值 a，引入布尔变元 $x : a$. 如果它为真，就表示 x 的值是 a. 对每个变量 x_i，我们需要生成下面的子句:

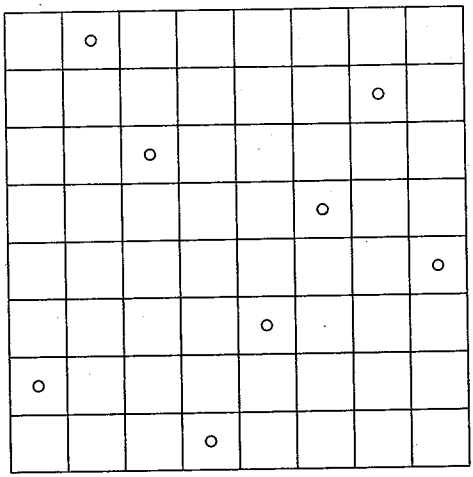
$$x_i : a_{i1} \vee \ldots \vee x_i : a_{in_i}$$
$$\neg x_i : a_{ij} \vee \neg x_i : a_{ik} \ \ (\text{对任何 } j, k,\ 1 \leq j < k \leq n_i)$$

其意义分别是，变量 x_i 必须有一个值，它属于集合 D_i；一个变量不能有两个不同的值. 此外，对每个约束条件 C，生成若干个如下形式的子句:

$$\neg x_1 : v_1 \vee \ldots \vee \neg x_m : v_m$$

这里 $x_1, \ldots, x_m$ 是 C 中所含的变量. 每个子句对应着不满足条件 C 的取值组合 $\langle x_1 = v_1, \ldots, x_m = v_m\rangle$.

例 著名的八皇后问题是，是否可以将 8 个皇后放在 8×8 的棋盘上，使得每行每列都有且仅有一个皇后，并且每条对角线上只有一个皇后. 下面是该问题的一个解 (其中的圆圈代表皇后).



在一般情况下，我们考虑 n 皇后问题（ n 为正整数）. 它可以看成是约束满足问题. 一种经典的表示方式是，引入 n 个变量 x_i（ $1 \leq i \leq n$ ），其中每个变量的取值范围是 1 到 n 之间的整数. 如果 $x_i = j$ ，就说明第 i 行、第 j 列有一个皇后. 要满足的约束条件如下：

$$x_i \neq x_j, \quad |x_i - x_j| \neq |i - j| \quad (1 \leq i < j \leq n)$$

我们也可以用命题逻辑来描述皇后问题. 为此引入 n^2 个命题变元 $p_{ij}(1 \leq i \leq n,\ 1 \leq j \leq n)$ ，其中变元 p_{ij} 为真表示在第 i 行、第 j 列有一个皇后（即上面的变量 x_i 取值为 j ）. 需要满足的命题逻辑公式包括如下子句：

- 每行至少有一个皇后：对每个 $i\ (1 \leq i \leq n)$ ， $p_{i1} \vee p_{i2} \vee \ldots \vee p_{in}$.
- 每行最多只能有一个皇后：对任何 i, j 和 $k\ (1 \leq i \leq n,\ 1 \leq j < k \leq n)$ ， $\neg p_{ij} \vee \neg p_{ik}$.
- 每列最多只能有一个皇后：对任何 i, j 和 $k\ (1 \leq j \leq n,\ 1 \leq i < k \leq n)$ ， $\neg p_{ij} \vee \neg p_{kj}$.

• 每条对角线上最多只能有一个皇后：对任何 i,j 和 k $(1 \leq i,j,k \leq n,\ k \neq i)$，

(a) 如果 $1 \leq j+i-k \leq n$，那么 $\neg p_{ij} \vee \neg p_{k,j+k-i}$.

(b) 如果 $1 \leq j+i-k \leq n$，那么 $\neg p_{ij} \vee \neg p_{k,j+i-k}$.

例 鸽笼 (pigeon-hole) 问题 [74]. 对正整数 n，公式 P_n 表示将 $(n+1)$ 个鸽子放到 n 个笼子中，并且每个笼子中最多只能有一个鸽子. 我们用整数串来表示子句（参看 §1.1 ）. 正整数 $n*(i-1)+j$ 表示将第 i 个鸽子放在第 j 个笼子中（ $1 \leq i \leq n+1, 1 \leq j \leq n$ ）. 例如，公式 P_3 等价于下面的子句集：

```
( 1 2 3 )      ( 4 5 6 )
( 7 8 9 )      ( 10 11 12 )
( -1 -4 )      ( -1 -7 )      ( -1 -10 )
( -2 -5 )      ( -2 -8 )      ( -2 -11 )
( -3 -6 )      ( -3 -9 )      ( -3 -12 )
( -4 -7 )      ( -4 -10 )     ( -5 -8 )
( -5 -11 )     ( -6 -9 )      ( -6 -12 )
( -7 -10 )     ( -8 -11 )     ( -9 -12 )
```

例 图着色问题. 给定图 1.1，是否能将两种颜色（红和蓝）分配给每个顶点，使得相邻顶点具有不同的颜色？

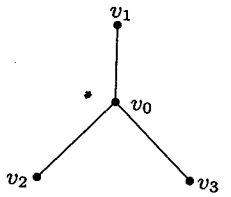


图 1.1

如果将问题看成是 CSP，那么有 4 个变量，每个变量有两种可能的取值. 要是按照上面所讲的方法转换成 SAT，我们需要引入 8 个命题变元： $p_1, \ldots, p_8$. 对于 $0 \leq i \leq 3$， p_{2i+1}（ p_{2i+2} ）为真表示顶点 v_i 的颜色为红（蓝）. 转换得到的子句如下：

$$
\begin{aligned}
&p_1 \vee p_2, \quad p_3 \vee p_4, \quad p_5 \vee p_6, \quad p_7 \vee p_8, \\
&\neg p_1 \vee \neg p_2, \quad \neg p_3 \vee \neg p_4, \quad \neg p_5 \vee \neg p_6, \quad \neg p_7 \vee \neg p_8, \\
&\neg p_1 \vee \neg p_3, \quad \neg p_2 \vee \neg p_4, \quad \neg p_1 \vee \neg p_5, \quad \neg p_2 \vee \neg p_6, \\
&\neg p_1 \vee \neg p_7, \quad \neg p_2 \vee \neg p_8.
\end{aligned}
$$

共有两个解（模型）：

$$
\begin{aligned}
&\{\, p_1, \neg p_2, \neg p_3, p_4, \neg p_5, p_6, \neg p_7, p_8 \,\} \\
&\{\, \neg p_1, p_2, p_3, \neg p_4, p_5, \neg p_6, p_7, \neg p_8 \,\}
\end{aligned}
$$

在第一个解中，顶点 v_0 染红色，其他三个顶点染蓝色；第二个解正好相反.

对于图着色这样的问题，我们也可以不要求每个 CSP 变量只取一个值. 也就是说，上面的第二行公式可以不要. 如果所得到的解中有某个顶点对应了两个颜色（比如说 p_1 和 p_2 都为真），那么我们可以任取其中一个. 采取这样的措施，可以减少子句的个数，使问题变得更简单.

上面所讲的这种转换方法未必对所有问题都是一种好方法. 有时并不需要那么多的命题变元. 看一个特殊情形. 如果把一个有 n 个命题变元的 SAT 问题实例当成是 CSP，再作上述转换，结果需要 $2n$ 个命题变元.

Iwama 和 Miyazaki[91] 提出另一种将 CSP 转换成 SAT 的办法. 他们的方法将整数值表示成二进制，每一位对应一个命题

变元. 这样会得到比较少的变元. 例如, CSP 的某个变量 x 有 8 种可能取值, 不妨设为 $0, 1, \cdots, 7$. 因此可以用 3 个命题变元 p_0, p_1, p_2 来表示 x 的所有取值情况, 而不需要 8 个变元. 这里 p_0 对应于二进制数的低位. 比如 $p_2 = 0, p_1 = 1, p_0 = 1$ 就表示整数 3.

在一般情况下, 对于命题变元 $p_0, p_1, \ldots, p_k$ 和非负整数 $m < 2^k$, 令 $C(p_k, \ldots, p_0, m)$ 表示这样一个子句, 它的值为假当且仅当二进制数 $(p_k \ldots p_0)$ 等于十进制数 m. 例如, $C(p_2, p_1, p_0, 3)$ 就是子句 $(p_2 \vee \neg p_1 \vee \neg p_0)$.

对于图的顶点着色问题, 如果有 n 个顶点和 m 种颜色, 那么采用上述办法至多需要 kn 个命题变元 ($k = \lceil \log_2 m \rceil$). 每个顶点 v_i $(0 \le i < n)$ 对应于变元 $p_{i0}, p_{i1}, \ldots, p_{i,k-1}$. 所要满足的条件包括如下子句:

1. 对于每个整数 t $(m \le t < 2^k)$ 和 i $(0 \le i < n)$, 构造子句 $C(p_{i,k-1}, \ldots, p_{i0}, t)$.

2. 如果第 i 个顶点和第 j 个顶点相邻 ($0 \le i, j < n$), 对每个整数 t $(0 \le t < m)$, 构造子句

$$C(p_{i,k-1}, \ldots, p_{i0}, t) \vee C(p_{j,k-1}, \ldots, p_{j0}, t)$$

注: Iwama 和 Miyazaki[91] 的转换方法中, 将 v_0 作为一个特殊顶点, 并假定它的颜色为 0. 这样会减少 k 个变元.

§1.3 Davis-Putnam 算法

解决约束满足问题的算法有很多, 其中最常用的是回溯搜索法 [106,146,157]. 既然 SAT 是 CSP 的特例, 这些方法当然也可以用来判定命题公式的可满足性. 但这样就没有考虑到约束条件的特殊性. 另外, CSP 的很多算法假定约束条件中只有两个变量;

而对于 SAT 来说，一般每个子句都有多于两个的变元. 所以 CSP 求解程序不宜直接用于判定命题公式的可满足性. 尽管如此，我们可以借鉴 CSP 领域的一些思想和技巧，设计出好的 SAT 算法.

§1.3.1 DP 算法

解决 SAT 问题的一个重要算法是 Davis 和 Putnam[40] 提出的方法. 后来被称为 Davis-Putnam 算法，简称 DP. Davis, Logemann 和 Loveland[39] 对它作了进一步的描述，所以它有时候也被称为 DPLL.

假定所给的 CNF 公式为 S（一组子句）. DP 算法不断地按照如下规则对 S 作变换，直到不能再进行为止.

重言式规则 删去 S 中所有的重言式，剩下的子句集记为 S'. S 可满足，当且仅当 S' 可满足. 一种常见的做法是，检查每个子句，看其中有没有互补的文字. 例如，$S = \{p \vee \neg p,\ q \vee r\}$，则 $S' = \{q \vee r\}$.

单子句规则 如果 S 中有一个单子句 L，那么 L 必须取真值. 我们可从 S 中删去所有包含 L 的子句（包括单子句本身），得到集合 S_1. 如果它是空集，则 S 可满足. 否则对 S_1 中的每个子句，如果它包含文字 $\neg L$，则从该子句中去掉这个文字. 由此得到子句集合 S_2. S 可满足，当且仅当 S_2 可满足. 例如，子句集 $\{p,\ p \vee \neg q,\ \neg p \vee r\}$ 的可满足性等同于 $\{r\}$ 的可满足性.

纯文字规则 如果文字 L 的补 $\neg L$ 不出现在子句集 S 中，那么 L 被称为纯文字（pure literal）. 从 S 中删去包含 L 的所有子句，得到集合 S'. 那么 S 可满足，当且仅当 S' 可满足. 例如，$S = \{p \vee q \vee \neg r \vee s,\ \neg p \vee q \vee \neg r \vee \neg s,\ \neg p \vee \neg q\}$，则 $\neg r$ 是纯文字，而 $S' = \{\neg p \vee \neg q\}$.

分裂规则 假设 S 可写成如下形式：

$$(A_1 \vee L) \wedge \ldots \wedge (A_m \vee L) \wedge$$
$$(B_1 \vee \neg L) \wedge \ldots \wedge (B_n \vee \neg L) \wedge$$
$$C_1 \wedge \ldots \wedge C_l$$

这里 A_i, B_j, C_k 都是子句，而且都不含 L 或 $\neg L$. 在这种情况下，我们可以生成两个比 S 简单的子句集：

$$S_1 = A_1 \wedge \ldots \wedge A_m \wedge C_1 \wedge \ldots \wedge C_l$$
$$S_2 = B_1 \wedge \ldots \wedge B_n \wedge C_1 \wedge \ldots \wedge C_l$$

那么 S 可满足，当且仅当 S_1 可满足，或者 S_2 可满足.

我们也可以这样来看分裂规则. 选取一个文字 L. 如果它取真值，则根据单子句规则，可将 S 化为 S_2；而 L 取假值（即 $\neg L$ 成立）时， S 可化为 S_1.

按照上述规则，我们可以不断地对子句集合进行化简. 算法显然是终止的. 其执行过程可以用二叉搜索树来表示. 每个结点对应着一组子句. 每个分支代表分裂规则，其左右子树分别对应所选文字取假值和真值的情况，其他边代表其他规则. 叶结点有两种：

(a) 子句集为空（ $\emptyset$ ），可满足.

(b) 子句集含有一个空子句（ $\square$ ），因而不可满足.

例 假定 $S = \{p \vee \neg q,\ p \vee r,\ \neg p \vee \neg q,\ q \vee \neg r\}$. 可以看出，其中没有重言式，也没有纯文字或者单子句. 所以我们先必须用分裂规则. 选取文字 p. 如果它取假值，则得到 $S_1 = \{\neg q,\ r,\ q \vee \neg r\}$. 其中有两个单子句. 利用第一个（即 $\neg q$ ），可得到 $S_{11} = \{r,\ \neg r\}$. 对 S_{11} 再使用一次单子句规则，得到空子句，因而 S_1 不可满足.

我们再看 p 取真值的情况. 这时 S 可化为 $S_2 = \{\neg q,\ q \vee \neg r\}$. 对它使用单子句规则（即让 $\neg q$ 成立），得到 $S_{21} = \{\neg r\}$.

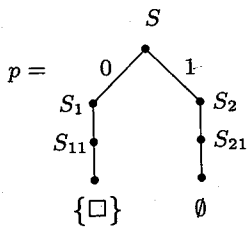


图 1.2　DP 算法搜索树

显然它可满足（只需给 r 赋假值）. 因此 S 可满足. 它的一个模型是 $\{p=1,\ q=0,\ r=0\}$.

上述搜索过程可用图 1.2 来表示.

在实现 DP 算法时，有些人（见文献 [15] ）认为纯子句规则会使程序变慢. 大家用得比较多的是单子句规则和分裂规则. 这时算法可写成如下的递归过程：

```
boolean function DP(S)
/* S 是子句集. 如果它可满足，返回 TRUE；
   否则返回 FALSE. */
{
  while (1) {
    从 S 中选单子句 L；
    如果选不到，则跳出循环;
    否则，根据单子句规则，利用 L 化简 S；
    如果 S 为空，那么 return(TRUE)；
    否则，如果 S 中有空子句，那么 return(FALSE)；
  }
```

从 S 中选一个文字 L；
return $\mathrm{DP}(S \cup \{L\}) \vee \mathrm{DP}(S \cup \{\neg L\})$；
}

§1.3.2 分支策略

在使用 DP 算法判定公式的可满足性时，一般都要使用分裂规则. 其执行过程可看成是二叉搜索树. 每使用一次分裂规则，也就是说在树的每个分叉点，需要选定一个未被赋值的文字. 如果选的方法不好（比如说随机地选一个文字），就有可能得到一棵很大的树，增加算法的执行时间. 为此，人们提出了各种各样的分支策略（branching strategy/rules）.

出现次数最多的变元优先 一种简单的分支策略是，根据变元在子句中的出现次数来选择. 在统计变元出现次数时，已经被满足的子句忽略不计. 出现次数越多，越有可能被选中. 我们看一个例子 [159].

例 对于给定的正整数 n，定义公式 E_n 如下：

$$E_n = \varphi_1 \wedge \varphi_2 \ldots \wedge \varphi_n \wedge \psi$$

这里

$$\varphi_i \equiv (p_i \vee \neg q_i) \wedge (\neg p_i \vee q_i),$$
$$\psi \equiv (r \vee s) \wedge (\neg r \vee s) \wedge (r \vee \neg s) \wedge (\neg r \vee \neg s)$$

不难看出，ψ 是不可满足的. 而 φ_i 实际上表示 $p_i \leftrightarrow q_i$，所以 $(\varphi_1 \wedge \ldots \wedge \varphi_n)$ 是可满足的，并且有 2^n 个解. 如果按照这样的顺序来选择变元：$p_1, \ldots, p_n, r$，那么 DP 算法搜索树的大小显然是指数级的. 但是，r 和 s 都分别在 4 个子句中出现，而其他每个变元只出现在两个子句中. 如果我们先选 r，那么立即就可以得出矛盾.

对上面的策略还可以作一些微小的改变. 例如，统计文字（而

不是变元）的出现次数. 再如，我们只统计每个变元在长度为 2 的子句中出现的次数. 如果变元 p 在子句 C 中出现，并且 C 只有两个文字，那么给 p 赋值后，就确定了 C 或者另外一个变元的真假值. 这样的推理非常有效.

Dubois 等人 [49] 讨论了几种分支策略. 算法 A-SAT 采用的策略是，优先考虑在最短子句中出现次数最多的变元. 它也被称为 Mom（ Maximum Occurences in clauses of Minimum size ）[54]. 另一种策略是，选取使下面的表达式达到最大值的变元 x：

$$f(x) + f(\overline{x}) + \alpha \min(f(x), f(\overline{x}))$$

这里 $f(l)$ 是文字 l 在最短子句中出现的次数，而参数 α 的值可通过实验确定，比如取 1.5. 如果再精确一点，除了最短子句以外，还可以考虑其他子句.

最短正子句优先 前一节中我们提到， SAT 是 CSP 的特例，而有限 CSP 可以转换成 SAT. 在解决 CSP 的算法中有一个常用的策略，就是选择那些可取值个数最少的变量. 由此得到 SAT 算法的一个分支策略：优先选择最短正子句中的文字. 下面的算法就采用了这种策略：

```
boolean function search(S)
/* S 是子句集. 如果它不满足，返回 FALSE；
   否则返回 TRUE. */
{
  对 S 使用纯文字规则和单子句规则;
  如果发现矛盾，则 return(FALSE)；
  如果 S 为空，则 return(TRUE)；
  如果 S 中没有正子句，则 return(TRUE)；
```

否则找一个最短的正子句 $x_{i_1} \vee x_{i_2} \vee \ldots \vee x_{i_l}$;

return(search($S \cup \{x_{i_1}\}$) $\vee$

search($S \cup \{\neg x_{i_1}, x_{i_2}\}$) $\vee$

$\ldots$

search($S \cup \{\neg x_{i_1}, \ldots, \neg x_{i_{l-1}}, x_{i_l}\}$));

}

如果 S 中没有正子句(即每个子句中都至少有一个负文字), 那么只需将所有变量取假值, 就可使 S 中所有子句被满足.

上面的算法采用了多分支搜索 [142], 其搜索树的每个结点可能有多于两个的子结点. 在第一个子结点中, x_{i_1} 取真值; 在第二个子结点中, x_{i_1} 取假值, x_{i_2} 取真值; …… 不过, 这也可以看成是若干个二元分叉的缩写.

Jeroslow-Wang Jeroslow 和 Wang[97] 提出一种分支策略, 我们简称为 J-W 策略. 其思想是, 尽可能得到可满足的子问题. 具体说来, 对每个文字 L , 定义 $J(L) = \sum_{i:\ L \in C_i} 2^{-n_i}$. 这里 C_i 是第 i 个子句, 而 n_i 是其长度 (即该子句所含文字的个数). J-W 策略建议, 在分支时, 先选择 $J(L)$ 值最大的那个文字 L.

他们的理由如下. 假设子句集中有 n 个布尔变元, 那么总共有 2^n 种可能的赋值. 其中有些使子句集被满足, 而另一些赋值使至少一个子句不成立. 不妨设子句 C_i 不成立, 也就是说其中每个文字都取假值. 这样, 另外的 $(n-n_i)$ 个变元不管取什么值, 子句集都是不可满足的. 所以, 在总共 2^n 个可能赋值中, 有 2^{n-n_i} 个赋值使 C_i 不成立. 如果将子句 C_i 去掉, 会减少这么多的 "坏" 赋值. 因此在使用分裂规则时, 如果选择文字 L , 使它为真, 就会去掉 $\sum_{i:\ L \in C_i} 2^{n-n_i} = 2^n J(L)$ 个坏的赋值. J-W 策略建议选取 $J(L)$ 值最大的文字 L. 这样, 所得到的子问题将很可能是可满足的.

Hooker 和 Vinay[87] 分析了 J-W 策略. 他们认为，该规则的成功之处在于，所得到的子问题比较简单，易于解决. 他们还研究了与它类似的一些策略，例如：

1. 选择使得 $J(L) - J(\neg L)$ 达到最大值的文字 L.

2. 选择使得 $J(\neg L)$ 达到最大值的文字 L.

3. 选择使得 $J(x_j) + J(\neg x_j)$ 达到最大值的变量 x_j. 如果 $J(x_j) \geq J(\neg x_j)$，就先让 x_j 取真值，否则先让 x_j 取假值.

此外还有其他一些更复杂的办法，详见文献 [87].

Hooker 和 Vinay 所做的实验结果表明，将第三条策略和正子句策略相结合，效果比较好. 也就是说，考察 S 中所有正子句中的变量，在其中选择使 $J(x_j) + J(\neg x_j)$ 达到最大值的变量 x_j. 如果 $J(x_j) \geq J(\neg x_j)$，就先让 x_j 取真值；否则，先让 x_j 取假值.

除了上面的策略以外，还有其他一些. Freeman 在他的博士论文 [54] 中，研究了几种分支策略，并实现了可满足性判定的原型工具 POSIT (Propositional Satisfiability Testbed)，其中一种策略是前面提到的 Mom. 另外一种是选择这样的文字，如果它被赋以真值，将使子句集含有尽可能多的长度为 2 的子句.

SATO 程序 [201] 采取一种混合型的办法，将前面提到的一些策略结合起来. 假设当前子句集中有 m 个最短非 Horn 子句，而 a 是一个介于 0 和 1 之间的参数. 令 $k = \lceil a * m \rceil$，$f_2(l)$ 为文字 l 在长度为 2 的子句中出现的次数，而 X 是前 k 个最短正子句中出现的所有变元构成的集合. 在使用分裂规则时，从 X 中选出使 $f_2(x) * f_2(\neg x)$ 达到最大值的那个变元.

§1.3.3 其他提高效率的手段

对于回溯搜索算法来说，分支策略非常重要. 除此之外，还

有其他方面也值得考虑. 比方说, 在搜索过程中, 应保持子句集中无多余文字. 如果子句 C_1 中的所有文字都出现在子句 C_2 中, 那么 C_2 可以被删掉. 这时我们称 C_2 被 C_1 所包含(subsumed). 例如, 子句集 $S = \{(p, \neg r),\ (\neg p, q, r),\ (p, \neg q, \neg r)\}$ 可简化为 $S' = \{(p, \neg r),\ (\neg p, q, r)\}$.

另外, 在遇到矛盾时, 如何进行回溯? 人们在研究 CSP 时, 提出了不少办法 [106,146,157]. 它们也可用于改进 SAT 算法. 例如, Silva 和 Sakallah[177] 描述了一个可满足性判别算法, 叫作 GRASP (Generic Search Algorithm for the Satisfiability Problem). 它的一个主要特点是进行 **冲突分析** (conflict analysis). 也就是说, 在算法执行过程中, 当出现矛盾时, 分析其原因. 如果引起矛盾的原因是在搜索树的较高层, 就直接回溯到那一层. 不一定要回到最近的分叉点. Bayardo 和 Schrag[6] 也采用了类似的方法.

对称性 Benhamou 和 Sais[7] 提出利用公式中的对称性来减少无效的搜索. 简单地说, 子句集 S 上的一个对称映射(symmetry) σ 是从 S 中所有文字构成的集合到它自身的一一映射, 使得 S 在此映射下保持不变, 而且对每个文字 l, $\sigma(\neg l) = \neg\sigma(l)$. S 是在下述意义下保持不变: S 中子句的次序可互换, 同一子句中的文字也可以交换位置, 而 $\neg\neg p$ 等同于 p. 比如 $S = \{p \vee \neg q,\ r \vee s\}$, 那么下面就是 S 上的一个对称映射:

$$\sigma(p) = \neg q,\quad \sigma(q) = \neg p,\quad \sigma(r) = r,\quad \sigma(s) = s$$

对于 S 中的文字 l_1 和 l_2, 如果有一个对称映射 σ 使得 $\sigma(l_1) = l_2$, 那么就称这两个文字是对称的 (记为 $l_1 \sim l_2$). 对上面的子句集, 我们有 $p \sim \neg q$.

给定子句集 S, 我们可将它的模型 M 表示成一个集合, 它

包含所有在 M 中为真的文字. 假设 σ 是 S 上的一个对称映射, 我们可以证明如下结论 [7] :

1. 对 M 中所有文字作 σ 变换, 所得到的集合也是 S 的模型.

2. 对于 S 中的文字 l_1 和 l_2 , 如果 $l_1 \sim l_2$, 那么下面两句话是等价的:

(a) S 有一个模型 M_1 , 它使得 l_1 为真 (即 $l_1 \in M_1$).

(b) S 有一个模型 M_2 , 它使得 l_2 为真 (即 $l_2 \in M_2$).

因此, 如果在搜索过程的某个分支点, 将变元 p 赋以真值会得到空子句, 那我们就将 $\neg p$ 以及所有与它对称的文字都加入到子句集中, 这样会得到更多的信息.

Benhamou 和 Sais 的方法中需要动态地寻找互相对称的命题变元, 这会带来较大的额外开销. 在下一章我们将会介绍更好的办法.

数据结构 软件的效率不仅取决于它的算法, 还受实现技术 (如数据结构) 以及编程语言等因素的影响. 对同样的算法, 一个好的实现和一个粗糙的实现, 其运行时间可能相差几十倍. 但是一般的文献中很少提到实现细节.

就 SAT 而言, 很多程序用链表或者类似的结构来表示子句和子句集. 至于表中含有哪些信息, 值得仔细考虑. 有的结构比较简单, 有的很复杂. Böhm 和 Speckenmeyer[15] 采用这样的数据结构:

(1) 对每个文字, 设一个链表, 记录它在哪些子句中出现.

(2) 每个子句由一个表头和若干个指向文字的指针来表示.

(3) 一个子句集也是一个链表, 其元素是各个子句的表头. 所有的子句按照其长度从小到大排列.

Rauzy[159] 描述了 DP 算法的一种实现. 该程序用稀疏矩阵

存放子句集. 矩阵的每行表示一个子句, 每列给出一个变元在子句集中的所有出现. 在算法执行过程中, 子句的状态分成四种:

(1) 被满足 (子句中有一个文字变为真),

(2) 取假值 (子句中所有文字都变为假),

(3) 长度缩短 (子句中有一部分文字变为假),

(4) 未受影响.

在后两种情况下, 我们称子句是 **活跃的**. 对于整个子句集, 我们统计每种子句的个数. 如果所有的子句都被满足, 就找到了解.

对每个变元 p, Rauzy 的程序记录文字 p 和 $\neg p$ 出现在哪些单子句中. 如果 p 出现在某个活跃的单子句中, 而 $\neg p$ 也出现在某个活跃的单子句中, 那么子句集就是不可满足的. 除了变元外, 还统计每个文字出现在活跃子句中的次数. 如果是 0, 它的补就是纯文字.

在实现 DP 算法时, 还可以采用其他数据结构. 张瀚涛和 Mark Stickel 在他们的程序 SATO 和 DDPP 中 [182,200], 都用 trie[103] 来表示子句集合. 这种数据结构本来是用在词典搜索中的. de Kleer[43] 首先将它用于存放合取范式形式的命题逻辑公式.

在 §1.1, 我们提到, 可以用一串正整数来表示一个子句. 如果用 trie 来表示子句集, 它就是一棵树. 树的每条边都有一个整数标记, 代表某个文字. 我们要求离树根越近的边, 它所带标记的绝对值越小. 每个叶结点对应着一个子句, 它所包含的文字就是从树根到这个叶结点的路径上所有的标记. 例如, 子句集 {(1, 2, 3), (-2, -3), (1, 2, 4)} 可表示为图 1.3.

采用 trie 结构的好处是, 节省空间, 并且可以有效地查出单子句和多余的子句. 另外, 由于多个子句可能共享几条边, 而且

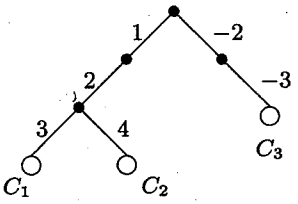


图 1.3　用 trie 表示子句集

边上的标记是从小到大排序的，所以能比较简单地对多个子句实施单子句规则. 当然，在 trie 上进行一些操作要复杂些，这给编程增加了难度.

并行处理　对于 NP 难的搜索问题来说，采用多处理机或者分布式系统，并不能从根本上解决问题. 但是，在有些情况下会得到比较好的效果. 在对多个分支同时搜索时，有可能沿着某一分支会很快找到解.

Böhm 和 Speckenmeyer[15] 实现了一个可在并行机(如 T800 Transputer)上运行的 SAT 软件. 每个处理器上有两个进程，其中一个负责处理子句集，另一个负责与其他处理器之间的通讯和协调. 前者以 DP 过程为基本算法. 它根据子句集中所含变元的个数来估计 DP 的运行时间. 如果时间不长，就调用串行的 DP 程序判定子句集的可满足性；否则，使用分裂规则得到两个较简单的子句集，并把它们放到队列中，以便进一步处理. 第二个进程根据公式队列的长度，进行负载均衡 (load balancing). 如果某个处理器的队列很长 (也就是说有很多子句集要处理)，而另一个处理器的队列很短，那么就可将一些子句集由前一个处理器发送到后一个处理器.

Zhang, Bonacina 和 Hsiang[202] 将 SATO 并行化，使之能

在多台工作站上同时执行. 新程序 PSATO 是在美国阿尔贡国家实验室开发的 C 软件包 P4 之上运行的. 它在解决一个 SAT 问题实例时, 有一个主进程 (master) 负责分配任务, 将搜索空间划分为一些互不相交的子空间; 若干个从属进程 (slave) 在所分配的子空间内进行搜索. 每个任务可用二元组 $\langle S, P\rangle$ 来表示, 其中 S 是子句集, P 是搜索树中由根到其他结点的一条路径. 当然, 搜索子空间并不是事先划分好的, 而是动态生成的. PSATO 的特点是: 通迅量小, 从属进程之间基本上互相独立, 因而不需要很复杂的负载均衡算法. 另外, P4 软件使得 PSATO 可在普通计算机上运行, 而不需要特殊的并行机.

§1.4 局部搜索法

局部搜索 (local search) 是解决很多 NP- 难问题 (特别是优化问题) 的一种常用方法. 回溯法是一个一个地确定变量的值, 逐渐地将部分解扩充为完整的解. 而局部搜索法的基本思想是, 先任意地给每个变量取一个值, 得到一个赋值. 它可能使一部分子句的值为真, 另一部分子句的值为假. 然后反复地对现有的赋值作局部调整 (例如改变一、二个变量的值), 使得被满足的子句个数越来越多. 在理想情况下, 最终将得到一个解, 它使所有的子句被满足.

基于上述思想, 顾钧 (Jun Gu) 给出了解决 SAT 的一系列搜索过程 [69,70], 分别命名为 SAT1.0 、 SAT1.1 、 SAT2.0 、 SAT3.0 , 等等. Russell 和 Norvig[165] 认为, 顾钧最早将局部搜索法用于 SAT.

Bart Selman 等人 [172,170] 也于 1992 年前后提出贪心搜索过程 GSAT. 他们实现了这一过程, 并将源代码公开. 后来很多人以此程序为基础, 对局部搜索法作进一步的研究.

在国内，也有不少人研究如何用局部搜索过程解决 SAT. 例如，李未、黄文奇等人 [89,112] 采用拟物拟人的思想，实现了高效率的 SAT 程序. 所谓拟物，就是观察物理世界，寻找与 SAT 等价的物质运动形式，从中找出解决问题的有效办法. 在某种意义下，SAT 等同于带电质子在静电场中的势函数是否为零的问题. 拟物途径实际上也是采用优化的办法. 而拟人途径是从人类在几千年社会实践中积累的丰富经验中得到启发，从而设计出解决数学问题的算法.

假设子句集 S 中有 n 个命题变元. 对这些变元的一种赋值可看成是 n 维空间中的一个点. 总共有 2^n 个点. 对每一个点（赋值）θ，在该赋值下取假值的子句个数记为 $g(\theta)$. 为了判定子句集 S 的可满足性，我们需要找出一个点 α，使得 $g(\alpha)=0$.（往往称 g 为目标函数.）开始时我们从搜索空间中随机地选取一点. 然后在每一步，我们检查当前点的相邻点，看看函数 g 在某个点的值是否比在当前点的值小. 如果是，就以那一点替换当前点.

究竟什么样的点算是相邻，这有不同的定义. 对 GSAT 来说，如果两个赋值仅有一个变元的值不同，那么它们就为邻点. 因此，该过程的一个基本操作是将某个变元的值取反，称为翻转（flipping）.

对变元 p 和赋值 θ，我们引入整型量 $fi(\theta,p)$ 和 $br(\theta,p)$. 它们的含义是这样的. 如果当前赋值是 θ，那么当 p 翻转后，有 $fi(\theta,p)$ 个子句的值由假变为真，有 $br(\theta,p)$ 个子句的值由真变为假. 因此，翻转 p 会使被满足子句的个数增加 $\delta(\theta,p)=fi(\theta,p)-br(\theta,p)$.

GSAT 的具体定义见图 1.4. 该过程中的 MAX_TRIES 和 MAX_FLIPS 是事先给定的常数.

```
GSAT()
{
  for(i = 0; i < MAX_TRIES; i++) {
    随机地为每个变量选择一个真假值,
      从而得到初始赋值 θ0 ;
    for(j = 0; j < MAX_FLIPS; j++) {
      如果 θj 使得所有子句都为真, return(可满足) ;
      如果集合 {p | δ(θj,p) ≥ 0} 为空,
        就跳出内循环;
      否则, 从该集合中选择 δ 值最大的变元 p ;
      将 p 的值取反, 得到新的赋值 θj+1 ;
    }
  }
}
```

图 1.4 局部搜索过程 GSAT

我们把外循环的每次执行叫做一遍（ try ）. GSAT 的一个特征是，如果对当前点 θ 的所有邻点 θ_1 ，都有 $g(\theta_1) \geq g(\theta)$ ，那么也可以从那些具有相同目标函数值的邻点中随机地选一个，作为新的赋值. 换句话说，既可以沿着使函数值下降的方向走，也可以作平移动作. 当然，也有可能所有邻点的目标函数值都比 $g(\theta)$ 大，这时如果 $g(\theta) \neq 0$ ，那我们就称搜索过程掉进了局部极小点. 例如，图 1.5 中的点 P_1 就是局部极小点，因为其函数值大于点 P_0 处的值. 在这种情况下需要重新选一个初始赋值，再做一遍搜索.

很显然，如果给定的子句集不可满足，局部搜索过程不能给我们任何有益的信息. 当过程结束时，如果还没有找到解，那么我们不能判定子句集是否可满足. 所以严格地讲，像 GSAT 这样的过程不能称为 SAT 算法. 但是，Koutsoupias 和 Papadimitriou[104] 用概率论的分析手段得出结论：对于绝大多数可满足的 3SAT 问

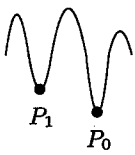


图 1.5　局部极小点

题，局部搜索法几乎总能成功地找到解.

例　假设我们有 6 个子句 $c_1, c_2, \ldots, c_6$，其中

$$
\begin{aligned}
c_1 &: \neg p_1 \vee p_2 \vee \neg p_3 \\
c_2 &: \neg p_2 \vee p_3 \vee p_4 \\
c_3 &: p_1 \vee p_2 \\
c_4 &: p_1 \vee \neg p_4 \\
c_5 &: \neg p_2 \vee \neg p_4 \\
c_6 &: \neg p_3 \vee p_4
\end{aligned}
$$

开始时我们随机地得到一个赋值，如 $\theta_0 = \{\neg p_1, p_2, \neg p_3, \neg p_4\}$. 它使得子句 c_2 为假，而其他子句都为真. 因此 $g(\theta_0) = 1$. θ_0 有四个邻点，分别是

$$
\begin{aligned}
\theta_1 &= \{p_1, p_2, \neg p_3, \neg p_4\}, \\
\theta_2 &= \{\neg p_1, \neg p_2, \neg p_3, \neg p_4\}, \\
\theta_3 &= \{\neg p_1, p_2, p_3, \neg p_4\}, \\
\theta_4 &= \{\neg p_1, p_2, \neg p_3, p_4\}.
\end{aligned}
$$

经过简单的计算可知，$g(\theta_1) = g(\theta_2) = g(\theta_3) = 1$，$g(\theta_4) = 2$. 因此我们从 θ_1, θ_2 和 θ_3 中随机地选一个，作为新的赋值. 假设

θ_2 被选中. 于是我们检查 θ_2 的四个邻点 θ_0, θ_5, θ_6 和 θ_7, 其中 $\theta_5 = \{p_1, \neg p_2, \neg p_3, \neg p_4\}$ 使得 6 个子句都被满足. 因而它就是一个解, 所给的子句集合可满足.

改进措施 与其他局部搜索过程类似, GSAT 有不少缺点. 它很容易陷入局部极小点. 后来人们提出了各种改进措施 [170,171], 其中之一是给子句加权. 子句 C_i 的权(weight)是正整数 k_i, 其含义是在计算目标函数时把 C_i 当成 k_i 个子句. 当某个赋值使得 C_i 取假值时, 我们将 g 的值加上 k_i , 而不是 1. 改进的 GSAT 过程 [170] 按照下面的方法给子句加权. 每一遍开始时, 所有子句的权都是 1 ; 在一遍结束时, 如果某个子句在当前赋值下取假值, 就把它的权加 1.

执行局部搜索过程中, 如果碰到局部极值点, 很可能会有一些变元的值不断地翻过来转过去. 为避免这种现象, Mazure 等人 [131] 提出 TSAT 过程. 它采用一个先入先出的禁忌队列(tabu list), 将最近翻转过的变元按照先后次序记录下来. 在搜索过程的每一步, 如果按照贪心策略所选出的变元已经在队列中, 那就不翻转它, 而是翻转另一个不在队列中的变元. 这样我们就能在一定程度上减少重复翻转. 当然, 队列的长度是一个需要仔细考虑的重要参数. 在极端情况下, 如果它等于 0 , 那么 TSAT 就和 GSAT 一样.

对 GSAT 过程的一个比较明显的改进是, 在其中加上随机游走 (random walk) [170,171]. 我们简要介绍 Selman, Kautz 和 Cohen 提出的 WSAT 过程 [171]. 在每一步按照下面的方法确定一个布尔变元 q , 然后将其值取反. 从未被当前赋值满足的所有子句中, 随机地选择一个子句 C ; 从 C 中随机地或者是根据贪心策略选一个变元 q. 也就是说, 除了下降、平移以外, 还允许一定的爬升动作, 以便逃出局部极值点. Bram Cohen 用 C 语

言实现的 WalkSAT 程序虽然只有 1000 行左右的源代码，但是其效率很高.

Parkes 和 Walser[151] 描述了如下两种从子句 C 中选变元的具体办法：

1. WSAT/G 的做法是，以一定的概率 ρ 随机地从 C 中选一个变元，而以 $(1-\rho)$ 的概率从 C 中选一个使 $\delta(\theta,q)$ 取最大值的变元 q.

2. WSAT/SKC 的做法是，令 $u=\min\{\, br(\theta,p) \mid p$ 是出现在 C 中的变元 $\}$. 如果 $u=0$，则从 C 中选一个满足条件 $br(\theta,q)=0$ 的变元 q. 当 $u>0$ 时，以概率 ρ 随机地从 C 中选一个变元，而以 $(1-\rho)$ 的概率从 C 中选一个使 $br(\theta,q)$ 达到最小值的变元 q.

上面我们提到，以概率 ρ 做某事，而以概率 $(1-\rho)$ 做另一件事. 这是用一个随机变量 r 实现的. 它的取值范围是 [0,1]. 在当前时刻究竟做哪件事，这取决于 r 的值是否比 ρ 大.

McAllester 等人 [133] 还提出一种基于"新颖性"(novelty) 的策略. 它的基本做法和 GSAT 类似，只是在有若干个变元的 δ 值都为最大值时，挑选最早被翻转的那个，其动机和采用禁忌队列的策略一样，都是为了减少重复翻转.

§1.5 有序二叉判定图

前面两节我们介绍了解决 SAT 的算法和过程. 它们要求先把命题逻辑公式（或者叫布尔表达式、布尔函数）转换成合取范式. 这在有些情况下不是很方便. 本节我们介绍另一种范式.

首先我们引入布尔算子（逻辑连接符）**if-then-else**，简记为 ITE. 它可定义为

$$\mathrm{ITE}(p,e_1,e_2)=(p\wedge e_1)\vee(\neg p\wedge e_2)$$

这里， p 是布尔变量， e_1 和 e_2 是布尔表达式.

对于布尔表达式 f 和其中的变量 x , 用 $f[x \leftarrow 0]$ 和 $f[x \leftarrow 1]$ 分别表示将 x 代换以 0 或 1 后所得到的表达式. 我们不难证明下面的恒等式:

$$f \equiv x \cdot f[x \leftarrow 1] + \overline{x} \cdot f[x \leftarrow 0] .$$

这就是所谓的 Shannon 展开式. 它实际上就是

$$f \equiv \mathrm{ITE}(x, f[x \leftarrow 1], f[x \leftarrow 0])$$

据此很容易将布尔表达式表示成 ITE 范式. 这种形式的公式中只有 ITE , 没有其他的连接符, 而且变量总是作为 ITE 的第一个参量. 例如

$$\neg x = \mathrm{ITE}(x, 0, 1), \quad x \vee y = \mathrm{ITE}(x, 1, \mathrm{ITE}(y, 1, 0))$$

我们往往用二叉判定树来形象地表示 ITE 范式. 树的每个叶节点标以 0 或 1 (表达式的值); 而非叶节点 v 以变量 $\mathrm{var}(v)$ 为标记. 非叶节点 v 的两个子节点记为 $\mathrm{low}(v)$ 和 $\mathrm{high}(v)$, 它们分别对应于该变量取值 0 或 1 的情况. 把节点 v 代表的布尔表达式记为 $be(v)$. 它可以递归地定义如下: 如果 v 是叶节点， $be(v)$ 就是它所标记的真假值; 否则

$$be(v) = \mathrm{ITE}(\mathrm{var}(v), be(\mathrm{high}(v)), be(\mathrm{low}(v)))$$

例 公式 $x \vee \neg y$ 等价于 $\mathrm{ITE}(x, 1, \mathrm{ITE}(y, 0, 1))$. 它可以用图 1.6(a) 中的判定树来表示.

我们也可将判定树看成是有向图. 每个非叶节点到它的每个子节点有一条边, 而树根和叶节点可分别看成是有向图中的源点和终点. 所有的叶节点可合并成两个: 其中一个标记为 0 , 另一个

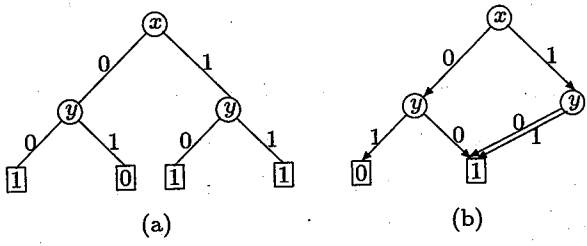


图 1.6　二叉判定树和 BDD

标记为 1. 如图 1.6(b) 所示. 这样的图称为 **二叉判定图**（Binary Decision Diagrams，简称 BDD）[2]. （注：在有些文献中，用虚线表示由 v 到 low(v) 的边，用实线表示由 v 到 high(v) 的边.）

在从源点到终点的每条路径中，每个变量最多只出现一次. 这样的路径对应着给变量的一种赋值，而其终点表明在这种赋值下函数所取的值. 每条终点为 1 的路径对应一种使布尔函数得到满足的赋值.

判定树（图）和真值表比较直观，但是它们的大小随着变量的个数成指数级增长. 因此，只有在变量不太多的情况下，才用这两种方式来表示布尔函数. 后来人们对 BDD 进行改进，以得到更简洁的表示形式.

Bryant[18,19] 对二叉判定图加以限制，得到所谓的 **有序二叉判定图**（Ordered Binary Decision Diagrams，简称 OBDD）. 它是一个与 BDD 类似的有向图. 所不同的是，它事先对变量加以排序，并要求：如果节点 u 到节点 v 有一条边，那么 u 的变量号必须比 v 的变量号小（除非 v 是终点）. 越靠近终点的内节

点，其变量的序号越大．在从源点到终点的每条路径中，变量都必须按照给定的顺序出现．例如，图 1.6(b) 中，变量的排列顺序为 $x < y$.

除了变量序的限制以外，还要对判定图进行一系列的化简．这主要是为了删除多余的顶点．通常采用两种方法：

1. 对两个不同的节点 u 和 v，如果它们标记的变量相同，而且其子节点分别相同，即 $\mathrm{low}(u) = \mathrm{low}(v), \mathrm{high}(u) = \mathrm{high}(v)$（如图 1.7(a) 所示），那么这两个顶点只需要保留一个（比如 u）．而所有指向 v 的边改为指向 u.

2. 如果某个节点 v 的两个子节点为同一个节点，那么 v 和它的两条输出边可以去掉，而所有指向 v 的边改为指向 v 的子节点．比如图 1.6(b) 可以简化为图 1.7(b).

可以看出，上面这两条措施并不改变判定图所代表的布尔函数．反复对一个 OBDD 作这样的简化，最终得到的图被称为 **精简的有序二叉判定图**（Reduced Ordered Binary Decision Diagrams，简称 ROBDD）．不过，有关文献中的 OBDD 通常就是指 ROBDD.

OBDD 有一些比较好的性质.

唯一性 OBDD 是一种规范的（canonical）表示形式．给定一个布尔函数及其变量上的一种排序，相应地只有一个 OBDD．这个性质可用来判定两个函数的等价性．我们只要在同样的变量序下分别构造出它们的 OBDD．如果所得到的图相同，那么两个函数就等价．在为一个布尔函数构造出相应的 OBDD 之后，我们还可以很容易地判断它是不是永真公式，是不是可满足.

复杂性 OBDD 的一个主要好处在于，它比较简洁，布尔运算易于实现．对很多常见的布尔函数来说，其 OBDD 的节点数不多．比如加法器函数就可以有线性复杂度的 OBDD．当然，

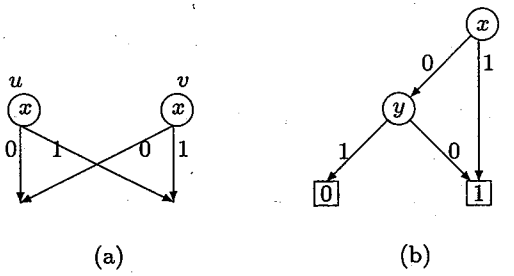


图 1.7 简化 OBDD

也有些布尔函数，不管采用什么样的变量排序，其 OBDD 的大小都是指数级的. Bryant[20] 指出，n 位乘法器的输出中至少有一位，它所对应的 OBDD 需要 $\Omega(1.09^n)$ 个节点. Devadas[45] 给出了另一个布尔函数. 用经典的 DNF 形式来表示它，公式的长度为多项式量级；但如果用 OBDD 来表示，则其大小为指数级的.

OBDD 的构造 给定一个布尔表达式和一种变量排序，如何构造出相应的 OBDD 呢？对于真假值 0 和 1 以及单个变量 x，其 OBDD 是显而易见的. 在一般情况下，不妨设表达式的形式是 $f\ op\ g$. 这里，op 是逻辑连接符，f 和 g 是子表达式. 在得到 f 和 g 的 OBDD 之后，我们需要将它们合二为一. 这个运算通常被记为 APPLY.

我们可以递归地定义 APPLY 运算. 假设 f 和 g 所对应的 OBDD 分别以 r_f 和 r_g 为根. 如果 r_f 和 r_g 都是终节点（对应于布尔常量），那么直接就可以算出 $(f\ op\ g)$ 的值. 它所对应的节点就是 APPLY 运算的结果. 在含有变量的情况下，APPLY 运

算的实现依赖于下面的事实：

$$f\ op\ g \equiv \mathrm{ITE}(x, f[x \leftarrow 1]\ op\ g[x \leftarrow 1], f[x \leftarrow 0]\ op\ g[x \leftarrow 0])$$

也就是说，我们可以选一个变量 x（称为分裂变量），分别考虑它等于 1 和等于 0 的情况，构造出两个子树．每个子树是对 $f[x \leftarrow b]$ 和 $g[x \leftarrow b]$ 所对应的 OBDD 施以 APPLY 运算而得到（这里的 b 取值 0 或 1 ）．显然这是个递归过程．

怎样确定分裂变量呢？如果 $\mathrm{var}(r_f) = \mathrm{var}(r_g)$ ，那么就选它作为 x．因为这时我们很容易找到 $f[x \leftarrow 1]$，$f[x \leftarrow 0]$，$g[x \leftarrow 1]$ 和 $g[x \leftarrow 0]$ 所对应的节点．它们分别是 $\mathrm{high}(r_f)$，$\mathrm{low}(r_f)$，$\mathrm{high}(r_g)$，$\mathrm{low}(r_g)$．如果 $\mathrm{var}(r_f)$ 和 $\mathrm{var}(r_g)$ 不相等，比如说 $\mathrm{var}(r_f) > \mathrm{var}(r_g)$ ，那么就选比较小的那个作为分裂变量，即 $x = \mathrm{var}(r_g)$．之所以如此，是因为不管 b 等于 0 还是 1 ，$f[x \leftarrow b]$ 对应的节点就是 r_f．而 $g[x \leftarrow 1]$ 和 $g[x \leftarrow 0]$ 所对应的节点分别是 r_g 的两个子节点．假如我们让 $x = \mathrm{var}(r_f)$ ，那就不太好确定 $g[x \leftarrow b]$ 对应的节点．

对上面所述的 APPLY 运算还可以作一些优化．例如，如果 op 是“与”，而 r_f 是终节点 0 ，那就不需要选分裂变量，而是直接返回终节点 0．此外，还可以用一张表记录中间结果，以避免重复计算．该表的每一项对应于二元组 $\langle i, j \rangle$ ，表示 $\mathrm{APPLY}(i, j)$ 所得到的节点号．

由于 OBDD 是精简的判定图，所以在构造过程中一般还要用到一个哈希表（ hash table ），其作用是避免生成同样的节点．我们可以根据三个参数 x, l, h 对表进行查询．这里，x 是变量号，l 和 h 是节点号．如果目前已经有一个节点 v ，满足 $\mathrm{var}(v) = x$，$\mathrm{low}(v) = l$，$\mathrm{high}(v) = h$，那么查询结果就是 v．这时就不需要分配新的节点号．

除了 APPLY 运算外，还有其他运算，如 RESTRICT. 它从函数 f 中选择一个变量 x，将其赋以真假值，得到一个新的函数，也就是本节开始时提到的 $f[x \leftarrow 0]$ 或 $f[x \leftarrow 1]$. 给定 f 的一个 OBDD，我们对其进行深度搜索. 在为 $f[x \leftarrow 0]$ 构造 OBDD 时，对于 $\mathrm{var}(v) = x$ 的节点 v，我们将指向 v 的边改成指向 $\mathrm{low}(v)$. 如果要为 $f[x \leftarrow 1]$ 构造 OBDD，就将指向 v 的边改成指向 $\mathrm{high}(v)$. 同 APPLY 运算一样，为了保证所得到的判定图是精简的，在 RESTRICT 的实现过程中也要用到哈希表.

利用 APPLY 和 RESTRICT 运算，我们可以对布尔函数进行各种操作，比如两个函数的复合. 此外，我们还可以根据下述等式判断 QBF 的真假值.

$$\begin{aligned} \exists x f &= f[x \leftarrow 0] + f[x \leftarrow 1] \\ \forall x f &= f[x \leftarrow 0] \cdot f[x \leftarrow 1] \end{aligned}$$

变量排序 给定两个不同的变量排序，由同一个布尔函数可以得到不同的 OBDD. 它们的节点数可能相差很大. 一个经典的例子 [18,19] 是布尔函数 $x_1y_1 + x_2y_2 + \ldots + x_ny_n$. 如果所采用的序是 $x_1 < y_1 < \ldots < x_n < y_n$，那么得到的 OBDD 只有 $2n+2$ 个节点；而采用的序如果是 $x_1 < \ldots < x_n < y_1 < \ldots < y_n$，那么所得到的 OBDD 将有 2^{n+1} 个节点. 图 1.8 是 $n = 2$ 的情形.

在很多情况下，都能找到变量的一种排序，使得 OBDD 节点的个数相对于变量的个数是多项式的关系（而不是指数级的）. 但是，怎样找到比较好的排序，这没有一个通用的办法. 很多人在这方面进行了有益的探索. 所采用的方法大致有这几种：

1. 基于电路拓扑结构的启发式方法 [124,57]. 对于不少常见的电路（特别是树状结构的电路），从其输出端往输入端进行深度优先搜索，可以得到较好的变量排序.

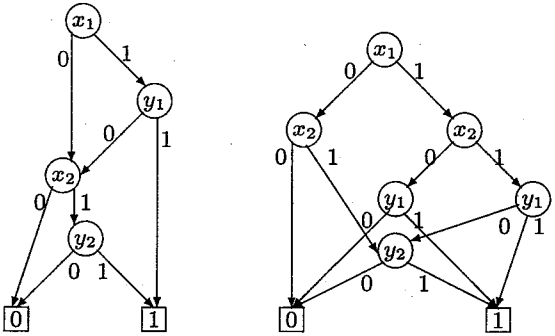


图 1.8　变量排序对 OBDD 大小的影响

2. 交换变量法 [92] 或者叫动态排序法 [163]. 这实际上是一种局部搜索、逐步改进的过程. 它通过交换相邻的两个变量达到减少节点数的目的. 这种交换操作只影响两层节点, 因此不会太费时间.

3. 寻找最优排序的穷举法.

此外, Fujii 等人 [56] 针对多输出端的电路提出了一种获得变量排序的办法.

目前 OBDD 主要用于门级数字电路的正确性验证 [19]. 在这方面, 它是最有效的方法. Moore[144] 指出, OBDD 之所以具有比较强的功能, 主要有以下原因:

(1) 采用固定的变量排序使得对布尔函数的操作变得简单;

(2) 通过哈希表实现结构共享;

(3) 在进行 APPLY 操作时, 存放中间结果以便将来使用.

软件工具的效率当然还和实现技术有关. Brace 等人 [17] 描述了高效实现 OBDD 的一些细节, 包括哈希表和内存管理技

术.

最近几年，有些学者对 OBDD 进行了各种扩展，以适应不同的应用. 例如， Bryant 和 Chen[21] 提出 Binary Moment Diagram (BMD). 它能比较方便地表示乘法器.

§1.6 语义表和 Stålmarck 方法

§1.6.1 语义表

前面介绍的 Davis-Putnam 算法和 OBDD 方法都可以理解为根据变量的取值进行穷举，从而判定公式的可满足性. 除此而外，我们也可以根据子公式来穷举. 例如，为了确定公式 $(p \to q) \vee (q \to p)$ 是否可满足，我们可以分别检查子公式 $(p \to q)$ 和 $(q \to p)$. 只要它们中有一个是可满足的，那么原来的公式也可满足. 这也是一种按情形分析，把大问题化成小问题的手段. 但它不是按照某个变量的真假值来分别处理.

基于上面这种思想，逻辑学家们通过构造 **语义表**(semantic tableau) 来研究公式的性质. 给定命题逻辑公式 φ，它的语义表实际上可看成是一棵树，其中每个结点包含一些公式. 从树根到树叶的一条路径被称为一个分支 (branch). 它代表可能存在的 φ 的一个模型. 如果一个分支中同时含有某个命题变元和它的补，该分支就被称为 **封闭的** (closed). 我们用 $\otimes$ 来表示. 如果表的每个分支都是封闭的，那么公式 φ 就是不可满足的.

构造语义表的过程是这样的. 开始时只有根结点，它所包含的公式为 φ. 然后按照一定的规则逐步地对表进行扩展. 每次使用一条规则，由表的某个叶结点 v 生成它的若干个子结点. 规则的一般形式是，如果当前分支 (即从树根到 v 的路径) 中有某种形式的公式，那么就可以往新结点中加入某些公式. 在同一分支中，不要重复地对某个公式使用同样的规则.

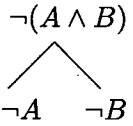
对经典的命题逻辑来说，我们有如下一些规则：

规则(1) $A \wedge B$.

如果分支中有公式 $A \wedge B$，则我们可生成一个新结点，其中包含公式 A 及 B. 因为当 $A \wedge B$ 成立时，很显然 A 和 B 都成立. 如下所示：

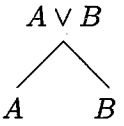
规则(2) $\neg(A \wedge B)$.

如果分支中有公式 $\neg(A \wedge B)$，则我们可生成两个新结点，分别包含公式 $\neg A$ 和 $\neg B$. 其意思是，当 $\neg(A \wedge B)$ 成立时，有两种可能：一是 $\neg A$ 成立，一是 $\neg B$ 成立.



规则(3) $A \vee B$.

如果我们有公式 $A \vee B$，则有两种可能： A 成立，或者 B 成立.



规则(4) $\neg(A \vee B)$.

如果我们有公式 $\neg(A \vee B)$，则可加上新公式：$\neg A$，$\neg B$.

$$\begin{array}{c} \neg(A \vee B) \\ | \\ \neg A \\ \neg B \end{array}$$

规则(5) $A \to B$.

如果我们有公式 $A \to B$，则有两种可能：$\neg A$ 成立，或者 B 成立.

$$\begin{array}{ccc} & A \to B & \\ & \diagup \quad \diagdown & \\ \neg A & & B \end{array}$$

规则(6) $\neg(A \to B)$.

如果我们有公式 $\neg(A \to B)$，则可加上新公式：A，$\neg B$.

$$\begin{array}{c} \neg(A \to B) \\ | \\ A \\ \neg B \end{array}$$

规则(7) $\neg\neg A$.

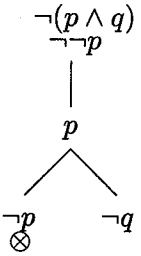
如果我们有公式 $\neg\neg A$，则可加上新公式 A.

$$\begin{array}{c} \neg\neg A \\ | \\ A \end{array}$$

当然，我们还可以有更多的类似的规则，来处理其他形式的公式，如 $(A \leftrightarrow B)$，$\neg(A \leftrightarrow B)$，等等.

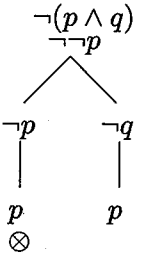
给定一个（或一组）公式，反复运用上述规则对一个分支进行扩展，直到不能再进行下去为止. 如果该分支不是封闭的，就得到一个模型. 否则，继续扩展其他分支. 如果所有分支都是封闭的，那么所给定的公式是不可满足的.

例 假设我们有两个公式 $\neg(p \wedge q)$ 和 $\neg\neg p$. 先后利用规则 (7) 和规则 (2)，可得到如下的语义表：



其中第一个分支含公式 p 和 $\neg p$，因此是封闭的（用 $\otimes$ 表示）. 而从另外一个分支可得到一个模型：$\{p, \neg q\}$. 容易验证，当 p 为真、q 为假时，所给的两个公式都为真.

一般说来，应该尽量推迟使用产生多个新结点的规则，如规则 (2)、(3)、(5). 对于上面的例子，如果先用规则 (2)，再用规则 (7)，则语义表如下：



其中规则 (7) 使用了两次.

§1.6.2 Stålmarck 方法

瑞典学者 Gunnar Stålmarck 于 20 世纪 80 年代末发明了一种命题逻辑定理证明方法, 90 年代中期获得了美国和欧洲的专利. 目前, Prover 公司已利用这种方法开发出商业化的证明器. 但是, 公开发表的关于 Stålmarck 方法的资料却很少 (主要是文献 [173]). 下面我们简单地介绍它的基本思想.

我们先回忆一下 Davis-Putnam 过程. 该算法的执行情况可以看成是一颗二叉树. 在每个分叉处, 我们假定某个变量取真值或假值. 然后根据一定的推导规则 (主要是单子句规则), 得到其他变量的值.

Stålmarck 方法比 DP 过程更复杂. 它不仅考虑输入公式中各个变量的取值, 还考虑公式的所有子公式及其间的逻辑关系 (主要是等价关系). 在分叉处, 可以假定某个子公式 (而不仅仅是某个变量) 取真值或假值. 然后使用很多推导规则, 导出子公式之间新的关系.

子公式原理 如果公式 φ 是永真的, 那么一定有个证明, 其中只含 φ 的子公式或其否定, 而不牵涉到其他公式.

一个布尔表达式可以用二叉树来表示 (假定只有一元或二元连接符), 其树叶是布尔变量, 而每个非叶节点对应一个连接符. 树中每个节点代表一个子公式. 在 Stålmarck 方法中, 为每个非叶节点引入一个新的临时变量名, 以便对相应的子公式作处理. 假如我们有表达式 $(a \vee \neg b) \wedge c$, 可引进变量 $t_1 : \neg b$, $t_2 : a \vee t_1$, $t_3 : t_2 \wedge c$. 在搜索过程中, 既可以选择表达式中原有的变量进行分支, 也可以选择某个临时变量 (如 t_3).

公式之间的等价关系 所有的子公式及其补 (以及真值和假

值）可以划分成一些等价类。在同一类中的公式具有相同的真假值，每个子公式和它的补属于不同的等价类。等价类的集合定义了一种等价关系。

如果我们要判定公式 φ 的可满足性，可以先假定它和真值（1 或 ⊤）等价。假设我们有公式 $(a \wedge b)$。开始时等价关系如下：

$$\{(a \wedge b) \equiv 1,\ \neg(a \wedge b) \equiv 0,\ a,\ b,\ \neg a,\ \neg b\}$$

也就是说有 6 个等价类。我们要找的解（即公式的模型）也可表示成等价关系。例如，公式 $(a \wedge b)$ 只有一个模型，就是 $a = b = 1$。对应于下面的等价关系：

$$\{(a \wedge b) \equiv a \equiv b \equiv 1,\ \neg(a \wedge b) \equiv \neg a \equiv \neg b \equiv 0\}$$

关系推导规则 那么怎样从已知的等价关系推得新的关系呢？这可以使用一定的推导规则。例如，由 $a \oplus b = 0$ 可推出 $a \equiv b$。这可以写成

$$\frac{a \oplus b \equiv 0}{a \equiv b}$$

一般说来，推导规则具有这样的形式：

$$\frac{L_1 \equiv R_1, \cdots, L_n \equiv R_n}{L_0 \equiv R_0}$$

这里，L_i 和 R_i 可以是真假值，也可以是布尔变量或其否定，还可以是简单的布尔表达式 $x \circ y$（$\circ$ 是二元连接符）。

当然，这些规则要满足一定的条件。首先前提之间不能互相矛盾，否则由它们可推出任何结论。这样的规则没什么意义。其次，从给定的 n 个前提关系中去掉任何一个，都不能推出结论 $L_0 \equiv R_0$。

下面是一些具体的推导规则：

$$\frac{a \wedge b \equiv 1}{a \equiv 1} \qquad \frac{a \wedge b \equiv 1}{b \equiv 1} \qquad \frac{a \equiv b}{a \wedge b \equiv a} \qquad \frac{a \equiv b}{a \wedge b \equiv b}$$

$$\frac{a \vee b \equiv 0}{a \equiv 0} \qquad \frac{a \vee b \equiv 0}{b \equiv 0} \qquad \frac{a \equiv b}{a \vee b \equiv a} \qquad \frac{a \equiv b}{a \vee b \equiv b}$$

$$\frac{a \equiv \neg b}{a \wedge b \equiv 0} \qquad \frac{a \equiv \neg b}{a \vee b \equiv 1} \qquad \frac{a \equiv (a \to b)}{a \equiv 1}$$

$$\frac{a \equiv 1,\ (a \to b) \equiv 1}{b \equiv 1} \qquad \frac{a \equiv 0,\ b \equiv 0}{(a \vee b) \equiv 0}$$

上面这些规则被称为 **简单规则** 。

两难推理规则（ dilemma rule ） 这是一种分支及合并（ branch and merge ）规则。给定公式之间的关系 R，从 R 的两个不同（但不互补）的等价类中选取两个公式 A 和 B 。我们分别考虑两种可能：一是 A 与 B 等价，一是 A 与 $\neg B$ 等价。一个特殊情况是， B 为真。这时我们考虑的两种可能性是： A 为真，或者 A 为假。对每一种可能，我们通过逻辑推导，对 R 进行扩充，分别得到 R_1 和 R_2 。在第一个分支的推导过程中，如果发现 R_1 的某个等价类中包含了互补的公式（即矛盾），那么使用两难规则后， R 被扩充为 R_2 。反之亦然。如果在两个分支都没有发现矛盾，那么将 R_1 和 R_2 的交作为结果，也就是将 R 扩充为 $R_1 \cap R_2$ 。

该规则的运用可以形象地用下图表示：

$$\begin{array}{c} R \\ \hline \begin{array}{cc} \text{假定} A \equiv B & \text{假定} A \equiv \neg B \\ \Downarrow & \Downarrow \\ R_1 & R_2 \end{array} \\ \hline R_1 \cap R_2 \end{array}$$

在 $A \equiv B$ 的假设下，由 R 推出 R_1 的过程中，可以再使用两难推理规则。也就是说，这种规则的使用可以互相嵌套。证明过程中所嵌套的最多假设的个数被称为 **证明深度**（ proof depth ）。如果把证明过程看成是一棵树，把两难推理规则的每次使用看成是一个非叶节点（分支点），那么证明深度就相当于树的深度。当然，在由 R 推出 R_1 或 R_2 的过程中，一般也要使用大量的简单规则。但是，这样的推理被看成是在某一条边中进行，不影响证明深度。这和 DP 算法类似，其搜索树的深度决定于分裂规则的使用。

Stålmarck 方法按照公式的证明深度来寻找其证明。关于该方法的详细情况，可参看文献 [173] 。基于 Stålmarck 方法的证明器目前可以处理非常复杂的命题逻辑公式（比如含有几十万个连接符的公式）。它已在欧洲的铁路系统得到应用。

§1.7 其他途径

本节我们谈谈求解 SAT 问题的其他途径。有关思想也可推广到非子句形式的命题逻辑公式。

§1.7.1 随机探索算法

Wu 和 Tang[198] 提出一种蒙特卡罗式的随机算法。对于事先

选定的失败概率 ϵ，该算法给出正确答案的可能性至少是 $(1-\epsilon)$。给定子句集 C 及失败概率 ϵ，过程 $W(C,\epsilon)$ 如下：

在 C 中找空子句；
若找到，则 return(不可满足)；
从 C 中删掉那些显然可满足的子句
（即同时含一个文字及其补的子句）；
剩下的文字个数记为 m；
$k = ln(\epsilon)/ln(1-(1-\frac{1}{m})^m)$；
for($i=1$; $i \leq k$; $i++$) {
  随机生成一个赋值；
  如果该赋值使 C 中所有子句为真，
  则 return(可满足)；
}
return(C 不可满足)；

可以看出，如果算法给出的答案是“可满足”，那么子句集 C 确实可满足；但如果答案是“不可满足”，该答案未必正确。

从理论上讲，上述过程能以任何给定的概率来近似地解决 SAT 问题。但从实用的角度来看，它显然不是一个好方法。所有赋值都是随机生成的。而在局部搜索过程中，一般说来，后一个赋值比前一个有所改进。

另外一种较好的办法是所谓的“重复随机取样”（iterative sampling）。它可以看成是由 DP 过程演变而来。它保留了单子句推理规则，但是没有回溯。如果碰到矛盾，就从头（即树根）开始，而不是一步步地往回走。

下面是 Crawford 和 Baker[37] 用来解决 SAT 问题的过程

Isamp(C)：

```
for(i = 0; i < MAX_TRIES; i++) {
    取消所有变量的值;
    while (1) {
        如果所有变量都有值， return(可满足)；
        随机选一个未被赋值的变量 p；
        随机地给 p 取一个真假值;
        对子句集 C 反复地施以单子句规则;
        如果得到矛盾，就跳出循环;
    }
}
return(失败)；
```

§1.7.2 转换为多项式问题

我们可以将 SAT 转换为其他类型的问题，比如说 0-1 整数规划（ Integer Programming ）问题[85,86]。我们用一个简单的例子来说明一种常见的转换方法。给定子句 $(p_1 \vee p_2 \vee \neg p_3)$，我们得到不等式 $x_1 + x_2 + (1 - x_3) \geq 1$，也即 $x_1 + x_2 - x_3 \geq 0$。这里的每个变量 x_i 只能取值 0 或 1。

整数规划也是比较难的问题。一种解决办法是将它看成线性规划（ Linear Programming ）问题，再用有关工具得到一个实数解，然后对其取整而得到整数解。当然，只有在一定条件下，才可以这样做。 Chandru 和 Hooker[29] 定义了“扩展（ extended ） Horn 子句集”。对这种子句集合，如果将其转换成线性不等式组，那么恰好符合上述条件。换句话说，扩展 Horn 子句集的可满足性可以用线性规划软件来判定。

Schlipf 等人 [166] 给出了判定扩展 Horn 子句集可满足性的算法。他们的算法具有多项式时间复杂度，且不需要识别过程。也就是说，对任何子句集都可使用此过程。如果所给的是扩展 Horn 子句集，则肯定能判定其可满足性，否则有可能会失败（即不能给出结论）。

其他学者也提出了另外的方法，将 SAT 转换为多项式优化问题，比如 Gu 提出的问题模型 UniSAT 和 UniSAT7 等 [71,72]。假设所给的 SAT 实例中有 n 个变元，m 个子句。互补的文字不出现在同一个子句中。UniSAT7 通过下述方式将 n 维布尔空间上的约束求解问题转换成实空间上的全局优化问题。对应于每一个命题变元 p_j，我们引入一个实变量 x_j（$1 \leq j \leq n$）。每个子句 C_i（$1 \leq i \leq m$）对应于实函数：

$$c_i(\vec{x}) = d_{i1} * d_{i2} * \ldots * d_{in}$$

其中

$$d_{ij} = \begin{cases} (x_j - 1)^2, & \text{如果 } p_j \in C_i \\ (x_j + 1)^2, & \text{如果 } \neg p_j \in C_i \\ 1, & \text{否则} \end{cases}$$

整个子句集对应于实函数

$$s(\vec{x}) = c_1 + c_2 + \ldots + c_m.$$

如果我们能找到一组值 $x_j \in \{-1, 1\}$，使得上述目标函数能达到最小值 0，那么原来的 SAT 实例就是可满足的。使它满足的赋值是这样定义的：

$$p_j = \begin{cases} \mathbf{T}, & \text{如果 } x_j = 1 \\ \mathbf{F}, & \text{如果 } x_j = (-1) \end{cases}$$

举一个简单的例子。子句集 $\{p_1 \vee p_2 \vee \neg p_3,\ \neg p_2 \vee p_3\}$ 将被翻译成多项式 $(x_1-1)^2(x_2-1)^2(x_3+1)^2+(x_2+1)^2(x_3-1)^2$。它的零点对应了原来的 SAT 问题的解。

非线性规划是非常困难的一类问题，直接用这方面的算法去解 SAT 未必合适。但是，可以借鉴已有的非线性优化技术，针对命题逻辑公式设计出新的方法（如各种局部搜索法）。

我国著名数学家吴文俊在研究初等几何定理的机器证明时，提出求解多项式方程组的一套算法 [199]。贺思敏 [80] 研究了如何利用这种方法来解决 SAT。他采用下述变换 f 得到一组多项式方程组：

- 对每个布尔变元 p_i，引入变量 x_i。
- $f(1)=0,\ f(0)=1.$
- $f(p_i)=1-x_i,\ f(\neg p_i)=x_i.$
- $f(l_1 \vee l_2 \vee \cdots \vee l_k)=f(l_1)*f(l_2)*\cdots*f(l_k).$

这里，每个 l_i 是文字。

子句集 $S=\{C_1,C_2,\ldots,C_m\}$ 对应的方程组为

$$f(C_j)=0,\quad j=1,2,\ldots,m$$
$$x_i(1-x_i)=0,\quad i=1,2,\ldots,n$$

这种变换方法的特点是，包含一组方程 $x_i(1-x_i)=0$。这些方程限制了 x_i 的取值只能是 0 或者 1。不难看出，由方程组的解可以得到一个使 S 满足的赋值。

§1.7.3 有向图

Aspvall 等人 [5] 提出一个解决 2SAT 的线性时间算法，其基本思想是将子句集 S 转换成有向图。每个文字对应一个顶点。如果有子句 (l_1,l_2)，就构造两条边：一条由 $\neg l_1$ 到 l_2，另一条由

$\neg l_2$ 到 l_1。然后找出图的强连通分量(这可在线性时间内做到). 如果某个分量中含一个文字和它的补，那么 S 不可满足；否则 S 可满足。在可满足的情况下，如果有一条从文字 l 到 $\neg l$ 的路径，就将 l 的值定为假。

Dowling 和 Gallier 也基于图论的思想，提出了解决 Horn 子句集可满足性问题的两个算法 [46,168]。它们将子句集 S 转换成图 G_S。每个命题变元对应图中的一个顶点，此外还有两个顶点分别表示布尔常量 **T** 和 **F**。S 不可满足，当且仅当有一条路径将 **T** 和 **F** 连起来。

Gallo 等人 [60,61] 通过超图(hypergraph)来研究 SAT 问题。一个超图由一个顶点集合和一个超弧集合组成。每个超弧(hyperarc)，或者叫超边(hyperedge)，包括头顶点集合和尾顶点集合。这两个集合的交集为空。

将合取范式转换成超图的方法是这样的。如前所述，超图的顶点是所有命题变元加上真假值。每个子句对应一个有向超边。假设子句采用前件蕴涵后件的形式(参看 §1.1)，那么前件中的变元构成尾顶点集合，而后件中的变元构成头顶点集合。例如，子句 $p \vee \neg q \vee r$ 可转换为 $q \rightarrow p \vee r$，相当于如下的超边它的头顶点集合是 $\{p, r\}$，尾顶点集合是 $\{ q \}$, 如图 1.9 所示。用超图来表示子句集，就将可满足问题转换成图的路径问题。

超图这种表示方式还可用来进行问题分解。在实际应用中往往遇到很大的子句集。如果我们能事先将其划分为若干个比较小的子句集，再利用前面几节的算法来求解，就比较容易。Park 和 Van Gelder[150] 研究了如何得到这种划分。他们也把子句集看成是超图，然后利用比较成熟的超图划分技术。与 Gallo 等人不同的是，他们将命题变元看成是超边，而不是顶点。每条边连接该变元所出现的所有子句。

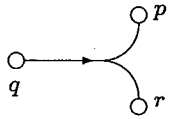


图 1.9　用超边表示子句.

§1.7.4　利用易解的特殊可满足性问题

前面（§1.2）我们提到，Horn 子句集的可满足性问题和 2SAT 可以用线性时间复杂度的算法来解决。这样的特殊算法也可作为 SAT 算法的一部分，以提高其效率。比如说，子句集 S 中有很多 Horn 子句，而我们用线性时间算法判断出这些子句是不可满足的，那么 S 也就不可满足。另外，在搜索过程中，随着某些变元得到值，子句集不断被简化，也有可能变成 Horn 子句集。

一个子句集虽然不是 Horn 子句集，但有可能通过一定的转换，变成 Horn 子句集。一种办法如下：选择一些变元，将其中每个变元在子句中的所有出现都换成其补，而将变元的补换成变元自身。例如，对于下面的子句集：

$$\{x_1 \vee x_2 \vee \neg x_4,\ \ x_1 \vee \neg x_3,\ \ x_3 \vee x_4\}$$

我们可以选择变量 x_1 和 x_3，进行以上置换，得到 Horn 子句集：

$$\{\neg x_1 \vee x_2 \vee \neg x_4,\ \ \neg x_1 \vee x_3,\ \ \neg x_3 \vee x_4\}$$

Chandru 等人 [28] 以及 Hébrard[83] 分别给出了线性时间的算法，来判断一个子句集是不是能通过上面的办法变成 Horn 子句

集。如果能这样做，我们就可以利用判定 Horn 子句集可满足性的线性时间算法 [46,168]，来解决原来的可满足性问题。

很多非 Horn 子句集不能直接通过上述办法转换成 Horn 子句集。Chandru 和 Hooker[30] 提出了这样的办法：选择其中一部分变元并对其赋值，然后化简子句集，再作上述转换。当然，应该选择尽可能少的变元。遗憾的是，怎样找到最小的变元集，这本身就是一个 NP 难问题。

Gallo 和 Urbani[59] 提出两个方案，利用 Dowling 和 Gallier 的算法来解决一般子句集的可满足性问题。它们的思想都是通过引入新的变元，将非 Horn 子句转换成特定形式. 其中第二个方案 HORN2 效率比较高。对于每个非 Horn 子句

$$C_i: \quad p_1 \wedge \ldots \wedge p_l \rightarrow q_1 \vee \ldots \vee q_m$$

它引入新的变元 w_i，并用下面两个子句代替 C_i：

$$p_1 \wedge \ldots \wedge p_l \rightarrow w_i$$
$$w_i \rightarrow q_1 \vee \ldots \vee q_m$$

其中前一个是 Horn 子句，后一个子句中只有一个负文字。给定一组子句 S，HORN2 按照上述变换得到集合 $S_H \cup S_N$，其中 S_H 是 Horn 子句集，而 S_N 中的每个非 Horn 子句只含一个负文字。对于前一个集合，可以用线性时间算法求解；而对于后一个集合，采用类似于“最短正子句优先”的策略进行分支。如果开始时 S 中有 n 个非 Horn 子句，那么采用算法 HORN2 时，搜索树的深度不超过 n。

在 Crawford 和 Auton 的程序 TABLEAU[36] 中，Horn 子句和非 Horn 子句也是分开处理的，其分支策略之一也是优先考虑含正文字最少的非 Horn 子句。

Larrabe[109] 将组合电路测试码模式的自动生成问题转换为 SAT，并且利用 2SAT 来解决该问题。Hermann Stamm 编写了一个基于 DP 算法的程序 [24]，其中也用到 2SAT 求解过程。它的做法是这样的：如果 S 中有单子句，就采用单子句规则；否则找出其中所有长度为 2 的子句，并为它们构造一个有向图 G，然后采用 Aspvall 等人 [5] 的算法。当然，在一般情况下，还是要运用分裂规则。

Gent 和 Walsh[64] 对 DP 算法作了这样的改进：对每个文字 l，先假设 l 为真，使用单子句规则之后所得到的子句集记为 S_l；如果 S_l 中所有长度为 2 的子句所构成的集合是不可满足的，那么 l 的值就只能是假。

§1.7.5 约束逻辑程序设计

在人工智能领域关于约束满足问题的研究中，有相当长一段时间，人们并不关心约束条件的具体形式。大家只是假设有某种方式，能够检查一个赋值是否使约束条件得到满足。当然，也有少数约束求解系统，它们有自己的约束描述语言。

在 20 世纪 80 年代，逻辑程序设计方式受到很大的关注，Prolog 语言颇为流行。但是人们在解决一些实际问题时逐渐发现，它的效率不太高。后来，一些研究人员对它进行扩充，将约束求解的机制加进去。这就是所谓的“约束逻辑程序设计”（Constraint logic programming，简称 CLP）[95;189]。

近 15 年来，世界各地研制出了各种不同的 CLP 系统。有的系统将线性规划算法结合到逻辑推理中，如 CLP($\mathcal{R}$)；有的系统擅长解决有穷域上的约束满足问题，如 CHIP, clp(FD) …。

命题逻辑公式的可满足性问题可看成是有限 CSP 的特殊情形。因此，针对有穷域的 CLP 系统也可解决像 SAT 这样的问

题。 CLP 系统中常用的一种推理手段是值传播。假设我们有约束条件 $y = \neg x$ 。那么当 x 被赋值时，就可推出 $y = 1 - x$ 。另一方面，如果 y 被赋值，则 x 等于 $1 - y$ 。这里我们将布尔值当成整数 0 和 1 。实际上， Stålmarck 方法中就采纳了这样的思想。

在变量的值不确定时，也能进行推理。比如说，如果我们有约束条件 $x + y < 3$ ，而 x 和 y 的取值范围分别是 $[0, 4]$ 和 $[3, 5]$ ，那么就可推出矛盾。 在一般情况下，对每个变量 x ，我们用 $\max(x)$ 和 $\min(x)$ 表示它的上界和下界。当某些变量的上下界发生变化时，可以推出矛盾，或者改变另一些变量的上下界。

§1.8 各种方法的分析、比较与结合

前面几节我们介绍了命题逻辑公式可满足性判定的一些方法。一般说来，很难从理论上证明某个方法比另一个更优越。有的在这一类问题上很有效，而有的则适合于另一类问题。每一种方法又都有一些缺点。那么怎样对它们进行分析和比较呢？有没有可能将它们结合起来呢？

§1.8.1 理论分析

衡量一个算法好坏的经典办法是从理论上对它进行分析。从复杂度上讲，命题逻辑公式的可满足性问题是 NP 难的，因而不太可能有多项式时间的算法来解决这一类问题。这可以说是可满足性判定算法时间开销的“下界”。再看上界。给一个有 n 个变元的公式，最简单的可满足性判定办法就是穷举 2^n 种可能。对 2^n 这个界限还可作一些改进。有些算法 (如文献 [142, 210, 105]) 的最坏时间复杂度要比它小。也就是说，对于任何给定的公式，算法都能在不到 2^n 步内终止，并判断出公式的可满足性。对于 3SAT

来说，目前最好的上界大约是 1.5^n 。

除了最坏情况外，还可以研究算法的平均复杂度。Purdom[158] 分析了一些判定命题公式可满足性的回溯算法。它们包括最简单的回溯过程及其变种（比如加上单子句规则或者纯文字规则）。他采用这样的随机问题实例。假设有 n 个命题变元，那么就有 $2n$ 个不同的文字。在构造一个子句时，以概率 p 选取每个文字。这样得到的子句可能会含有互补文字。

§1.8.2 通过实验结果来比较

最近 10 年，人们越来越注重可满足性判定方法的有效实现，以及根据实验结果对它们进行比较。比较的标准主要是运行时间和空间。但是这不仅取决于算法和策略的好坏，也受编程语言和硬件的影响。有的程序用 C 语言写，有的则用 LISP 语言。因此除了程序的执行时间外，我们也要看其他指标，比如回溯过程所生成的搜索树的大小（结点或树枝的个数）以及在每个结点上所花的时间。

在比较两种方法或程序好坏时，我们还要考虑输入数据（即问题实例）。一种简单的办法是随机地生成命题逻辑公式。但是由此而得到的实验结果有时候难以令人信服 [140] 。

在对 SAT 算法做实验时，常常是事先固定好若干参数（如子句长度、子句个数、变元个数），然后根据它们随机地生成一些问题实例。上面我们提到 Purdom 所用的生成随机实例的办法。除此之外，大家还采用其他方法，比如子句长度固定的 随机 k-SAT 实例 [141] 。也就是说，假设每个子句中都有 k 个文字。将子句和命题变元的个数分别记为 m 和 n 。每次随机地选取 k 个变元，并将每个变元以 0.5 的概率取反，这样就得到一个长度为 k 的子句。独立地重复做 m 次，就得到一个有 m 个子句的集合。

随机 SAT 问题实例的难度随着有关参数的变化而变化。最近几年，大家通过实验发现了影响难度的一个重要参量，就是子句个数与变元个数之比 $\frac{m}{n}$ 。当 $\frac{m}{n}$ 很小时，变元很多，子句很少（比如说只有一个子句），很容易满足；而当 $\frac{m}{n}$ 很大时，变元很少，子句很多，很容易出现矛盾，子句集往往不可满足。这两种极端情形都比较容易解决，而处于中间的则较难。

对于 3SAT 来说，很多学者所做的大量实验表明，在 $\frac{m}{n}$ 大约为 4.25 时，可满足概率为 0.5 。一般认为，在这个比值下随机生成的 3SAT 问题实例非常难解决，这就是所谓的相变(phase transition)现象。 它原本是物理学中的概念。在人工智能领域，Cheeseman 等人于 1991 年通过实验发现，CSP 具有相变现象 [32] 。他们指出，不少约束满足问题都有一个参数。当它处于某个值时，问题实例比较难；而参数值太大或太小时都很容易解决。后来 Mitchell 等人 [141] 针对 SAT 问题进一步研究了这种现象，并描述了怎样生成比较难解的问题实例。国际《人工智能》杂志还针对相变现象出了一个专辑 [84] 。

需要指出的是，对于现实世界中的结构性问题，问题实例的难度变化不一定符合上述规律 [130] 。

问题实例的“难”与“易”也和搜索算法有关。有些人认为，局部搜索法适合于有很多解的问题。所以在测试、比较这类软件的性能时，也可以采用只有一个解的问题实例。Asahiro, Iwama 和 Miyano[4] 提出了一套随机生成 SAT 实例的方法，后来被称为 AIM 。通常的随机实例既可能是可满足的，也可能是不可满足的。而 AIM 可以专门生成不可满足的实例，也可生成只有 1 个解的可满足实例。 Gent 和 Walsh[64] 用 DP 算法做了很多实验。他们发现，即使在随机 SAT 的易解区域（比如说 $\frac{m}{n} = 2$ ），也有非常难的 SAT 实例。

还有人专门提出具有某种特殊结构的难解性问题 [74,188] 。比较有名的是鸽笼问题。因为公式是不可满足的，所以不能采用局部搜索法。而对于 DP 算法来说，目前大多数程序在几分钟的时间里只能解决 $n \leq 10$ 的情形。由此可见，变量不多的 SAT 实例也可能非常难。

同样的问题可以有不同的表示方法。变量个数多、搜索空间大的子句集，不一定就难解。 Iwama 和 Miyazaki[91] 提出一种基于二进制的方法，将 CSP 转换成 SAT（参看 §1.2.2 ）。这种方法的特点是，可以得到比较少的命题变元。但是后来人们做的实验结果表明，这种转换方法对于像 WalkSAT 这样的搜索过程未必有好处 [88] 。

Uribe 和 Stickel[187] 用几类问题实例比较了 OBDD 方法和 DP 过程。他们选用了当时（ 1994 年）具有代表性的软件工具。他们的实验结果表明，在数字电路问题上， OBDD 很有效；但对于皇后问题和一些随机生成的 3SAT 问题， DP 算法更好。一般来讲， OBDD 适合于有很多解而且解之间有重复结构的问题。

1993 年德国人 Michael Buro 和 Hans Kleine Büning[24] 举办了 SAT 程序比赛。他们随机地生成了 7 组公式，每组包括 40 个合取范式。取得最好成绩的是德国 Düsseldorf 大学的 Max Böhm 和 Hermann Stamm 实现的基于 DP 算法的程序。 1996 年在北京也举办了基于局部搜索法的 SAT 程序比赛。黄文奇、金人超等人实现的算法 Solar[89] 获得第一名。

目前在可满足性问题研究中影响比较大的是 DIMACS 问题集。 DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) 是美国自然科学基金会设在 Rutgers 大学的一个科技中心，它是该校和 Bell 实验室、 Princeton 大

学、NEC 研究所等的合作项目。它在 1993 年、1996 年曾先后两次举办关于 SAT 的研讨会。

随机生成的问题实例比较简单，很容易获得，对于算法研究和程序实现也有一定的帮助。但我们认为，更重要的是针对特定的应用，确定哪些方法能提高效率，并理解为什么会这样。可满足性算法和工具最终是要用来解决实际问题的。因此，我们更看重各种应用领域中的典型问题实例。实际中常见的问题不一定具有很高的难度，但它们可能不太规整，也可能有很多变量。我们认为，只要在某一类问题上有所贡献，这样的算法就有意义。

§1.8.3 回溯法与不完备方法的结合

张健和张瀚涛 [209] 提出一种将回溯法和不完备方法结合起来的方案。在极端情形，它可看成是回溯过程或局部搜索过程。它将问题求解过程分为两个阶段：在前一阶段，选取一部分变元，采用不完备的方法得到一个部分赋值；在后一阶段采用回溯法将这个部分赋值扩展为完整的解。

Mazure 等人 [132] 用另外一种方式将 DP 算法和局部搜索过程 TSAT[131] 结合起来。该方法的主要框架是 DP，而 TSAT 被用于分裂变元的选择。他们认为，不可满足的子句集合 S 往往有个不可满足的子集 $S' \subset S$，在局部搜索过程中取假值次数最多的子句很可能属于 S'。因此，DP+TSAT 采取这种做法：先对 S 施以单子句规则，然后调用 TSAT 过程；再选择那些在未满足子句中出现次数最多的变元作分支，得到两个新的子句集，并重复上述过程。

吕卫锋和张玉平 [122] 研究了怎样针对给定的问题实例，选择不同的求解过程（包括 DP 算法、局部搜索法和随机试探法）。他们提出两个参量 ϕ 和 δ，并通过实验发现了一些经验规则。对

不同的问题实例，可根据这些规则以及参量值的大小，选用不同的算法。

第二章　一阶谓词逻辑

前一章所涉及的命题逻辑比较简单，它只可用来分析和表示原子命题之间的逻辑关系．本章将讨论更复杂些的一阶逻辑，对原子命题的内部结构进行剖析，以便能更细致、更准确地推理．但这也是有代价的．我们没有一个算法，能够判断任意给定的一阶逻辑公式是否可满足．因此，我们只能在一定的限制条件下来解决可满足性问题．

§2.1　一阶谓词逻辑简介

一阶谓词逻辑又称一阶谓词演算或者量词理论，简称一阶逻辑．它研究一些对象（称为 **个体** ）以及它们之间的关系，在这些对象上可进行运算（不仅仅是逻辑运算）．除了前一章所提到的那些逻辑连接符外，还可以用全称量词或存在量词来组成公式．本节我们简要介绍一阶逻辑中的基本概念和结论．

语法

一阶逻辑语言中通常有以下四个可数集合：

- 个体变量（或变元）集 $X = \{x_1, x_2, \cdots\}$；
- 个体常量（或常元）集 $C = \{c_1, c_2, \cdots\}$；
- 函数符集 $F = \{f_1^1, f_2^1, \cdots, f_1^2, f_2^2, \cdots, f_1^3, f_2^3, \cdots\}$，其中 f_i^n 表示第 i 个 n 元函数符；
- 谓词符集 $R = \{P_1^1, P_2^1, \cdots, P_1^2, P_2^2, \cdots, P_1^3, P_2^3, \cdots\}$，其中 P_i^n 表示第 i 个 n 元谓词符．

有时我们也把常量看成是 0 元函数符．

由这些集合我们可以递归地构造出项集合和公式集合.

定义（项）

1. 个体常量和个体变量是项.

2. 如果每个 t_i 都是项，则 $f_i^n(t_1, \cdots, t_n)$ 也是一个项.

3. 只有由上面两条规则生成的表达式是项.

例 假设 a 和 b 是常量，x, y 和 z 是变量，g 是一元函数符，f 是二元函数符，那么下面三个表达式都是项：

$$f(a, g(b)) \quad f(x, f(y, z)) \quad g(f(f(x, y), z))$$

为了构造一阶公式，我们需要引入全称量词符 $\forall$ 和存在量词符 $\exists$，其含义分别是“对任何个体”和“存在某个个体”.

定义（公式）

1. 如果每个 t_i 都是项，则 $P_i^n(t_1, \cdots, t_n)$ 也是一个公式，称为原子公式.

2. 如果 ψ 是公式，则 $(\neg\psi)$ 也是公式.

3. 如果 ψ_1 和 ψ_2 是公式，则 $(\psi_1 * \psi_2)$ 也是公式. 这里 $*$ 可以是任何一个二元连接符，如 $\vee, \wedge, \rightarrow, \leftrightarrow$.

4. 如果 ψ 是公式，而 x 是个体变量，那么 $(\forall x\, \psi)$ 和 $(\exists x\, \psi)$ 都是公式.

5. 只有由上面四条规则生成的表达式是公式.

在不引起混淆的情况下，我们可以省略公式中的一些括号. 连接符的优先级与命题逻辑中的规定（§1.1）相同. 量词的优先级比它们都高. 因此，$\forall x\, \psi_1 \vee \psi_2$ 代表 $((\forall x\, \psi_1) \vee \psi_2)$ 而不是 $(\forall x\, (\psi_1 \vee \psi_2))$. 在极少数情形，如果括号实在太多，我们也使用中括号来代替一部分小括号.

定义 如果某个变量 x 的出现不在量词 $\forall x$ 或 $\exists x$ 的范围中，那就叫做 **自由出现**.

定义 如果一个公式（项）中不含自由出现的变量，它就被称为 **闭公式**（**闭项**）或者 **句子**（ sentence ）.

例 假设 x 和 y 是变量，f 是二元函数符，P 是一元谓词符，那么下面是两个公式：

$$\forall x\, P(f(x,x)), \qquad \forall x\,(P(x) \rightarrow \exists y\, P(f(x,y)))$$

它们都是闭公式. 如果 R 是二元谓词符，那么下面是三个闭公式：

$$\forall x\, R(x,x)$$
$$\forall x \forall y\,(\, R(x,y) \rightarrow R(y,x)\,)$$
$$\forall x \forall y \forall z\,(\, R(x,y) \wedge R(y,z) \rightarrow R(x,z)\,)$$

它们通常叫做自反律、对称律、传递律.

在一阶逻辑中可以按照一定的推理规则由给定的公式推出新的公式. 限于篇幅，我们不详细叙述. 读者可参考数理逻辑的有关书籍 [50,121,190,191]. 这里我们仅举两个简单的例子.

例 用 $P(x)$ 表示 "x 是人"，$Q(x)$ 表示 "x 长生不老". 我们可以写出下面的公式：

$$\forall x\,(\, P(x) \rightarrow \neg Q(x)\,)$$

假设 a 是某个具体的人，如秦始皇. 由公式 $P(a)$ 和上面的公式可以推出 $\neg Q(a)$. 也就是说，秦始皇不是长生不老的.

在很多情况下，我们还要用到一个特殊的二元谓词符，就是 "等词"（ EQ ）. 对含有等词的公式，可以使用一些特殊的推导规则. 例如，由 $EQ(t_1,t_2)$ 可推出 $EQ(g(t_1),g(t_2))$. 我们也常把 $EQ(t_1,t_2)$ 写为 $t_1 = t_2$.

例 给定公式 $\forall x\, P(f(x,g(x)))$ 和 $\forall x \forall y\,(f(x,y) = f(y,x)$ 我们可以得到新公式 $\forall x\, P(f(g(x),x))$.

例 假定 f 是二元函数符，x 和 y 是变量，那么下面是两个闭公式：

$$\forall x \forall y (f(x,y) = f(y,x))$$
$$\forall x \forall y \forall z (f(x,f(y,z)) = f(f(x,y),z))$$

它们就是我们所熟知的交换律和结合律.

如果一个公式具有这样的形式：$\forall x_1 \cdots \forall x_k (t_1 = t_2)$，它可被简写成 **等式** $t_1 = t_2$. 这里，t_1 和 t_2 是两个项，而且它们所含的变量都在集合 $\{ x_1, \cdots, x_k \}$ 中.

例 群论公理. 抽象代数中的群结构可用如下几条等式来刻画：

$$f(x,e) = x$$
$$f(e,x) = x$$
$$f(x,g(x)) = e$$
$$f(g(x),x) = e$$
$$f(x,f(y,z)) = f(f(x,y),z)$$

其中 e, f, g 分别表示单位元、乘法和逆运算. 第二条和第四条等式可以由其他等式推导出来.

语义 一阶公式中只是一些抽象的符号，它们必须和我们所感兴趣的对象联系起来才有意义. 这是通过一种映射来完成的. 为方便起见，把我们感兴趣的所有对象组成的集合称为 **论域**. 通常要求它不是空集.

定义 个体变量的一个 **指派** 是变元集合 X 到论域 D 的一个映射 $s : X \to D$.

我们用 $s(x/a)$ 表示由 s 得到的一个新的指派 s'，它把变量 x 映射到对象 a，其他变量的值与 s 相同. 也就是说，对任何变量 v，如果 $v = x$，那么 $s'(v) = a$；否则 $s'(v) = s(v)$.

定义 一阶逻辑公式的一个 **解释** 包括一个非空论域 D 和满足如下条件的映射 I：

1. 对常量 a， $I(a)$ 是 D 中的元素.

2. 对 n 元函数符 f_i^n， $I(f_i^n)$ 是 D 上的 n 元函数，其定义域是 D^n，值域是 D.

3. 对 n 元谓词符 P_i^n， $I(P_i^n)$ 是 D^n 到真值集合 $\{\mathbf{T}, \mathbf{F}\}$ 的映射.

这里， D^n 表示笛卡儿积 $D \times \cdots \times D$（重复 n 次）.

一般我们要求将等词解释为“相等”关系.

根据解释 I 和指派 s，可以递归地确定一个项 t 在论域 D 中的值 $t^{I,s}$：

1. 当 t 是常量 a 时， $t^{I,s} = I(a)$.

2. 当 t 是变量 x 时， $t^{I,s} = s(x)$.

3. 当 t 形如 $f(t_1, \cdots, t_n)$ 时， $t^{I,s} = I(f)(t_1^{I,s}, \cdots, t_n^{I,s})$.

我们也可以递归地确定一阶公式 φ 在 I 和 s 之下的真假值 $\varphi^{I,s}$：

1. 如果 φ 是原子公式 $P(t_1, \cdots, t_n)$，那么 $\varphi^{I,s}$ 的值等于 $I(P)(t_1^{I,s}, \cdots, t_n^{I,s})$. 特别地，当 P 是等词时，如果 $t_1^{I,s} = t_2^{I,s}$，那么 $(t_1 = t_2)^{I,s}$ 为真，否则为假.

2. $(\neg\psi)^{I,s} = \neg(\psi^{I,s})$. 等号左边的 $\neg$ 是连接符，而右边的 $\neg$ 代表真值函数“非”.

3. $(\psi_1 * \psi_2)^{I,s} = \psi_1^{I,s} * \psi_2^{I,s}$. 等号左边的 $*$ 是二元连接符，而右边的 $*$ 则代表相应的真值函数.

4. 假设 φ 的形式为 $\forall x\,\psi$ 或者 $\exists x\,\psi$. 如果对论域 D 中的每个元素 e， $\psi^{I,s(x/e)}$ 均为真，则 $(\forall x\,\psi)^{I,s}$ 为真，否则为假. 类似地，如果 D 中有某个元素 e 使得 $\psi^{I,s(x/e)}$ 为真，则 $(\exists x\,\psi)^{I,s}$ 为真，否则为假.

定义 对一阶公式 φ，如果有论域 D 以及相应的解释 I 和指派 s，使得 $\varphi^{I,s}$ 为真，那么 φ 就是 **可满足的**.

下面我们假定所有的公式都是闭公式. 这时可以忽略掉对变量的指派 s.

定义 公式 φ 的一个 **模型** 由一个论域 D 以及相应的解释 I 组成，使得 φ^I 为真. 对于一组公式 S，其模型使得 S 中的每个公式都为真. 模型的大小是指集合 D 的大小.

一个公式（集）如果有模型，那么它（们）就是协调的或者相容的（ consistent ），否则就有矛盾.

例 我们看公式 φ_1：$\forall x \exists y\, R(y,x)$. 如果选择论域 D 为自然数集合 $\mathcal{N}$，而将 R 解释为“大于”关系，就得到 φ_1 的一个模型. 它是个无穷模型. 但如果再加上公式 φ_2：$\forall x\, R(x,x)$，那么 $\mathcal{N}$ 就不是 φ_1 和 φ_2 的模型了. 但是这两个公式并不矛盾. 如果把 R 解释为“大于或等于”，那么 $\mathcal{N}$ 是 φ_1 和 φ_2 的模型. 我们也可为这两个公式构造有穷模型 M_n，其论域是任何一个大小为 n 的有限集合（ n 是正整数），而 R 被解释为“等于”关系.

例 给定下面两个公式：

$$\forall x \neg R(x,x)$$

$$\forall x \forall y \,(\, R(x,y) \to R(y,x) \,)$$

如果我们将论域的元素看成顶点，$R(x,y)$ 表示顶点 x 和顶点 y 之间有边相连，那么每个无向图都是上述公式的模型. 换句话说，上面两个公式是无向图的公理. 公理中的全称量词往往省略.

例 对任何给定的正整数 n，下面的公式只有大小为 n 的模型.

$$\exists x_1 \cdots \exists x_n \,[$$

$$\forall y \,(y = x_1 \vee y = x_2 \vee \cdots \vee y = x_n) \wedge$$

$$
\begin{aligned}
&(x_1 \neq x_2) \wedge (x_1 \neq x_3) \cdots \wedge (x_1 \neq x_n) \wedge \\
&(x_2 \neq x_3) \wedge \cdots \wedge (x_2 \neq x_n) \wedge \\
&\cdots \\
&(x_{n-1} \neq x_n)
\end{aligned}
$$

]

定义 给定公式 φ 和论域 D. 如果对 D 上的所有解释 I, φ 都为真, 那么就说 φ 在 D 中是 **有效的** (valid). 如果 φ 在所有论域中都是有效的, 则称 φ 是 **有效的**.

公式 φ 是有效的, 当且仅当它的否定 $\neg\varphi$ 不可满足.

定义 给定一组公式 $\alpha_1, \alpha_2, \cdots, \alpha_n$ 和 φ. 如果对任何使得前 n 个公式成立的解释 I, φ 都成立, 那么 φ 就是公式 $\alpha_1, \alpha_2, \cdots, \alpha_n$ 的 **逻辑推论** (logical consequence).

定义 给定一个闭公式集合 S, 它所定义的 **理论**(theory) 是指 S 的所有逻辑推论所构成的集合.

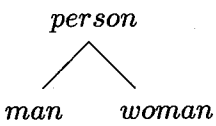
多种类逻辑 对很多应用来说, 采用多种类的(many-sorted, 或者说带类型的) 一阶逻辑公式更自然一些. 我们不打算详细介绍这种公式. 如有兴趣, 可参看有关文献 (如文献 [143] 第 29 章以及文献 [50] 第三章第七节). 简单地说, 就是有若干个类型. 每个项都是带类型的, 而每个函数符 (谓词符) 的参量也具有特定类型. 例如, 在谈到夫妻关系时, 可以有两个类型: man 和 $woman$. 假设有两个函数:

$$
\begin{aligned}
wife\_of &: man \to woman \\
husband\_of &: woman \to man
\end{aligned}
$$

那么我们可以有这样的公式:

$$
\forall x : man \ [\ husband\_of(wife\_of(x)) = x\ ]
$$

一般说来，带类型公式的语义牵涉到多个论域，而且它们都不是空集．在最简单的情况下，它们之间互不相交．但在有些场合，其中一个论域可能包含其他论域．例如，我们可以有一个新的类型 *person*．按照常识， *man* 和 *woman* 都是它的子类型．也就是说，在任何一个解释中， *man* 和 *woman* 所对应的论域都是 *person* 所对应论域的子集．在树状的类型结构中，每个类型的直接子类型之并集等于它自己，两个不同子类型之交集为空．如下所示：



多种类的公式可以转换成一般公式．我们只需要为每个种类 s 引入一个相应的谓词 P_s，将 $\forall x : s,\ \varphi$ 变为 $\forall x\,(P_s(x) \to \varphi)$，将 $\exists x : s,\ \varphi$ 变为 $\exists x\,(P_s(x) \wedge \varphi)$．当然，还需要一定的公理来描述类型之间的关系．例如，

$$\forall x\,[\ P_{person}(x) \to (P_{man}(x) \vee P_{woman}(x))\ ]$$
$$\forall x\,[\ P_{man}(x) \to P_{person}(x)\ ]$$
$$\forall x\,[\ P_{woman}(x) \to P_{person}(x)\ ]$$
$$\forall x\,[\ \neg P_{man}(x) \vee \neg P_{woman}(x)\ ]$$

§2.2 自动定理证明

长期以来，数学家们一直梦想着能有一种机械的方式来证明所有的数学定理．但是后来 Gödel 的研究结论给这一努力泼了冷水．尽管如此，在电子计算机诞生不久，人们就开始尝试用它来证明

一些定理. 通常人们将这类工作称为自动定理证明(Automated Theorem Proving , 简称 ATP)或者机器定理证明(Mechanical Theorem Proving).

A. Newell, J. C. Shaw 和 H. A. Simon 于 1956 年提出的"逻辑理论机"可以算是 ATP 方面最早的工作. 他们采用的是启发式方法, 模仿人类证明定理的推导过程. 他们的程序能证明《数学原理》一书中的几十条定理. Newell 和 Simon 以及 M. Minsky 、J. McCarthy 后来被认为是人工智能领域的创始人. H. Gelernter 于 1959 年实现了"几何定理证明机". 它采用反向推导的办法, 由要证的结论出发, 将每个目标分解成子目标.

与此同时, 人们也开始以经典逻辑为基础研究 ATP. 王浩(Hao Wang)先后实现了基于命题逻辑和谓词演算的证明程序, 并证出了《数学原理》一书中的几百条定理. 他后来获得了美国数学会颁发的首届 ATP 里程碑奖. 我们知道, 证明一个公式为永真公式, 这等价于证明其否定式为不可满足的. Gilmore 根据这一点, 并利用 Herbrand 定理, 将谓词演算中的定理证明问题转换为命题逻辑公式的可满足性问题. Davis 和 Putnam 后来提供了比较有效的办法, 来判定命题公式的可满足性. 这就是前一章中谈到的 DP 算法.

这一节我们简单介绍目前流行的基于一阶谓词演算的定理证明方法. 如果想更详细地了解有关技术, 可参看文献 [31, 117, 196, 115, 116]. Siekmann 和 Wrightson[176] 收录了 ATP 领域早期的一些重要文献, 包括王浩和 Gilmore 等人的文章.

§2.2.1 前束范式和子句形式

具有以下形式的公式称为 **前束范式**(prenex normal form) :

$$Q_1x_1\cdots Q_nx_n\ \varphi$$

这里每个 Q_i 是 $\forall$ 或者 $\exists$，而 φ 中没有量词. 如果所有的 Q_i 都是全称量词，我们就称公式为 **全称前束范式**.

对于一般形式的一阶公式，可以利用以下变换规则将量词左移，得到前束范式.

$$\neg\forall x\,\varphi(x) \;\Rightarrow\; \exists x\,\neg\varphi(x)$$
$$\neg\exists x\,\varphi(x) \;\Rightarrow\; \forall x\,\neg\varphi(x)$$
$$\varphi_1 * Qx\,\varphi_2(x) \;\Rightarrow\; Qx\,(\varphi_1 * \varphi_2(x))$$
$$\forall x\,\varphi_1(x) \wedge \forall x\,\varphi_2(x) \;\Rightarrow\; \forall x\,(\varphi_1(x) \wedge \varphi_2(x))$$
$$\exists x\,\varphi_1(x) \vee \exists x\,\varphi_2(x) \;\Rightarrow\; \exists x\,(\varphi_1(x) \vee \varphi_2(x))$$

在第三条变换规则中，Q 代表 $\forall$ 或者 $\exists$，而 $*$ 代表 $\vee$ 或者 $\wedge$，并且要求变量 x 不在公式 φ_1 中自由出现. 尽管我们上面只提到了 $\neg$、$\vee$ 和 $\wedge$，但其他连接符可以转换成这三种，因为它们构成了一个完备的连接符集合（参看 §1.1 ）.

当然，我们也可以利用命题逻辑中的推理规则. 比如说，由 $\neg(\psi_1 \wedge \psi_2)$ 可得到 $(\neg\psi_1 \vee \neg\psi_2)$. 此外，如果需要的话，还可对变量进行换名. 例如，将 $\forall x\,P(x,x)$ 变换为 $\forall y\,P(y,y)$.

得到前束范式后，我们可以进一步去掉其中的存在量词. 这一过程叫做 **Skolem化**，其做法如下：从左到右依次检查每个量词. 假设最左边的存在量词是 $\exists y$，而其左方的全称量词是 $\forall x_1 \cdots \forall x_n$，那我们就去掉 $\exists y$，同时引入新的 n 函数符 f，并以 $f(x_1, \cdots, x_n)$ 代替 y 的所有出现. 注意，0 元函数符被看成是常量. 因此当 $\exists y$ 处于公式的最左方，也即 $n = 0$ 时，我们用常量 a 去代替 y. 新引入的函数符（常量）通常被称为 Skolem **函数（常量）**.

例

$$\begin{aligned}
&\neg\exists x\,(P(x)\ \wedge\ \forall y\,Q(x,y))\ \Rightarrow\\
&\forall x\,\neg(P(x)\ \wedge\ \forall y\,Q(x,y))\ \Rightarrow\\
&\forall x\,(\neg P(x)\ \vee\ \exists y\,\neg Q(x,y))\ \Rightarrow\\
&\forall x\,\exists y\,(\neg P(x)\ \vee\ \neg Q(x,y))
\end{aligned}$$

该前束范式中有一个存在量词. 为此引入 Skolem 函数 f，得到全称前束范式 $\forall x\,(\neg P(x)\ \vee\ \neg Q(x,f(x)))$.

经过上述转换后得到的全称前束范式与原来的公式并不一定等价. 但是我们有这样的结论：一阶公式是可满足的，当且仅当它的全称前束范式可满足.

对于全称前束范式 $\forall x_1\cdots\forall x_n\ \varphi(x_1,\cdots,x_n)$，我们可以把 φ 化成更规范的形式，如合取范式，以便在计算机上实现有效的推理. 通常我们将公式前面的全称量词忽略不写. 也就是说，我们由每个一阶公式得到一组子句.

在第一章，我们已经看到子句形式的命题逻辑公式. 一阶子句和命题子句的主要区别在于，其原子公式一般不是简单的命题变量，而是以若干个项为参量的谓词. 目前，有些自动推理程序可以将含有量词的公式转换成一组子句. 下面我们看看 William McCune 实现的证明器 OTTER[136].

例 考虑公式

$$\forall x\exists y\exists z\,((\neg P(x,y)\vee Q(x,z))\to R(x,y,z)).$$

我们可以写出下面几行作为 OTTER 的输入：

```
set(auto).
formula_list(usable).
all x exists y exists z ((-P(x,y)
```

```
                    | Q(x,z)) -> R(x,y,z)) .
    end_of_list.
```

这里的第一行说明，OTTER 将按缺省的设置运行. 而第三行就是所给的公式，其中的符号 '-' 、 '|' 和 '->' 分别表示 "非"、"或"、"蕴含". 第二行表示，公式被放在 " usable " 列表中. (OTTER 将所有的公式分成几类，包括 usable 、 sos 等. 后面我们将对此作进一步的解释.)

给定上面的输入， OTTER 会将输入公式化为子句，并输出如下信息：

```
-------> usable clausifies to:

list(usable).
P(x,$f2(x))|R(x,$f2(x),$f1(x)).
-Q(x,$f1(x))|R(x,$f2(x),$f1(x)).
end_of_list.
```

由此可见，原来的带量词的一阶公式被自动地转换成两个子句，其中 `$f1` 和 `$f2` 是新引入的 Skolem 函数.

需要指出的是，采用经典的方法将公式转换为合取范式，可能会生成非常多的子句. 因此，有些学者抛开合取范式，而直接对非子句形式的公式进行推理. 另外，也有些人研究更好的转换办法 [154,14,149,148] ，以得到比较少的子句. 一种常见的做法是引入新的谓词符号来代替子公式，比如将 $\varphi_1 \vee \forall x\, \varphi_2$ 转换为

$$(\varphi_1 \vee \forall x\, P(x)) \ \wedge \ \forall x\, (P(x) \rightarrow \varphi_2)$$

这里， x 是 φ_2 中唯一的自由变元， P 是新谓词符. 如果由 φ_1 可得到 m_1 个子句，由 φ_2 可得到 m_2 个子句，那么采用上述办

法可将子句个数由 m_1m_2 个减为 m_1+m_2 个（我们假设 m_1 和 m_2 都大于 1 ）.

例 给定下面的公式作为输入：

```
(all x (-R(x,a) & -R(x,b))) |
(all u (Q1(a,u) <-> Q2(a,u))) |
(all w (Q1(w,b) <-> Q2(w,b))).
```

OTTER 会得到如下 8 个子句：

```
-R(x,a) | -Q1(a,y) | Q2(a,y) | -Q1(z,b) | Q2(z,b
-R(x,a) | -Q1(a,y) | Q2(a,y) | Q1(z,b) | -Q2(z,b
-R(x,a) | Q1(a,y) | -Q2(a,y) | -Q1(z,b) | Q2(z,b
-R(x,a) | Q1(a,y) | -Q2(a,y) | Q1(z,b) | -Q2(z,b
-R(x,b) | -Q1(a,y) | Q2(a,y) | -Q1(z,b) | Q2(z,b
-R(x,b) | -Q1(a,y) | Q2(a,y) | Q1(z,b) | -Q2(z,b
-R(x,b) | Q1(a,y) | -Q2(a,y) | -Q1(z,b) | Q2(z,b
-R(x,b) | Q1(a,y) | -Q2(a,y) | Q1(z,b) | -Q2(z,b
```

如果引入新谓词 P_1 和 P_2，将原公式变为下面 3 个公式的合取：

```
((all x (-R(x,a) & -R(x,b))) |
 (all x P1(x)) | (all x P2(x))).
(all u (P1(u) -> (Q1(a,u) <-> Q2(a,u)))).
(all w (P2(w) -> (Q1(w,b) <-> Q2(w,b)))).
```

那么就会生成 6 个子句：

```
-R(x,a) | P1(y) | P2(z).
-R(x,b) | P1(y) | P2(z).
```

```
-P1(x) | -Q1(a,x) |  Q2(a,x).
-P1(x) |  Q1(a,x) | -Q2(a,x).
-P2(x) | -Q1(x,b) |  Q2(x,b).
-P2(x) |  Q1(x,b) | -Q2(x,b).
```

这里的 a 和 b 是常量.

例 P. Andrews 提出一个挑战性问题 [152]，就是证明如下的公式 φ 为定理：

$$[\,(\,\exists x\forall y\,(p(x)\leftrightarrow p(y))\,)\;\leftrightarrow\;(\,\exists x\,q(x)\leftrightarrow\forall y\,q(y)\,)\,]\;\leftrightarrow$$
$$[\,(\,\exists x\forall y\,(q(x)\leftrightarrow q(y))\,)\;\leftrightarrow\;(\,\exists x\,p(x)\leftrightarrow\forall y\,p(y)\,)\,]$$

虽然这个公式看起来并不复杂，但如果用经典的方法将 $\neg\varphi$ 转换成子句形式，会得到 1600 个子句. Guha 和 Zhang[73] 通过引入新的谓词符号，得到 36 个子句.

§2.2.2 Herbrand 定理

要证明公式是永真的，只需要证明其否定是不可满足的. 而公式的可满足性定义中牵涉到论域. Herbrand 的研究结果表明，在判定某公式的可满足性时，我们只要考察一个特殊的域，就是所谓的 Herbrand 域.

定义 给定一个子句集合 S，其 **Herbrand 域** 是一个集合，它包含所有由 S 中的常量和函数符构成的项. 如果 S 中没有常量，就引入一个新的常量 c.

我们可以按照一定的过程，从简单到复杂，依次生成 Herbrand 域中所有的项. 读者不难写出这样的算法.

例 对于子句集：$\{\,P(g(x))\,\}$，其 Herbrand 域是 $\{\,c,\ g(c),\ g(g(c)),\ \cdots\,\}$. 对于子句集：$\{\,P(g(x)),\ Q(f(x,a))\,\}$，其 Herbrand 域是 $\{\,a,\ f(a,a),\ g(a),\ f(a,g(a)),\ f(a,f(a,a)),\ \cdots\,\}$.

定义 一个**替换**是有限集合 $\{t_1/v_1, \cdots, t_n/v_n\}$，其中 $v_i (1 \leq i \leq n)$ 是互不相同的变量符号，而每个 t_i 是不同于 v_i 的项. 若 $n = 0$，则是空替换.

假设 E 是一个表达式(项或子句)，而 $\theta = \{t_1/v_1, \cdots, t_n/v_n\}$ 是一个替换. 将 E 中变量 v_i 的每次出现同时代换成项 t_i，所得到的表达式记为 $E\theta$. $E\theta$ 称为 E 的一个**实例** (instance).

对两个替换 α 和 β 可进行**复合** (composition) 运算，得到的新替换记为 $\alpha \circ \beta$. 对任何表达式 E，$E(\alpha \circ \beta) = (E\alpha)\beta$. 其直观意思是，先作 α 替换，再作 β 替换. 具体地说，假设 $\alpha = \{s_1/x_1, \cdots, s_m/x_m\}$，$\beta = \{t_1/y_1, \cdots, t_n/y_n\}$，那么 $\alpha \circ \beta$ 是从集合 $\{s_1\beta/x_1, \cdots, s_m\beta/x_m, t_1/y_1, \cdots, t_n/y_n\}$ 中删掉以下元素而得到的：

1. t_i/y_i，如果 y_i 等于某个 x_j；

2. $s_i\beta/x_i$，如果 $s_i\beta = x_i$.

例 对于替换 $\alpha = \{g(y)/x, b/y\}$ 和 $\beta = \{c/x, a/y, d/z\}$，我们有 $\alpha \circ \beta = \{g(a)/x, b/y, d/z\}$. 如果项 $T = P(x, f(y, y))$，那么 $T(\alpha \circ \beta) = P(g(a), f(b, b))$.

不含变量的项和子句分别被称为**基项** (ground term) 和**基子句** (ground clause). 用 Herbrand 域中的基项去替换一个子句中的所有变量，就得到该子句的**基例** (ground instance).

定理 (Herbrand) 一组子句是不可满足的，当且仅当有这些子句的有限个基例，它们是不可满足的.

Herbrand 定理为我们提供了一种证明公式不可满足性的方法. 就是不断地生成所给公式的基例，然后去测试基例集是否可满足. 如果得到一个不可满足的集合，那么原来的公式也就不可满足. 在判定基例集的可满足性时，我们可以把基项 (如 $P(a)$, $Q(g(b))$) 看成是命题变元，然后采用命题逻辑公式的判定过程

（如 DP 算法）.

Gilmore 最早按照上述想法，在计算机上实现了证明过程. 不过，他使用了效率不太高的办法去判定命题逻辑公式的不可满足性. Davis 和 Putnam[40] 改进了命题逻辑的判定过程. 但是，直接根据 Herbrand 定理来证明公式的不可满足性，其效率还是不高. 主要原因在于，可能要生成非常多的基例，才能得到一个不可满足的集合. 下面我们简要介绍目前认为比较有效的证明方法以及流行的自动定理证明器.

§2.2.3 基于消解原理的否证法

前面我们看到，利用一定的变换规则并进行 Skolem 化，可以将任意形式的一阶公式转换成全称前束范式，并进一步得到一组子句. 而且这种转换过程不改变公式的可满足性. 因此，自动定理证明问题可转化成一组子句的不可满足性判定.

那么怎样才会发现一组子句不可满足呢？通常采用的办法是，由这些子句出发，不断地按照一些合理的规则推出新的子句. 如果经过一定的推导，我们得到空子句（即矛盾），那么问题就解决了. 当然对某些子句集合，这一推理过程可能不终止.

具体地说，给定子句集合 S，它的一个 **演绎推导序列** 是一组子句 $C_1, C_2, \cdots$ 其中每个 C_i 要么属于 S，要么是由序列中 C_i 之前的子句按照一定的规则推出的.

在演绎推理过程中，需要用到 **推导规则**（inference rule）. 这样的规则使我们能从已有的几个公式推出一个新公式. 一个简单的例子是命题逻辑中的规则：由公式 $p \to q$ 和 p，我们可以推出 q. 推导规则一般应具有两个重要性质. 其一是可靠性，就是说，新公式必须是原公式的逻辑推论. 另一个性质是完备性，就是说推导规则应有足够强的推理能力：对任何不可满足的（或者

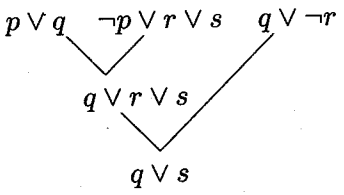
说矛盾的）子句集，都能由该规则推出空子句．所有可用的规则都必须是可靠的，但也有少数规则是不完备的．除了这两个性质以外，推导规则最好比较简单，易于在计算机上有效地实现．

目前影响最大、应用最广的推导规则是 J.A. Robinson[162] 提出的“消解”（resolution，也被译成“归结”）及其改进．我们先看命题逻辑中的消解方法．它可以说是 DP 算法中单子句规则的推广．根据单子句规则，由子句 p 和 $\neg p \vee q_1 \vee q_2 \cdots$ 能推出子句 $q_1 \vee q_2 \cdots$．这里要求有一个单子句 p．而消解方法舍弃了这一限制．

消解原理（Resolution Principle） 假设有两个子句 C_1 和 C_2，其中分别含文字 L_1 和 L_2，并且这两个文字互补．分别将 L_1 和 L_2 从 C_1 和 C_2 中删掉，再将两个子句中的其余文字合并，所得到的新子句 C 称为 C_1 和 C_2 的 **消解式**（resolvent）．C_1 和 C_2 是 C 的父子句．对任意两个子句，它们的消解式必是其逻辑推论．

例 子句 $p \vee q$ 和 $\neg p \vee r \vee s$ 的消解式为 $q \vee r \vee s$．子句 $p \vee q$ 和 $p \vee \neg r$ 没有消解式，因为它们中没有互补的文字．

可以用图来形象地表示推导过程，如下图所示．对于不可满足的子句集来说，如果推导规则足够强，最后应该能得出空子句．这时的推导图是一个树，称为演绎树．每个叶结点是事先给定的公式，而非叶结点是导出的公式．与搜索树不同的是，演绎树通常是叶子在上，树根（即空子句）在下．



在上面所讲的命题逻辑情形，容易判断两个子句中是否含有互补的文字. 对于两个一阶子句，我们想知道，它们是否有基例，其中含有互补的文字. 这就需要寻找满足一定条件的替换.

定义 给定一组表达式（项或子句）$\{E_1, \cdots, E_m\}$，如果有替换 θ 使得 $E_1\theta = \cdots = E_m\theta$，那么这些表达式被称为 **可合一的**（unifiable），而替换 θ 被称为是它们的 **合一**（unifier）. 如果对于每个其它的合一 α，都存在一个替换 β，使得 $\alpha = \theta \circ \beta$，那么 θ 就被称为 **最一般合一**（most general unifier，简称 MGU）.

给定一组表达式，可以按照一定的步骤去寻找其 MGU. 这样的过程称为 **合一算法**（unification algorithm）. 比如 Martelli 和 Montanari 提出的算法 [128].

定义 假设 C_1 和 C_2 是两个无公共变量的子句，其中分别含有文字 L_1 和 L_2. 如果 L_1 和 $\neg L_2$ 有最一般合一 σ，那么子句 $(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$ 称为 C_1 和 C_2 的 **二元消解式**.

这里，$C - L$ 表示从子句 C 中删去文字 L 所得到的子句. 如果 C_1 和 C_2 有公共变量，我们就对变量重新命名，以避免冲突.

例 假设有两个子句 C_1：$P(a) \vee \neg Q(x, y)$ 和 C_2：$\neg P(y) \vee Q(b, c)$. 由于它们有公共变量 y，我们将 C_2 中的变量 y 换名为 z. 如果让 C_1 和 C_2 的第一个文字合一，可得到二元消解式 $\neg Q(x, y) \vee Q(b, c)$. 如果让两个子句的第二个文字合一，可得到二元消解式 $P(a) \vee \neg P(z)$.

在上面所说的二元消解中，父子句只有两个. 其他规则也可由多个父子句推导出新子句，例如“超消解”（hyperresolution）.

对于 Robinson 的消解原理，后来人们还作了很多改进，以提高推理程序的效率. 其中比较著名的是 L. Wos 等人 [195] 提出

的“支撑集策略”（ Set of Support strategy ）. 我们介绍一下它的基本思想. 定理证明问题的初始形式一般是这样的：证明由假设 $A_1, \cdots, A_m$ 可推导出猜想 C. 在使用消解方法以及其他否证法时，先将 C 取反，然后试图从公式集合 $\{A_1, \cdots, A_m, \neg C\}$ 导出矛盾. 为此，我们反复地使用推导规则，由这些公式不断地推出新公式. 一般来说， $A_1, \cdots, A_m$ 是公理，它们之间是相容的. 如果对它们中的某几个公式施以推导规则，很可能白费精力，不会由此推出矛盾.

支撑集策略就是要对推导规则的使用作一些限制，以避免无用的推理. 具体地说，我们将子句集合 S 划分成两部分，其中子集 T 称为“支撑集”. 该策略禁止在 $S-T$ 中进行推理，也就是说，在每个推导步骤，必有一个父子句属于 T 或由 T 中的子句推出. 在使用上述策略时，支撑集的选择并不是任意的. 显然，当 T 是空集时，该策略不能导出任何新子句. 除了 T 非空外，还要求 $S-T$ 可满足. 通常我们可选猜想的否定作为支撑集.

支撑集策略的主要提出者 Larry Wos 博士长期在美国阿尔贡国家实验室（ Argonne National Laboratory ）从事逻辑推理自动化方面的研究. 他和同事们不断地研制出世界领先的自动定理证明器. 目前他们的主要工具是 William McCune 博士实现的 OTTER[136]. 它的早期版本要求用户将公式划分为几组：一组称为可用公式（ usable list ），通常包括公理；另一组就是支撑集（ set of support list ），包括猜想的否定；还有一组公式（ list of demodulators ）是用来对复杂的项进行化简. 此外，用户还需要指明采用什么样的推理规则. 这些信息使得人们能比较有效地控制推理过程，但也给一般用户带来了不便. 后来， OTTER 提供了更加自动化的方案，使得用户只需要将所有的公式写出来即可.

例 在前面我们给出了群论公理. 大家知道, 对群结构中的每个元素, 它的逆元素之逆就是它自己. 也就是说, 对任何 x, $g(g(x)) = x$. 如果用否证法来证明这个定理, 就需要将它取反, 得到子句 $g(g(a)) \neq a$. (这里的 a 是 Skolem 常量) 因此我们可写出如下的问题说明:

```
set(auto).
list(usable).
f(x,e) = x.
f(e,x) = x.
f(x,g(x)) = e.
f(g(x),x) = e.
f(x,f(y,z)) = f(f(x,y),z).
g(g(a)) != a.
end_of_list.
```

其中 != 表示不相等. OTTER 很快就会由上述公式推出子句 $F, 即矛盾. 因此定理得证.

例 有两个人: 陈先生和李小姐. 他们中一人是护士, 一人 是医生. 要求护士必须是女的. 问他们二人的职业分别是什么? 这个问题非常简单, 其答案是显然的.

如果用自动推理程序来解决它, 先要将有关的事实和规则用逻辑公式表示出来. 我们引入谓词 F 和 P, 其中 $F(x)$ 表示 x 是女性, $P(x,y)$ 表示 x 的职业是 y. 因此我们有以下子句:

$$P(Chen, nurse) \vee P(Chen, doctor).$$
$$P(Li, nurse) \vee P(Li, doctor).$$
$$\neg P(Chen, nurse) \vee \neg P(Chen, doctor).$$
$$\neg P(Li, nurse) \vee \neg P(Li, doctor).$$

$$P(Chen, nurse) \vee P(Li, nurse).$$
$$P(Chen, doctor) \vee P(Li, doctor).$$
$$\neg P(Chen, nurse) \vee \neg P(Li, nurse).$$
$$\neg P(Chen, doctor) \vee \neg P(Li, doctor).$$
$$F(Li).$$
$$\neg F(Chen).$$
$$\neg P(x, nurse) \vee F(x).$$

其中前四个子句表示：陈先生（李小姐）是护士或者医生，但不同时有两个职业．紧接着的四个子句表示：陈先生和李小姐中有且仅有一人是护士（医生）．

像 OTTER 这样的定理证明器是采用否证法的思想．也就是说，先必须给出猜想，将它取反，再推出矛盾．对于本问题，如果我们的猜想是“陈先生以医生为职业”，就将 $\neg P(Chen, doctor)$ 加入子句集，然后就能得出矛盾．这说明猜想是对的．但如果我们猜测陈先生是护士，OTTER 就会在经过几步推理之后停下来，而得不到任何结论．

除了基于消解原理的否证法之外，还有其他证明方法和系统，比如连接法（ connection method ）、自然推导法和一些归纳定理证明方法．目前在欧洲特别是德国， ATP 方面的研究仍然很活跃．近 10 年来，德国学者研制出一些高效率的定理证明器，如 SETHEO[111] 和 SPASS[192]．美国 Stanford 大学的 Carolyn Talcott 博士提供了一个网页，其中含有很多自动推理系统的信息．其网址是 `http://www-formal.stanford.edu/pub/clt/ARS/ars-db.html`

§2.2.4 判定过程

在前一章我们看到，有一个算法（如 DP ），它可以判定任何命题逻辑公式是否可满足，或者是否为永真公式. 这样的算法被称为一个 **判定过程**（ decision procedure ）. 给定任何一个命题公式作为 DP 过程的输入，如果公式可满足，那么过程终止，并给出肯定的结果（即“可满足”）；如果公式不可满足，那么过程也终止，并给出否定的结果. 判定过程也能帮助我们判断由一组命题公式是否能推导出另外的公式.

对一阶谓词逻辑来说，判定过程不存在. 因此我们说一阶逻辑是不可判定的. 有时候我们也说一阶逻辑是半可判定的. 因为对任何不可满足的公式，基于消解法的推理过程总能推出矛盾. 但如果公式可满足，该过程不一定终止.

对于具有某些特殊形式的一阶逻辑公式，可能会有判定过程 [1,47,16]. 例如，只含一元谓词的公式集合是可判定的，具有如下形式的前束范式集合也是可判定的：

$$\exists x_1 \cdots \exists x_m \forall y_1 \cdots \forall y_n \ \varphi$$

这里 φ 不含量词. 这类公式被称为 Bernas-Schönfinkel 类. 此外，下面这样的公式（叫做 Ackermann 类）也是可判定的.

$$\forall y \exists z_1 \cdots \exists z_n \ \varphi$$

其中 φ 不含量词，且只有二元谓词.

在经典的逻辑学研究中，只关心一类子公式是否可判定，而不注意判定过程的效率. 后来人们开始研究较为实用的算法. Fermüller 等人 [52] 针对各种不同的公式类，对消解方法进行一些改进，使之成为有效的判定过程.

在前面的讨论中，所有的函数符和谓词符（除等词外）都是未解释的（ un-interpreted ）. 也就是说，我们不假定它们是什么特殊的函数或谓词. 但在有些应用中，我们只关心某个特殊的论域及其上的运算和关系. 比如说，只考虑整数域和加法、减法、乘法以及常见的比较运算. 这样的公式在一般情况下是不可判定的. 但有个著名的可判定子集，即 Presburger 算术，其中没有乘法运算. 例如，下面的公式就属于这个子集：

$$\forall n \exists m \, (2m \geq n \ \wedge \ 2m < n + 2)$$

这里，$2m$ 只是 $(m+m)$ 的缩写. Shostak[175] 给出了 Presburger 公式的一个判定算法.

除了基于逻辑的比较通用的机器定理证明技术外，在某些专门的数学领域也发展了一些高效的证明方法. 例如，我国著名数学家吴文俊在几何定理证明方面作出了开创性的工作. 他提出的方法后来被人们称为“吴方法” [199]. 基于该方法的证明器可以快速地证明数千条几何定理. 1997 年吴文俊获得了自动推理界的 Herbrand 奖.

目前 ATP 方面的主要学术组织是国际自动推理学会（ Association for Automated Reasoning, 简称 AAR ）. 它每年举办国际自动演绎大会（ Conference on Automated Deduction, 简称 CADE ）. 1974 年第一届 CADE 在美国阿尔贡国家实验室召开，其论文收录于 IEEE Transactions on Computers, Vol. C-25, No. 8. 自第五届起，CADE 会议录由 Springer 出版社出版，作为 Lecture Notes in Computer Science（后转为 Lecture Notes in Artificial Intelligence ）的一部分.

自动推理领域一直注意用一些具体问题来测试定理证明器的效率 [152]. CADE 还曾经专门征集一些未解决的问题. 最近几

年，澳大利亚 James Cook 大学的 Geoff Sutcliffe 博士和德国慕尼黑技术大学的 Christian Suttner 博士收集了大量的这类问题，组成了一个问题库，称为 TPTP (Thousands of Problems for Theorem Provers)[185]，其中有的问题来自数论和代数，有的来自拓扑、分析、几何，还有的是电路设计、程序验证和可计算性理论方面的问题. Suttner 和 Sutcliffe 还组织自动定理证明比赛 [184]，用 TPTP 中的题目来测试自动推理程序的效率.

§2.3 模型的自动构造

前一节我们主要讲了机器定理证明的否证法，其基本思想是：要证明从 A 可推出 B，就试图由 $A \wedge \neg B$ 推出矛盾. 有些情况下，所给的猜想不是定理，或者说从 A 不能推出 B. 那么怎样来说明这一点呢？一种办法就是证明 $A \wedge \neg B$ 是可满足的. 我们可以为它找个模型，或者说是反例.

我们在 §2.2.4 提到，一阶谓词逻辑是不可判定的. 从理论上讲，没有一个算法，可以判定任何一阶逻辑公式是否可满足（或者为任何可满足的公式构造出模型）. 逻辑学家曾提出一些构造模型的办法，如 consistency family 和 ultraproduct （参看文献 [143] 第 18 章）. 但从自动化的角度看，这些方法并不实用.

§2.3.1 用搜索法构造有限模型

在很多情况下，可满足公式具有有限模型. 如果其大小是某个给定的常数 n，我们可以用穷举搜索的办法来得到模型的具体表示（类似于乘法表）. 大家也许会想，这样做的复杂度太大，不切实际. 从最坏情况看，的确如此. 但是如果采用适当的推理过程和策略，并恰当地安排推理步骤，时间代价往往达不到最坏情况. 事实上，DP 算法也可看成是一种穷举搜索.

我们举一个简单例子来说明这一点. 给定公式 $\forall x \forall y\,(P(x,y) \rightarrow P(y,x))$, 这里 P 是二元谓词. 我们要找一个大小为 2 的模型. 不妨设模型的论域为 $\{a, b\}$. 那么我们要确定 $P(a,a)$, $P(a,b)$, $P(b,a)$ 和 $P(b,b)$ 的真假值. 按照一般的想法, 这有 16 种可能性. 但实际上, 每当我们确定了 $P(a,b)$ 的值, 我们也就确定了 $P(b,a)$ 的值; 反之亦然. 也就是说, 4 个未知量之间并不是完全互相独立的.

现在我们看一般的情形. 不妨设所给定的公式是一组子句 S, 其中有这样一些函数符和谓词符:

$$\{f_1^{k_1}, f_2^{k_2}, \cdots, P_1^{l_1}, P_2^{l_2}, \cdots\}$$

这里的上标 k_i 和 l_j 表示这些符号可带的参量个数. 不妨用集合 D_n: $\{0, 1, \cdots, n-1\}$ 来表示大小为 n 的论域. 这些非负整数也可被看成是特殊的常量. 需要指出的是, 对于 D_n 中两个不同的元素 i, j, 我们将 $(i = i)$ 和 $(i \neq j)$ 化简为真 ($\mathbf{T}$), 而 $(i = j)$ 和 $(i \neq i)$ 化简为假 ($\mathbf{F}$).

我们可以把有穷模型的自动构造看成是约束满足问题. 其未知量集合 $VT(S,n)$ 是如下两个集合的并集:

$$\{f_i^{k_i}(a_1, \cdots, a_{k_i}) \mid 0 \leq a_1, \cdots, a_{k_i} < n\}$$
$$\{P_j^{k_j}(b_1, \cdots, b_{l_j}) \mid 0 \leq b_1, \cdots, b_{l_j} < n\}$$

(为了不引起混淆, 我们把 CSP 中的变量叫未知量, 以区别于公式中的变量.) 前一个集合中的未知量取值范围为 D_n, 后一个集合中未知量的值只能为真或假. 每个未知量都是一个特殊的项. 所要满足的约束条件 $CF(S,n)$ 是用如下方式得到的所有基子句: 将所给子句中的每个变量都替换成 D_n 中的某个整数.

举一个例子. 假设 $n = 2$, S 中只有一个子句: $\neg P(x,y) \vee P(y,x)$. 那么 $VT(S,n) = \{P(0,0), P(0,1), P(1,0), P(1,1)\}$,

而 $CF(S,n)$ 包含如下公式：

$$\neg P(0,0) \vee P(0,0)$$
$$\neg P(0,1) \vee P(1,0)$$
$$\neg P(1,0) \vee P(0,1)$$
$$\neg P(1,1) \vee P(1,1)$$

对于这个具体的例子，我们可将第一个和最后一个公式去掉. 此外，还可以做其他形式的预处理. 比如说，如果 S 中有公式 $f(x,y) = x$ ，那么就直接得出：

$$f(0,0) = 0, f(0,1) = 0, \cdots, f(1,0) = 1, \cdots$$

在做完预处理之后，可以用下面这样一个简单的搜索过程来构造有穷模型.

(1) 如果 $VT(S,n)$ 中每个未知量都有值，就说明 S 有一个大小为 n 的模型，过程结束.

(2) 否则，选一个还没有值的未知量 vt ，并从其取值范围中为它选一个值 e.

(3) 根据新的赋值 $vt = e$ ，对 $CF(S,n)$ 中的每个公式进行化简. 将所有的项 vt 替换成 e. 在化简时，也要用到目前已知的其他未知量的值.

(a) 如果某条公式被简化为 $\mathbf{F}$ ，那么需要回溯. 给 vt 选一个新的值. 如果所有的值都被考虑过了，就回到以前的某个未知量. 如果 vt 是第一个被选择的未知量，那么模型不存在，过程结束.

(b) 如果某条公式被简化为 $vt_1 = e_1$ ，那么转到 (3). 这里， vt_1 是一个还没有值的未知量， e_1 是其可取值范围中的一个值.

(c) 否则转到 (1).

例　考虑下面两条公理：

(A)　$f(x,x)=x$

(B)　$f(f(x,y),x)=y$

这里 f 是一个二元函数符，x 和 y 是受全称约束的变量.

假设我们要找一个大小为 3 的模型. 首先我们根据公理 (A) 可以确定 $f(0,0)=0$, $f(1,1)=1$, $f(2,2)=2$. 我们还要找出下面 6 个未知量的值：$f(0,1)$, $f(0,2)$, $f(1,0)$, $f(1,2)$, $f(2,0)$ 和 $f(2,1)$. 对公理 (B) 实例化，得到下面 9 个公式：

(B1)　$f(f(0,0),0)=0$

(B2)　$f(f(0,1),0)=1$

(B3)　$f(f(0,2),0)=2$

(B4)　$f(f(1,0),1)=0$

(B5)　$f(f(1,1),1)=1$

(B6)　$f(f(1,2),1)=2$

(B7)　$f(f(2,0),2)=0$

(B8)　$f(f(2,1),2)=1$

(B9)　$f(f(2,2),2)=2$

根据 $f(0,0)$, $f(1,1)$ 和 $f(2,2)$ 的值，可以知道 (B1), (B5) 和 (B9) 显然成立. 剩下的 6 个公式就是我们要继续考虑的条件.

首先考虑 $f(0,1)$ 的值. 如果它等于 0，那么由 (B2) 可推出 $f(0,0)=1$. 这与我们已知的结果矛盾. 如果 $f(0,1)=1$，则由 (B2) 可推出 $f(1,0)=1$，再由 (B4) 可推出 $f(1,1)=0$. 这也是个矛盾. 最后看 $f(0,1)=2$ 这种情形. 由 (B2) 可推出 $f(2,0)=1$，再由 (B7) 可推出 $f(1,2)=0$. 把这个赋值代入公式 (B6) 中，得不出矛盾.

因此我们再找一个没有值的未知量，假设是 $f(0,2)$. 按照与上面类似的过程，可以知道 $f(0,2)=1$, $f(1,0)=2$, $f(2,1)=0$.

最后我们得到如下的模型：

f	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

上面的搜索过程很简单，可以利用 CSP 领域的很多技术对它加以改进．比如说，在选择未知量作赋值时，应优先考虑那些可取值较少的未知量；对于某个选定的未知量，可以先将每个可能的值赋给它，看看有没有什么不好的后果，等等．

除此而外，在搜索过程中，还应该尽量避免互相同构的模型．两个大小为 n 的模型 M_1 和 M_2 被称为 **同构**，如果能定义一个从 D_n 到它自身的一一映射，使得 M_1 经映射后与 M_2 一样．对于前面的例子，如果 $n=4$，它有下面两个模型：

f	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

f	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

它们是互相同构的．只要将 2 和 3 互换，就可由一个模型得到另一个．

为了避免重复搜索，作者提出了所谓的“最小取数法”（least number heuristic，简称 LNH）[204]．它的基本思想是这样的．对大多数问题来说，它的每条子句的含义是，对所有的个体变元，某某条件应该被满足．因此，D_n 中所有元素在搜索开始之前是

平等的（或者说是对称的）. 这样，在给某些未知量选择值时，对于互相对称的元素，我们只选其中的一个代表，比如说最小的那个.

我们来看一种简单的情形. 假设所给的子句集合中只有一个二元函数符 f 和一些变元. 在一般情况下，没有特殊的等式可以让我们直接确定某些未知量的值. 因此假定所有未知量都是通过穷举实验而得到值的. 我们先考虑 $f(0,0)$. 按照通常的想法，它有 n 个可选择的值： $0,1,\cdots,n-1$. 但如果考虑到对称性，它的值实际上只有两种可能： 0 或者不等于 0 的某个值. 在后一种情况，我们不妨把 $f(0,0)$ 的值指定为 1. 再看 $f(0,1)$. 这时，0 和 1 已经是特殊的元素，它们之间有一定的关系；而其他元素（ $2,\cdots,n-1$ ）还是互相平等的. 因此， $f(0,1)$ 的取值有三种可能： 0, 1, 非 0 非 1 的某个值（设为 2 ）. 如此等等.

为了方便起见，我们用一个特殊的程序变量 mdn 来表示当前用到的 D_n 中的最大数. 其含义是， $0,\cdots,mdn$ 这些数代表了论域中的特定元素；而 $mdn+1,\cdots,n-1$ 是互相对称的. 变量 mdn 的初始值是 (-1). 在搜索过程中，每当我们引入论域中的一个新整数时，就对 mdn 的值进行更新. 可以证明，如果给定的子句集合不含有 D_n 中的数，那么使用 LNH 是安全的，它不会丢掉解 [204]. 有时候，用户可以事先指定某个常量为 0 ，另一个常量为 1 ，等等. 这时，需要将 mdn 的初始值调整为子句中出现的最大数.

例 我们再看前面两条公理 (A) 和 (B). 假设要找大小为 4 的模型. 首先根据 (A) 可知， $f(0,0)=0$, …, $f(3,3)=3$. 然后开始搜索，过程如图 2.1 所示，其中 $\otimes$ 表示矛盾. 经过若干次回溯，最终在选择 $f(2,1)=3$ 之后，得到一个模型（也就是上面两个模型中左边的那个）.

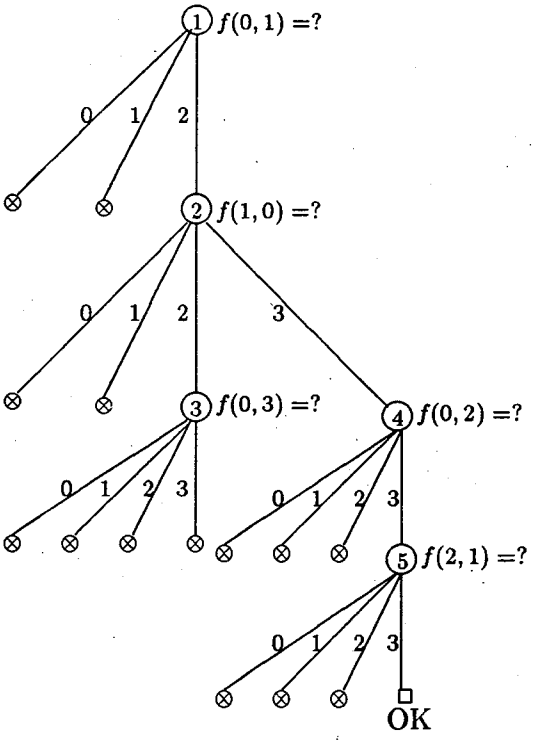


图2.1 有限模型搜索树

对于很多问题来说，采用最小取数法，可以非常有效地避免相互同构的模型. 但是还能作进一步的改进，参看文献 [153].

基于约束满足的思想，澳大利亚国立大学的 John Slaney 博士自 1991 年开始实现模型搜索程序 FINDER[181]. 作者也独立地设计并实现了构模软件 FALCON（开始叫 Mod/E ）[203,204]. FALCON 主要针对等式逻辑，应用范围较窄. 在这种逻辑中，所有的公式都是等式形式： $\alpha = \beta$ ，这里 α 和 β 都是项. 所有的变量都受全称量词约束.

后来，我们综合了 FALCON, SATO 等工具的优点，实现了 SEM[208]. 它可以处理带类型的一阶逻辑子句，并且采用了复杂的数据结构以提高效率. SEM 的输入文件一般由四部分组成：类型、函数和变量的说明以及子句. 在不致于引起混淆的情况下，可以忽略变量说明. FINDER, FALCON 和 SEM 都是用 C 语言实现的.

例 鸽笼问题. SEM 的输入文件如下：

```
%% Sorts
( pigeon [30] )
( hole [29] )

%% Functions
{ h  : pigeon -> hole }
{ in : pigeon hole -> BOOL }

%% Variables
< x, y : pigeon >
< z : hole >

%% Clauses
[ in(x,h(x)) ]
[ -in(x,z) | -in(y,z) | x=y ]
```

这里，- 表示“否定”，| 表示“或”，以 % 开头的行是注释. 函数符 h 的意思是，鸽子 x 在笼子 $h(x)$ 里. 本问题中有 30 只鸽子. 利用最小取数法，在 SUN SPARCstation 2 上只需几秒就可判断出它无解.

例　八皇后问题. SEM 的输入文件如下：

```
( dom [8] )
{ q : dom -> dom }
[ x = y | DIFF(q(x),q(y)) != DIFF(x,y) ]
[ x = y | q(x) != q(y) ]
```

这里的 DIFF 表示两个数的差. 函数 q 的意思是，在第 x 行、第 $q(x)$ 列有皇后.

例　我们看一个猜工作的智力游戏(jobs puzzle). 这是人工智能研究中一个比较有名的问题. 有四个人 (Roberta, Thelma, Steve, Pete) 和八种工作 (厨师、门卫、护士、办事员、警察、演员、拳击手、教师). 每个人有且仅有两个工作. 除此而外，还知道其他一些事实. 例如， Roberta 不是拳击手，厨师的丈夫是办事员，等等. 问每个人的工作是什么？

解决该问题的 SEM 输入文件见图 2.2, 其中省略了变量说明部分. 第一个子句的意思是，每个人最多有两个工作. 如果某个人 x 从事工作 z_1, z_2 和 z_3 ，那么其中必有两个工作是一样的. 由于本问题的输入说明很直观，我们不对它作更详细的解释.

Wos 等人 [196] 描述了如何用 OTTER 解决上述问题，其输入文件差不多有 3 页. 我们认为，用多种类的模型搜索器来解决这类问题更合适.

如果公式不是多种类的， SEM 的输入格式可以更简单一些. 下面是一个例子：

```
8.
f(x,e) = x.
f(e,x) = x.
```

```
( job : chef, guard, nurse, clerk, police,
        actor, boxer, teacher )
( person : Roberta, Thelma, Steve, Pete )
( sx2 : female, male )
{ job_holder : job -> person }
{ sex : person -> sx2 }
{ husband : person person -> BOOL }
{ educated : person -> BOOL }
[ job_holder(z1) != x | job_holder(z2) != x |
  job_holder(z3) != x | z1=z2 | z2=z3 | z3=z1 ]
[ sex(Roberta) = female ] [ sex(Steve) = male ]
[ sex(Thelma) = female ]  [ sex(Pete) = male ]
[ -husband(x,y) | sex(x) = female ]
[ -husband(x,y) | sex(y) = male ]
[ sex(job_holder(nurse)) = male ]
[ sex(job_holder(actor)) = male ]
[ educated(job_holder(nurse)) ]
[ educated(job_holder(police)) ]
[ educated(job_holder(teacher)) ]
[ -educated(Pete) ]
[ job_holder(boxer) != Roberta ]
[ job_holder(chef) != Roberta ]
[ job_holder(police) != Roberta ]
[ job_holder(chef) != job_holder(police) ]
[ husband(job_holder(chef), job_holder(clerk)) ]
```

图 2.2 SEM 输入文件 jobs.in

```
f(x,g(x)) = e.
f(g(x),x) = e.
f(x,f(y,z)) = f(f(x,y),z).
```

其中第一行指明要构造的模型的大小，其他行是群论公理. 假定把上面的问题说明放在文件 group.IN 中，那我们就可以用命令“sem group.IN”来找一个 8 阶群. 我们也可以用命令行参数来指明模型的大小，比如用命令“sem group.IN -z6”来找一个 6 阶群.

例 本问题选自文献 [120] 习题 3.1.6 . 公式 φ :

$$\exists x \forall y \left(\left(P(x,y) \wedge \neg P(y,x)\right) \rightarrow \left(P(x,x) \leftrightarrow P(y,y)\right) \right)$$

在大小不超过 3 的论域中是有效的，但在大小为 4 的论域中不是有效的.

大家知道，如果 φ 是有效的，则 $\neg\varphi$ 不可满足. 我们先利用 OTTER 将 $\neg\varphi$ 化成一组子句. 为此，给出下面的输入：

```
set(auto).
formula_list(usable).
-(exists x all y ( P(x,y) & -P(y,x) ->
                   (P(x,x) <-> P(y,y)) )).
end_of_list.
```

OTTER 将会输出如下内容：

```
-------> usable clausifies to:

list(usable).
```

```
P(x,$f1(x)).
-P($f1(x),x).
P(x,x)|P($f1(x),$f1(x)).
-P(x,x)| -P($f1(x),$f1(x)).
end_of_list.
```

再将上面的四个子句作为 SEM 的输入. 我们会发现, 没有大小为 1,2,3 的模型, 但是有如下的大小为 4 的模型:

P	0	1	2	3
0	0	1	0	0
1	0	1	1	0
2	0	0	0	1
3	1	0	0	1

	0	1	2	3
$\$f1$	1	2	3	0

大家可以验证一下, 上面这种解释确实使 $\neg\varphi$ 得到满足. 也就是说, φ 在论域 D_4 上不是有效的.

对于单类型的公式, 如果我们事先不知道模型的大小, 也可以这样来搜索: 先看有没有大小为 1 的模型; 如果没有, 再看大小为 2 的模型, 等等. 假设用户的根目录是 `/home/zj`, 可执行文件是 `sem`, 问题描述放在文件 prob.IN 中, 那么就可以用图 2.3 中的 Perl 程序进行这种搜索 (Perl 是 Larry Wall 创建的程序设计语言). 程序的意思是, 按照模型的大小为 1 、为 2 、为 3 …… 依次执行构模程序, 直到输出结果中出现 " Model " (也就是模型被找到) 为止.

例 非交换群. 假定文件 `prob.IN` 中有五条群论公理, 再加上子句 $f(a,b) \neq f(b,a)$. 上面的 Perl 程序将输出这样的信息:

```
Trying size 1 ...
```

```
$cmd = "/home/zj/sem";
$arg[0] = "/home/zj/prob.IN";

$n = 1;
while (1) {
  print "  Trying size $n ...\n";
  $arg[1] = "-z"."$n";
  @w = `$cmd @arg`;
  $nl = $#w;
  $end = 0;
  for($i = 0; $i < $nl; $i++) {
    if ($w[$i] =~ /Model/) {
      $end = 1;
      last;
    }
  }
  if ($end) {
    print @w;
    last;
  }
  $n ++;
}
```

图 2.3　由小到大搜索模型的 Perl 程序

```
Trying size 2 ...
Trying size 3 ...
Trying size 4 ...
Trying size 5 ...
Trying size 6 ...
```

然后给出一个大小为 6 的非交换群：

```
******** Model 1 ********

 f | 0 1 2 3 4 5
---+-------------
 0 | 0 1 2 3 4 5
 1 | 1 0 3 2 5 4
 2 | 2 4 0 5 1 3
 3 | 3 5 1 4 0 2
 4 | 4 2 5 0 3 1
 5 | 5 3 4 1 2 0

 e = 0

   | 0 1 2 3 4 5
---+-------------
 g | 0 1 2 4 3 5

 a = 1
```

```
b = 2

**************************
```

有些作者认为，用回溯方法只能找出很小的模型. Tammet ([52] , 第 156 页)举了一个例子来说明这一点. 如果我们有一个二元函数符，并且要找一个大小为 10 的模型，那么将会有 10^{100} 种可能性. 但实际情况是，对于很多问题，现有的工具能很快地找到有 10 个元素的模型，或者断定它不存在.

§2.3.2 转换成 SAT

一阶逻辑公式的有穷模型构造问题也可转换成 SAT. Kim 和 Zhang[102] 讨论了怎样利用 Davis-Putnam 算法去判定一阶逻辑公式在有限论域中的可满足性. 他们实现了一个软件工具，叫 ModGen. 它能接受类似于 OTTER 输入文件的问题说明，将其转换为命题逻辑公式，再用 SATO 判定可满足性. 美国阿尔贡国家实验室的 McCune 博士也用 C 语言实现了 DP 算法. 他的程序叫 ANL-DP. 后来他又编写了转换器 MACE (Models And Counter-Examples)[137].

从一阶公式到命题公式的转换过程基本上可分为两大步. 第一步将一阶子句转换成不含函数符和常量的关系型子句. 具体操作方法如下：

(1) 对每个 k 元函数符 f , 引入一个 $(k+1)$ 元谓词符号 F. 常量可看成是 0 元函数符.

(2) 对子句 C 中每个不是变量的项 t , 如果它不是等词的参量，我们就引入新的变量 x , 并将 $C[t]$ 替换为 $(x \neq t) \vee C[x]$.

(3) 对于子句 C 中每个等式形式的文字 l : $(\alpha = \beta)$, 如果 α 是带函数符的项，比如 $f(x,y)$, β 是变量，则将 l 替换成

$F(x,y,\beta)$. 如果 α 和 β 都不是变量, 则引入新的变量 x, 并将 C 替换成两个子句 C_1 和 C_2. 前者将文字 l 换为 $\alpha \neq x \vee \beta = x$, 后者将文字 l 换为 $\beta \neq x \vee \alpha = x$.

例如, 给定子句 $f(g(x),h(x)) = a$, 我们先引入变量 y 和 z, 得到子句 $y \neq g(x) \vee z \neq h(x) \vee f(y,z) = a$. 然后再引入变量 u, 并得到两个子句

$$y \neq g(x) \vee z \neq h(x) \vee u \neq f(y,z) \vee u = a$$
$$y \neq g(x) \vee z \neq h(x) \vee u \neq a \vee u = f(y,z)$$

它们可以变为关系型公式

$$\neg G(x,y) \vee \neg H(x,z) \vee \neg F(y,z,u) \vee A(u)$$
$$\neg G(x,y) \vee \neg H(x,z) \vee \neg A(u) \vee F(y,z,u)$$

在第二步, 我们由一组关系型子句以及模型的大小 n 得到命题逻辑公式. 假设模型中的元素为 $0,1,\cdots,n-1$. 对每个子句 C, 如果它含 m 个变量, 我们就让它们取各种可能的值, 对 C 进行实例化, 得到 n^m 个基子句. 比如当 $n=3$ 时, 由子句 $F(x,x,x)$ 可以生成如下三个基子句:

$$F(0,0,0) \qquad F(1,1,1) \qquad F(2,2,2)$$

上面每个基项可以对应到一个命题逻辑变元. 除了实例化所有的子句外, 我们还要对每个函数符加上一些条件, 以保证其定义的合理性. 例如对二元函数符 f, 我们需要下面这些命题公式:

$$F(x,y,0) \vee F(x,y,1) \vee \cdots \vee F(x,y,n-1)$$
$$\neg F(x,y,v_1) \vee \neg F(x,y,v_2) \quad (0 \leq v_1 < v_2 < n)$$

在得到一组命题逻辑子句后, 就可以用 DP 算法或者其他方法判定其可满足性. 如果可满足, 一般还要将解翻译成乘法表的形式.

MACE 的输入文件格式也和 OTTER 几乎一样. 举一个小例子. 如果要找非交换群, 我们可以写出下面的问题说明, 并把它放在文件 ncg.in 中.

```
list(usable).
f(e,x) = x.
f(g(x),x) = e.
f(f(x,y),z) = f(x,f(y,z)).
f(a,b) != f(b,a).
end_of_list.

list(passive).
assign(e, 0).
assign(a, 1).
assign(b, 2).
end_of_list.
```

这里 != 表示不相等. 我们很容易推断出, e, a 和 b 必须互不相同, 否则不可能有 f(a,b) != f(b,a). 采用和最小取数法类似的思想, 我们可以事先假定 e, a 和 b 分别取值 $0, 1, 2$.

现在可以敲入如下命令: mace -n6 -m1 -p < ncg.in 这里, -n6 和 -m1 表示要找大小为 6 的模型, 且最多只要一个; -p 要求把模型打印出来. 输出结果如下:

```
f :
     | 0 1 2 3 4 5
   --+------------
   0 | 0 1 2 3 4 5
```

```
  1 | 1 0 3 2 5 4
  2 | 2 4 0 5 1 3
  3 | 3 5 1 4 0 2
  4 | 4 2 5 0 3 1
  5 | 5 3 4 1 2 0

e: 0

g :

      0 1 2 3 4 5
  ---------------
      0 1 2 4 3 5

a: 1

b: 2
```

将一阶公式的模型构造问题转换成 SAT，其好处是，可以充分利用 SAT 算法研究中取得的成果和有关的高效工具. 但是这种方法的一个重要问题是，生成的命题逻辑公式可能太大，变元数量太多. 例如，MACE（版本 1.2.0）在搜索 6 阶非交换群时，生成了 9 万多个子句. 在使用单子句规则进行处理后，还剩下将近 6 万个子句. 如果将模型的大小改为 8，就会有几十万个子句，需要非常多的内存. 但是，利用 SEM 很容易构造大小为 18 的非交换群. 如何采用更好的方式得到命题逻辑公式，这是值得研究的一个重要问题.

§2.3.3 SATCHMO 和 MGTP 及其他方法

上面我们介绍的有限模型搜索技术通常假定论域的元素是非负整数，并将模型用类似于乘法表的形式表示出来. 但实际上，更传统、更一般的模型表示方法是，列出在模型中成立的基原子公式. 欧洲有不少学者研究如何构造公式的 Herbrand 模型.

Manthey 和 Bry[127] 于 1988 年提出了一种模型构造和定理证明方法，并用逻辑程序设计语言 Prolog 实现了软件 SATCHMO (SATisfiability CHecking by MOdel generation). 他们的方法适合于判定所谓的 range-restricted 子句是否可满足. 当用前件和后件的形式表示子句时，如果子句的每个变量都在前件中至少出现一次 (或者说后件中的每个变量都在前件中出现)，那么它就被称为是 range-restricted[127]. 对这样的子句作推理时，只需考虑单向的合一操作，即匹配 (matching).

后来人们对 SATCHMO 作了各种改进，如文献 [118]. 比较引人注目的是，20 世纪 80 年代初，日本提出了新一代计算机研究计划. 实施该计划的研究所 ICOT 用他们自己的并行逻辑程序设计语言 KL1 实现了自动推理系统 MGTP(Model Generation Theorem Provers) [78,77]. 它可在多达 256 个处理器的并行推理机 PIM 上运行. MGTP 实际上包括两个互相独立的程序：MGTP/G 和 MGTP/N (G 和 N 分别代表 Ground 和 Non-ground). 前者基本上采用 SATCHMO 的算法；后者则采用消解法，和 OTTER 类似.

下面我们简要介绍 SATCHMO 和 MGTP/G 构造模型的方法. 给定一个子句集 S，其 Herbrand 域的一个子集 I_S 叫做 S 的一个 **部分解释**. 通常我们在讨论某个固定的子句集 S 时，可以忽略下标 S. 假设 I 是个部分解释或模型，称为模型候选 (model candidate). 如果 S 中有一个子句未被 I 满足，则修改 I 使之得

到满足. 修改的方法是采用如下两条规则:

• 模型扩展规则 (model extension rule) 假定有子句 $A \to C$ 及替换 σ. $A\sigma$ 在 I 中满足, 而 $C\sigma$ 未被满足, 则扩展 I, 将 $C\sigma$ 加进去.

• 模型抛弃规则 (model rejection rule) 如果有负子句 A 及替换 σ, $A\sigma$ 在 I 中满足, 则去掉 I.

具体的算法可用如下的递归过程 $MG(S, I)$ 来描述:

1. 从 S 中找一个未被 I 满足的子句. 如果找不到, 则 S 可满足, I 是它的一个模型. 过程结束.

2. 否则假设子句 C : $A_1, \cdots, A_n \to B_1; \cdots; B_m$ 未被满足. 也就是说, 有一个基替换 σ, 使得对所有 $i : 1 \le i \le n$, $A_i\sigma \in I$; 而对所有 $j : 1 \le j \le m$, $B_j\sigma$ 不属于 I.

3. 如果 C 是一个负子句 ($m = 0$), 则抛弃 I.

4. 否则, 依次对每个 j $(1 \le j \le m)$, 将 I 扩展为 $I' = I \cup \{B_j\sigma\}$, 然后调用过程 $MG(S, I')$. 如果某一种扩展方式使得 S 被满足, 则过程结束.

5. 如果上面 m 种方式中, 每个模型候选都被抛弃, 则 S 不可满足.

第一次调用过程 $MG(S, I)$ 时, I 为空.

在使用 SATCHMO 和 MGTP/G 构造有穷模型时, 往往用谓词而不是函数, 并且用一个特殊的谓词 dom 来列出模型中的元素. 对每个 k 元函数符号 f, 可以引入一个 $(k+1)$ 元谓词符号 F, 使得 $f(e_1, e_2, \cdots, e_k) = val$ 当且仅当 $F(e_1, e_2, \cdots, e_k, val)$ 成立. 例如, 如果要找一个大小为 3 的模型满足等式 $f(x, x) = x$, 我们可以写出如下的子句:

$\mathrm{dom}(0).\ \mathrm{dom}(1).\ \mathrm{dom}(2).$

$\mathrm{dom}(x) \wedge \mathrm{dom}(y) \to F(x, y, 0) \vee F(x, y, 1) \vee F(x, y, 2).$

$\mathrm{dom}(x) \wedge \mathrm{dom}(y) \wedge F(x,y,0) \wedge F(x,y,1) \rightarrow \mathtt{FALSE}.$ ⋯

$F(x,x,x).$

上面第一行给出模型中的元素名： $0,1,2$. 而第二行是说，对任何 x 和 y， $f(x,y)$ 的值是上述三个元素之一. 第三行则表明，$f(x,y)$ 不能同时具有两个不同的值.

除了上面的工作外，欧洲还有其他一些学者也提出了自动构造模型的各种办法. Tammet[52] 研究了如何为一类公式构造有限模型. 这类公式具有以下两种形式之一.

1. $\exists x_1 \cdots \exists x_m \forall y \exists z_1 \cdots \exists z_n\ \varphi$, 其中 $m \geq 0, n \geq 0, \varphi$ 不含量词. 例如 $\forall x \exists y \exists z\ P(a,y,z)$. 它可转化成子句 $P(a, f(x), g(x))$.

2. 满足这样条件的前束范式：它经 Skolem 化之后，每个文字的参量集合中最多只有一个元素不是常量. 例如这两个公式：$P(a,b,c)$, $\forall x \forall y \exists z\ Q(a,z,b,z)$. 其中第二个公式可转化成子句 $Q(a, f(x,y), b, f(x,y))$.

Tammet 采用 Herbrand 域中的元素来构造模型，其优点是不需要回溯. 但是它得到的模型可能很大，而且只适合于具有上述形式的公式.

Caferra 等人 [26,13] 也提出了一种基于等式推理的模型构造方法，并在 OTTER 的基础上加以实现. 他们的方法可看成是消解法的扩展，对受约束的公式进行推理. 所构造的模型也是基于 Herbrand 域. 模型可以是有穷的，也可以是无穷的.

§2.4 模型构造与定理证明

可满足性判定与通常所说的自动定理证明是紧密相连的. 对于命题逻辑系统来说，其可满足性判定过程可以用来证明一个猜

想是定理. 给定一组假设 A 和结论 φ，如果 $(A \wedge \neg\varphi)$ 不可满足，那么我们就知道由 A 可以推出 φ.

至于一阶逻辑, 大家一提到自动定理证明, 就会想到消解法. 但是我们经常遇到要证明的猜想不是定理这种情况. 很多国际知名专家在自己的文章中提到这个问题. 目前绝大多数基于消解法的定理证明器不能找反例. 但是，如果我们能为前提条件和猜想的否定构造一个模型，则该猜想不是定理.

一阶谓词逻辑公式的可满足性是不可判定的. 但是我们可以用两个并发执行的进程来模拟一个不完备的 "判定过程". 其中一个进程不断地使用消解法和其他类似的推理规则试图从 $(A \wedge \neg\varphi)$ 推导出矛盾，而另一个进程试图用搜索法为它构造出有限模型. 如图 2.4 所示(注: 在图中我们采用类似于并发 Pascal 的文法, 而不是 C 语言的文法来描述过程). 从这个意义上讲, 模型构造和定理证明是互为补充的. 当然, 图 2.4 中的过程 SDP 并不是整个一阶谓词逻辑的判定过程，它不能判定所有一阶公式的可满足性. 对于只有无穷模型的公式，它不终止. 进程 P_1 可以由 OTTER 这样的定理证明器来完成，而进程 P_2 可以用图 2.3 中的 Perl 程序实现.

另外我们在 §2.1 提到，在自动定理证明研究的早期，人们就考虑过将命题逻辑公式可满足性判定方法 (如 DP 算法) 用于一阶定理证明. 基本做法是，先根据定理对子句集实例化，再使用 DP 来判断基子句集的可满足性. 最近 David Plaisted 领导的研究小组提出超链接 (hyperlinking) 的方法，对实例化过程进行改进 [110]. 他们的工具在某些定理证明问题上取得了不错的效果，但目前其总体性能还比不上基于消解法的证明器.

即使是对消解法来讲，模型构造也是有意义的. 人在证明定理时往往会使用一些例子. 基于这样的考虑，一些研究人员提出

```
program SDP(ψ)
cobegin
   P1::
     repeat
        使用消解等规则由 ψ 推出新公式;
     until (出现矛盾);
     kill(P2) ;
   ||
   P2::
     var n;
     n := 0;
     repeat
       n := n + 1 ;
        为 ψ 找一个大小为 n 的模型;
     until (找到模型);
     kill(P1) ;
coend
```

图 2.4 模拟判定过程

在证明过程中使用例子(即模型)[178,160]. 所谓的"语义消解法"(semantic resolution)就是用前提 A 的模型来引导搜索. 它可看成是支撑集策略的推广. 它只生成满足这种条件的消解式, 其父子句中至少有一个在模型中不成立. McCune[134] 对这类推理规则做了很多实验, 不过他用的模型都是手工构造的. John Slaney 将模型搜索器 FINDER 结合到定理证明器 OTTER 中, 得到一个新的工具 SCOTT[179]. SCOTT 采用 OTTER 的基本框架, 但它包含了动态的语义消解法. 也就是说, 它使用的模型不是事先由人给定的、固定不变的, 而是由 FINDER 动态生成的. 在某些问题上, SCOTT 的性能比 OTTER 好.

除了在定理证明器的执行过程中使用模型外, 也可事先利用

它们来生成一些有用的猜想. 我们设计并实现了猜想搜索工具 MCS (Model-based Conjecture Searching)[206]. 其基本思想是, 用穷举的办法或者是随机地生成一些公式, 再用给定的一组模型对它们测试. 在所有模型上都成立的那些就被作为猜想. 在得到猜想后, 可以用图 2.4 中的过程 SDP 去判断它们是否为定理. 其中一些比较简单的定理可被当成引理, 在证明其他更复杂的定理时帮助提高效率. 当然, 如果用穷举的办法生成公式, 需要对公式的长度加以限制. 我们采取的办法是, 先构造一些简单的项, 再由这些项构造出公式. 用户可以指定项中变量和函数符出现的最大次数.

例 我们再看群论. 假如我们想知道, 在什么样的条件下, 一个群是 Abel 群 (即交换律成立). 为此我们先给出 11 个不同的有限群. 在构造项时, 假设每个项中的变量只能是 x, y , 函数符出现的次数不超过 2. 用 MCS 可以得到这些项:

$$x, \quad f(x,x), \quad f(x,y), \quad f(x,f(x,x)), \cdots$$

并发现, 当群满足下面的任何一个等式时, 它就是 Abel 群.

$$\begin{aligned} f(x,x) &= e \\ g(x) &= x \\ f(x,f(x,x)) &= x \cdots \end{aligned}$$

不难证明, 它们确实是定理.

对于 MCS 来说, 一个主要问题是生成的猜想可能太多. 我们采取了这样的策略, 就是用重写规则简化所生成的项. 这一般会大大减少项和公式的个数. 至于如何从生成的猜想中选择比较重要的, 这需要用户利用领域知识来作决定.

第三章　模态逻辑

模态逻辑是一种非经典逻辑. 它是关于可能性和必然性的逻辑. 在这里, 我们讨论的命题可能为真(假), 或者必然为真(假). Hughes 和 Cresswell[90] 以及 Chellas[33] 详细地介绍了各种模态逻辑系统. Halpern 和 Moses[75] 综述了与人工智能密切相关的几种逻辑及其可满足性问题的复杂度.

关于模态逻辑的自动推理, 目前大致有以下几种方法. 一种是基于语义表的 [53,27,68]. 另一种是对经典逻辑推理中的消解原理进行推广, 使之能处理模态算子 [51]. 由于这种方法未见到有效的实现, 我们将不加以介绍. 此外, 还可以直接将模态公式翻译成经典逻辑公式, 再用传统的方法和工具来证明它的有效性或者可满足性 [145].

§3.1　命题模态逻辑

语法　与经典逻辑相比, 模态逻辑多了两个算子: 必然算子(记为 $\Box$)和可能算子(记为 $\Diamond$). 它们也被称为模态词. 命题模态逻辑公式的构造规则如下:

1. 命题变元是公式.

2. 如果 ψ 是公式, 则 $(\neg\psi), (\Box\psi), (\Diamond\psi)$ 都是公式.

3. 如果 ψ_1 和 ψ_2 是公式, 则 $(\psi_1 * \psi_2)$ 也是. 这里 $*$ 可以是任何一个二元连接符, 如 $\vee, \wedge, \rightarrow, \leftrightarrow$.

4. 只有由上面三条规则生成的表达式是公式.

在不引起混淆的情况下，我们可以省略一些括号. 连接符的优先级与经典逻辑中的规定（ §1.1 ）相同. 模态词 $\Box$ 和 $\Diamond$ 的优先级最高. 因此， $\Box p \to \Diamond p$ 表示 $((\Box p) \to (\Diamond p))$，而不是 $(\Box(p \to (\Diamond p)))$.

下面是一些合法的命题模态逻辑公式：

$$\Box p \to \Diamond p$$
$$\Box p \to p$$
$$p \to \Box\Diamond p$$
$$\Box p \to \Box\Box p$$
$$\Diamond p \to \Box\Diamond p$$

它们通常被记为 (D) 、 (T) 、 (B) 、 (4) 、 (5).

在不同的应用中，模态词的确切含义和性质可能有所区别，因而需要不同的模态逻辑系统. 这些系统的公理和推理规则互不相同. 详见文献 [33]. 常用的一类系统称为正规模态逻辑（ normal modal logics ）. 这类逻辑系统一般包括如下三条公理和两条规则：

(A1) $\Diamond p \leftrightarrow \neg\Box\neg p$.

(A2) $\Box(p \to q) \to (\Box p \to \Box q)$.

(A3) 经典命题逻辑中的永真公式.

(R1) 如果已证得 φ，就可以推出 φ.

(R2) 如果已证得 φ_1 和 $(\varphi_1 \to \varphi_2)$，就可以推出 φ_2.

当然，我们可根据需要，添加其他公式作为公理. 例如，如果把公式 (T) 和 (4) 也看成是公理，就得到模态逻辑系统 $S4$；如果再往公理集中加上公式 (B)，就得到模态逻辑系统 $S5$.

语义 正规模态逻辑的语义通常借助于可能世界 (possible worlds) 的概念. 每个世界是事物的一种状态. 同一个命题可以在世界 A 取真值, 在世界 B 取假值. 但各个世界之间并不是毫无关联的. 我们可以在世界集合上定义二元的 "可达" 关系 R: $R(x,y)$ 表示由世界 x 可以到达世界 y. 对模态逻辑系统 K 而言, 可达关系 R 没有什么特别的性质. 而对于其他的正规系统, 其可达关系满足一定的性质. 例如,

逻辑系统	R 的性质
T	自反
B	自反和对称
S4	自反和传递
S5	自反、对称和传递

一个 Kripke 结构是三元组 $\langle W, R, L\rangle$, 其中 W 是一些可能世界的集合, $R \subseteq W \times W$ 是 W 上的可达关系, $L : W \times P \to \{\mathbf{T}, \mathbf{F}\}$ 是对原子命题的解释. 这里 P 是所有命题变元的集合, $L(w,p)$ 表示命题变元 p 在世界 w 中是否成立.

给定一个 Kripke 结构 $M = \langle W, R, L\rangle$ 和一个命题模态逻辑公式 ψ, 我们用 $(M,w) \models \psi$ 来表示 ψ 在 M 的世界 w 中成立. 它可以按照公式 ψ 的结构递归地定义如下:

- $(M,w) \models p$, 当且仅当 $L(w,p) = \mathbf{T}$. 这里 p 是命题变元.
- $(M,w) \models \neg\varphi$, 当且仅当 $(M,w) \models \varphi$ 不成立.
- $(M,w) \models \varphi_1 \wedge \varphi_2$, 当且仅当 $(M,w) \models \varphi_1$ 并且 $(M,w) \models \varphi_2$.
- $(M,w) \models \Box\varphi$, 当且仅当对任何可从 w 到达的世界 v (也就是满足 $\langle w,v\rangle \in R$ 的 v), 都有 $(M,v) \models \varphi$.

含有其他连接符和模态算子的公式可转换成上述形式的公式.

如果对于 M 的任何世界 w，都有 $(M,w) \models \psi$，那么 ψ 在 M 中为 **有效**（ valid ）. 如果有一类结构 C，对于其中任何一个结构 M，公式 ψ 在 M 中都为有效，那么称 ψ 是 C 有效的（ C-valid ）. 比方说，假如 ψ 在所有可达关系满足自反律和传递律的 Kripke 结构中为有效，那么它就是 $S4$ 有效公式.

在计算机科学中的应用 模态逻辑为知识工程和并发程序的研究提供了一种形式化的途径. 模态词可用来表示“知道”、“相信”等概念. Burrows 等人[25] 提出了一个多种类的模态逻辑系统，用于认证协议的形式化描述，以便对其安全性进行推理. 后来人们称之为 BAN 逻辑. 它包含一些特殊形式的公式，可以表示“ P 相信命题 X ”、“ K 是 P 的公开密钥”，等等. BAN 逻辑有一些不足之处，后来人们进行了各种改进.

时序逻辑（ temporal logics，又称时态逻辑）也是一种常见的模态系统，可用来描述并发程序的某些性质[155,126]. 在所谓的线性时间时序逻辑中，除了 $\Diamond$ 和 $\Box$ 以外，还有模态词 $\bigcirc$（下一时刻）. 假如我们用命题变元 p 表示“北京市中关村地区今天下雨”，并且以天为时间单位，那么公式 $\Diamond p$ 的意思是“中关村地区总要下雨”，公式 $\bigcirc p$ 表示“明天下雨”，而公式 $(p \to \Diamond\neg p)$ 则代表“雨总是要停的”.

Pnueli[155] 提出用时序逻辑公式来表示并发程序及其性质（比如无死锁），从而将程序验证转换为逻辑推理. 给定并发程序 P，按照他的方法可以得到时序逻辑公式 $W(P)$. P 具有某种性质 R，当且仅当 $W(P) \to R$ 为定理. XYZ 系统[186] 就是受此思想影响而研制成的软件工程环境.

§3.2 命题模态逻辑公式的可满足性判定方法

§3.2.1 语义表方法

在前面（ §1.6.1 ）我们介绍了判定命题经典逻辑公式可满足性的语义表（ semantic tableau ）方法. 对命题模态逻辑公式也可以用类似的方法. 经典逻辑可看成是模态逻辑的特殊情形，它的有关推理是在同一个世界里进行的. 而对模态逻辑公式来说，还要考虑不同的世界. 每个公式都应该有一个“世界”标记. 我们用 $w:\varphi$ 表示某公式 φ 在某世界 w 成立. 在模态逻辑的语义表中，每个结点包含一些像这样的带标记的公式. 有时我们也用 $w:S$ 来表示集合 S 中的每个公式都在 w 中成立.

我们先看一个小例子. 比如说要判断公式 $\Diamond p \wedge \Box q$ 的可满足性. 不妨假设有一个 Kripke 结构 M，其中有一个世界 w_0 使得公式成立，即 $(M, w_0) \models \Diamond p \wedge \Box q$. 我们的目的就是要按照一定的规则构造出 M，或者证明它不存在. 在后面的讨论中，我们忽略 M.

对于本例来讲，我们可根据经典逻辑的推理规则得出公式 $w_0:\Diamond p$ 以及 $w_0:\Box q$. 怎样进一步对语义表作扩展呢？根据模态算子 $\Diamond$ 的定义，可引入某个从 w_0 可达的新世界 w_1， p 在 w_1 中为真. 除了 p 以外，还会有其他公式（比如 q ）也在新世界中成立. 事实上，如果有公式 $w_0:\Box\varphi$，那么根据模态算子 $\Box$ 的定义可往表中加入公式 $w_1:\varphi$，因为 w_1 是一个从 w_0 可达的世界. 类似地，如果有公式 $w_0:\neg\Diamond\varphi$，就可以把公式 $w_1:\neg\varphi$ 放到新结点中.

一般说来，除了经典逻辑中那些规则以外，还可以使用如下规则：

(1) 如果当前分支中有带标记公式 $w_i : \Box\varphi$，那么对任何可由 w_i 到达的世界 w_j，往新结点中添加公式 $w_j : \varphi$. 同样地，由 $w_i : \neg\Diamond\varphi$ 可以推出 $w_j : \neg\varphi$.

(2) 如果当前分支中有带标记公式 $w_i : \Diamond\varphi$，那么引入一个新的世界名 w_k，将 $\langle w_i, w_k\rangle$ 加到可达关系中，并把公式 φ 放入新结点中.

在使用第二条规则引入新世界 w_k 之后，可以检查 w_i（以及其他可到达 w_k 的世界）中成立的那些公式. 如果可能的话，就对其中一些公式使用第一条规则.

很明显，在构造语义表时，需要考虑各个世界之间的可达关系. 对同样的命题公式，如果模态逻辑系统不同，也会得到不同的语义表.

给定一个命题模态逻辑公式 φ，假若要判定它是否可满足，我们就为它构造一个语义表. 公式是可满足的，当且仅当表中有一个分支不是封闭的. 要想知道 φ 是不是有效公式，就为 $\neg\varphi$ 构造语义表. 如果所有分支都是封闭的，那么公式 φ 是有效的；否则我们就得到一个使 $\neg\varphi$ 满足的 Kripke 结构，它有时候被称为 φ 的反模型（counter-model）. 要想更进一步了解模态语义表，可参看文献 [53,27,68]. 下面举两个简单的例子.

例 我们来看公式 $(p \to \Box\Diamond p)$ 是否为 $S4$ 有效公式. 假设它不是，也就是说有一个世界 w_0 使公式不成立. 因此在 w_0 中，p 为真，而 $\Box\Diamond p$ 为假. 也就是说，$\Diamond\neg\Diamond p$ 在 w_0 中成立. 根据上面的规则，我们引入一个由 w_0 可以到达的新世界 w_1，使得 $\neg\Diamond p$ 在其中为真. $S4$ 的可达关系具有自反性和传递性. 所以由 w_1 可以到达它自己，而 $\neg p$ 在 w_1 也就为真. 除此而外，我们不能推出更多的信息. 因此，开始所给的公式不是 $S4$ 有效公式. 图 3.1 是它的一个反模型.

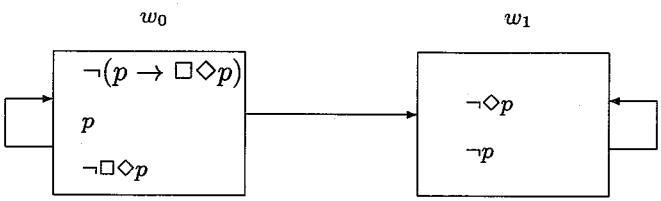


图 3.1　Kripke 结构

例　判断下面的公式 φ 是否为 K 有效.

$$\Diamond(p \wedge q) \to (\Diamond p \wedge \Diamond q)$$

我们假定有一个结构，$\neg\varphi$ 在其中的某个世界 w_0 取真值. 也就是说，$\Diamond(p \wedge q)$ 在 w_0 中为真，而 $(\Diamond p \wedge \Diamond q)$ 为假. 对后一个公式使用经典逻辑的推理规则，可以得到两种情形：w_0：$\neg\Diamond p$ 或者 w_0：$\neg\Diamond q$.

于是我们得到两个分支. 其中一个含公式 $\Diamond(p \wedge q)$ 和 $\neg\Diamond p$，另一个含公式 $\Diamond(p \wedge q)$ 和 $\neg\Diamond q$，它们所带的标记都是 w_0. 先看第一个分支. 根据其中第一个公式的形式，我们引入一个新的由 w_0 可以到达的世界 w_1，并进行如下推导：

$$\begin{aligned} w_0 &: \ \{\Diamond(p \wedge q),\ \neg\Diamond p\} \\ w_1 &: \ \{(p \wedge q),\ \neg p\} \\ w_1 &: \ \{p,\ q,\ \neg p\} \end{aligned}$$

公式 p 和 $\neg p$ 在 w_1 中都成立，这是矛盾的. 类似地，可由第二分支中的公式推出矛盾. 因此，公式 φ 为 K 有效公式. 整个推理过程如图 3.2 所示.

基于语义表方法，Catach[27] 用 Prolog 语言实现了一个命题模态逻辑定理证明器 TABLEAUX，并给出了一些实验结果.

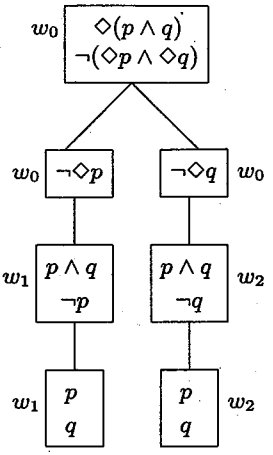


图 3.2　模态语义表

TABLEAUX 的基本数据结构是带标记的公式 $\psi = \langle w : \alpha : b \rangle$. 其中 w 代表某个世界，α 在该世界中成立；b 是一个布尔变量，表示 ψ 是否已经被检查过.

Giunchiglia 和 Sebastiani[65] 对语义表方法进行改进，建议采用 Davis-Putnam 算法来判定多个公式是否矛盾. 他们提出了一个判定命题模态逻辑公式 K 可满足性的过程（称为 KSAT ），并用 Lisp 语言加以实现. 他们对一些随机生成的命题模态逻辑公式做了实验，得到了比较好的结果.

§3.2.2　翻译法

我们知道，以经典逻辑为基础的自动定理证明已经研究了近 40 年. 这方面的理论已比较成熟，也开发了不少高效率的工具. 因此，利用已有的经典逻辑推理工具来进行模态逻辑推理，能充

分利用传统的自动定理证明器和有关的经验. 模态逻辑推理的翻译法, 就是把模态逻辑公式按照一定规则翻译成经典逻辑公式, 再用传统的自动工具进行推理. 对于多数命题模态逻辑系统来说, 翻译法是可行的. 一般说来, 可以通过两种途径来得到经典逻辑公式. 下面分别加以介绍.

语义途径 从直观上看, 必然算子 $\Box$ 和可能算子 $\Diamond$ 分别对应于谓词逻辑中的全称量词和存在量词. 因此, 一种直接了当的翻译方法是, 把模态逻辑公式 ψ 翻译成经典逻辑公式 $\sigma(\psi, w)$. 其直观意思是, ψ 在世界 w 成立. 此外, 我们用一个特殊的二元谓词符号 R 来表示可能世界之间的可达关系, 用一组一阶经典逻辑公理来表示 R 所应满足的性质.

具体的翻译规则可根据公式的语法结构递归地定义如下:

$$
\begin{aligned}
\sigma(p, w) &\equiv p'(w) \\
\sigma(\neg\varphi, w) &\equiv \neg\sigma(\varphi, w) \\
\sigma(\Box\varphi, w) &\equiv \forall v\,(R(w, v) \to \sigma(\varphi, v)) \\
\sigma(\Diamond\varphi, w) &\equiv \exists v\,(R(w, v) \wedge \sigma(\varphi, v)) \\
\sigma(\varphi_1 * \varphi_2, w) &\equiv \sigma(\varphi_1, w) * \sigma(\varphi_2, w)
\end{aligned}
$$

这里, p 是命题变元, 而 p' 是与之相对应的 (一元) 谓词符号; v 是不同于 w 的 "世界" 变元; $*$ 代表二元连接符, 如 $\vee, \wedge, \to$. 举一个简单的例子. 命题模态逻辑公式 α: $p \to \Box\Diamond p$ 将被翻译为

$$
p'(w) \to \forall u\,(R(w, u) \to \exists v\,(R(u, v) \wedge p'(v)))
$$

对于正规模态逻辑系统 S, 刻画 S 中可达关系的公理集合记作 $Ax(S)$. 例如, $Ax(T)$ 为 $\{\,\forall w\,R(w, w)\,\}$, 而 $Ax(S4)$ 包

含以下两条公理：

$$\forall w\, R(w,w)$$
$$\forall x \forall y \forall z\, (R(x,y) \wedge R(y,z) \rightarrow R(x,z))$$

定理 模态公式 ψ 在正规模态系统 S 中有效，当且仅当一阶逻辑公式 $Ax(S) \rightarrow \forall w\, \sigma(\psi, w)$ 在一阶谓词逻辑中是有效的.

Morgan[145] 简要地叙述了证明该定理的思想.

这样，我们就可以利用经典逻辑的自动推理工具来证明模态逻辑定理或者判断命题模态逻辑公式的可满足性. 对于前面的公式 α： $p \rightarrow \Box\Diamond p$ ，我们看它在 $S4$ 中是不是有效的. 为此，将 $\forall w\, \sigma(\alpha, w)$ 取反，得到一阶经典逻辑公式 α_1:

$$\exists w\ [\, p'(w) \wedge \neg \forall u\, (R(w,u) \rightarrow \exists v\, (R(u,v) \wedge p'(v)))\,]$$

公式集 $Ax(S4) \cup \{\alpha_1\}$ 有一个大小为 2 的模型：

$$R(a,a).\ \ R(b,b).\ \ R(a,b).\ \ \neg R(b,a).\ \ p'(a).\ \ \neg p'(b)$$

这里 a 和 b 是两个世界的名字. 从 a 可以到达 b ，但反之不行；p' 在 a 世界成立，在 b 世界不成立 (类似于图 3.1). 上述模型表明，α 在 $S4$ 中不是有效的. 如果我们要看 α 在其他正规模态逻辑系统中是不是有效，那我们只需要换一下有关的公理即可.

模态逻辑推理的翻译法比较容易实现，而且它能充分利用经典逻辑中已有的推理技术和工具. 它的适用范围也很广. 对于人工智能中常用到的模态系统 (如 S4, S5, KD45)，其可达关系的性质可用有限个经典逻辑公式表示，因而能用语义翻译法.

目前很少见到能处理 15 个基本模态系统的自动推理工具. TABLEAUX[27] 是个例外，但其作者给出的例子较简单. 多数工具只针对个别系统 (如 Demri[44] 的工具只适用于 S4). 当然，

翻译法也有些缺点，比如翻译的结果往往很复杂，所获得的证明可读性较差，等等. 很多研究人员专门针对某个模态逻辑系统（如 K ）设计出可满足性判别方法，其效率比翻译法往往要高.

某些研究人员对翻译法有一些误解. 一种误解是，采用翻译法会丧失命题模态逻辑的可判定性，因为一阶谓词逻辑是不可判定的. 我们认为这是不正确的，翻译法并不具有这样的“缺点”. 实际上，由命题模态逻辑公式按照上述规则翻译所得到的结果只是一阶经典逻辑公式的子集. 很多命题模态逻辑系统（包括 K, S4, S5 ）具有小模型性质（ small model property ）[33] ，也就是说，可满足的公式肯定具有不超过一定大小的有限模型.

给定正规模态逻辑系统 S 和命题模态逻辑公式 α ，我们用 ψ 表示一阶谓词逻辑公式 $\neg(\ Ax(S) \rightarrow \forall w\, \sigma(\alpha, w)\ )$. 我们可以用图 2.4 中的过程 SDP 来判定 α 是否为 S 有效公式. 在 α 为有效公式的情况下， SDP(ψ) 的第一个进程必然终止. 否则，由于 S 具有小模型性质，第二个进程将在有穷时间内结束. 当其中一个进程结束时，就发信号给另一个进程，使其终止. 我们可以假定进程调度是公平的. 也就是说，不会出现某个进程一直不能被执行的情况. 因此，过程 SDP 总会终止，采用翻译法并不会失去可判定性.

对翻译法的另一个误解是，它根本不实用. 例如， Ohlbach 和 Weidenbach[149] 曾以一个例子来说明上面所描述的直接翻译法不实用. 他们试图证明公式 $(\Diamond\Box p \leftrightarrow \Diamond\Box\Diamond\Box p)$ 在 $KD45$ 中为有效公式. 他们报告说，将该公式翻译后交给 OTTER[136] 去证明，花费了数个小时都未成功. 我们认为其主要原因是， OTTER 不善于处理形如 $(\alpha \leftrightarrow \beta)$ 的公式. 但这不能说明翻译法效率很低. 我们可以利用经典逻辑中的推理规则，将上述例子化成两个公式： $(\Diamond\Box p \rightarrow \Diamond\Box\Diamond\Box p)$ 和 $(\Diamond\Box\Diamond\Box p \rightarrow \Diamond\Box p)$ ，再分别用翻

译法去证明. 这样总共只需要不到 2 秒即可完成.

翻译法的效率在很大程度上取决于经典逻辑中推理工具的能力. 我们曾在 SUN SPARC station 2 上运行定理证明器 OTTER[136], 对有关文献中的数百个公式进行了实验. 其中绝大多数都能很容易地被处理. 对于不少比较复杂的有效公式, OTTER(3.0 版) 可以在几秒之内得到证明 [205].

有些公式不是有效公式. 我们可以利用 SAT 工具来构造一个 Kripke 结构, 使其否定式得到满足. 很多可满足的公式具有非常小的模型. 例如, 下面的公式在 $KT4$ 中不是有效的.

$$(\Box(\Box(p \to \Box p) \to p) \to (\Diamond\Box p \to p)) \\ \to (\Box(\Box(p \to \Box p) \to \Box p) \to (\Diamond\Box p \to \Box p)).$$

其否定式可在一个含 3 个世界的结构中得到满足.

上面所讲的方法是关系型的翻译法, 它引入新的二元关系符. 也有学者提出函数型的翻译(functional translation)方法 [147], 以得到更简洁的谓词逻辑公式.

语法途径 我们还可以从语法的角度来进行翻译. 前面我们看到, 很多命题逻辑系统可由一组公理和一组推导规则来定义. 它们指明了哪些公式是定理, 怎样由已知的定理推出新的定理. 这也能用一阶谓词逻辑的语言来描述. 为此我们引入一个一元谓词 P, 表示某个模态公式是可证的. 二元连接词和模态算子被看成是一阶谓词逻辑中的函数符号, 而命题模态逻辑公式对应于一阶谓词逻辑中的项.

具体地说, 我们采用下述规则由命题模态逻辑公式 ψ 得到一阶谓词逻辑公式 $\tau(\psi)$:

$$\tau(p) \equiv p$$
$$\tau(\neg\alpha) \equiv n(\tau(\alpha))$$

$$\tau(\alpha \to \beta) \equiv i(\tau(\alpha), \tau(\beta))$$
$$\tau(\Box\alpha) \equiv B(\tau(\alpha))$$
$$\tau(\Diamond\alpha) \equiv D(\tau(\alpha))$$

这里，p 代表一个变量；α 和 β 代表命题模态逻辑公式；n, i, B 和 D 是新引入的函数符，分别表示“否定”、“蕴含”、“必然”、“可能”. 在 $\tau(\psi)$ 中的变量都受全称量词约束.

例如，模态公式 $(x \to \Box x)$ 将被翻译成一阶谓词逻辑公式 $\forall x\, P(i(x, B(x)))$. 如果命题模态逻辑系统中有这样的公理：$x \to x$，那么我们可得到一阶经典逻辑公式 $\forall x\, P(i(x, x))$.

假设原来的命题逻辑系统中有这样的推理规则：

由 $\alpha_1, \cdots, \alpha_k$ 可推出 β

那么将它翻译成如下形式的一阶公式：

$$P(\tau(\alpha_1)) \wedge \cdots \wedge P(\tau(\alpha_k)) \to P(\tau(\beta))$$

这里我们省略了全称量词. 看一个简单例子，就是三段论：如果 p 和 $(p \to q)$ 都可证，则 q 也可证. 这对应于一阶公式

$$\forall p \forall q\, [\, P(p) \wedge P(i(p, q)) \to P(q)\,]$$

定理 模态公式 α 在命题模态逻辑系统 S 中有一个证明，当且仅当一阶逻辑公式 $Ax(S) \wedge Ir(S) \wedge \neg\tau(\alpha)$ 不可满足. 这里的 $Ax(S)$ 和 $Ir(S)$ 分别表示 S 的公理和推理规则经翻译后得到的一阶逻辑公式.

例 我们考虑这样的命题模态逻辑系统. 其公理包括：

(A1) $p \to (q \to p)$

(A2) $(\neg p \to \neg q) \to (q \to p)$

(A3) $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$

(A4) $\Diamond p \leftrightarrow \neg\Box\neg p$

(A5) $\Box p \to p$

(A6) $\Diamond p \to \Box\Diamond p$

(A7) $\Box(p \to q) \to (\Box p \to \Box q)$

(A8) $(p \to q) \to (\Box p \to \Box q)$

有两条推理规则：

(R1) 如果 α 和 $(\alpha \to \beta)$ 都可证，则 β 也可证.

(R2) 如果 α 可证，则 $\Box\alpha$ 也可证.

实际上，(A1)、(A2)、(A3) 和 (R1) 刻画了经典的命题逻辑.

假设我们要证明定理 ψ: $A \to \Box A$，那么我们可以试图在一阶谓词逻辑中证明下述公式集合的不可满足性：

$P(i(x, i(y, x)))$

$P(i(i(n(x), n(y)), i(y, x)))$

$P(i(i(x, i(y, z)), i(i(x, y), i(x, z))))$

$P(i(D(x), n(B(n(x))))).\ P(i(n(B(n(x))), D(x)))$

$P(i(B(x), x))$

$P(i(D(x), B(D(x))))$

$P(i(B(i(x, y)), i(B(x), B(y))))$

$P(i(i(x, y), i(B(x), B(y))))$

$P(i(x, y)) \wedge P(x) \to P(y)$

$P(x) \to P(B(x))$

$\neg P(i(a, B(a)))$

为了和习惯一致，我们用 x,y,z 分别代替 p,q,r. 上面最后一个公式是由 $\tau(\psi)$ 取反后得到的，其中 a 是常量.

在 SUN SPARCstation 5 上运行 OTTER 3.0.4，可在数分钟内找到证明.

翻译法非常简单，但它却有不少优点. 它能充分利用高效率的经典逻辑推理工具，而且适用于很多不同的模态系统，也能处理一阶模态公式. 基于语义的翻译法对很多具有小模型性质的命题模态逻辑系统来说，并没有丢掉可判定性. 而且实验结果表明，它能有效处理相当多的常见公式. 它的缺点在于，要求可达关系的性质能用有限条一阶逻辑公式刻画. 语法翻译法只适合于命题模态逻辑，但是它可以解决非正规模态逻辑系统中的定理证明问题. 在语义不清晰的情况下也能用.

本章我们着重介绍了命题模态逻辑公式的可满足性判定算法，而没有讨论关于一阶模态逻辑的自动推理方法. 后一方面的研究比较少. 而且对于常用的正规模态逻辑系统来说，一阶公式可以翻译成经典逻辑公式.

目前已有一些效率较高的模态推理软件，其中绝大多数是直接对模态逻辑公式进行推理. 例如，瑞士 Bern 大学的 Peter Balsiger 等人实现了 LWB 系统（ the Logics Workbench，网址 `http://www.lwb.unibe.ch/` ）. 它能证明相当复杂的命题逻辑公式. LWB 所包含的命题模态逻辑系统有：K, KT, S4 和 S5，经典命题逻辑以及一些非单调逻辑.

近几年，意大利罗马大学的 Fabio Massacci 博士开始组织非经典逻辑（特别是模态逻辑）自动推理软件的比赛 [129]. 但是目前参加者还比较少. 1999 年获得冠军的是美国贝尔实验室 Peter Patel-Schneider 用 ML 语言编写的软件 DLP（ Description Logic Prover ）. 它主要针对 $S4$ 和命题动态逻辑（ dynamic logic ）.

第四章　应用

从理论上讲，很多问题都可在多项式时间内转换成 SAT. 本章介绍命题逻辑公式可满足性判定方法和一阶逻辑公式有限模型构造技术在离散数学研究、电路辅助设计、软件工程和人工智能等领域的一些应用. 侧重点在于我们认为比较有意义、比较有趣或者最近被其他学者研究得比较多的问题. 当然，有时候用命题逻辑来描述一个实际问题，可能会得到非常多的公式. 本章所讲的应用并不一定都很成功. 有些问题只是被当作 benchmarks，用来测试和比较各种不同的 SAT 软件. 尽管如此，我们还是认为，应该结合一定的应用背景来研究可满足性判定方法. 随着计算机硬件性能的不断提高，相信有关技术会解决更多的问题.

§4.1　数学研究

自动定理证明技术的一个主要应用是数学问题求解. 可满足性判定算法也可用于这方面，尤其是离散数学. 本节将给出若干例子.

§4.1.1　组合学

一个 n 阶拉丁方是 $n \times n$ 方阵，其中每个元素都是不大于 n 的正整数，而且每一行（列）都没有重复的数. 换句话说，每一行是整数 $\{1, 2, \cdots, n\}$ 的一个排列，每一列也是. 例如，下面的

两个方阵分别是 3 阶和 4 阶拉丁方：

$$\begin{array}{ccc} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{array} \qquad \begin{array}{cccc} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{array}$$

我们也可以用整数集 $\{0, 1, \cdots, n-1\}$ 来代替 $\{1, 2, \cdots, n\}$.

如果我们用 $f(i,j)$ 来表示方阵中第 i 行、第 j 列的元素值，那么可以用下面的公式来描述拉丁方的性质：

$$\forall x \forall y \forall z \ [\, y \neq z \rightarrow f(x,y) \neq f(x,z) \,]$$
$$\forall x \forall y \forall z \ [\, x \neq y \rightarrow f(x,z) \neq f(y,z) \,]$$

或者说

$$\forall x \forall y \forall z \ [\, f(x,y) = f(x,z) \rightarrow y = z \,]$$
$$\forall x \forall y \forall z \ [\, f(x,z) = f(y,z) \rightarrow x = y \,]$$

满足上述公理的代数结构通常被称作拟群（ quasigroup ）.

给定一组逻辑公理 AS，人们常常关心它的谱(spectrum)，也就是正整数集合 $\{\, n \mid$ 存在 AS 的大小为 n 的模型 $\}$. 上述公理的谱就是所有正整数的集合，因为对于任何正整数 n，都有 n 阶拉丁方. 但是如果再加上一定的条件，就可能会有某个正整数 k，使得满足那些条件的 k 阶拟群不存在.

组合学家们所感兴趣的条件往往由一些恒等式表示 [10]. 一个简单的例子是幂等性（ idempotency ）：$f(x,x) = x$ （如果把 f 看成是乘法，该等式也可写为 $x^2 = x$ ）. 换句话说，幂等拟群对应着这样的拉丁方，其 i 行、i 列的元素值为 i. 其他一些重

要的恒等式包括：

$$
\begin{aligned}
&f(x,f(x,y))=f(y,x)\\
&f(f(f(y,x),y),y)=x\\
&f(f(x,y),f(y,x))=x\\
&f(f(y,x),f(x,y))=x\\
&f(f(x,y),y)=f(x,f(x,y))\\
&\cdots
\end{aligned}
$$

其中第二个等式后来被记为 QG5[58,182]. 对正整数 n，问题 QG5.n 指的是，满足等式 QG5 的 n 阶幂等拟群是否存在.

构造有限拟群的数学方法大致有两类. 一类是直接构造，一般是利用抽象代数中的有限域. 另一类是递归地构造，也就是通过已有的拟群得到新的拟群. 一个典型的办法是直积法，由一个 m 阶拟群和一个 n 阶拟群得到一个 mn 阶拟群. Bennett[8] 详细地研究了 QG5 的谱. 除了 56 个数外，对于其他所有的自然数 n，他确定了满足 QG5 的 n 阶幂等拟群的存在性. 这 56 个数中，最小的是 9，最大的是 174.

我们认为，计算机方法可以是数学方法的一个很好的补充. 对于比较小的 n，可以通过穷举搜索的方法来判断 n 阶幂等拟群是否存在. 基于这样的思想，作者于 1990 年编制了一个搜索程序，得出一个新的结论：满足 QG5 的 9 阶幂等拟群不存在. 后来，其他研究人员采用各种不同的办法和软件工具，得到了更多的结果. 这些软件包括：日本五代机研究所的一阶逻辑推理系统 MGTP，澳大利亚国立大学的有限模型搜索工具 FINDER，以及基于 DP 算法的程序 DDPP、SATO，等等. 关于这方面的详细情况，可参看文献 [180,182].

值得一提的是，解决搜索问题所需要的时间受到多种因素的

影响，特别是搜索算法和策略．除此而外，还有问题表示方式，搜索工具的数据结构、程序设计语言和硬件等因素．上面提到的软件中，DDPP 是用 Lisp 语言写的，MGTP 是用并行逻辑程序设计语言 KL1 实现的，其他都采用 C 语言．根据 Fujita 等人 [58] 1993 年报告的结果，在 256 个处理器的并行机 PIM-m 上运行 MGTP，经过 1 分多钟可以得到结论，满足 QG5 的 10 阶幂等拟群不存在．1994 年底我们用 C 语言实现的搜索工具 SEM[208]，在 SUN SPARC station 2 上运行不到 1 分钟也可得出同样的结论．

1994 年之前，人们在研究 QG5 问题时只用到这些约束条件：我们要找的是个拟群，它具有幂等性，并且满足恒等式 QG5．后来人们发现，如果再添加下面两个等式

$$f(y, f(f(x,y),y)) = x, \quad f(f(y, f(x,y)),y) = x,$$

搜索效率会得到提高，并且不改变解的存在性 [182]．作了这种改变之后，在 SPARCstation 2 上 SEM 只需 0.4 秒就可解决问题．我们设计的 MCS 系统 [206] 可以自动地寻找像上面两个等式那样的冗余约束条件．

§4.1.2 抽象代数和逻辑

例 三元布尔代数（ternary Boolean algebra ）．它有一个三元函数符 f 和一个一元函数符 g，满足如下公理：对任何 v, w, x, y, z,

(T1) $f(f(v,w,x),y,f(v,w,z)) = f(v,w,f(x,y,z))$

(T2) $f(y,x,x) = x$

(T3) $f(x,y,g(y)) = x$

(T4) $f(x,x,y) = x$

(T5) $f(g(y),y,x)=x$

在这些公理中，(T2) 独立于其他公理. 也就是说，(T2) 不能由 (T1) 和 (T3)、(T4)、(T5) 推出来. Winker[194] 用一个基于消解原理的定理证明器，半自动地证明了这一结果. 但是，一个有限模型搜索工具很容易完全自动地找出一个大小为 3 的模型，它不满足 (T2)，但满足其他四条公理 [203]. 如果使用 SEM，其输入文件如下：

```
3.
f(f(v,w,x),y,f(v,w,z)) = f(v,w,f(x,y,z)).
f(x,y,g(y)) = x.
f(x,x,y) = x.
f(g(y),y,x) = x.
f(b,a,a) != a.
```

其中第一行指明模型的大小，a 和 b 是引入的两个常量，“ != ”表示不相等. 在 SPARCstation 2 上 SEM 只要大约 0.1 秒就可找到解.

例 组合逻辑 (combinatory logic). 有人建议把组合逻辑作为数学和程序设计语言的基础. 它可被看成是一个等式系统，满足如下公理：对任何 x, y, z,

$$a(a(a(S,x),y),z)=a(a(x,z),a(y,z)),\qquad a(a(K,x),y)=x.$$

这里，a 是一个二元函数符，S 和 K 是两个常量，叫作组合子 (combinator). 除了 S 和 K 以外，还有其他一些组合子，如 B, I, L, M, N_1, Q 和 W. 它们分别满足以下等式：

$$a(a(a(B,x),y),z) \;=\; a(x,a(y,z))$$

$$
\begin{aligned}
a(I,x) &= x \\
a(a(L,x),y) &= a(x,a(y,y)) \\
a(M,x) &= a(x,x) \\
a(a(a(N_1,x),y),z) &= a(a(a(x,y),y),z) \\
a(a(a(Q,x),y),z) &= a(y,a(x,z)) \\
a(a(W,x),y) &= a(a(x,y),y)
\end{aligned}
$$

这里的 x 和 y 都是受全称量词约束的变量.

Wos 和 McCune 利用他们的定理证明器 OTTER 研究了组合子的一些性质，特别是强不动点性质 ψ：

$$\exists y \forall x \, [\, a(y,x) = a(x,a(y,x)) \,]$$

他们得到了不少结果，比如 B 和 W 具有强不动点性质. 也就是说，如果把上面的第一个等式和最后一个等式作为公理，可以推出 ψ 为定理. 但他们不知道 $\{B, N_1\}$ 是否也有强不动点性质 [197]. 我们利用 FALCON 程序 [204] 构造出一个模型，否定地回答了这个问题.

例 时间推理. Allen 和 Hayes[3,79] 为了刻画关于时间的推理，提出了一个区间逻辑系统. 它的基本谓词是 MEETS，记为 $\|$. 每个区间可表示为 $[a,b]$，其中 a 和 b 分别是区间的起点和终点，要求 $a < b$. 对于两个时间区间（ temporal interval ）i 和 j，$i \| j$ 表示 i 的终点与 j 的起点重合. 例如，区间 i 是 [1,3]，而区间 j 是 [3,8]，则 $i \| j$ 为真.

Allen 和 Hayes 的逻辑系统中有如下五条公理：

$$\text{(M1)} \quad \forall p,q,r,s \ (p \| q \wedge p \| s \wedge r \| q \ \rightarrow \ r \| s)$$

$$\text{(M2)} \quad \forall p,q,r,s \ (p \| q \wedge r \| s \ \rightarrow$$
$$p \| s \oplus \exists t\,(p \| t \| s) \oplus \exists t\,(r \| t \| q))$$

(M3) $\forall p \exists q, r\ (q \parallel p \parallel r)$

(M4) $\forall p, q, r, s\ (p \parallel q \parallel s \wedge p \parallel r \parallel s\ \rightarrow\ q = r)$

(M5) $\forall p, q\ (p \parallel q\ \rightarrow\ \exists r, s, t (r \parallel p \wedge q \parallel s \wedge r \parallel t \parallel s))$

这里，$p \parallel q \parallel r$ 代表 $(p \parallel q) \wedge (q \parallel r)$. Ladkin[108] 声称，公理 (M5) 可以由 (M1)、(M2) 和 (M3) 推出来. （Ladkin 所引用的第二条公理有笔误.）但是 Galton[62] 手工地构造了一个反例，说明 (M5) 不是前三条公理的推论. 我们的构模工具 SEM 也能在不到 1 秒的时间内自动地找到如下的反例：

$\parallel$	0	1	2
0	**F**	**T**	**F**
1	**F**	**F**	**T**
2	**T**	**F**	**F**

也就是说，有三个区间：i_0, i_1 和 i_2，满足 $i_0 \parallel i_1 \parallel i_2 \parallel i_0$. 不难验证，它们满足前三条公理，而公理 (M5) 不成立.

需要指出的是，目前 SEM 只接受合取范式形式的公式. 因此我们先必须将一般形式的公式转化成一组子句.

例 一般说来，群的公理集中有五个或者三个等式. 有没有可能只用一个等式呢？事实上，这种等式有很多，被称为单公理集（single axioms）. 它们都比较复杂，因而在证明时很容易出错. 数学家 B.H. Neumann 曾声称下面的等式足够作为群理论的公理：

$$((x \cdot ((y \cdot z) \cdot (y \cdot (u^{-1} \cdot z^{-1}))^{-1})) \cdot x)^{-1} = u$$

但是，McCune[135] 利用 FINDER[181] 发现，上述公理有一个大小为 2 的模型，其中没有单位元.

如果使用 SEM 的话，可以将以下两行放在文件 grp-ax.in 中（第一行指明模型大小，第二行是公理）：

```
2.
i(f(f(x,f(f(y,z),i(f(y,f(i(u),i(z))))),x)) = u .
```

这里 $i(x)$ 相当于 x^{-1}，而 $f(x,y)$ 相当于 $x \cdot y$. 然后执行命令 "sem grp-ax.in -m0"，就会得到如下 4 个模型：

	0	1
i	0	1

f	0	1
0	0	1
1	1	0

	0	1
i	0	1

f	0	1
0	1	0
1	0	1

	0	1
i	1	0

f	0	1
0	0	1
1	1	0

	0	1
i	1	0

f	0	1
0	1	0
1	0	1

大家知道，每个群结构中都有一个单位元 e 满足 $i(e) = e$. 但是在上面的最后两个模型中，元素 0 和 1 都不满足这个条件，所以这两个模型都不是群，而 Neumann 的等式也就不是群论公理.

前面所举的例子大多是计算机领域的研究人员独立（或以他们为主）完成的工作. 要用自动推理技术解决复杂的数学问题，

领域专家（即数学家）的积极参与是非常重要的. 事实上，组合数学家 Frank Bennett、朱烈和人工智能学者 Masayuki Fujita、John Slaney、Mark Stickel、张瀚涛等合作，在拟群（拉丁方）和组合设计的存在性方面得到了很多新的结果 [58,182,9]. 自动推理专家 William McCune 和数学家 R. Padmanabhan 的合作也取得了丰硕的成果 [138]. 当然，也有些数学家在自己的研究工作中，独立地使用自动推理程序. 例如，美国威斯康星大学的 K. Kunen 教授将 SEM 用于抽象代数研究 [107].

§4.2 硬件测试与验证

目前，集成电路和操作系统被称为信息领域的核心技术. 它们的可靠性非常重要. 即使有些微小的错误，也可能引起严重的后果. 例如，Intel 公司刚推出奔腾处理器不久，有人发现它在做浮点运算时可能会出错. 后来公司不得不收回很多芯片，因而蒙受了巨大的经济损失.

保障硬件可靠性的主要手段包括测试和形式验证以及容错. 本节我们主要介绍组合电路的等价性验证与测试生成技术.

等价性验证

对于组合电路，很容易用布尔逻辑来描述它在门一级的行为. “与”、“或”、“非”门直接对应于命题逻辑中的连接符. 例如，图 4.1 中的电路可用下面的公式来表示：

$$t = \neg(x_1 \wedge x_2), \quad z = x_3 \wedge t$$

或者用这种写法：

$$t = \overline{x_1 x_2}, \quad z = x_3 t$$

因此，电路的输入信号和输出信号之间的关系是：$z = (\overline{x_1} + \overline{x_2})x_3$.

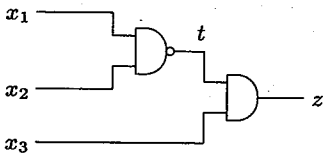


图 4.1　组合电路

另一方面，组合电路的功能也可用逻辑表达式来描述．我们可以将电路看成是一个黑盒子，直接将输出信号表示为输入信号的函数．例如，一位全加器的输入线有 x_i, y_i 和 c_{in}，输出线有 z_i 和 c_{out}．这些变量的取值都是 0 或 1，其中 c_{in} 是低一位的进位，c_{out} 是本位的进位．输入和输出之间应该满足关系

$$x_i + y_i + c_{in} = z_i + 2c_{out}.$$

（这里，我们把布尔值当成整数．加法和乘法也是在整数域上进行的．加号不表示“或”运算．）　根据上面的等式，我们可以得到如下真值表：

x_i	y_i	c_{in}	z_i	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

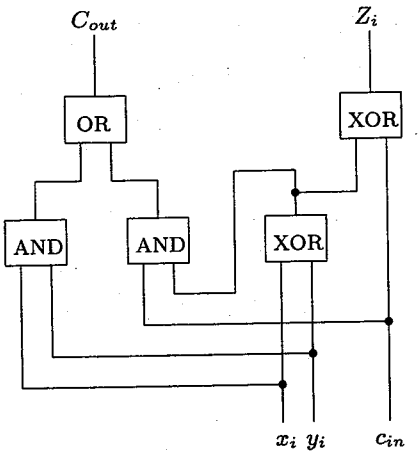


图 4.2　全加器

因此我们有如下的布尔逻辑表达式：

$$z_i = \overline{x_i y_i} c_{in} + \overline{x_i} y_i \overline{c_{in}} + x_i \overline{y_i c_{in}} + x_i y_i c_{in}$$
$$c_{out} = x_i y_i + x_i c_{in} + y_i c_{in}$$

图 4.2 是一位全加器的一种实现. 其输入输出关系可以用如下两个布尔函数来表示：

$$Z_i = c_{in} \oplus (x_i \oplus y_i)$$
$$C_{out} = x_i y_i + c_{in}(x_i \oplus y_i)$$

我们不难证明， Z_i 与 z_i 、 C_{out} 与 c_{out} 分别等价.

现在，基于 OBDD 的 CAD 软件已经广泛地用于数字电路的设计中.

测试生成

在很多情况下，输入线太多，我们不能够完全验证电路的正确性，而只能选取输入值的某些组合对电路进行测试. 也就是说，在电路的输入端送入一定的二进制码组合，然后观测电路的输出值，看看是否有错. 为了提高测试的有效性，必须针对一些具体的故障类型，选择合适的输入输出模式.

电路中常见的一种故障是固定“ 0 ”故障（ stuck-at-0, 简称 s-a-0 ）或者固定“ 1 ”故障（ stuck-at-1, 简称 s-a-1 ）. 也就是说，某个逻辑门的输入或输出信号总是 0 或者 1. 我们希望能通过观察输入和输出值检测出这类故障. 为方便起见，我们假定所给定的电路有 n 个输入信号（ $x_1, \cdots, x_n$ ）， 1 个输出信号（ z ）.

例 我们有一个“或非”门，它有三条输入线，一条输出线：

$$z = \overline{x_1 + x_2 + x_3}$$

在正常情况下，只有在三个输入信号都为 0 时，输出才为 1. 但如果其中一个输入（如 x_2 ）有 s-a-0 故障，那么当输入模式为 $(0,1,0)$ 时，输出也为 1. 因此下面的输入输出模式就是对 x_2 固定为“ 0 ”故障的一个测试：

$$x_1 = 0,\ x_2 = 1,\ x_3 = 0,\ z = 1.$$

所谓的测试码模式自动生成(Automatic Test Pattern Generation , 简称 ATPG ）就是根据电路的结构，自动地构造出测试某种故障的输入输出模式. 这里我们只考虑单个 s-a-0 故障或 s-a-1 故障，也就是说，假定电路中不会有两个或多个地方有这类故障. 在这样的前提下，如果电路中有 k 条连接线，我们只要考虑 $2k$ 个可能的故障.

如果电路中某处 w 出现了固定“0”故障，那么什么样的输入输出模式可以测试出这种故障呢？首先我们将输入、输出和 w 的关系用布尔表达式表示出来．不失一般性，假设

$$z = F(w, x_1, \cdots, x_n), \quad w = G(x_1, \cdots, x_n).$$

我们需要的输入信号必须使 w 的值为 1，而且 w 取值为 0 与取值为 1 时所得到的输出值应该不同．具体地说，就是要找出满足下述条件的输入：

$$G(x_1, \cdots, x_n) = 1$$
$$F(1, x_1, \cdots, x_n) \oplus F(0, x_1, \cdots, x_n) = 1$$

（假如要测的是固定“1”故障，就将第一个方程改为 $G = 0$．）上述方程组的解构成了 w 固定“0”故障的测试集．如果无解，那么该故障是不可测的．

例 对于前面提到的三输入“或非”门，假定要测 x_2 固定“0”故障，就要解下面的方程组：

$$x_2 = 1, \quad \overline{x_1 + 0 + x_3} \oplus \overline{x_1 + 1 + x_3} = 1$$

我们很容易得到解：$x_1 = 0,\ x_2 = 1,\ x_3 = 0.$

对于图 4.1 中的电路，函数 F 和 G 分别是

$$z = x_3 t, \quad t = \overline{x_1 x_2}.$$

如果我们要测试 t 固定“1”故障，输入数据必须满足以下条件：

$$\overline{x_1 x_2} = 0, \quad x_3 \oplus 0 = 1.$$

因此所要的测试模式为

$$x_1 = 1,\ x_2 = 1,\ x_3 = 1,\ z = 1$$

如果我们观测到这样的输入输出对，就找到了一个 s-a-1 故障.

从本质上看， ATPG 就是上面的布尔方程组求解问题. 但是在历史上，学者们先后提出了一些不同的解决办法：

• 代数方法 [169]. 利用布尔逻辑中的等值公式，对方程组的左边进行化简，从而得到解. 这种方法很难自动化.

• 结构化方法 [66]. 它不直接化简上述布尔方程，而是用一个数据结构来表示电路. 它对输入信号值的各种可能组合进行枚举，采用分支定界法得到符合条件的输入数据. 这实际上是一个回溯搜索算法.

• 转换成 SAT. Larrabee[109] 由布尔方程组得到一组子句，再用 SAT 算法求解. Stephan 等人 [183] 对其加以改进，减少了所生成子句的个数.

OBDD 和 SAT 算法也可以用于时序电路的验证与测试. 例如， Chen 和 Gupta[34] 提出一种方案，将路径延迟故障的测试生成问题转换为合取范式的可满足性问题. 另一方面，在对时序电路进行形式验证时，一种很有效的办法是用 OBDD 来表示状态转移图 [23]. McFarland[139] 详细介绍了时序电路验证的各种方法和工具，包括归纳定理证明、模型检测(model checking)，等等.

最近， Biere 等人 [11] 用命题经典逻辑公式的可满足性判定程序帮助寻找硬件设计中的错误. 他们的方法主要用来检查并发系统的安全性，即判断系统是否在可到达的任何状态都满足某个条件 φ. 该方法的基本思想和图 2.4 中的过程 SDP 类似. 一方面试图用自动定理证明技术证明 φ 的不变性，也就是说， φ 在每个状态都成立. 另一方面，试图用可满足性判定算法构造一个长度有限的反例. 给定自然数 n ，一个长度为 n 的反例是指状态转移序列： $s_0, s_1, \cdots, s_n$, 使得 φ 在某个状态 s_i $(0 \leq i \leq n)$ 不成

立. 这里的 s_0 是系统的初始状态.

本节所讲的方法主要针对门级电路. 对于寄存器级电路的验证, 一般要用到一阶逻辑.

§4.3 软件开发中的形式化方法

在大多数软件的开发过程中, 需求说明一般是用自然语言来书写的. 但是很多计算机科学家建议采用形式化或半形式化的方法, 也就是说用数学和逻辑来精确地描述软件系统的功能和其他特性. 其好处是, 避免需求说明中的二义性, 并尽可能早地发现错误. 至于使用什么样的描述手段比较好, 这在各个应用领域不尽相同.

形式化方法的基础是形式规范 (或者叫形式化规格说明, 即 formal specification). 由于它比较抽象和复杂, 一般人在写这种说明时, 很容易犯各种各样的错误. 对通常的 C 语言程序, 我们可以选一组输入数据, 执行程序, 看看它有没有错. 那么怎样发现形式化规格说明中可能出现的错误呢? 我们把错误分成两类: 一种情况是, 规格说明与实际需求不相符; 另一种情况是, 规格说明本身有毛病, 比如它的各部分之间不一致. 如果我们用一组逻辑公式作为需求说明的话, 可满足性判定和模型构造技术可以用于上述错误的检测.

§4.3.1 逻辑和关系型规范的验证

在群论单公理集的例子 (见 132 页) 中我们看到, 可以通过公式的模型来研究公式本身的性质. 如果给定公式的某些模型不符合人们想要刻画的实际情况, 那么必须对公式进行修改. 我们也可用类似的技术来发现规格说明的不足之处.

例 我们讨论人类中的配偶关系. 用 $gender(x)$ 代表 x 的

性别，$spouse(x,y)$ 表示 x 和 y 互为配偶．首先我们写出规则

$$\text{(R0)}\quad \forall x\forall y\,[\,spouse(x,y)\wedge spouse(x,z)\;\rightarrow\; y=z\,]$$
$$\text{(R1)}\quad \forall x\forall y\,[\,spouse(x,y)\;\rightarrow\; spouse(y,x)\,]$$

我们为 (R0) 和 (R1) 生成一些小模型，以便对它们进行检测．假设论域为 { John, Mary }，并且

$$gender(\text{John})=\text{male},\quad gender(\text{Mary})=\text{female}.$$

我们得到 5 个模型，其中两个如下：

spouse	John	Mary
John	F	F
Mary	F	F

spouse	John	Mary
John	F	F
Mary	F	T

显然第二个模型不符合现实．因此我们再加一条规则

$$\text{(R2)}\quad \forall x\;\neg spouse(x,x)$$

这时只有两个大小为 2 的模型：

spouse	John	Mary
John	F	F
Mary	F	F

spouse	John	Mary
John	F	T
Mary	T	F

它们都是合理的．

我们再增加一个人，将论域扩充为 { John, Mary, Bill }．得到如下几个模型：

spouse	John	Mary	Bill
John	F	F	F
Mary	F	F	F
Bill	F	F	F

spouse	John	Mary	Bill
John	F	F	F
Mary	F	F	T
Bill	F	T	F

spouse	John	Mary	Bill
John	F	T	F
Mary	T	F	F
Bill	F	F	F

spouse	John	Mary	Bill
John	F	F	T
Mary	F	F	F
Bill	T	F	F

在目前世界上绝大多数地方，同性婚姻是不合法的，因而最后一个模型不合适．为了避免它，我们再增加一条规则

$$\text{(R3)} \quad \forall x \forall y \, [\, spouse(x) = y \;\rightarrow\; gender(x) \neq gender(y) \,]$$

这时就只有三个模型，也就是上面四个中的前三个．

上述规则比较简单，正好是子句形式．在其他情况下，需要先将一般形式的公式化为子句．这会引入 Skolem 函数符，从而增加不必要的模型．我们对 SEM 稍加修改，让它将 Skolem 函数符和其他函数符区别对待，就可避免这种情况 [207].

在比较通用的形式规范语言 (formal specification language) 中，VDM 和 Z 是目前世界上影响比较大的两个．Z 语言是基于集合论和一阶逻辑的．在一个 Z 规范中，一般有若干个实体集合以及与之相关的一些函数和操作．这些集合和函数的定义构成了系统的状态．描述一个操作的方法是，刻画这个操作前后系统状态之间的关系．

我们看一个例子．一个简单的符号表 st 可表示为从符号集合 SYM 到值集合 VAL 的部分函数，其定义域和值域分别记为 $\mathrm{dom}(st)$ 和 $\mathrm{rng}(st)$．例如，$SYM = \{$ Zhang, Li, Wang $\}$，$VAL = \{\, 1, 2, \cdots, 200 \,\}$，而 st 表示人的年龄．在系统的初始状态中，st 的定义为：$\{$ Zhang $\mapsto$ 30, Li $\mapsto$ 35 $\}$．那么，$\mathrm{dom}(st) = \{$Zhang, Li $\}$，$\mathrm{rng}(st) = \{$ 30, 35 $\}$.

对符号表可以定义一些操作，比如“更新”，就是将某个符号 $s?$ 所对应的值改为 $v?$．操作的输入变量名习惯上以问号为结

尾. 更新后的符号表记为 st'. 它和旧表之间的关系为: $st' = st \oplus \{s? \mapsto t?\}$. 这里的 $\oplus$ 不表示 "异或", 而是 Z 语言中的特殊符号. 它作用于两个函数, 得到一个新函数. 其定义如下:

$$
\begin{aligned}
&\textbf{if } x \in \mathrm{dom}(g)\\
&\textbf{then } (f \oplus g)(x) = g(x)\\
&\textbf{else if } x \in \mathrm{dom}(f)\\
&\textbf{then } (f \oplus g)(x) = f(x)\\
&\textbf{else } (f \oplus g)(x) = \mathrm{undefined}.
\end{aligned}
$$

还可以用类似的公式定义其他操作, 如 "查表".

像 Z 这样的语言表示能力很强, 因而可以用于各种软件的开发和验证. 但是它的通用性和复杂性也带来了一些问题. 迄今为止, 没有很好的支持工具能对规范进行自动分析 (如判断它是否有矛盾). 有关问题往往是不可判定的.

Daniel Jackson 等人 [38,94] 定义了形式规范语言 NP. 这实际上相当于 Z 的一个子集. 该语言中的表达式 (也称为项) 是带类型的. 它可以是集合或其中的元素, 也可以是二元关系. 对表达式可定义一些特定的运算, 包括集合运算以及前面提到的 dom 和 rng. 例如, 下面两个都是合法的表达式:

$$S_1 \cup S_2, \qquad S_1 \cap \mathrm{dom}(r).$$

这里, S_1 和 S_2 是集合, 而 r 是二元关系. NP 语言所允许的公式是原子公式通过与、或、非等连接符组合而成. 每个原子公式是两个表达式之间的比较, 比如 $S_1 \subseteq S_2$, $x \in \mathrm{dom}(r)$. NP 语言中没有量词, 也没有普通的函数, 因而表达能力有限.

Jackson 等人还实现了一个形式规范检查工具, 叫 Nitpick. 它也是在有限集合中进行搜索, 并且借鉴了逻辑公式可满足性判

定的一些方法. 例如, 在穷举搜索时, 采用避免同构模型的策略来减小搜索空间 [94]. 此外, 他们还尝试了用 OBDD 和局部搜索过程 WalkSAT 来分析 NP 规范说明 [38,93].

§4.3.2 状态转换式规范的一致性检查

对很多软件系统而言, 可以比较自然地用一种基于状态转换的机制来描述其功能. 系统有若干个状态 (或者叫模式). 当某个事件发生或者某些条件得到满足时, 系统从一个状态转到另一个状态.

对于这样的需求说明, 一般要求它具有下面的性质.

• 完备性: 对每种可能发生的情况, 都定义系统的行为. 假设系统下一步要进入的状态完全受风向的影响. 那我们必须说清楚, 在刮东风时就转到某某状态, 在刮西北风时转到某某状态 …… 在没有风时转到某个状态 (或者保留在原状态), 不能有遗漏.

• 一致性: 不能有两个或更多的条件同时得到满足. 比如说, 如果满足甲条件, 汽车往左转; 如果满足乙条件, 汽车往右转. 一般来讲, 这两个条件不能同时成立. 虽然有时候允许一定程度的不确定性, 但应尽量避免这种情况.

在系统的行为比较简单时, 可以手工地判别需求说明是否满足上述要求. 但如果系统的状态数较多, 条件较复杂, 自动分析工具就很有必要.

假定系统在状态 S 可以转向 n 个后继状态: 当条件 c_i 成立时转向状态 S_i $(1 \le i \le n)$. 那么上面的要求可以分别表示为

$(c_1 \vee c_2 \vee \cdots \vee c_n)$ 为永真公式;

对任何 $i \neq j$, $(c_i \wedge c_j)$ 不可满足.

反过来说, 如果 $\neg(c_1 \vee c_2 \vee \cdots \vee c_n)$ 可满足, 或者对某两个不相等的 i 和 j, $c_i \wedge c_j$ 可满足, 需求说明就违背了一致性和完备性

的要求.

美国一些学者将状态转换机用于飞机控制部件的功能描述，并用可满足性判定工具来检查其中的错误. Heitmeyer 等人[82] 采用 SCR 需求说明方法，并实现了一个基于语义表的检查工具. Heimdahl 和 Leveson[81] 用的是一个基于 OBDD 的分析工具. 在他们的描述语言 RSML 中，为了使用户更容易理解，将每个条件 c_i 表示为一个与 / 或表（ AND/OR table ）. 这实际上是一个析取范式形式的命题逻辑公式. 下面是一个简单的与 / 或表.

		OR	
A	pc_1	**T**	**F**
N	pc_2	.	**T**
D	pc_3	**F**	.

这里的 pc_i（ $i = 1, 2, 3$ ）是基本条件. 表中右边的两列是析取关系，而每列是若干个文字的合取. 在第 j 列中，如果其中第 i 行是 **T**，则 pc_i 以正文字出现；如果是 **F**，则 pc_i 以负文字出现；否则 pc_i 不出现. 上表的右部第一列代表公式 $pc_1 \wedge \neg pc_3$，第二列代表公式 $\neg pc_1 \wedge pc_2$. 整个表所对应的条件是

$$(pc_1 \wedge \neg pc_3) \vee (\neg pc_1 \wedge pc_2)$$

在满足该条件时，系统进行一定的状态转换.

由于使用布尔变量（如 pc_i ）对实际情况进行抽象，因此往往会出现假报错（ spurious error reports ）[81]. 也就是说，检查出的错误实际上并不可能发生. 比如说，我们使用可满足性判定程序得到结论：当 b_1 为真、b_2 为假、b_3 为真时，违反了一致性或完备性的要求. 但是，如果基本条件 b_1 和 b_3 分别代表

(WaterPres> 1000) 和 (10 <WaterPres< 1000)，那么它们不可能同时为真. 这里， WaterPres 表示水压. 为了避免假报错，需要使用比命题逻辑更复杂的描述手段以及相应的推理工具.

§4.3.3 程序验证与测试

在传统的形式化方法研究中，最受重视的是形式验证，即证明一个给定的程序具有某种性质. 在验证过程中，往往需要证明大量的定理. 因此，具有一定自动化程度的定理证明器十分重要. 在理想情况下，要证的猜想都是定理. 但是，很多实例研究表明，不少猜想并不成立. Rushby 和 Srivas[164] 在谈到他们使用 PVS 的经验时说：当找不到想要的证明时，有可能是证明器比较弱或者时间不够，也有可能猜想本身就是假的；但很难确定究竟是哪种情况. 如果有比较好的反例生成器，就会帮助我们得到明确的结论.

约束求解技术也可用于程序测试数据的自动生成. 大家知道，程序测试方法可分为白盒测试和黑盒测试两大类. 前者的出发点是程序的内部结构，后者则只考虑程序的规格说明. Mandrioli 等人 [125] 提出了一个描述实时系统的一阶时序逻辑语言，并将形式说明的模型看成是系统的测试数据. 因此，模型构造器可作为黑盒测试的一个辅助工具. 当然，规格说明的模型非常多（即使对其大小作一定的限制）. Mandrioli 等人 [125] 为此提出了一些选择标准. 不过他们使用的工具是交互式的，其自动化程度不高.

在白盒测试方法中，一般先构造出程序的控制流图，然后选择一组符合测试标准的路径，再确定合适的输入数据，使得程序能沿着某条路径执行下去. 常用的测试标准包括：语句覆盖，条件覆盖，等等. 按照这些标准选出的某些路径实际上是不可行的（ infeasible ），也就是说，不存在输入数据，使得程序的执行轨

迹就是选定的路径. 例如, 在下面的 C 程序段中, L1-L2-L3 是不可行路径. 其中 i 是整型变量.

```
L1: if (i > 3) {
      /* other statements */
L2:   if (i*i < 8)
L3:     proc1();
      else
L4:     proc2();
    }
```

这里假定 i 的值在语句 L_1 和 L_2 之间不被改变.

为了查出不可行路径, 一般要根据数据变量的类型说明和语句的语义进行推理. 但是由于程序中数据类型的复杂性, 目前还没有合适的自动推理软件. Goldberg 等人 [67] 把语句中的基本条件抽象为布尔变量, 然后将布尔表达式的可满足性判定工具用于程序路径的可行性分析和测试数据生成. 当然, 这样并不能完全解决问题.

§4.4 人工智能及其他

§4.4.1 机器人规划

规划(planning)问题要求我们找一个动作序列(即规划), 以达到某个给定的目标 (比如使系统的状态满足一定的条件). 以积木世界 (blocks world) 为例. 假设有三个大小相同的积木: A, B, C. 系统的初始状态 Si 是这样的: A 和 B 都在桌子上, 而 C 在 B 之上. 我们想达到状态 Sf: A 在 B 上, B 在 C 上, C 在桌子上. 如图 4.3 所示.

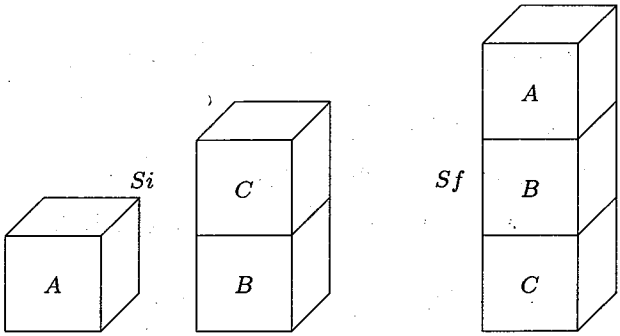


图 4.3

怎样使系统由状态 Si 到状态 Sf 呢？一个符合要求的规划（动作序列）是，$puton(C, Table)$, $puton(B, C)$, $puton(A, B)$. 这里，$puton(x, y)$ 表示将 x 放到 y 之上.

Kautz 和 Selman[100,101] 将规划问题看成是逻辑公式的可满足性问题. 他们采用下述谓词：

- $\text{on}(x, y, t)$：在时刻 t，积木 x 在积木 y 之上.
- $\text{clear}(x, t)$：在时刻 t，积木 x 上是空的，可以将其他积木放上去.
- $\text{move}(x, y, z, t)$：在时刻 t 和 $(t+1)$ 之间，将积木 x 从 y 上面挪到 z 上面.

他们使用多种类的逻辑，包括两个类型：积木和时间. 一般只考虑有限时间段. 公理如下：

1. 动作的前提条件和后果. 例如，

$$\text{move}(x, y, z, t) \rightarrow \text{clear}(x, t) \wedge \text{clear}(z, t) \wedge \text{on}(x, y, t)$$
$$\text{move}(x, y, z, t) \rightarrow \text{on}(x, z, t+1) \wedge \text{clear}(y, t+1)$$

2. 框架公理（ frame axioms ）. 执行一个动作不会影响所

有的谓词. 例如

$$\text{move}(x,y,z,t) \rightarrow (\text{on}(z,w,t+1) \leftrightarrow \text{on}(z,w,t))$$

3. 每一时刻有且仅有一个动作发生.

在这些公理中, 变量都受全称约束. 此外, "积木" 这个类型有一个特殊元素 "桌子" (Table). 它有如下性质:

$$\neg\ \text{move}(\text{Table},x,y,t). \quad \text{clear}(\text{Table},t).$$

也就是说, 我们不搬动桌子, 而桌子上总有空间, 总可以放积木.

满足初始状态、目标状态和上述公理的有限模型就是一个规划. 因此我们可以用可满足性判定过程来构造规划. 为了减少命题变元的个数, Kautz 和 Selman 引入新的二元谓词 object, source 和 destination 来代替四元谓词 move. 它们的对应关系如下:

$$\text{move}(x,y,z,t) \equiv (\text{object}(x,t) \wedge \text{source}(y,t) \wedge \text{destination}(z,t))$$

§4.4.2 资源调度

在调度 (scheduling) 问题中, 我们要找出一组操作的执行时间及其占用的资源 (如哪台机器), 使得一定的约束条件得到满足. 根据约束条件的不同, 可以将调度问题分成几类 (包括 job-shop, open-shop 等等).

Crawford 和 Baker[37] 讨论了如何将一种调度问题转换成 SAT. 这类问题中有如下一些约束条件:

1. 先后次序. 例如, 操作 i 必须在操作 j 之前完成.

2. 资源冲突. 如果操作 i 和操作 j 都只能在某一台机器上完成, 那么这两个操作不能同时进行.

3. 时间限制. 操作 i 必须在时刻 D_i 之前完成，而其起始时间不能早于时刻 B_i.

为了能用命题逻辑来描述问题，他们引进了这样一些命题变元：

- $pr_{i,j}$（操作 i 在操作 j 之前执行）
- $sa_{i,t}$（操作 i 的起始时间大于或者等于 t ）
- $eb_{i,t}$（操作 i 在时间 t 之前结束）

不妨用非负整数 $\{0,1,\cdots,d-1\}$ 来表示时间. 假设操作 i 的执行时间已给定，记为 P_i. Crawford 和 Baker[37] 按照下述方式由调度问题的约束条件得到命题逻辑公式：

1. 对任何操作 i 和时间 t（ $0 < t < d$ ），有公式 $sa_{i,t} \rightarrow sa_{i,t-1}$.

2. 对任何操作 i 和时间 t（ $0 \leq t < d-1$ ），有公式 $eb_{i,t} \rightarrow eb_{i,t+1}$.

3. 对任何操作 i 和时间 t（ $0 \leq t$, $t+P_i-1 < d$ ），有公式 $sa_{i,t} \rightarrow \neg eb_{i,t+P_i-1}$.

4. 对任何操作 i 、操作 j 和时间 t（ $0 \leq t$, $t+P_i < d$ ），有公式 $sa_{i,t} \wedge pr_{i,j} \rightarrow sa_{j,t+P_i}$.

5. 如果操作 i 必须在操作 j 之前完成，那么有子句 $pr_{i,j}$.

6. 如果操作 i 和操作 j 有资源冲突，那么有子句 $pr_{i,j} \vee pr_{j,i}$.

7. 从时间限制条件可以得到子句 sa_{i,B_i} 以及 eb_{i,D_i}.

采用上面这种转换方法，可以得到一些结构化的（而不是随机生成的）可满足性问题实例. 这有助于我们更全面地、客观地比较 SAT 算法和有关的软件工具. Crawford 和 Baker[37] 将一些调度问题实例转换成 SAT ，并比较了三种不同类型的算法：GSAT, Tableau（基于 DP 过程）和 Isamp（重复随机取样）. 它们的结论是： Isamp 最好.

不过就调度问题本身而言，可能用整数域上的约束求解方法更合适. 因为用上述办法得到的子句集合中，变元的个数直接依赖于时间段的长短. 假设有 10 个操作，100 个时间点，就有几千个命题变元. 如果我们考虑 1000 个时间点，就有几万个命题变元.

§4.4.3 图像理解

Reiter 和 Mackworth[161] 提出一个图像领域知识表示的逻辑框架，从而将图像理解形式化. 他们用一阶逻辑公式来描述有关的知识，并把对图像的一个解释看成是这些公式的一个模型. 为了方便起见，我们采用多种类的逻辑公式.

Reiter 和 Mackworth 将要讨论的对象分为图像对象(image-object)和场景对象(scene-object)两大类. 前者是物理概念，后者是人脑中的概念. 每类又可分成若干个互不相交的子类，如图 4.4 所示.

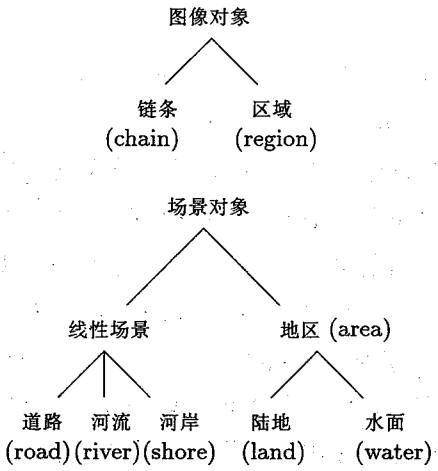


图 4.4

对于图像引入如下谓词：

• $\mathrm{tee}(c_1, c_2)$ 和 $\mathrm{chi}(c_1, c_2)$ 表示链条 c_1 和链条 c_2 相交，其形状分别为 T 形和 χ 形.

• $\mathrm{closed}(c)$ 表示链条 c 是封闭的（即首尾相接，形成一个圈）.

• $\mathrm{bounds}(c, r)$ 表示区域 r 的一个边界是链条 c.

• $\mathrm{interior}(c, r)$ 和 $\mathrm{exterior}(c, r)$ 分别表示区域 r 在链条 c 的内侧和外侧.

对场景对象也可以定义一些谓词，如 $\mathrm{loop}(s)$, $\mathrm{cross}(s_1, s_2)$, $\mathrm{inside}(s_1, s_2)$, $\mathrm{outside}(s_1, s_2)$, $\mathrm{beside}(s_1, s_2)$, $\mathrm{joins}(s_1, s_2)$, 等等. 它们必须满足一些公理，如，$\forall x : river\ \neg loop(x)$.

用 $\Delta(i, s)$ 表示将图像 i 理解为场景 s. 通过它我们可将两类对象联系起来. 当然，关于该谓词也有一些公理. 例如，对于任何图像 i_1 和 i_2，任何场景 s_1 和 s_2，

$$(\Delta(i_1, s_1) \wedge \Delta(i_2, s_2)) \rightarrow (\mathrm{tee}(i_1, i_2) \leftrightarrow \mathrm{joins}(s_1, s_2))$$
$$(\Delta(i_1, s_1) \wedge \Delta(i_2, s_2)) \rightarrow (\mathrm{chi}(i_1, i_2) \leftrightarrow \mathrm{cross}(s_1, s_2))$$

Reiter 和 Mackworth 只讨论了比较简单的图像，如手工绘制的地图. 按照他们提出的逻辑框架，对这种图像的一种解释就是一组逻辑公式的模型. 这组公式包括对图像本身的形式描述和一些公理. 给定一个图像，可以用 CSP 或 SAT 算法来获得对它的解释. 所得到的解释可能不是唯一的.

§4.4.4 其他

在知识工程中，我们往往把关于客观世界的一些知识存放到知识库中. 很多学者用一组命题逻辑公式 S 来描述知识库. 由于各种原因，我们所掌握的信息不一定是完备的. 因此对某个公式（询问）φ，可能会出现这种情况：S 不能推出 φ，也不能推

出 $\neg\varphi$. 这时应不应该相信 φ 呢？一种办法就是看看 S 有多少模型支持 φ（即 φ 在其中成立）. 如果支持 φ 的模型比支持 $\neg\varphi$ 的模型多，那么就认可 φ.

随着时间的推移，我们对现实世界的认识也在改变. 新的知识很可能和已有的知识相矛盾. 怎样修改知识库（比如删掉一些公式），使它没有矛盾，这是个很值得研究的问题. 一般要求尽可能多地保留公式. 因此，上述问题实际上就是 Max-SAT.

在前一章我们提到，模态逻辑可用来描述安全协议. Johann Schumann[167] 采用基于语法的翻译法，将 BAN 逻辑翻译到一阶谓词逻辑，再用传统的定理证明器 SETHEO[111] 来验证安全协议的正确性.

Fabio Massacci[130] 建议将密钥搜索作为测试 SAT 程序的一个挑战性问题. 他用命题逻辑来描述美国数据加密标准（Data Encryption Standard 即 DES）的加密算法. 经过一定的转换，明文、密钥和密文之间的关系可以表示为一组命题逻辑公式. 如果我们的 SAT 程序效率足够高，就能够进行已知明文攻击，根据一部分明文和密文找到密钥. 但是，Massacci 用 WalkSAT[171] 和 relsat[6] 所进行的实验结果表明，目前的 SAT 技术离这一目标还差得很远.

Kamps[99] 将数理逻辑用于社会学的研究，对组织机构的活动和行为进行形式化. 利用定理证明器 OTTER[136] 和模型构造程序 MACE [137]，可以得到合适的公理.

结 束 语

逻辑公式的可满足性问题是计算机科学界的一个重要问题. 近 40 年来研究人员在这一领域做了大量工作. 有关成果大致可分为这几方面: 理论复杂度、算法、实现技术和应用. 我们侧重于算法及应用, 尤其是判定命题逻辑公式可满足性和构造一阶逻辑公式有限模型的搜索算法.

我们知道, 关于可满足性问题, 有一些不太好的理论结果. 对命题公式来讲, 它是 NP 难的. 而一阶逻辑公式的可满足性是不可判定的. 因此对这类问题, 不能指望有什么完美的解决办法. 我们只能退而求其次. 比如说, 给 SAT 加以一定的限制, 得到一个多项式复杂度的算法. 再如, 对一阶逻辑公式, 我们只关心它的有穷模型是否存在.

经典的理论结果一般是考虑所有情况, 特别是最坏情况. 虽然它们很令人失望, 但我们也不必悲观. 作一个类比. 在可计算性理论研究中, 有很多问题被证明是不可判定的. 但是今天越来越多的人使用计算机. 实际上, 大多数 SAT 问题实例是容易解决的. 因此最近 10 年, 人们往往用公认比较难的随机实例来比较各种算法和过程. 另外, 很多人也针对各自所感兴趣的应用领域, 期望算法在尽可能多的一类问题实例上, 有比较好的性能.

在可满足性判定算法的研究中, 人们想方设法减少时间开销. 例如, 很多分支策略被提出来, 用以改善回溯算法. 至于局部搜索过程, 我们关心的一个重要问题是, 怎样避免重复搜索, 如何尽快跳出局部极值点. 当然对于 OBDD 方法来说, 如何减少空

间开销则是个关键问题.

在§1.8我们比较了命题公式可满足性问题求解的各种方法. 它们各有一些特点. 局部搜索法的一个缺点是不完备性. 除此而外, 很难说某个方法比其他方法具有明显的优越性. 与命题逻辑相比, 用一阶逻辑公式来描述问题有一定的好处. 它使问题的结构变得更清楚, 并且允许比较大的推理步骤. 对于著名的鸽笼问题[74]和一些抽象的数学问题, 采用避免同构子模型的技术具有明显的效果. 当然, 关于一阶公式的高效推理还需要作更深入的研究.

SAT可看成是约束满足问题(CSP)的特例. 在SAT算法和一阶公式有限模型构造方法中, 也能用到解决CSP的一些思想, 特别是选择未知量的策略. 通过增加约束条件(公式), 可能会提高搜索效率. 比如在解QG5问题时, 如果增加两个等式, 就会大大缩短程序的运行时间. 此外, Stålmarck方法中的简单规则与CSP求解算法中的布尔值传播技术类似.

本书多次提到转换法或者叫翻译法, 比如将模态逻辑公式翻译成经典逻辑公式(§3.2.2), 将一阶公式的有穷模型构造问题转换成SAT(§2.3.2), 将SAT转成多项式问题(§1.7.2), 等等. 这种做法是不是一种好办法呢? 我们认为, 这取决于有关技术的当前水平. 比如说, 在1990年以前, SAT算法和实现技术很落后, 而运筹学中的有关方法比较成熟. 因此人们提出将SAT转为整数规划问题. 但是目前的状况是, 直接处理命题公式的程序效率更高. 不管怎样, 在各种不同问题模型之间建立联系还是有益的. 从解决一类问题的方法中可能会得到启发, 从而设计出解决另一类问题的办法.

除了问题转换之外, 还可以通过其他一些途径来提高算法的效率. 比如说, 充分利用一些结构特性, 以及将算法并行化. 当

然, 对于 NP 难问题来说, 处理器的速度和数量并不是最重要的. 一个串行程序的运行时间有可能少于或者接近另一个并行程序的时间.

在过去十年内, 世界各国的研究人员实现了很多用于可满足性判定的软件工具. 在大家的共同努力之下, 它们的性能得到迅速提高. 例如, 1993 年初, SATO[201] 在 SUN SPARC station 2 上解决 QG5.9 所花的时间大约是 25 个小时, 而到 1994 年底只需不到 1 秒. 除了算法和计算机硬件的因素之外, 数据结构和其他实现技术也很重要. 但是有关文献很少描述实现细节, 以前人们也不太注重这方面. 1993 年举办的 SAT 比赛中, 不少参赛程序有各种各样的错误 [24]. 后来的程序一般都经过各种测试, 在性能和可靠性方面都比较好.

如果想进一步了解与 SAT 有关的研究, 可参看文献 [48,98]. 另外, 最好自己动手去实现算法, 或者用已有的软件工具做实验. 这样会得到更深刻的体验. 虽然很多人已经研究过 SAT 算法, 但是仍有可能进一步提高效率.

目前, 一些高效率的 SAT 程序可以处理很多含有几千个变量、上百万个子句的问题实例. 可是它们仍然解决不了一些具有实际意义的问题. 我们认为, 其中一个原因在于约束条件的表示. 对有些问题来说, 用一阶公式可能更好. 如果牵涉到数值的约束条件, 一般应该用约束程序设计系统 [189] 或者其他专门的约束求解软件. 另一个办法是, 对原问题进行抽象, 用命题变元代替某些简单的约束条件. 但这会丧失一些有用的信息, 导致像“假报错”这样的后果. 如何采用比较好的转换方式, 或者将命题逻辑公式和数值约束条件有机地结合起来, 这是值得进一步研究的课题.

此外, 我们觉得, 应该和具体应用结合起来研究可满足性问

题．逻辑公式具有丰富的表示能力．基于逻辑的工具比较灵活，可用于解决来自不同领域的、表面上看起来没有关系的问题．在研究某一类问题时发明的技术可能对其他很多问题都有用．这些都是逻辑的好处．但是从长远来看，也不能离开具体的应用，纯粹地研究一般形式的可满足性问题．根据具体应用领域的特点，往往可得到比较强的改善算法性能的办法．

也许很多人觉得，处理逻辑公式的软件没什么用，但是在欧美的一些国家，这方面的发明已经获得专利，也有不少公司用这类技术获取利润．即使某些研究成果在目前看起来还没有什么实用价值，但是其中的一些思想可以被用于其他类似的研究中．

另一种极端的观点是，用逻辑的框架可以解决一切问题．但实际上并非如此．描述性的逻辑公式并不是在任何情况下都很好用．它有一些弱点，比如所谓的“框架问题”（ frame problem ）．另外，对一个具体的应用，如果有更好的专门算法，通常人们不会使用一般性的工具．

本书的重点尽量放在比较成熟的技术或者有意思的应用上．当然这也和作者自己的知识面和研究兴趣相关．限于篇幅，我们没有涉及很多内容，包括：一阶谓词逻辑公式无限模型的构造，关于高阶逻辑、多值逻辑、模糊逻辑的自动推理，等等．

参 考 文 献

[1] W. Ackermann, Solvable Cases of the Decision Problem, North-Holland, Amsterdam, 1954.

[2] S.B. Akers, Binary decision diagrams, IEEE Trans. on Computers, C-27(6), 509~516, 1978.

[3] J.F. Allen and P.J. Hayes, A common-sense theory of time, Proc. IJCAI-85, 528~531.

[4] Y. Asahiro, K. Iwama and E. Miyano, Random generation of test instances with controlled attributes, In [98], 377~393.

[5] B. Aspvall, M. F. Plass and R. E. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, IPL, 8(3), 121~123, 1979.

[6] R. J. Bayardo Jr. and R. C. Schrag, Using CSP look-back techniques to solve real-world SAT instances, Proc. 14th AAAI, 203~208, 1997.

[7] B. Benhamou and L. Sais, Tractability through symmetries in propositional calculus, JAR 12(1), 89~102, 1994.

[8] F. Bennett, Quasigroup identities and Mendelsohn designs, Canadian Journal of Mathematics 41(2), 341~368, 1989.

[9] F.E. Bennett, H. Zhang and L. Zhu, Self-orthogonal Mendelsohn triple systems, J. of Combinatorial Theory A73, 207~218, 1996.

[10] F. Bennett and L. Zhu, Conjugate-orthogonal latin squares and related structures, in: J. Dinitz & D. Stinson (eds.), Contemporary Design Theory: A Collection of Surveys, John Wiley & Sons, 41~96, 1992.

[11] A. Biere *et al.* Symbolic model checking without BDDs, Proc. TACAS'99, LNCS, 1579, 1999.

[12] A. Blass and Y. Gurevich, On the unique satisfiability problem, Information and Control 55(1-3), 80~88, 1982.

[13] C. Bourely, R. Caferra and N. Peltier, A method for building models automatically: Experiments with an extension of OTTER, Proc. CADE-12 LNAI 814, 72~86, 1994.

[14] T. Boy de la Tour, An optimality result for clause form translation, J. Symbolic Computation 14, 283~301, 1992.

[15] M. Böhm and E. Speckenmeyer, A fast parallel SAT-solver ~ efficient workload balancing, Annals of Mathematics and Artificial Intelligence 17(3-4), 381~400, 1996.

[16] E. Börger, E. Grädel and Y. Gurevich, The Classical Decision Problem, Springer Verlag, Berlin, 1997.

[17] K.S. Brace, R.L. Rudell and R.E. Bryant, Efficient implementation of a BDD package, Proc. 27th DAC, 40~45, 1990.

[18] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Computers, C-35(8), 677~691, 1986.

[19] R.E. Bryant, Symbolic Boolean manipulation with ordered-binary decision diagrams, ACM Computing Surveys 24(3), 293~318, 1992.

[20] R. Bryant, On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication, IEEE Trans, Computers 40(2), 205~213, 1991.

[21] R.E. Bryant and Y.-A. Chen, Verification of arithmetic circuits with binary moment diagrams, Proc. 32nd DAC, 535~541, 1995.

[22] 卜东波，命题逻辑的可满足性问题：复杂性和算法，中国科学院计算技术研究所硕士论文，北京， 1997 .

[23] J.R. Burch et al. Sequential circuit verification using symbolic model checking, Proc. 27th DAC, 46~51, 1990.

[24] M. Buro and H.K. Büning, Report on a SAT competition, Bulletin of the European Association for Theoretical Computer Science (EATCS), No. 49, 143~151, 1993.

[25] M. Burrows, M. Abadi and R.M. Needham, A logic of authentication, Proc. Royal Society of London, A 426, 233~271, 1989.

[26] R. Caferra and N. Zabel, A method for simultaneous search for refutations and models by equational constraint solving, J. of Sym-

bolic Computation 13(6), 613~641, 1992.

[27] L. Catach, TABLEAUX: A general theorem prover for modal logics, J. of Automated Reasoning 7, 489~510, 1991.

[28] V. Chandru et al. On renamable Horn and generalized Horn functions, Annals of Mathematics and Artificial Intelligence 1, 33~47, 1990.

[29] V. Chandru and J.N. Hooker, Extended Horn sets in propositional logic, J ACM 38(1), 205~221, 1991.

[30] V. Chandru and J.N. Hooker, Detecting embedded Horn structure in propositional logic, Info. Proc. Letters 42(2), 109~111, 1992.

[31] C.L. Chang and R.C.T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.

[32] P. Cheeseman, B. Kanefsky and W.M. Taylor, Where the *really* hard problems are, Proc. IJCAI-91, 331~337, 1991.

[33] B.F. Chellas, Modal Logic: An Introduction, Cambridge University Press, 1980.

[34] C.-A. Chen and S.K. Gupta, A satisfiability-based test generator for path delay faults in combinatorial circuits, Proc. 33rd DAC,209~214, 1996.

[35] S.A. Cook, The complexity of theorem-proving procedures, Proc. 3rd ACM Symposium on Theory of Computing, 151~158, 1971.

[36] J.M. Crawford and L.D. Auton, Experimantal results on the crossover point in satisfiability problems, Proc. 11th AAAI, 21~27, 1993.

[37] J.M. Crawford and A.B. Baker, Experimental results on the application of satisfiability algorithms to scheduling problems, Proc. 12th AAAI, 1092~1097, 1994.

[38] C.A. Damon, D. Jackson and S. Jha, Checking relational specifications with binary decision diagrams, ACM Software Engineering Notes, 21(6), 70~80, 1996.

[39] M. Davis, G. Logemann and D. Loveland, A machine program for theorem proving, Communications of the ACM 5(7), 394~397, 1962.

[40] M. Davis and H. Putnam, A computing procedure for quantification theory, J ACM 7(3), 201~215, 1960.

[41] M.D. Davis and E.J. Weyuker, Computability, Complexity and Languages, Academic Press, New York, 1983.

[42] J. de Kleer, A comparison of ATMS and CSP techniques, Proc. IJCAI-89, 290~296, 1989.

[43] J. de Kleer, An improved incremental algorithm for generating prime implicates, Proc. 10th AAAI, 780~785, 1992.

[44] S. Demri, Efficient strategies for automated reasoning in modal logics, in: Logics in AI, LNAI 838, 182~197, 1994.

[45] S. Devadas, Comparing two-level and ordered binary decision diagram representations of logic functions, IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, 12(5), 722~723, 1993.

[46] W.F. Dowling and J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. of Logic Programming 1(3), 267~284, 1984.

[47] B. Dreben and W.D. Goldfarb. The Decision Problem: Solvable Classes of Quantificational Formulas, Addison-Wesley, Reading, Massachusetts, USA, 1979.

[48] D.-Z. Du, J. Gu, P.M. Paroalos (eds.) Satisfiability Problem: Theory and Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35, American Mathematical Society, Providence, R.I., USA, 1997.

[49] O. Dubois et al. SAT versus UNSAT, In [98], 415~436.

[50] H.-D. Ebbinghaus, J. Flum and W. Thomas, Mathematical Logic, Springer-Verlag, New York, USA, 1984.

[51] P. Enjalbert, and L. Fariñas del Cerro, Modal resolution in clausal form, Theoret. Comp. Sci. 65, 1~33, 1989.

[52] C. Fermüller et al. Resolution Methods for the Decision Problem, LNAI 679, 1993.

[53] M. Fitting, Proof Methods for Modal and Intuitionistic Logics, D. Reidel Publishing Company, Dordrecht, Holland, 1983.

[54] J.W. Freeman, Improvements to Propositional Satisfiability Search Algorithms, PhD Dissertation, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA, 1995.

[55] 傅京孙、蔡自兴、徐光佑, 人工智能及其应用, 清华大学出版社, 北京, 1987.

[56] H. Fujii, G. Ootomo and C. Hori, Interleaving based variable ordering methods for ordered binary decision diagrams, Proc. ICCAD, 38~41, 1993.

[57] M. Fujita, H. Fujisawa and Y. Matsunaga, Variable ordering algorithms for ordered binary decision diagrams and their evaluation. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 12(1), 6~12, 1993.

[58] M. Fujita, J. Slaney and F. Bennett, Automatic generation of some results in finite algebra, IJCAI-93, 52~57, 1993.

[59] G. Gallo and G. Urbani, Algorithms for testing the satisfiability of propositional formulae. J. of Logic Programming 7(1), 45~61, 1989.

[60] G. Gallo et al. Directed hypergraphs and applications. Discrete Applied Math. 42, 177~201, 1993.

[61] G. Gallo and D. Pretolani, A new algorithm for the propositional satisfiability problem. Discrete Applied Math. 60(1-3), 159~179, 1995.

[62] A. Galton, Note on a lemma of Ladkin, J. of Logic and Computation 6(1), 1~4, 1996.

[63] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, CA, USA, 1979.

[64] I.P. Gent and T. Walsh, The hardest random SAT problems, in: KI-94: Advances in Artificial Intelligence, LNAI 861, 355~366, 1994.

[65] F. Giunchiglia and R. Sebastiani, Building decision procedures for modal logics from propositional decision procedures——the case study of modal K, Proc. CADE-13 LNAI 1104, 583~597, 1996.

[66] P. Goel, An implicit enumeration algorithm to generate tests for combinatorial logic circuits, IEEE Trans. Computers, C-30(3), 215~222, 1981.

[67] A. Goldberg, T.C. Wang and D. Zimmerman, Applications of feasible path analysis to program testing, Proc. ISSTA 94, ACM SIGSOFT Software Engineering Notes 19, Special Issue, 80~94, 1994.

[68] R. Goré, Tableau methods for modal and temporal logics, in: Handbook of Tableau Methods, (eds.) M. D'Agostino et al. Kluwer, 1999.

[69] J. Gu, Efficient local search for very large-scale satisfiability problems, ACM SIGART Bulletin 3(1), 8~12, 1992.

[70] J. Gu, Local search for satisfiability (SAT) problem. IEEE Trans. on Systems, Man, and Cybernetics, 23(4), 1108~1129, 1993. Corrigenda, IEEE Trans. on Systems, Man, and Cybernetics, 24(4), 709, 1994.

[71] J. Gu, Global optimization for satisfiability (SAT) problem, IEEE Trans. Knowledge and Data Engineering 6(3), 361~381, 1994. Corrections, 7(1), 192, 1995.

[72] J. Gu, Q. Gu and D. Du, On optimizing the satisfiability (SAT) problem. J. of Computer Science and Technology, 14(1), 1~17, 1999.

[73] A. Guha and H. Zhang, Andrews' challenge problem: Clause conversion and solutions, Association for Automated Reasoning Newsletter, No. 14, 5~8, Dec. 1989.

[74] A. Haken, The intractability of resolution, Theoret. Comp. Sci 39, 297~308, 1985.

[75] J.Y. Halpern and Y. Moses, A guide to completeness and complexity for modal logics of knowledge and belief, Artificial Intelligence, 54, 319~379, 1992.

[76] P. Hansen and B. Jaumard, Uniquely solvable quadratic boolean equations, Discrete Applied Math. 12(2), 147~154, 1985.

[77] R. Hasegawa, Parallel theorem-proving system: MGTP, Proc.

FGCS'94, 51~65, 1994.

[78] R. Hasegawa and H. Fujita, A model generation theorem prover in KL1 using ramified-stack algorithm, Proc. 8th Int'l Conf. on Logic Programming, 535~548, 1991.

[79] P.J. Hayes and J.F. Allen, Short time periods, Proc. 10th IJCAI, 981~983, 1987.

[80] 贺思敏，可满足性问题的算法设计与分析，清华大学计算机科学与技术系博士论文，北京，1997。

[81] M.P.E. Heimdahl and N.G. Leveson, Completeness and consistency in hierarchical state-based requirements, IEEE Trans. Softw. Eng. 22(6), 363~377, 1996.

[82] C.L. Heitmeyer et al. Automated consistency checking of requirements specification, ACM Transactions on Software Engineering and Methodology 5(3), 231~261, July 1996.

[83] J.-J. Hébrard, A linear algorithm for renaming a set of clauses as a Horn set, Theoret. Comp. Sci. 124(2), 343~350, 1994.

[84] T. Hogg, B.A. Huberman and C.P. Williams, Phase transitions and the search problem, Editorial of the special volumn, Artif. Intel. 81, 1~15, 1996.

[85] J.N. Hooker, A quantitative approach to logical inference, Decision Support Systems 4(1), 45~69, 1988.

[86] J.N. Hooker, Resolution vs. cutting plane solution of inference problems: some computational experience, Operations Research Letters 7(1), 1~7, 1988.

[87] J.N. Hooker and V. Vinay, Branching rules for satisfiability, J. of Automated Reasoning 15(3), 359~383, 1995.

[88] H.H. Hoos, SAT-encodings, search space structure, and local search performance, Proc. 16th IJCAI, 296~302, 1999.

[89] 黄文奇、金人超，求解 SAT 问题的拟物拟人算法 ——Solar，中国科学（E 辑），27(2)，179~186，1997。

[90] G.E. Hughes and M.J. Cresswell, An Introduction to Modal Logic, Methuen and Co., 1968.

[91] K. Iwama and S. Miyazaki, SAT-variable complexity of hard com-

binatorial problems, Proc. 13th World Computer Congress, IFIP, Vol. 1, 253~258, 1994.

[92] N. Ishiura, H. Sawada and S. Yajima, Minimization of binary decision diagrams based on exchanges of variables, Proc. ICCAD, Santa Clara, CA, USA, 472~475, 1991.

[93] D. Jackson, An intermediate design language and its analysis, ACM Software Engineering Notes, 23(6), 121~130, 1998.

[94] D. Jackson, S. Jha and C.A. Damon, Isomorph-free model enumeration: A new method for checking relational specifications, ACM Transactions on Programming Languages and Systems 20(2), 302~343, 1998.

[95] J. Jaffar and M.J. Maher, Constraint logic programming: A survey, J. of Logic Programming 19/20, 503~581, 1994.

[96] B. Jaumard and B. Simeone, On the complexity of the maximum satisfiability problem for Horn formulas, Info. Proc. Letters 26, 1~4, 1987.

[97] R. Jeroslow and J. Wang, Solving propositional satisfiability problems, Annals of Mathematics and Artificial Intelligence 1, 167~187, 1990.

[98] D.S. Johnson and M.A. Trick, (eds.) Cliques, Coloring, and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, The American Mathematical Society, Providence, RI, USA, 1996.
`http://dimacs.rutgers.edu/Challenges/index.html`

[99] J. Kamps, On criteria for formal theory building: Applying logic and automated reasoning tools to the social sciences, Proc. 16th AAAI, 285~290, 1999.

[100] H. Kautz and B. Selman, Planning as satisfiability, Proc. European Conf. on Artificial Intelligence, 359~363, 1992.

[101] H. Kautz and B. Selman, Pushing the envelope: Planning, propositional logic, and stochastic search, Proc. 13th AAAI, 1194~1201, 1996.

[102] S. Kim and H. Zhang, ModGen: Theorem proving by model gen-

eration, Proc. 12th AAAI, 162~167, 1994.

[103] D.E. Knuth, The Art of Computer Programming, Vol.3: Sorting and Searching, Addison-Wesley, Reading, Massachusetts, USA, 1973.

[104] E. Koutsoupias and C.H. Papadimitriou, On the greedy algorithm for satisfiability, Info. Proc. Letters 43(1), 53~55, 1992.

[105] O. Kullmann, New methods for 3-SAT decision and worst-case analysis, Theoret. Comp. Sci. 223(1-2), 1~72, 1999.

[106] V. Kumar, Algorithms for constraint satisfaction problems: A survey. AI magazine, 32~44, Spring 1992.

[107] K. Kunen, *G*-loops and permutation groups, J. of Algebra 220(2), 694~708, 1999.

[108] P. Ladkin, Models of axioms for time intervals, Proc. 6th AAAI, 234~239, 1987.

[109] T. Larrabee, Test pattern generation using Boolean satisfiability, IEEE Trans. on Computer-Aided Design 11(1), 4~15, 1992.

[110] S.-J. Lee and D.A. Plaisted, Eliminating duplication with the hyper-linking strategy, J. of Automated Reasoning 9(1), 25~42, 1992.

[111] R. Letz et al. SETHEO: A High-Performance Theorem Prover, J. of Automated Reasoning 8, 183~212, 1992.
http://wwwjessen.informatik.tu-muenchen.de/~setheo/

[112] 李未、黄文奇，一种求解合取范式可满足性问题的数学物理方法，中国科学(A 辑)， 24(11)， 1208~1217， 1994 .

[113] 梁东敏、吴晔、马绍汉，一个求解结构 SAT 问题的高效局部搜索算法，计算机学报， 21(增刊)， 92~97， 1998 .

[114] 刘涛、李国杰，求解 SAT 问题的分级重排搜索算法，软件学报， 7(4)， 201~210， 1996 .

[115] 刘叙华、姜云飞，定理机器证明，科学出版社，北京， 1987 .

[116] 刘叙华，基于归结方法的自动推理，科学出版社，北京， 1994 .

[117] D.W. Loveland, Automated Theorem Proving: A Logical Basis, North-Holland, Amsterdam, 1978.

[118] D.W. Loveland et al. SATCHMORE: SATCHMO with RElevancy, J. of Automated Reasoning 14(2), 325~351, 1995.

[119] 陆汝钤，人工智能（上、下），科学出版社，北京， 1989 .

[120] 陆钟万，数理逻辑与机器证明，科学出版社，北京， 1983 .

[121] 陆钟万，面向计算机科学的数理逻辑，北京大学出版社，北京， 1989 .

[122] W. Lu and Y. Zhang, Experimental study on strategy of combining SAT algorithms, J. of Computer Science and Technology 13(6), 608~614, 1998.

[123] A.K. Mackworth, Constraint satisfaction, in: Encyclopedia of Artificial Intelligence, (ed.) S.C. Shapiro, Vol. 1, John Wiley, New York, 205~211, 1990.

[124] S. Malik et al. Logic verification using binary decision diagrams in a logic synthesis environment, Proc. ICCAD, 6~9, 1988.

[125] D. Mandrioli et al. Generating test cases for real-time systems from logic specifications, ACM Trans. on Computer Systems, 13(4), 365~398, 1995.

[126] Z. Manna and A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer Verlag, New York, 1992.

[127] R. Manthey and F. Bry, SATCHMO: A theorem prover implemented in Prolog, Proc. CADE-9, LNCS 310, 415~434, 1988.

[128] A. Martelli and U. Montanari, An efficient unification algorithm, ACM Trans. on Prog. Lang. and Syst. 4(2), 258~282, 1982.

[129] F. Massacci, Design and results of the Tableaux-99 non-classical (modal) systems comparison, Proc. TABLEAUX'99, LNAI 1617, 1999.

[130] F. Massacci, Using Walk-SAT and Rel-SAT for cryptographic key search, Proc. 16th IJCAI, 290~295, 1999.

[131] B. Mazure et al. Tabu search for SAT, Proc. 14th AAAI, 281~285, 1997.

[132] B. Mazure et al. System description: CRIL platform for SAT, Proc. CADE-15, LNAI 1421, 124~128, 1998.

[133] D. McAllester, B. Selman and H. Kautz, Evidence for invariants in local search, Proc. 14th AAAI, 321~326, 1997.

[134] W. McCune, Experiments with semantic paramodulation, J. of Automated Reasoning 1, 231~261, 1985.

[135] W.W. McCune, Single axioms for groups and Abelian groups with various operations, J. of Automated Reasoning 10(1), 1~13, 1993.

[136] W. McCune, OTTER 3.0 reference manual and guide, Argonne National Laboratory Tech. Rep. ANL-94/6, Argonne, IL., USA, 1994.
`http://www-unix.mcs.anl.gov/AR/otter/`

[137] W. McCune, A Davis-Putnam program and its application to finite first-order model search: Quasigroup existence problems, Tech. Memo ANL/MCS-TM-194, Argonne National Laboratory, Argonne, IL., USA, 1994.
`http://www-unix.mcs.anl.gov/AR/mace`

[138] W. McCune and R. Padmanabhan, Automated Deduction in Equational Logic and Cubic Curves, LNAI 1095, 1996.

[139] M.C. McFarland, Formal verification of sequential hardware: A tutorial, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 12(5), 633~654, 1993.

[140] D.G. Mitchell and H.J. Levesque, Some pitfalls for experimenters with random SAT, Artif. Intel. 81, 111~125, 1996.

[141] D. Mitchell, B. Selman and H. Levesque, Hard and easy distributions of SAT problems, Proc. 10th AAAI, 459~465, 1992.

[142] B. Monien and E. Speckenmeyer, Solving satisfiability in less than 2^n steps, Disc. Applied Math. 10, 287~295, 1985.

[143] J.D. Monk, Mathematical Logic, Springer, New York, 1976.

[144] J.S. Moore. Introduction to the OBDD algorithm for the ATP community, J. of Automated Reasoning 12(1), 33~45, 1994.

[145] C.G. Morgan, Methods for automated theorem proving in nonclassical logics, IEEE Trans. on Computers, C-25(8), 852~862, 1976.

[146] B.A. Nadel, Constraint satisfaction algorithms, Computational Intelligence 5(4), 188~224, 1989.

[147] A. Nonnengart, First-order modal logic theorem proving and functional simulation, Proc. IJCAI-93, 80~85, 1993.

[148] A. Nonnengart, G. Rock and C. Weidenbach, On generating small clause normal forms, Proc. CADE-15, LNAI 1421, 397~411, 1998.

[149] H.J. Ohlbach and C. Weidenbach, A note on assumptions about Skolem functions, J. of Automated Reasoning 15, 267~275, 1995.

[150] T.J. Park and A. Van Gelder, Partitioning methods for satisfiability testing on large formulas, Proc. CADE-13, LNAI 1104, 748~762, 1996.

[151] A.J. Parkes and J.P. Walser, Tuning local search for satisfiability testing, Proc. 13th AAAI, 356~362, 1996.

[152] F.J. Pelletier, Seventy-five problems for testing automatic theorem provers, J. of Automated Reasoning 2: 191~216, 1986. Errata: Vol. 4, 235~236, 1988; Vol. 18, 135, 1997.

[153] N. Peltier, A new method for automated finite model building exploiting failures and symmetries, J. of Logic and Computation 8(4), 511~543, 1998.

[154] D. Plaisted and S. Greenbaum, A structure-preserving clause form translation, J. of Symbolic Computation 2, 293~304, 1986.

[155] A. Pnueli, The temporal semantics of concurrent programs, Theoret. Comp. Sci. 13(1), 1~20, 1981.

[156] D. Pretolani, A linear time algorithm for unique Horn satisfiability, Info. Proc. Letters 48(2), 61~66, 1993.

[157] P. Prosser, Hybrid algorithms for the constraint satisfaction problem, Computational Intelligence 9(3), 268~299, 1993.

[158] P. Purdom, A survey of average time analyses of satisfiability algorithms, J. of Information Processing, 13(4), 449~455, 1990.

[159] A. Rauzy, Polynomial restrictions of SAT: What can be done with an efficient implementation of the Davis and Putnam's procedure? Proc. of the First Int'l Conf. on Principles and Practice of Constraint Programming, LNCS 976, 515~532, 1995.

[160] R. Reiter. A semantically guided deductive system for automatic theorem proving. IEEE Trans. on Computers, C-25(4), 328~334, 1976.

[161] R. Reiter and A.K. Mackworth, A logical framework for depiction

and image interpretation, Artif. Intel. 41, 125~155, 1989/90.

[162] J.A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12, 23~41, 1965.

[163] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, Proc. ICCAD, 42~47, 1993.

[164] J. Rushby and M. Srivas, Using PVS to Prove Some Theorems of David Parnas, In: Higher Order Logic Theorem Proving and Its Applications, LNCS 780, 163~173, 1994.

[165] S.J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1995.

[166] J.S. Schlipf et al. On finding solutions for extended Horn formulas, Info. Proc. Letters 54(3), 133~137, 1995.

[167] J. Schumann, Automatic verification of cryptographic protocols with SETHEO, Proc. CADE-14, LNAI 1249, 87~100, 1997.

[168] M.G. Scutellà, A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability, J. of Logic Programming 8(3), 265~273, 1990.

[169] F.F. Sellers, Jr., M.Y. Hsiao and L.W. Bearnson, Analyzing errors with the Boolean difference, IEEE Trans. on Computers, C-17(7), 676~683, 1968.

[170] B. Selman and H.A. Kautz, Domain-independent extensions to GSAT: Solving large structured satisfiablility problems, Proc. IJCAI 93, 290~295, 1993.

[171] B. Selman, H.A. Kautz and B. Cohen, Noise strategies for improving local search, Proc. 12th AAAI, 337~343, 1994.

[172] B. Selman, H. Levesque and D. Mitchell, A new method for solving hard satisfiability problems, Proc. 10th AAAI, 440~446, 1992.

[173] M. Sheeran and G. Stålmarck, A tutorial on Stålmarck's proof procedure for propositional logic, In: Formal methods in computer aided design (FMCAD'98), LNCS 1522, 82~99, 1998.

[174] 史忠植，高级人工智能，科学出版社，北京， 1998 .

[175] R.E. Shostak, On the SUP-INF method for proving Presburger formulas, J ACM 24(4), 529~543, 1977.

[176] J. Siekmann and G. Wrightson, (eds.) Automation of Reasoning: Classical Papers on Computational Logic, Springer, Berlin, 1983.

[177] J.P.M. Silva and K.A. Sakallah, GRASp——A new search algorithm for satisfiability, Proc. ICCAD, San Jose, CA, USA, 220~227, 1996.

[178] J.R. Slagle, Automatic theorem proving with renamable and semantic resolution, J ACM 14(4), 687~697, 1967.

[179] J. Slaney, SCOTT: A model-guided theorem prover, Proc. 13th IJCAI, 109~114, 1993.

[180] J. Slaney, The crisis in finite mathematics: Automated reasoning as cause and cure, Proc. CADE-12, LNCS 814, 1~13, 1994.

[181] J. Slaney, FINDER: Finite domain enumerator. System description, Proc. CADE-12, LNCS 814, 798~801, 1994.

[182] J. Slaney, M. Fujita and M. Stickel, Automated reasoning and exhaustive search: Quasigroup existence problems, Computers & Math. with Appl. 29(2), 115~132, 1995.

[183] P. Stephan et al. Combinatorial test generation using satisfiability, IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, 15(9), 1167~1176, 1996.

[184] G. Sutcliffe and C. Suttner, (eds.) Special issue: The CADE-13 Automated Theorem Proving System Competition, J. of Automated Reasoning 18(2), 1997.

[185] G. Sutcliffe and C. Suttner, The TPTP problem library, J. of Automated Reasoning 21(2), 177~203, 1998.
`http://www.cs.jcu.edu.au/~tptp/`
`http://sunjessen24.informatik.tu-muenchen.de/~tptp/`

[186] 唐稚松等，时序逻辑程序设计与软件工程（上册：时序逻辑语言），科学出版社，北京，1999.

[187] T.E. Uribe and M.E. Stickel, Ordered binary decision diagrams and the Davis-Putnam procedure, Proc. of the Int'l Conf. on Constraints in Computational Logics, LNCS 845, 34~49, 1994.

[188] A. Urquhart, Hard examples for resolution, J ACM 34(1), 209~219, 1987.

[189] P. Van Hentenryck and V. Saraswat *et al.* Strategic directions in constraint programming, ACM Computing Surveys 28(4), 701~726, 1996.

[190] 汪芳庭，数理逻辑，中国科学技术大学出版社，合肥，1990．

[191] 王元元，计算机科学中的逻辑学，科学出版社，北京，1989．

[192] C. Weidenbach et al. System description: SPASS version 1.0.0, Proc. CADE-16, LNAI 1632, 378~382, 1999.
`http://spass.mpi-sb.mpg.de/`

[193] J.M. Wilson, Compact normal forms in propositional logic and integer programming formulations. Computers & Operations Research 17(3), 309~314, 1990.

[194] S. Winker, Generation and verification of finite models and counterexamples using an automated theorem prover answering two open questions, J ACM 29, 273~284, 1982.

[195] L. Wos, D. Carson and G. Robinson, Efficiency and completeness of the set-of-support strategy in theorem proving, JACM 12, 536~541, 1965.

[196] L. Wos et al. Automated Reasoning: Introduction and Applications, (2nd ed.) McGraw-Hill, New York, 1992.

[197] L. Wos, The kernel strategy and its use for the study of combinatory logic, J. of Automated Reasoning 10(3), 287~343, 1993.

[198] L.C. Wu and C.Y. Tang, Solving the satisfiability problem by using randomized approach, Info. Proc. Letters 41(4), 187~190, 1992.

[199] 吴文俊，初等几何判定问题与机械化证明，中国科学，507~516, 1977．

[200] H. Zhang and M. Stickel, Implementing the Davis-Putnam method, J. of Automated Reasoning, 24 (1~2), 277~296, 2000.

[201] H. Zhang, SATO: An efficient propositional prover, Proc. CADE-14, LNAI 1249, 272~275, 1997.
`http://www.cs.uiowa.edu/~hzhang/sato/`

[202] H. Zhang, M.P. Bonacina and J. Hsiang, PSATO: a distributed propositional prover and its application to quasigroup problems, J. of Symbolic Computation 21, 543~560, 1996.

[203] J. Zhang, Search for models of equational theories, Proc. of

the 3rd International Conference for Young Computer Scientists (ICYCS'93), 2, 60~63, Beijing, 1993.

[204] J. Zhang, Constructing Finite Algebras with FALCON, J. of Automated Reasoning 17(1), 1~22, 1996.

[205] 张健, 模态逻辑推理的翻译方法, 计算机研究与发展, 35(5), 389~392, 1998.

[206] J. Zhang, System description: MCS: Model-based conjecture searching, Proc. CADE-16, LNAI 1632, 393~397, 1999.

[207] 张健, 有限构模器的扩展及其在形式化方法中的应用, 计算机学报, 23(2), 190~194, 2000.

[208] J. Zhang and H. Zhang, SEM: a system for enumerating models, Proc. IJCAI-95, 298~303, 1995.
http://www.cs.uiowa.edu/~hzhang/sem

[209] J. Zhang and H. Zhang, Combining local search and backtracking techniques for constraint satisfaction, Proc. 13th AAAI, Vol.1, 369~374, 1996.

[210] W. Zhang, Number of models and satisfiability of sets of clauses, Theoret. Comp. Sci. 155(1), 277~288, 1996.

(TP·1349.1101)
责任编辑：毕　颖
封面设计：张　放

ISBN 7-03-008364-4/TP·1349
定 价：18.00 元