

复旦大学

硕士学位论文

可满足性问题DPLL算法研究

姓名：熊伟

申请学位级别：硕士

专业：微电子学与固体电子学

指导教师：唐璞山

20070522

## 摘 要

近十多年来随着许多有效的启发式算法的提出以及一系列数据结构和实现方法上的创新使得可满足性问题(SAT)的求解水平取得了突飞猛进的提高。尽管目前可满足性问题还属于NP问题,目前还没有任何一种算法能实现在最差情况下的多项式时间复杂度,但当前的SAT问题求解器以能够轻松解决含有几万个甚至几十万个变量的可满足性问题。由于可满足性问题求解水平的巨大进步, SAT的应用范围已越来越大,从形式验证到人工智能等许多领域均以SAT求解器作为其核心计算引擎。可满足性问题的研究已经从一个计算机理论学界的纯学术问题逐渐发展成为当前许多计算机领域中具有很大应用价值的计算问题。

本论文的贡献在于总结和分析了那些推动SAT问题发展的最主要的启发式算法和技术,并在此基础上提出了两点创新。其一,提出了一种新的正向推理技术:对称扩展的一元子句推导。与传统的一元子句推导技术相比,本文的方法通过在一元子句推导过程中添加对称的蕴涵关系从而能够推导出更多的一元子句。基于这项技术本文实现了一个可满足性问题预处理器Snowball。实验结果验证了这项新的正向推理技术的有效性,并表明该预处理器Snowball能够有效地化简SAT问题的规模并减少解决SAT问题的时间,特别是对不满足问题有不少例子可直接得到结果。其二,本文首次提出了一种采用双变量决策策略的可满足性问题DPLL算法以及其完整的实现方式描述。采用双变量决策策略能在理论上减少决策级数,进而能有效地减少SAT问题的搜索空间,加速SAT问题的求解。该双变量决策SAT算法的实现是以Minisat解决器为蓝本的。在其较完善的DPLL算法框架内本文对其中的各个主要的功能模块均进行了改造,使得改造后的SAT解决器首次具有了双变量决策功能,并与其中主要的软件模块:变量决策模块,蕴含推理模块,冲突分析和回溯模块相互配合,协调一致。实验结果验证了算法的正确性。

关键词: 可满足性问题, 可满足性算法

## Abstract

In the last decade, multiple effective heuristics algorithms and a series of innovations in data structure and implementation have made the Satisfiability problem solving ability significantly improved. Though the Satisfiability is a well known NP problem and no known algorithm for SAT is of polynomial time complexity, state of art SAT solver can easily solve the SAT with ten thousands or even hundreds of thousands variables. Due to the great improvement in SAT solving, SAT has been more and more widely applied. From Formal Verification to Artificial Intelligence SAT has become the core computing engine. Recently SAT has grown from a problem of academic interest to a core computational resource of immense value.

This thesis has summarized and analyzed the heuristics algorithms and technologies which are the corner stones for SAT development. Based on the previous technologies this thesis contributes to two innovations in SAT solving. First, this thesis proposes a new forward reasoning technique called Symmetric Extended Unit Propagation (SEUP). Compared to the ordinary UP this new technique can deduce more unit literals by adding the symmetric implications during unit propagating. A SAT preprocessor Snowball is implemented based on this new technique. Experimental results verify the efficiency of this new technique and show that Snowball can generally reduce the overall runtime in solving SAT problems and can even solve several hard instances independently. Second, this thesis has, for the first time, proposed a DPLL algorithm with dual-variable decision policy as well as its implementation details. The dual-variable decision policy can theoretically reduce the decision depth of searching so that the search space of SAT can be greatly reduced. With the Minisat solver as its blue print, the new DPLL algorithm of SAT with dual-variable decision policy has been implemented. All the main modules such as decision, propagation, conflict learning and backtracking modules have been modified in order to realize the function of dual-variable decision within the complete DPLL framework of Minisat. The experiment results verified the functional correctness of the new DPLL algorithm.

**Keywords:** Satisfiability, SAT Algorithms

## 论文独创性声明

本论文是我个人在导师指导下进行的研究工作及取得的研究成果。论文中除了特别加以标注和致谢的地方外，不包含其他人或其它机构已经发表或撰写过的研究成果。其他同志对本研究的启发和所做的贡献均已在论文中作了明确的声明并表示了谢意。

作者签名： 熊伟 日期： 2007.6.13

## 论文使用授权声明

本人完全了解复旦大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。保密的论文在解密后遵守此规定。

作者签名： 熊伟 导师签名： 唐臻 日期： 2007.6.14

# 第1章 绪论

## 1.1 集成电路验证危机与 SAT 问题

自从1958年Jack Kilby在美国TI公司发明了第一块集成电路以来,电子工业进入了集成电路(IC)的时代。经过近50年的发展,集成电路已经从最初的小规模集成电路(SSl)起步,先后经历了中规模(MSI)、大规模(LSI)、超大规模(VLSI)、巨大规模(ULSI),发展到目前的特大规模集成电路(GSI)和系统芯片(SOC),单个电路芯片集成的元件数也从当时的十几个发展到目前的几亿个甚至几十、上百亿个。Intel的创始人之一的摩尔在1965年预测了集成电路产业的发展规律:即集成电路的集成度每三年增长四倍,特征尺寸每三年缩小2倍。近半个世纪微电子产业的飞速发展验证了摩尔定律的正确性。有研究表明当前的集成电路产业的摩尔定律发展趋势还将会持续一段时间。然而随着IC设计的规模和复杂性的指数性增长,其对集成电路的验证技术的处理能力提出了极大的挑战。2006年度的国际半导体技术发展报告(International Technology Roadmap for Semi-conductor)指出当前验证在时间、人力及费用方面均已成为设计流程中开销最大的工作[1]。在目前的工程项目中,验证工程师远多于设计工程师,对于复杂的设计更是达到了3:1。造成这种局面的主要原因是设计规模和复杂性在不断增加,尤其是SOC产品集成了多种类型的模块,如数字逻辑、混合信号和存储器模块。另外,历史上对设计流程的其它环节(逻辑综合,布局布线,测试产生等问题)关注颇多,而对设计验证重视不够。目前,设计的概念和执行仅仅成为拉开验证主体工作的序幕。如果没有重大突破的话,验证将成为未来集成电路设计工业流程中的重大障碍,被称为“验证危机”。历史上由于对集成电路验证上的不完善而造成的重大损失甚至悲剧的案例有很多。1994年,Intel奔腾处理器被发现在执行某个特定的浮点运算时出现错误。对此,Intel付出4.75亿美元的巨额代价回收有缺陷的奔腾处理器。1996年6月4日,欧洲航天局研制的阿里亚娜五型火箭在发射升空后不到40秒爆炸。事后调查发现,错误发生于电路芯片中当一个很大的64位浮点数转换为16位带符号整数时出现异常。细微错误,使得十年的努力毁于一旦[2]。从以上的两个案例可以看到,设计验证在当前集成电路产业中非常重要,设计实现高效而完备的设计验证方法和工具对集成电路产业的发展有着

关键意义。

当前形式验证技术由于其不需要产生测试激励,具有数学完备性和逻辑推理严格的特点,能够证明一个系统是否实现了设计者的意图。相对于传统的模拟验证耗时长,验证不完备的缺点,形式验证能够提供高效而完备的验证解决方案,因而很有希望成为下一代集成电路验证手段,具有重大的研究意义和应用价值,现已成为国际上研究热点。其中可满足性问题算法(The Satisfiability Problem, 简称SAT)是目前形式验证的核心引擎,也是本文的研究重点。可满足性(SAT)问题属于命题逻辑的范畴,是约束满足问题的一个分支。在集成电路形式验证中,SAT问题求解器已被广泛应用于定界模型检验,无界模型检验和等价性检验,ATPG检验,微处理器验证,设计错误诊断。除此之外,SAT问题还应用于其它领域。例如:FPGA的布局,逻辑优化等。图1-1给出了SAT问题在EDA领域的广泛应用。

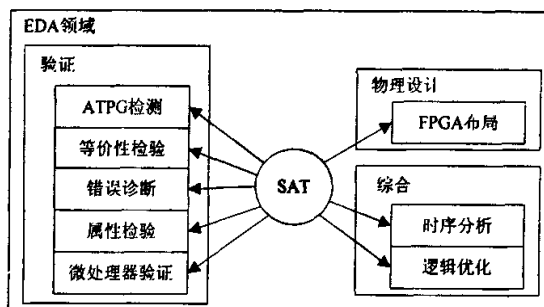


图 1-1 SAT 在 EDA 领域中的应用

可以看到,可满足性(SAT)问题研究是许多重大问题的基础,有着非常广泛的应用价值。

## 1.2 SAT 问题研究现状和挑战

近十多年来,可满足性问题研究逐渐升温,已成为了国际国内的研究热点,取得了一批相当重要的理论和实践成果,应该说当前的SAT问题研究比十多年前已取得了很大的突破,并直接或间接地推动了其他相关领域(比如形式验证,人工智能等领域)的发展。

当前的SAT问题的算法主要包含两类:局部搜索算法和回溯搜索算法。近十

多年来,国际上已提出了各种不同的局部搜索算法和回溯搜索算法,使得SAT解决器解决不同领域中的SAT问题的能力不断增强,能解决的问题的规模不断扩大。其中局部搜索算法显示出对于随机的SAT问题特别有用,而回溯搜索算法则被用来解决大规模实际应用领域中的SAT问题。事实上,国际上已提出了一大批采用回溯搜索算法的高效的SAT问题求解器,其中绝大多数提出来的回溯搜索算法是对原始的DPLL回溯搜索算法[3, 4]的改进算法。这些改进措施包括:新的变量决策策略,新的搜索空间剪除技术,新的推理和回溯技术以及新的更快的算法实现方案和数据结构等。当前水平的SAT问题求解器已能够轻松解决以前传统SAT问题求解器完全无法解决的可满足性问题。表1-1部分展示了SAT解决器在最近十多年里的发展经历。

表 1-1 最近的十年中的 SAT 解决器计算能力的演变

SAT 问题实例	Posit@1994	Grasp@1996	Sato@1998	Chaff@2001	Siege@2003
ssa2670-136	40.66 s	1.2 s	0.95 s	0.02 s	0.01 s
bf1355-638	1805.21 s	0.11 s	0.04 s	0.01 s	0.01 s
pret150_25	>7200 s	0.21 s	0.09 s	0.01 s	0.01 s
dubois100	>7200 s	11.85 s	0.08 s	0.01 s	0.01 s
aim200-2_0-no-1	>7200 s	0.01 s	0 s	0 s	0.01 s
2dlx_cc_mc_ex_bp_f2_bug005	>7200 s	>7200 s	>7200 s	2.9 s	0.22 s
c6288	>7200 s	>7200 s	>7200 s	>7200 s	6676.17 s

尽管当前的SAT问题求解器已取得了相当重要的进步,但是研究的脚步不会停止,我们还可以提出一些值得研究的问题。比如,是否存在新的更高效的SAT问题处理技术可以集成到DPLL算法框架内;是否可以找到除局部搜索,回溯搜索之外的其他SAT算法来更有效地解决SAT问题;是否能提出更好的SAT改进算法和实现方案。

### 1.3 本文的主要贡献

本文的工作主要体现为两个方面,一方面针对当前可满足性问题的规模越来越大,而由于SAT问题的NP特性,SAT解决器的求解时间因而也越来越长,特别是对于某些大规模SAT问题,连当前最先进的SAT解决器也无法在可以接受的时间内找到问题的解,限制了SAT问题在许多领域中的应用。为进一步增强求解大规模SAT问题的能力,本文从可满足性问题预处理入手进行了研究,设计并实现了一种新的SAT预处理器,以充分简化了原始问题的规模,减少了SAT问题总体

解决时间。另一方面，在吸收掌握已有的SAT研究成果和理论算法的基础上，提出了一种采用新的决策策略的SAT搜索算法，并设计并实现了基于该算法的SAT解决器。

本文工作的主要创新点如下：

- ◆ 提出一种基于对称蕴含推理的可满足性预处理算法，并与等价文字化简技术相结合，设计并实现了一个高效的预处理器Snowball。该预处理算法比以往的预处理算法更简洁，容易实现，并且理论和实验均表明该算法能发现更深层次的文字间的蕴含关系，因而该算法能更大地减小可满足性问题的规模。
- ◆ 在DPLL算法框架内提出了双变量决策点策略，革新了以往的单变量决策策略，理论分析表明，运用了双变量决策点策略的DPLL算法能够将决策深度减少一半，因而能大大减少SAT搜索树的规模，加速SAT问题的求解。在集成了现有的SAT启发式算法和技术的基础上，本文首次实现了采用双变量决策点的SAT解决器。实验结果验证了算法的正确性。

## 1.4 论文组织结构

本文的组织如下，第二章简要介绍和说明了有关可满足性问题的基础知识和定义。在第三章中，回顾了可满足性问题算法的发展，重点介绍了历史上基于DPLL算法框架的SAT问题研究的主要成果。第四章和第五章详细介绍了本文所提出来的新算法和技术，其中第四章介绍了对称蕴含的预处理技术，而第五章则详细阐述了双变量决策DPLL算法。在最后的第六章中，对全文做出了总结并对今后的工作进行了展望。



## 第2章 基础知识

### 2.1 布尔逻辑代数

布尔代数是本文工作的基础。在本节中将简单介绍一下与本文工作相关的一些布尔代数中的内容。布尔代数在数理逻辑中是属于命题逻辑（布尔逻辑），或称为0-1阶逻辑。更为抽象的逻辑包括1-阶逻辑和高阶逻辑。电路的形式验证理论中除了使用布尔逻辑外，也包括后两种抽象逻辑。这取决于形式验证应用在设计中的哪个抽象层次上。而本文所要阐述的可满足性问题以及电路逻辑层上的等价性验证均是以布尔代数理论为基础的[5]。

简单来说，布尔代数系统是由布尔变量和逻辑运算符（布尔关系）构成的代数系统。布尔变量的取值只能是离散的整数值0（假）或1（真）。离散量集合{0, 1}又称为布尔集合。基本的逻辑运算符有三个：“非”（用“ $\sim$ ”表示）、“与”（用“ $\cdot$ ”表示或者省略）和“或”（用“ $+$ ”表示）。其中“非”是单目运算符，“与”和“或”是双目运算符。组合的常用逻辑运算符还有：蕴含、恒等和异或。等价的逻辑描述如下：

表 2-1 组合逻辑运算符

运算名称	运算符	等价逻辑运算
蕴含	$a \rightarrow b$	$\sim a + b$
恒等	$a \equiv b$	$(a \cdot b) + (\sim a \cdot \sim b)$
异或	$a \oplus b$	$(a \cdot \sim b) + (\sim a \cdot b)$

定义在布尔集合上的映射又称为布尔函数。设B是一个布尔集合 $B=\{0, 1\}$ ，那么一个由n个布尔变量构成的布尔函数是一个映射： $B^n \rightarrow B$ （注意这里的有向箭头表示映射关系而非蕴含关系）。

### 2.2 计算复杂度

许多游戏或问题之所以有趣是由于它们的难度，即解决或完成任务需要人的聪明才智。一般而言在计算机科学中，一个问题的难度常可以用数学的形式表示出来，即计算复杂度。

一个处理过程或算法的复杂度是对该处理过程或算法所表现出来的困难程

度的度量。对于算法复杂度的研究,或者说复杂度理论的研究对象是在解决给定问题的计算过程中所需要的计算资源。最常见的计算资源是时间(需要多少步骤来解决问题)和空间(需要多大的存储空间),其他的计算资源还有如在并行处理过程中所需的处理器的个数等等。

表 2-2 时间复杂度比较[6]

n	$f(n)=n$	$f(n)=n^2$	$f(n)=2^n$	$f(n)=n!$
10	0.01us	0.1us	1us	3.63ms
20	0.02us	0.4us	1ms	77.1years
30	0.03us	0.9us	1s	$8.4 \times 10^{15}$ years
40	0.04us	1.6us	18.3min	
50	0.05us	2.5us	13days	
100	0.1us	10us	$4 \times 10^{13}$ years	
1,000	1.00us	1ms		

对于一个问题的时间复杂度是指为当使用最高效的算法解决该问题时所需要的步骤数,它是通常是问题规模的函数。表2-2即展示了不同的时间复杂度函数随着问题规模的生长的变化趋势。为直观地认识时间复杂度,可以考虑下面一个例子,如一个规模为 $n$ 的计算问题,需要计算机运行 $n^2$ 步才能解决,在这种情况下我们称该问题的时间复杂度为 $n^2$ 。当然精确的计算步数是依赖于所用的机器和程序语言的,为避免这个问题,通常使用大“O”表示法来表示所有在 $n^2$ 的数量级上的复杂度,  $O(n^2)$ 。大“O”表示法可以使我们抛开具体的计算细节,对不同的计算问题可以进行标准化的比较和研究。一般我们称那些时间复杂度为 $O(n^x)$ 的问题( $n$ 为问题的规模,  $x$ 为一常数)为有多项式时间复杂度的问题。这类问题通常都有高效的算法,因而能够求解的问题规模可以很大。

大多数复杂度理论研究的是判定性问题。所谓判定性问题是这样一类问题,它的解通常是要回答“是”或“否”。比如质数判定问题,给定一个整数,要求回答它是否为质数。判定性问题是基本并且相当重要的一类问题,这是因为许多其他的问题都可以简化为判定性问题。

判定性问题按计算复杂度的不同可以被分为几类,其中最著名的是P问题和NP问题。所谓的P问题是那些在确定型图灵机上有多项式时间复杂度的问题。这类问题都有高效的算法,即使是在最坏的情况下也能够进行高效的求解。而NP问题是指目前在确定型图灵机上经过多项式时间无法得到确定性解的一类问题[7]。P问题是NP问题的一个子集,判定是否P问题集合就是NP问题集合,即 $NP=P$ ,

或换句话说, 是否所有NP问题在确定型图灵机上都存在有多项式时间复杂度算法, 是当前计算机理论科学界最重要的一个开放式的问题。国际上甚至有研究机构专门悬赏一百万美元奖金征求对这个问题的解答。NP问题又可分为“NP难”(NP-hard)和“NP完全”(NP-complete)问题。所谓“难”和“完全”问题的概念可以用问题集合来说明, 给定问题集合X和问题集合Y, 如果Y中的每一个问题都能在多项式时间复杂度内转化为X中的某个问题而得到相同的解, 则我们称集合X是另一个问题集合Y的“难”问题集合。而如果集合X既是问题集合Y的“难”问题又是Y的子集, 则我们称X为集合Y的“完全”问题集。在计算机科学里最重要的一类完全问题即是“NP完全”问题。而可满足性(SAT)问题是第一个被证明具有NP完全性的问题[7]。从NP完全问题的定义可以看出, 如果能够找到对可满足性(SAT)问题的有效解法(希望是多项式时间复杂度的算法), 那么其他所有NP问题均能够迎刃而解, 因此SAT问题研究的意义和价值是巨大的。

## 2.3 SAT 问题的表示和定义

SAT问题属于布尔逻辑的范畴。而对于布尔逻辑的表示方法有很多种, 在介绍可满足性问题之前, 先对一些基本概念和定义作简单介绍。

**变量和文字:** 这里的所指的变量均指的是布尔逻辑变量。对于一个变量  $x$ , 其变量取值设为  $v(x) \in \{0, 1, X\}$ , 其中“X”代表变量  $x$  的值不确定, 或者说变量  $x$  处于未赋值状态。一个变量  $x$  有两个文字, 分别是正相位文字(可表示为  $x$ )和其反相位文字(可表示为  $\sim x$ )。

**合取范式(CNF, Conjunctive Normal Form):** 绝大多数可满足性(SAT)问题均是以合取范式的形式给出的。所谓合取范式(CNF)是指, 对于给定一个  $n$  维布尔变量集合  $V = B^n$ , CNF 表达式由众多“子句”构成, 而每个子句由集合  $V$  中的若干变量的“文字”构成, 其形式如右式:  $F = \prod_{i=1}^m c_i$ ,  $c_i = \sum l_k$

举例说明, 比如, 下式表示了一个 CNF 表达式。

$$F = (a + b + \sim c)(\sim b + e)(a + \sim f + c)$$

其中, 变量集合  $V = \{a, b, c, e, f\}$ , 该 CNF 表达式由三个子句组成, 分别是  $c_1 = (a + b + \sim c)$ ,  $c_2 = (\sim b + e)$  和  $c_3 = (a + \sim f + c)$ 。

可满足性问题(SAT)的描述十分简单, 可以用 CNF 表达式如下描述。

**可满足性问题定义:** 给定一个 CNF 表达式  $F$ , 问是否存在在变量集合  $V$  上的一组逻辑赋值, 使得表达式  $F$  的值为 1, 即使  $F$  满足。若存在这样的一组赋值, 则



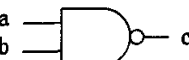
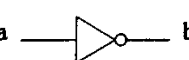



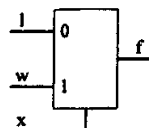
称原问题  $F$  可满足，否则称为原问题  $F$  不可满足。

正是由于可满足性问题形式简单而意义重大的特性，其获得了国际上广泛地研究。

## 2.4 电路验证与 SAT 问题转换

VLSI前端设计的最终形式就是逻辑电路。逻辑电路又分成组合电路，同步时序电路和异步时序电路。通常在形式验证中我们关心的是组合电路以及同步时序电路中组合部分向SAT问题的转化。组合逻辑电路通常定义为由一些基本逻辑门及其连接关系构成的多输入多输出的有向无环图。基本逻辑门包括：非门、与门和或门。通常还有一些常用的逻辑单元如异或门、选择器等等。基本逻辑门、常用逻辑门与CNF表达式存在一一对应的转换关系，见表2-3：

表 2-3 逻辑门与 CNF 的转换关系[5]

逻辑门	电路图	CNF 表达式
与门		$(\bar{c} + a)(\bar{c} + b)(c + \bar{a} + \bar{b})$
或门		$(c + \bar{a})(c + \bar{b})(\bar{c} + a + b)$
与非门		$(c + a)(c + b)(\bar{c} + \bar{a} + \bar{b})$
非门		$(a + b)(\bar{a} + \bar{b})$
或非门		$(\bar{c} + \bar{a})(\bar{c} + \bar{b})(c + a + b)$
同或门		$(\bar{c} + \bar{a} + b)(\bar{c} + a + \bar{b})(c + \bar{a} + \bar{b})(c + a + b)$
异或门		$(c + \bar{a} + b)(c + a + \bar{b})(\bar{c} + \bar{a} + \bar{b})(\bar{c} + a + b)$
选择器		$(x + l + \bar{l})(x + \bar{l} + f)(\bar{x} + w + \bar{l})(\bar{x} + \bar{w} + f)$

根据表2-3将一个组合电路转换得到的CNF表达式，它的可满足性可以描述为：满足该CNF表达式的每一组真值赋值对应一个电路输入输出关系。如果随意加入某个节点  $x$  的一元子句，如  $(x)$ ，则对应的CNF表达式的可满足性可以描述为，是否存在一个真值赋值（一个输入向量）使节点  $x$  的值恒为1。据此，通常组合电

路的等价问题可以构造如下的所谓乘积电路 (product machine, 又称为miter电路), 如图2-1所示:

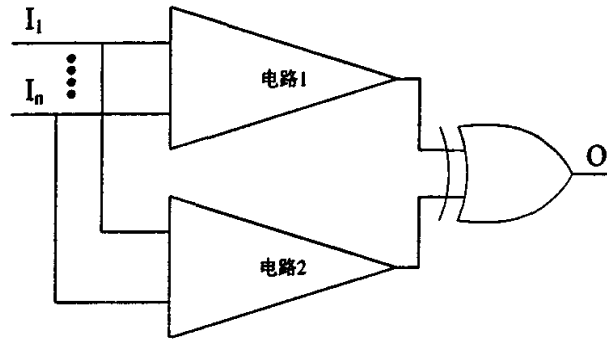


图 2-1 组合电路的乘积电路[5]

这样, 将图2-1的电路根据表2-3转化成CNF表达式并加入一元子句( $\sim O$ ), 则得到的CNF表达式的可满足性问题与电路的等价问题对应关系为: 如果CNF表达式不可满足, 则两个电路等价, 反之, 两个电路不等价。通常我们把这样由电路转换得到的可满足性问题又称为电路可满足性问题 (Circuit SAT)。

## 第3章 可满足性问题 DPLL 算法的发展

### 3.1 历史回顾

可满足性算法历来就有很多研究,综述性的文献就如[8]。通常,可满足性问题的算法可以分成两大类:不完全算法和完全算法。不完全算法的特点是算法终止时若能找到一组可满足的真值赋值,则可以断定问题是可满足的,但是若不能找到这样的真值赋值,则不能判断问题是否可满足。不完全算法的思想主要是源自最优化算法。它们通常以需要满足的子句数为优化目标,优化算法的目的是使最多子句被满足。算法从最初的猜测解,按一定的搜索策略希望逐步逼近使问题满足的最优解。最常见的是基于局部搜索思想的算法,也是较早的不完全算法[9]。以后又发展了多种算法包括基于贪婪搜索算法的GSAT[10],有基于拉格朗日松弛算法的DLM[11],有基于对噪声模型分析研究后的WalkSAT[12]等等。不完全算法通常对于某些事实上是可满足的问题有很好的表现,能够快速找到可满足的解。但是从大部分的实验比较,特别是国际互联网上公布的SAT竞赛的成绩看,目前为止最好的不完全算法如DLM,也没有在实际的SAT实例前有较好的排名。只是在随机的SAT实例上有好的成绩[13]。

而到目前为止, SAT算法还是以完全算法的研究为主,特别是一类被称为DPLL (Davis Putnam Logemann and Loveland) 的算法尤其突出,被认为是目前为止SAT问题的标准解决方案。DPLL算法最早可以追溯到1960年,由Davis和Putnam提出的DP算法[3]。到1962年, Davis, Logemann和Loveland又提出了基于深度优先搜索思想的DLL算法[4]。这两种算法所包含的思想到目前为止都一直被沿用,只是在采用的具体技术上又有改进和增加。因此统称这类算法为DPLL算法。虽然提出很早,但是最早的DP算法的空间复杂性也是指数的,而DLL的实际效率也很低,它们能解决的问题规模在10个变量的数量级上,因此很难应用于实际问题。在这之后,很多其他的技术被用来解决SAT问题,如上文提到的局部搜索算法,此外还有非主流研究的如BDD图[14],基于广度搜索的[15,16]等。DPLL算法本身的发展受到一定限制。直到1995年GRASP[17]的提出,系统总结了DPLL算法的基本流程,才使DPLL算法又重新成为研究的重点,而且被逐渐应用于实际问题的解决。1995年之后的DPLL算法的发展可以见图3-1。

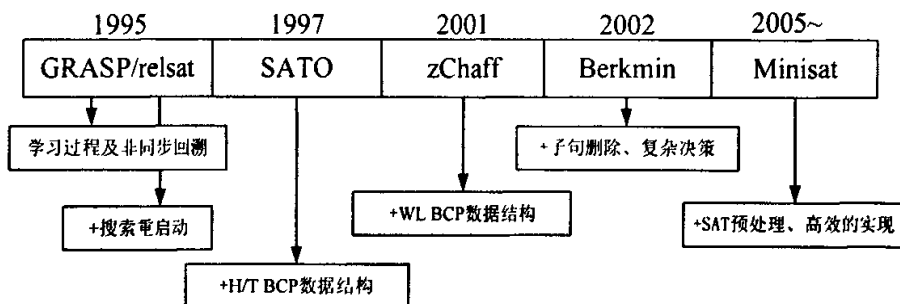


图 3-1 DPLL 算法近年来的发展

除了GRASP, 还有relsat[18], SATO[19], zChaff[20], Berkmin[21], Minisat[22]。GRASP最大的贡献在于系统的介绍了学习过程。学习过程虽然从原理上和DP算法没有多大区别, 但是实际上在配合DLL算法时候的剪除搜索空间上的效果非常出色, 使SAT算法有了一个质的飞跃。到2001年, Chaff和zChaff的提出更将SAT算法的实际运行速度提高了一个数量级。他们针对BCP(布尔约束传导)过程提出了懒散式的数据结构, 极大提高了BCP蕴含推理的效率。而且他们对于学习过程也做了很好的分析, 找出了到目前为止最为有效的学习方式。这之后虽然也有很多SAT算法公布, 但是大多数都是以zChaff为蓝本。如BerkMin, Limmat等。到2005的SAT竞赛, 结合了预处理技术的SAT求解器Minisat开始展露头脚, 成为了当年SAT竞赛的最大赢家。高效的SAT预处理技术以及集成了以往各种启发式技术的高效算法实现手段是Minisat求解器赢得近几年来SAT竞赛冠军的主要原因。下面我们将分节介绍各个主要的SAT处理和解决技术。

### 3.2 SAT 预处理方法

当前国际上已提出来了好几种可满足性问题预处理方法[23]。我们知道预处理可满足性问题存在化简程度与时间花费的折衷问题, 根据不同的预处理效果以及预处理时间我们可以把当前的这些预处理方法划分成两类。一类可称之为轻量级方法。该类型预处理方法花费的时间很短但其对原问题的化简程度也较浅。2Simplify[24]和NiVER[25]以及SatELite[26]就是属于该类型的两种可满足性问题预处理器。NiVER预处理器通过分析字句分布来进行变量消除。而2Simplify则运用了蕴涵图并结合了一些化简规则如有限的超级分解技术和等价化简[27]技术来化简原问题的规模。尽管这类预处理器使用了各自不同的化简技术, 它们均只

取得了有限的对原问题的化简程度，因此在降低SAT问题总体解决时间方面它们效果并不明显。而另一类预处理方法在预处理时可能花费较长的时间，然而相对于那些轻量级的预处理方法他们化简的程度更深，能更大程度地减少可满足性问题的规模。我们称之为重量级的预处理方法。预处理器Hypre[28]就是该类型预处理方法的典型代表。与2Simplify类似，Hypre同样运用了文字蕴涵图以及等价化简技术对原SAT问题进行化简。与2Simplify不同的是Hypre使用了更加完整和高效的超级分解技术，因此Hypre与2Simplify相比能取得更好的化简效果。这类重量级处理方法的最主要缺点在于它们往往需要较长的化简时间，对于一些大型的SAT问题这些方法就显得并不那么有效。这个问题可以通过下面两种途径得以改进，一是改进和优化预处理器的实现方式，或设计更新颖的算法来使得SAT预处理器越来越高效。另一种方法是给这些重量级的预处理器设置动态的超时停止功能，比如[29]即在程序预处理过程中监测程序化简效率 $p$ ，如果 $p$ 小于某个阈值或者预处理时间超过某个时间限制还没有结束则终止预处理程序，而保留目前所能得到的化简结果送给接下来的SAT解决器进行处理。当然后一种方法会牺牲一部分预处理器的化简效果，然而它限制了预处理时间，结合目前高效的SAT解决器，总体解决SAT问题的时间花费将会减少。接下来逐一介绍各个预处理器的算法和所用到的技术。

### 3.2.1 NiVER 预处理器

NiVER预处理器的全称为：Non Increasing Variable Elimination Resolution，正如其名字所言，它采用的预处理技术是非增长的变量消解算法[25]。变量消解算法（Variable Elimination Resolution）其实是一个非常传统的算法，最早应用于SAT解决器当中，不过由于其常常会带来子句数的增加，并且解决效率不高，后来基于这种变量消解技术的SAT解决算法就被DPLL算法所取代了。其变量消解的基本原理可举例说明如下：

比如假定原CNF表达式中所有包含文字“ $x$ ”的子句集合 $X_0$ 为：

$$(x + a + b)(x + b + c)(x + b + d)$$

而所有包含“ $\sim x$ ”的子句集合 $X_1$ 为：



$$(\sim x + c + y)(\sim x + c + z)(\sim x + c + w)$$

那么针对变量“x”进行变量消解后新生成的子句集合N如下所示：

$$(a + b + c + y)(a + b + c + z)(a + b + c + w)(b + c + y)(b + c + z)(b + c + w)$$

$$(b + c + d + y)(b + c + d + z)(b + c + d + w)$$

所谓的变量消解算法就是用新生成的子句集合“N”取代集合“ $X_0 \cup X_1$ ”以达到消除变量“x”的目的。

在NiVER预处理器中用到的算法给变量消解操作加了一个限制条件，即检验变量消解之后子句数量是否会增加，只有当变量消解后子句数目减少或保持不变时才接受变量消解的结果，否则对原CNF表达式不做改动。在NiVER中所用到的算法属于轻量级的预处理算法，除了进行非增长的变量消解技术处理SAT问题之外，其没有再应用其他的预处理技术，因而其预处理时间花费很少，但SAT简化效果相对有限。

### 3.2.2 SatELite 预处理器

SatELite预处理器是在NiVER预处理算法的基础上改进而成的另一种预处理器[26]。其与SAT解决器Minisat的组合解决策略曾获得过2004年国际SAT解决器竞赛的冠军。SatELite预处理算法主要也还是应用NiVER中提出的非增长的变量消解技术来化简冗余变量。在这基础之上，SatELite还采用了三种进一步的化简措施：子句包含消除，自包含消除和变量取代消除。

所谓子句包含消除可以用一个例子来说明。对于CNF中的两个子句 $(a + b + c)$ 和 $(a + c)$ ，可以看到子句 $(a + c)$ 对于解空间的约束比子句 $(a + b + c)$ 的约束要强，因此在子句 $(a + c)$ 存在的条件下子句 $(a + b + c)$ 为冗余子句，可以消去，这并不改变原问题的可满足性质。以上的子句消除技术即称为子句包含消除。

自包含消除技术是指对于CNF中的某些子句如C，由于有其他相关的二元子句存在，子句C中的某个文字可以从C中除去而不改变SAT问题的可满足性质。举例说明， $C_1 = (x + a + b)$ ， $C_2 = (\sim x + a)$ ，子句 $C_1$ 在 $C_2$ 存在的情况下可以化简成 $C_1 = (a + b)$ 这样可以将三元子句化简成二元子句。

最后的变量取代消除技术利用了逻辑门取代关系来化简原SAT问题。举例说明,有如下三个子句:  $(x + \sim a + \sim b)(\sim x + a)(\sim x + b)$ , 从中我们可以发现与门逻辑关系式:  $x = (a \cdot b)$ , 所谓变量取代消除技术即是将变量“x”用“a·b”来取代,之后再应用逻辑关系的分配率而得到相应的CNF表达式。比如,原子句  $(x + y + z)$ , 在用“a·b”取代“x”后,得到  $(a \cdot b + y + z)$ , 通过分配率转化成两个子句:  $(a + y + z)(b + y + z)$ 。可以看到变量取代消除技术可以减少变量个数,消除中间变量简化原问题。

### 3.2.3 2Simplify 预处理器

2Simplify预处理器主要采用的是二元子句蕴含图的方法来化简原SAT问题[24]。其具体的处理流程如下:首先利用CNF表达式中的二元子句构建蕴含图,每个二元子句可对应两个蕴含关系,比如  $(a + b)$  二元子句对应有两个蕴含关系“ $\sim a \rightarrow b$ ”和“ $\sim b \rightarrow a$ ”因此在蕴含图中一个子句对应两条有向边。构造完蕴含图后检查图中是否含有闭合环路,若存在闭合环路则所有在闭合环路中的文字存在等价关系,可以用一个文字代替。这一步可称为等价化简。当完成等价化简之后蕴含图成为一个有向无环图。然后再利用蕴含关系的传递性,完成蕴含图的蕴含闭包。即在蕴含图中将所有的文字节点与其孙子节点建立直接的蕴含关系。在完成蕴含闭包的过程当中可能会发现失败性文字或强制性文字,这时立即进行一元推理对原问题进行化简。当完成蕴含闭包后,在保持蕴含图连接关系不变的情况下对蕴含图进行化简,最后由简化后的蕴含图生成简化后的CNF表达式。

### 3.2.4 Hypre 预处理器

与2Simplify类似,Hypre预处理器同样运用了文字蕴涵图以及等价化简技术对原SAT问题进行化简[28]。与2Simplify不同的是Hypre使用了更加完整和高效的超级分解技术,因此Hypre与2Simplify相比能取得更好的化简效果。所谓的超级分解是一种对于两个以上的子句进行的一种分解操作。比如对于一个n元子句  $(n > 2)$ :  $(x_1 + x_2 + \dots + x_n)$  和  $n-1$  个二元子句  $(\sim x_i + y)$  (其中  $i=1, \dots, n-1$ ), 通过超级分解其能产生一个新的二元子句  $(y + x_n)$ 。超级分解操作等价于一系列普通的分解操作的

组合,并且其又能够避免生成许多由普通分解生成的中间态子句,因此它减少了子句的数目。Hypre预处理器属于重量级的预处理器,其在预处理时可能花费较长的时间,然而相对于那些轻量级的预处理方法它化简的程度更深,能更大程度地减少可满足性问题的规模,甚至直接解决某些SAT问题。不过,这类重量级处理方法的最主要缺点在于它们往往需要较长的化简时间,对于一些大型的SAT问题这些方法就显得并不那么有效。这个问题可以通过下面两种途径得以改进,一是改进和优化预处理器的实现方式,或设计更新颖的算法来使得SAT预处理器越来越高效。另一种方法是给这些重量级的预处理器设置动态的超时停止功能,比如文献[29]中所提到的HypreBinFast预处理器。

### 3.3 搜索的启发式策略

基本的DPLL算法主要采用的是深度优先搜索(DFS)的思想,即将搜索空间表示成由各个变量所构成的搜索树,以传统的深度优先搜索策略历遍所有的通路,找到能使问题满足的真值赋值。从对DPLL算法的简单分析中不难看出,DPLL算法的时间复杂度在最坏情况下是随着变量个数的增加成指数级增长的,这很容易让人对于目前主流的DPLL算法的发展前景产生悲观的态度,然而可喜的是从DPLL算法诞生到现在的四十多年里,研究人员并没有放弃对于SAT算法的研究,反之不断地进行探索和努力,提出了一系列有效的启发式算法和技术,使得原先的DPLL可满足性算法从原来的只能解决几十个变量发展到现在能够解决几十万甚至上百万个变量的水平,基于DPLL算法框架的SAT解决器的处理速度实现了数量级上的提高,特别是近十年来,SAT算法有了飞跃式的发展,逐渐引起了计算机界各个领域人士的重视,研究成果也逐渐从纯学术研究转到了各个应用领域,许多以SAT算法为基础的新的解决方案如雨后春笋般的发展起来,不仅促进了SAT算法的发展,同时更推进了其他领域的进步。本节将重点介绍在SAT算法的发展过程当中有着里程碑意义的研究成果和技术,与读者共同探讨这些推动SAT算法从实验室研究成功走向广泛应用的启发式策略。

### 3.3.1 布尔约束推导

布尔约束推导 (BCP, Boolean Constraint Propagation) 是最早针对 DPLL 算法提出的优化过程, 其又可称为一元推导 (Unit Propagation) [3,4]。其工作原理如下, 随着搜索过程的深入, 变量不断被赋值, 子句中未赋值的变量数的减少, 对于那些含有已被赋值为“1”的变量的子句, 根据布尔逻辑该子句已被满足, 可从子句库中删除, 而对于剩余的子句如果在子句中除了一个文字未被赋值外, 其他的文字均已被赋值为“0”, 则该子句即成为一元子句。举例说明, 对于子句  $(\sim a + \sim b + c)$ , 当在  $a=1, b=1$  的条件下时, 该子句即成为一元子句, 容易看出为使该子句满足, 子句中的文字“c”必须赋值为“1”, 这就是所谓一元子句规则。所谓 BCP 就是不断应用一元子句规则, 直到问题中不再存在一元子句, 从而无法进一步运用一元子句规则进行进一步的化简为止。BCP 过程是最简单的优化过程, 然而其也是到目前为止最有效的一种正向推理技术, 对于 DPLL 算法性能的提高有着决定性的作用。BCP 过程之所以重要, 是因为在实际的可满足性问题中, 平均每个子句所包含的文字数目是有限且通常很小。因此在搜索过程中一元子句出现的次数会很频繁。实验统计表明, 一般基于 DPLL 的可满足性问题算法的运行总时间有 80%—90% 耗费在 BCP 过程中。在本章的后面的章节还会介绍具体针对 BCP 过程的高效的实现方法, 可以看到提高布尔约束推导 BCP 的效率能极大地提高 DPLL 算法的效率。

### 3.3.2 变量决策策略

变量决策策略对于 DPLL 算法来说非常重要, 也历来是 SAT 算法研究者感兴趣的一个研究重点。我们可以通过下面一个 SAT 实例来观察不同的变量决策策略对于 DPLL 算法的影响。

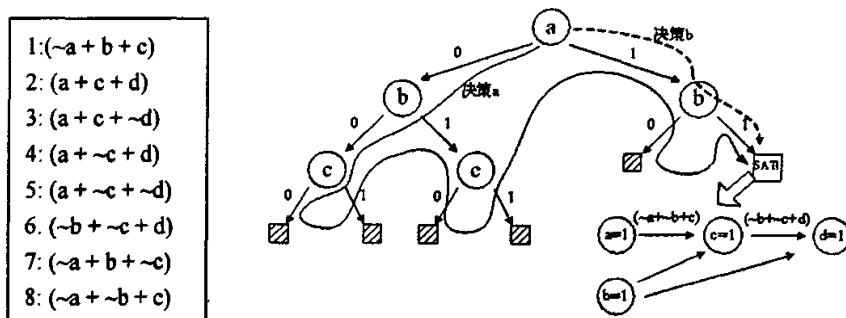


图 3-2 两种不同的决策策略效果比较图

以上的SAT实例是一个可满足的实例，即存在一组赋值使原问题满足。由图3-2可以看到决策a一开始选择了从“ $a=0$ ”的方向上搜索寻找，经历了许多曲折才最终找到问题的解，其中走了许多弯路，不是一个好的决策。而决策b则选择了一条捷径，只经历了两个变量就发现了问题的解，是一个很好的决策。由于DPLL算法本质上就是在由各决策点所构成的二叉树上进行深度优先搜索的过程，因此决策策略的不同直接决定了搜索的二叉树规模的大小，因此也直接影响了DPLL算法的效率。一个好的决策策略应该是能使搜索引擎能尽快的找到使问题满足的解，或能尽早地发现当前赋值的冲突，从而避免了搜索引擎继续在错误的路径上浪费时间。当前国际上对于决策策略上的研究主要是针对决策变量和决策相位的选择提出各种启发式算法。这里仅对在众多决策策略中最具有里程碑性质的两个计算方法做一介绍。

一是最短子句出现频率最大 (MOM, Maximum Occurrence in Minimization) 方法，也称为JW (Jeroslow-Wang) 算法[30]。其核心是计算所谓文字的J值。J值的计算公式如下：

$$J(l) = \sum_{\text{for all } C_i \text{ containing } l} 2^{-n_i}$$

其中 $n_i$ 是包含文字 $l$ 的子句 $C_i$ 中的文字数目。可见，文字所在子句的长度越长，对J值的贡献越小。从概念上分析，由于子句满足只需要其中某一个文字满足即可，一个子句文字数目越多，则可以使子句满足的“可选”文字数目越多，就越比较“容易”得到满足。相反，文字数目越少，越不容易满足。因此，在这些文字数少的子句中挑选出现频率大的文字先满足，目的就是优先满足不容易满足的子句，使搜索过程能尽快找到可满足解。许多所谓“高精度”的计算方式都是以

JW算法为基础的。JW算法在许多可满足性问题的解决器中有应用，且一度是主流的决策策略。但是，计算“精度”越高，带来的计算代价就越大。停留在决策过程中的时间也就越长，这无疑也是一个“精度”和速度的折中。就当前流行的趋势来讲，尽管MOM方法决策精度较高，但由于其决策的时间花费较长，成本较高，现在更倾向于采用计算量较低但还保留有一定精度的启发式决策算法，如下面将要介绍的独立下降式变量和（VSIDS）算法。

在2001年，文献[20]提出了一个全新的决策算法策略。之所以称它也具有里程碑意义，是因为该决策策略不仅仅从决策的角度出发选择“适合”的决策变量，更主要的是，它十分优秀的结合了DPLL算法的其他过程，如学习过程和BCP过程，使DPLL算法的整体表现有了提高，并且其计算代价十分低廉。这就是独立下降式变量和（VSIDS, Variable State Independent Decaying Sum）方法。VSIDS对每个文字设立一个计数器。当文字在某个子句中出现，包括原始问题中的子句和学习子句，该文字对应的计数器就加1。在一定的决策次数后，所有文字的计数器均除以一个常数（通常在2—4左右）。它并不需要如计算J值一样在每次选择变量时扫描所有子句，只需要根据当前文字的VSIDS值的排序选择一个具有最大VSIDS值的文字即可。VSIDS本身的想法也很简单，即满足最多的子句，使问题中还未满足的子句数目尽量少。可是不仅仅如此，VSIDS本身的重要在于它在一定的决策次数后所做的整体计数器下降的策略，如果没有这个策略，根据实验，VSIDS很容易失效。DPLL算法在加入了学习过程后，随着搜索的进行，学习得到的子句数目会远远大于原始问题中的子句数目。但是学习子句在搜索过程中的“贡献”却不是一直保持着。这是因为，当搜索从一个空间转向另一个空间时，原本学习到的子句也许没有多大用处，即学习子句有大部分会已经满足。已经满足的子句对于搜索的贡献是0，它不可能指导搜索过程在剩余还未得到满足的子句中寻找可满足解。然而从VSIDS的计算方法中可以知道，计数器本身的值并不考虑子句是否已经得到满足，它并没有在某个子句满足后将其对应的文字计数器减1。同时，下一个小结对数据结构的分析中可以知道，如今的BCP过程同样不能发现所有已经满足的子句。于是，计数器的“误差”就会随着搜索的进行很快增大，失去指导搜索的作用。VSIDS算法在一定的决策次数后做整体的数值下降，目的就是增加了最新学习到的子句对搜索的贡献。已学习到的子句对计数器的

贡献就会随着搜索过程的进行“指数”下降，从而也就避免了误差的累积。当前主流的可满足性问题算法的决策策略大多都是从VSIDS策略中变化而来。

除了以上提到的两种决策启发式算法，国内外还提出了许多其他的决策算法，比如骨干点搜索算法[31]，在决策过程当中选取骨干点变量作为决策点已减小搜索树的规模；根据计算卡诺图覆盖的方法确定决策相位[32]等。不断地开发新的更高效的决策启发式算法还会继续是国际上可满足性问题研究的热点。

### 3.3.3 子句学习和非同步回溯

在DPLL算法的发展中，有许多贡献是对旧有的DPLL算法中三大模块：决策，推理，回溯的改进，而接下来的将要介绍的研究成果是在DPLL算法框架内引入了一个新的模块——子句学习和非同步回溯[17]。在这个新的模块被引入DPLL算法框架之前，可以说有已有的贡献只是使DPLL算法的效率有量变上的提高，基于DPLL算法的SAT解决器的性能进展缓慢。而自从引入了子句学习和非同步回溯模块之后，基于DPLL算法的SAT解决器的性能有了大幅度的提升，并使SAT解决器拥有了解决大规模可满足性问题的能力，可以说而基于冲突的学习过程及其相伴随的非同步回溯是使DPLL算法能解决实际问题的关键。

为介绍子句学习和非同步回溯技术需要首先介绍一个新的概念——冲突分析，它是子句学习和非同步回溯思想的核心。所谓的冲突分析是指在假定在当前的变量赋值 $A_{curr}$ 的条件下，SAT算法在搜索的过程中遇到了一个冲突事件 $Conf$ （使SAT表达式中的一个子句 $c$ 不满足， $c$ 可被称为冲突子句），冲突分析的工作是分析并确定一个赋值文字的集合 $A_R(Conf) \subseteq A_{curr}$ ，该集合所包含的文字赋值是引起冲突 $Conf$ 的直接原因。特别需要指出的是所谓直接原因是指当仅进行 $A_R(Conf)$ 中的赋值时就会导致冲突 $Conf$ ，而无需 $A_{curr}$ 中除 $A_R(Conf)$ 以外的文字赋值。冲突分析的重要性体现在于在SAT搜索引擎搜索问题的解的过程当中所遇到的冲突往往仅是由一小部分变量的赋值引起的，这样通过冲突分析找到这一小部分变量赋值信息使得在将来的搜索过程中直接跳过由这些变量赋值后所限定的搜索子空间（因为这些子空间必然不包含问题的解，由冲突分析找到的变量个数越少排除的搜索子空间越大），极大地提高了搜索的效率。下面举一个从文献[17]提到的例

子来进一步说明冲突分析的概念。

考虑下面一个CNF表达式 $\Phi$ 中的一分子句集合：

$$\begin{array}{ll} c1 = (\sim X1 + X2) & c6 = (\sim X5 + \sim X6) \\ c2 = (\sim X1 + X3 + X9) & c7 = (X1 + X7 + \sim X12) \\ c3 = (\sim X2 + \sim X3 + X4) & c8 = (X1 + X8) \\ c4 = (\sim X4 + X5 + X10) & c9 = (\sim X7 + \sim X8 + \sim X13) \\ c5 = (\sim X4 + X6 + X11) \end{array}$$

假设当前的变量赋值情况如下：

$$A_{curr} = \{X9 = 0@1, X10 = 0@3, X11 = 0@3, X12 = 1@2, X13 = 1@2, \dots\}$$

以上变量赋值中“@”后面的数字代表决策级数，其表明该变量在哪个决策层被赋值，如 $X9 = 0@1$ 即表示变量 $X9$ 在第一层决策时被赋值为0。决策级数越大表示该变量越晚被赋值。在当前赋值情况下，现在再对变量 $X1$ 赋值， $X1 = 1@6$ ，此时经过BCP推导即可发现一个冲突子句，即 $c5 = (\sim X4 + X6 + X11)$ ，文字 $\sim X4$ ， $X6$ ， $X11$ 均被赋值为0，即子句 $c5$ 不满足。具体的文字间的蕴含推理关系可以参见下面的文字蕴含图。

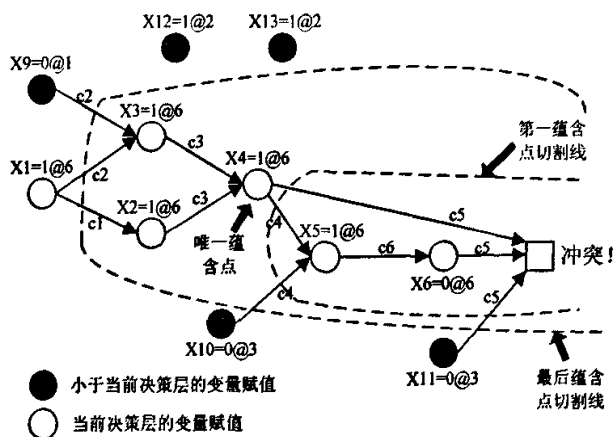


图 3-3 文字蕴含图以其切割线

图3-3中边的标号为子句编号。其中一元子句所对应的变量跟随当前决策变量的决策层。例如变量“ $X2=1$ ”的赋值是由于子句 $c1$ 在决策变量“ $X1=1@6$ ”时成为一元子句从而变量 $X2$ 的决策级数就与变量 $X1$ 相同为6。为了与决策变量 $X1$ 区别，我们把变量 $X2$ 称为决策变量 $X1$ 的BCP变量，或称为决策级数为6的蕴涵变



量。

定义3.1: 唯一蕴含点 (UIP, Unique Implication Point)。所谓唯一蕴含点, 是指这样一种白色节点, 通过白色节点的所有路径均通过该节点。

如果把一条路径喻为形成冲突的一个因素, 则通过白色节点的所有路径就是形成冲突的全部“原因”。那么, 唯一蕴含点的含义即为, 在当前已有的变量赋值情况下, 该节点所对应的变量赋值是在当前决策层中形成冲突的全部“原因”。比如上图中的唯一蕴含点  $X_4=1@6$ , 它表明了, 当决策层数低于6的变量赋值  $\{X_{10}=0, X_{11}=0, \dots\}$  存在的情况下,  $X_4=1$  一定会导致冲突的发生。值得注意的是, 唯一蕴含点在一般的蕴含图中并不是只有一个的, 而且根据唯一蕴含点的定义, 决策变量本身就是一个唯一蕴含点, 如上图中的  $X_1=1$  也是一个唯一蕴含点。

定义3.2: 切割线。所谓切割线, 是对所有通过白色节点的路径的一条割线。

一般而言, 对于切割线, 希望它能切割所有通过白色节点的路径。这样, 一条切割线通常可以将蕴含图分成两部份。包含冲突节点的部分我们可以称为“冲突部分”, 不包含冲突节点的可以称为“原因部分”。显然, “原因部分”中, 紧靠切割线的节点所对应的变量赋值, 就是形成冲突的全部原因。与唯一蕴含点不同的是, 这里的变量赋值是针对原始问题的。换句话说, 如果把其他所有变量的赋值取消, 这些变量的赋值也足以导致冲突的产生, 是形成冲突的“充分条件”。例如在图3-3中的最后蕴含点切割线, 它意味着当  $A_R(\text{Conf}) = \{X_9=0, X_1=1, X_{10}=0, X_{11}=0\}$  时, 原可满足问题即会产生冲突, 赋值文字集合  $A_R(\text{Conf})$  就是导致冲突  $\text{Conf}$  的直接原因。为保证在将来的搜索过程中避免遇到同样的冲突, 需要在此给将来的搜索过程增加一个限定条件, 不允许集合  $A_R(\text{Conf})$  中的各文字赋值同时成立, 对于此例来说即是规定 “ $X_9=0, X_1=1, X_{10}=0, X_{11}=0$ ” 不能同时成立。为实现这一限定条件, 可以增加一个子句  $c_{\text{learn}} = (X_9 + \sim X_1 + X_{10} + X_{11})$  到原始的子句库中, 这样当在以后的搜索的过程当中再遇到 “ $X_9=0, X_1=1, X_{10}=0, X_{11}=0$ ” 的赋值情况时, 子句  $c_{\text{learn}}$  就会不满足, 这时搜索引擎会立即转向其它的空间搜索, 从而避免陷入所有导致冲突的搜索空间。上式中的子句  $c_{\text{learn}}$  即被称为学习子句。

总结起来, 所谓子句学习过程, 就是在搜索遇到冲突时从冲突子句出发逆向考察先前的变量赋值, 比如在蕴含图中划出一条切割线, 收集蕴涵图原因部分最

邻近切割线的节点，构成冲突的原因集合。接着，按这些节点对应的变量赋值，取其相反相位（如 $X_9=0$ 对应的相反相位就是正相位的文字 $X_9$ ）构成一个学习子句。以上即是当前主流的基于冲突的子句学习的基本思想，而在具体算法实现中不同SAT解决器采取的学习策略也有少许不同，其主要集中在切割线选取策略的不同上。如图3-3所示，蕴涵图上可以有两条不同的切割线，最后蕴含点切割线是指以最后蕴含点为边界所画出一条切割线，而第一蕴含点切割线是以第一个蕴含点为边界所画出一条切割线。由不同的切割线所得到的学习子句的长度不同，如以第一蕴含点切割线所做出来的学习子句为 $(\sim X_4 + X_{10} + X_{11})$ ，子句长度为3；而以最后蕴含点切割线所做出来的学习子句为 $(X_9 + \sim X_1 + X_{10} + X_{11})$ ，子句长度为4。一般来讲学习子句长度越短对将来搜索空间的剪除效果越大，因此目前最先进的SAT解决器如zChaff，Minisat等均是采用第一蕴含点切割线的方法(1UIP)进行子句学习的。需要指出的是“学习子句”其实就是原始问题的蕴含项，或可称为冗余项，即加入“学习子句”并不影响原CNF式的性质。虽然不断添加学习子句可以保证搜索引擎在以后不会犯“同样”的错误，继而能使搜索过程越来越“智能化”，但是不断增长的学习子句集合本身也会给SAT解决器带来沉重的负担，特别是对于大规模的SAT问题，统计数据表明一般解决器在搜索的过程中会遇到上十万次甚至上百万次的冲突，如果每次冲突只是增加子句的话，那么SAT问题的规模会增加得很快反而影响了求解问题的速度。因此在目前先进的SAT问题解决器中还会引入子句删除的操作，这在下一节里会具体介绍。

“学习子句”带来的另一个很大的好处就是所谓非同步回溯。一般的，当决策变量的某一个相位赋值并执行BCP后发现了冲突，需要取消当前的赋值并选择该变量的另一个相位赋值。这是所谓的同步回溯。而非同步回溯意味着，我们甚至可以通过分析断定这另一个相位也一定存在冲突，从而不需要赋值运算直接回溯到更高的决策层中去。还是以图3-3为例，我们考察图中的“第一蕴含点切割线”。其所得到的学习子句已经分析过，即为 $(\sim X_4 + X_{10} + X_{11})$ 。可以发现，除了变量 $X_4$ 以外，其余变量的最深决策层为3。这说明，只要这些变量的赋值还存在，无论当前决策变量 $X_1$ 取何值，当 $X_4=1$ 时一定会遇到冲突。于是问题的根源在于，我们必须在决策层为3的时候令 $X_4$ 为0才能保证将来不会遇到同样的冲突。搜索过程因此可从决策级数为6处直接回溯至级数为3处，取消决策级数大于

3的所有变量（决策变量和BCP变量）的赋值，如图3-4所示。也即是说，回溯后的决策级数是由学习子句中除唯一蕴含点变量之外其余变量的最大决策级数所确定的。

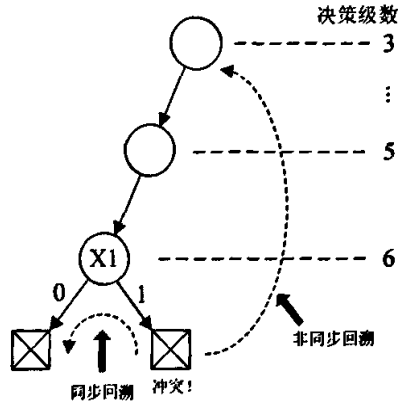


图 3-4 非同步回溯示意图

同时可以发现，当回溯到决策深度为3处时，子句 $(\sim X4 + X10 + X11)$ 除了变量唯一蕴含点变量 $X4$ 外，其余文字的值为0，此时学习子句成为了一个一元子句，根据BCP推导我们自然可以将变量 $X4$ 赋值为0。从中可以发现，以唯一蕴含点为指导的基于切割线的学习过程能与搜索过程中的BCP过程紧密结合，不但剪除了大量的搜索空间，而且有效提高了回溯的效率。

### 3.3.4 子句删除策略

上节中已经提到，学习子句对于搜索的贡献并不是在搜索过程中一直保持的。所谓贡献，可以有两种解释，一是对于搜索过程的BCP推理的支持，即学习子句在经过回溯后能变成一元子句参与BCP过程。这在上一节中已经介绍了。另一个贡献就是剪除不包含解的搜索空间并参与组织新的学习子句。即由于学习子句的存在可能会导致新的冲突，从而能进一步学习到新的子句来约束搜索空间。然而学习子句并不是在整个搜索过程中都有效的，无限制的增加学习子句会增加搜索引擎的负担，降低搜索的效率。观察发现学习子句的个数在最坏的情况下与SAT问题变量个数成指数增长的关系，如果不加以控制庞大的学习子句将会占用巨大的内存空间以至于使程序无法运行。因此有必要根据学习子句在搜索过程中

的表现,引入子句删除策略,对于那些没有什么贡献的学习子句予以删除,以控制SAT问题规模的增长,减轻搜索引擎的负担。

从分析中我们发现,许多“学习子句”如果有“贡献”,只是在很短的决策间隔内有贡献,随着决策级数间隔的增加,“学习子句”的贡献频率越来越低。换句话说,这些“学习子句”只是增加了BCP过程的负担,必须删去。由此也引出了子句的删除策略。哪些子句应该删去呢?到目前为止国际上并没有一个好的判断标准。在文献[20]中,删除策略的思想比较直观。一般而言,包含文字数少的短子句是不应该被删去的。因为子句长度越短对问题的约束能力越强,对搜索过程的贡献也会越久。而对于长子句,还必须看其是否已经满足,满足的文字数是否多。如果长子句中已经满足的文字数越多,证明它很难再回到不满足的状态,也就不会对搜索有贡献,必须删除。而在BerkMin[21]中,删除策略更为贪心。它认为,学习的子句数目必须控制在一定范围内,而且对于子句学习到的顺序也必须加以关心。因此将子句以学习到的先后入“栈”,应用删除的规则也以子句在栈中位置加以区别。栈顶的子句认为是最有贡献的,而越是栈底的子句,应用删除的标准就越严格,越会被优先删去。从而保持BCP过程的效率。总的来说,对于一个学习子句的删除需要考虑以下几个方面,一是学习子句的长度,二是该学习子句在冲突分析中的活跃性,三是产生该学习子句时的决策级数。一般来说,对于子句长度较长,活跃性较低,决策级数较大的学习子句越会被优先删除,反之则倾向于保留。

### 3.3.5 随机重启

在解决SAT问题的过程当中,SAT解决器由于最初的错误决策而时常会陷入一个局部的搜索空间中,耗费了大量的时间却找不到问题的解。为解决这个问题,近年来国际上提出了随机重启的启发式策略,并应用于当代的SAT求解器当中。所谓重新启动,即是在搜索的某一阶段放弃当前所有变量的赋值,重头开始问题的搜索。值得注意的是随机重启只是将目前变量的赋值全部取消,但还保留了记录了之前冲突的学习子句的信息,因此以前搜索的努力并不会因为重新启动而全部白费。相反另一方面,由于学习子句的添加会使得重启后的SAT解决器的变量决策策略得到调整,使其逐渐趋向于做出正确的决策。重新启动的时间

和每次重新启动的间隔可以调整。在文献[20]中,重新启动最初会较为频繁,随着时间增加,间隔会逐渐延长。实验数据表明随机重新启动能够提高SAT解决器的健壮性。

然而由于可满足性问题的复杂,有些规模不大但很“难”的例子,重新启动并不会带来想象中的效率提高,因此,通常这个策略是根据解决问题不同而可选的。

### 3.4 数据结构

DPLL算法原本的思想很古老,是分支定界思想的具体应用。算法本身的时间复杂性肯定是指数的。由于许多算法级别的启发式策略加入,从而改善了算法的实际效率。但是真正将算法推向实际应用,能解决实际规模例子的,其实现的数据结构的改善也起着决定性的作用。

上节中已经提及,无论加入什么优化策略,DPLL算法运行总时间的80%-90%要消耗在BCP过程上。简单来说,就是对子句中相关文字的赋值过程占据主要时间耗费。这个过程中子句的访问量决定了BCP过程的效率。BCP过程的数据结构主要分成邻接表形式和“懒散”(Lazy)形式。

所谓邻接表,即针对每个文字,记录包含该文字的所有子句号。当变量需要赋值时,对对应的文字的子句链表逐一访问子句,以判断是否有一元子句产生,或是否有子句满足/不满足。邻接表最主要的缺点就是子句访问量很大。尤其是采用了学习过程后,文字对应的子句数目增加很快,从而不断降低BCP过程的效率。为了克服这个缺点,也有些邻接表的变化形式。比如,对于已满足的子句,将它从对应文字的子句链表中“隐藏”,从而减少子句的访问量;对于不满足的文字(文字值为0),从所作的子句中“隐藏”,也减少一定的访问量。但是这些措施虽然在一定程度上提高了BCP过程的效率,却增加了回溯过程的开销,因此总体表现并不突出。

当代的可满足问题解决器的主要BCP数据结构是一类所谓“懒散”数据结构。之所以称之为“懒散”,因为对于某个文字,它并不关心所有包含该文字的子句,而仅仅是对需要“观察”的子句进行访问。“懒散”数据结构也主要分成两类。

一类所谓的“头尾”指针方式[19]，另一类所谓的“观察”指针方式[20]。在这两类“懒散”数据结构的基础上，后人又做出了一些改进，“头尾”指针加文字筛选数据结构和“观察”指针加文字筛选数据结构，但其基本思想还是基于上述的两种“懒散”数据结构。下面将分小节重点介绍这两种“懒散”数据结构。

### 3.4.1 头尾指针

“头尾”指针数据结构是第一个被提出来的针对BCP操作的懒散数据结构。它最初应用在SATO SAT求解器上[19]。正如其名字所蕴含的意义一样，该数据结构需要给每个子句附带两个指针，分别是“头”指针和“尾”指针，指向子句中的两个文字。图3-5展示了四种懒散数据结构的操作比较，第一列“HT”即表示了“头尾”指针的工作方式。图中所示了一个子句在不同时期的指针模式和子句的状态。每个方格代表子句中的一个文字。白色方格表示该文字还没有被赋值。而已划叉的灰色方格表示该文字的值为0。而灰色方格下方的数字表示该文字是在哪个决策层时被赋值的。只有当指针指向的文字对应的变量被赋值后，才访问该子句，并将相关的指针向中间移动，指向还未赋值的文字，或已经满足的文字。H和T有次序关系，H必须保持在T的左边。当某个指针在移动时遇到另一个指针，如图中当H指针原先所指向的文字已被赋值为0时，H指针在寻找未赋值文字时遇到了T指针，这表明该子句只有一个文字未赋值，是一元子句。注意，这里H指针只需要检查H和T之间的文字即可。在H和T两边的文字一定是不满足的。回溯时，为了保持这样的H和T的次序关系（H永远在T的左侧），如图中回溯到第2层，则必须移动指针，保持图中初始的第2层的指针状态。这点很重要。因为必须满足H和T两侧的文字都是不满足的，未赋值的文字一定在这两个指针中间。在具体实现时，用栈保留每个决策层时文字对应的子句链表，就能在回溯时实现上述功能。从以上的分析可以看到虽然头尾(HT)指针方式实现时也使用子句链表，但这个链表的长度要远小于邻接表中的子句链表。对于子句中没有被H和T指向的文字，它们的链表中并不会出现该子句的子句号。这样，所有文字的链表长度总和也只是子句数的两倍，而邻接表中子句链表的总和是子句数与子句平均长度的乘积，并且由于学习子句通常很长，因此“头尾”指针的BCP和回溯效率比邻接表的要高很多，实验表明采用“头尾”指针的数据结构可以比邻接表的数据结构

快一个数量级。

在头尾指针数据结构的基础上,文献[33]又提出了一种新的结构,头尾指针加文字筛选数据结构(见图3-5“htLS”列)。与头尾指针类似,该数据结构同样使用了“头尾”指针来指向未赋值的文字,此外在操作过程当中,它还动态地调整已赋值的文字在子句中的位置。按决策级数的大小排序,使得决策级数越小的文字放在越靠近子句的两端,而越靠近子句中间的文字的决策级数越大。除了头尾两个指针,该结构还另引入了两个指针“HB”和“TB”,其在操作过程中分别指向头指针(T)左边的一个文字和尾指针右边的一个文字。这样增加了文字筛选的好处是在子句成为一元子句时或成为冲突子句时,不必历遍整个子句来确认,而仅观察头尾指针之间的文字赋值情况即可,此外无论是在正向推理还是反向回溯的过程中,那些处在较低决策层的文字仅会被访问一次,之后即被筛选出头尾指针所限定的区域,避免了重复检查那些已被赋值的文字。

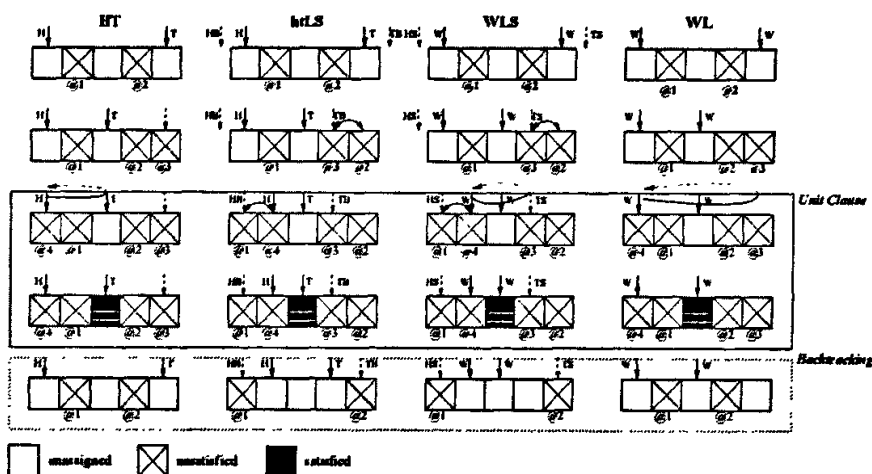


图 3-5 懒散数据结构的操作方式[33]

### 3.4.2 观察指针

“观察”指针方式是在“头尾”指针方式之后提出一种新的“懒散”数据结构[20]。同“头尾”方式不同的是,“观察”指针方式并不区分两个指针先后关系,见图3-5中第四列“WL”列所示。与“头尾”指针相同的是,在该结构下,子句同样使用两个指针指向两个尚未赋值的文字。没有被指向的文字的链表中不

会出现该子句的子句号,因此BCP的效率也是比较高的。但与“头尾”方式判定一元子句方式不同,由于它不保证指针两侧的文字都是已赋值的,因此,某个指针必须对子句中所有文字进行扫描才能判定是否除了另一个指针所指的文字外所有文字均不满足。当一个子句中包含文字数较多时,这样访问自然比“头尾”方式要费时。然而,“观察”指针方式比“头尾”指针优越在两点。一是在回溯上,“观察”指针方式几乎不用再访问子句,而是直接取消变量赋值即可,也无需对文字中的子句链表做任何操作,因此回溯效率非常高。二是,由于保持了“头尾”指针中的指针次序,文字的子句链表必须以栈方式保留每个决策层的链表内容,当决策较深时,带来的内存消耗和动态申请释放时间也是很可观的。而“观察”指针方式对于每个文字的链表只需要一个,因此内存效率很高。更为细致的分析后可以发现,其实“观察”指针方式的BCP效率比“头尾”指针高还体现在其自适应性,与决策策略相配合,如VSIDS[20]。决策策略表明在局部回溯后,变量的决策顺序也许不会有太大改变。可以看到“WL”列中,子句最右边的那个文字是在第3层被决策的。当回溯后,决策策略也许还会选择这个变量作为决策变量(由于其决策分值变化不大),然而从回溯后的子句指针状态可以看见,其实这个原先访问过的子句将不会被访问到(那个最右边的文字上没有指针!)。而且大多数子句都会存在这种情况(除非只有两个文字是未赋值的)。这就意味着,“观察”指针数据结构会自动将搜索定位在需要考察的子句上,而忽略在变量赋值后肯定不会是一元子句的子句。文献[34]对BCP过程在实际计算机中的指令级表现做了比较,可以发现,“观察”指针结构无论从缓存命中率还是缓存的访问量都优于“头尾”指针方式,自然优于邻接表方式。因而算法的整体运行时间得到了不少提高。与头尾指针类似,文献[33]也针对观察指针数据结构提出了改进措施,增加了文字筛选(见图3-5“WLS”列)。从图中可以看到改进后的数据结构不但继承了观察指针结构的优点,而且还避免了已赋值文字的重复访问(原因已在上节中介绍),因此其整体性能比观察指针(WL)结构又有了一定提高。

### 3.5 小结

本章主要回顾了完全算法中占有重要地位的一类基于DPLL的可满足性问题



算法，并从算法层和实现层介绍了这类算法的主要发展历史。总结之可以发现，在算法层次上的学习过程的提出以及实现层“懒散”数据结构的整合，使当代的可满足性问题算法越来越接近于实用化，并被其它类问题作为基本算法和引擎加以使用。虽然如此，算法本身还是有可以提高和改良的地方，比如预处理算法和决策策略的改进等方面。在下章中我们将对可满足性问题的算法和技术做更深入的探讨，主要阐述本文所作的工作，提出了新的SAT算法和技术以进一步加速SAT问题的求解，增强SAT解决器处理大规模SAT问题的能力。

## 第4章 对称蕴含预处理算法

在前面的章节中我们阐述了在可满足性算法和技术领域国际上已取得的研究成果和技术水平,介绍了前人的工作。从这一章起我们将陆续介绍新的SAT算法和技术,即重点阐述本文的工作。我们知道可满足问题有广泛而重要的实际应用,比如形式验证,数学规划,人工智能等领域中的问题均可以转化为可满足性问题。特别是由于近二十年来在可满足性问题解决方法上的巨大进步,越来越多的研究人员试图将他们所在的领域中的问题转化成可满足性问题的形式以达到在更短的时间内解决问题[35]。然而这些转换过来的可满足性问题的规模往往很大,即使是当前最先进的可满足性问题求解器也常常要花费很长的时间来解决,甚至对于有些工业界的例子当前的求解器还无法解决[13]。另一方面仔细观察这些从实际问题转换成合取范式(CNF)形式的可满足性问题表达式时我们可以发现这些表达式中包含有许多二元子句,这就给预处理化简带来了很大的空间。由于2SAT问题(所有的子句长度均不超过2)存在多项式时间复杂度算法,其包含许多有用的信息可被用来化简原问题的规模[13, 24],因此如何利用原始可满足性问题中的特性来有效进行预处理化简原始问题的规模,减少解决问题的时间是一个值得研究的课题。本章即就这一问题进行了研究,提出了一种新的可满足性问题预处理算法,对称蕴含预处理算法。

### 4.1 对称扩展的一元推导

在介绍对称扩展一元推导的概念之前,首先对其中会使用到的蕴含图的对称性做一简单介绍。

定义1: 蕴含图中的对称性:

在蕴含图中给定一个蕴含关系( $a \rightarrow b$ ), 它的对称的蕴含关系为( $\sim b \rightarrow \sim a$ ), 反之也一样。如果在蕴含图中对于每一个蕴含关系均有与其对称的蕴含关系存在, 则我们称该蕴含图具有对称性。所谓的蕴含关系如( $a \rightarrow b$ )是指当文字 $a$ 成立的条件卜(即 $a$ 取1时), 文字 $b$ 也必须随之成立, 否则会出现冲突。

与已有的SAT预处理器2Simplify[24]和Hypre[28]类似, 对称蕴含预处理算法

也使用了蕴含图来进行化简。接下来可以清晰的看到当只利用二元子句来构建蕴含图时，该二元蕴含图即具有对称性。然而当我们运用传统的一元推导来进一步构建蕴含图时将会失去蕴含图的对称性，此意味着传统的一元推导错失了一部分本来值得利用的蕴含关系来指导化简工作。以下面的例子作参考。给定CNF形式的布尔表达式：

$$\{(a, m), (a, n), (\sim m, \sim n, c), (\sim c, d), (\sim c, e), (\sim d, \sim e, k), \\ (b, h), (b, i), (\sim h, \sim i, \sim k), (a, \sim b)(\sim a, u, v)\}$$

利用其中的七个二元子句我们可以构建出一个二元蕴含图如图4-3所示。其中的蕴涵关系（用有向箭头表示） $\{(\sim i \rightarrow b), (\sim h \rightarrow b), (\sim b \rightarrow i), (\sim b \rightarrow h)\}$ 是由二元子句 $\{(b, h), (b, i)\}$ 产生的； $\{(\sim e \rightarrow \sim c), (\sim d \rightarrow \sim c), (c \rightarrow e), (c \rightarrow d)\}$ 是由子句 $\{(\sim c, d), (\sim c, e)\}$ 产生的； $\{(\sim n \rightarrow a), (\sim m \rightarrow a), (\sim a \rightarrow n), (\sim a \rightarrow m)\}$ 是由 $\{(a, n), (a, m)\}$ 产生的。而二元子句 $(a, \sim b)$ 产生了蕴涵关系 $\{(\sim a \rightarrow \sim b), (b \rightarrow a)\}$ 。从图4-3中我们可以直观地看到二元蕴含图的对称特性。每一个二元子句如 $(a, m)$ 可以导出两个对称的蕴涵关系，其规则为从二元子句所含文字的反向文字如 $\sim a$ （或 $\sim m$ ）出发分别添加蕴涵箭头指向二元子句中的另外一个文字如 $m$ （或 $a$ ）。这是因为从二元子句 $(a, m)$ 中可以得知文字 $\sim a$ 与文字 $\sim m$ 不能同时存在于一组可满足性的赋值当中（一组文字赋值由各个文字组成，而一组可满足性文字赋值指的是可使原表达式满足的这样一组文字赋值），因此有且仅有两种可能的文字蕴涵关系才能满足二元子句 $(a, m)$ ，即当文字 $\sim a$ 成立时文字 $m$ 必须成立，用 $(\sim a \rightarrow m)$ 表示；当文字 $\sim m$ 成立时文字 $a$ 必须成立，用 $(\sim m \rightarrow a)$ 表示。由此可见当只使用二元子句来构建蕴含图时蕴涵箭头的个数是二元子句个数的两倍并且每一个二元子句所产生的蕴涵关系均是以两两对称的形式存在的。正是由于这个原因我们称二元蕴含图具有对称性。在二元蕴含图的基础上我们可以进一步通过一元推导将更多的文字蕴涵关系添加到蕴含图中。因为通过一元推导能够利用三元甚至多元子句获得文字蕴涵关系而并不仅仅局限在二元子句中。举例说明，图4-3中新产生的蕴涵关系 $(c \rightarrow k)$ 是从三元子句 $(\sim d, \sim e, k)$ 推导出来的，因为当一元推导文字 $c$ 时，文字 $d, e$ 必然随之成立，即文字 $\sim d$ 和文字 $\sim e$ 在 $c$ 成立的情况下必须被赋值为0，将其带入三元子句 $(\sim d, \sim e, k)$ 中后，为满足该子句，则文字 $k$ 也必须成立，这样文字 $c$ 的成立蕴涵着文字 $k$ 的成立，因此在图4-3中用新添加的 $(c \rightarrow k)$ 来表示。同理可得新

添的 $(\sim b \rightarrow \sim k)$ 是从 $(\sim h, \sim i, \sim k)$ 产生的,  $(\sim a \rightarrow c)$ 是从 $(\sim m, \sim n, c)$ 产生的。这样通过一元推导我们可以得到更多的蕴含关系。然而我们发现传统的一元推导操作破坏了原有蕴含图的对称性。以图4-3中新产生的蕴含关系 $(\sim a \rightarrow c)$ 为例, 它的对称蕴含关系 $(\sim c \rightarrow a)$ 并没有存在于图4-3的蕴含图中。实际上这些对称的蕴含关系对于预处理化简来讲是十分重要和值得利用的信息, 它们可以使我们更深层次地发现失败性(或强制性)文字和等价文字进而能更好的化简原始SAT问题的规模。因此为了充分利用一元推导的化简能力我们还需要将由一元推导得到的蕴含关系的对称蕴含关系也添加到蕴含图当中, 始终保持蕴含图的对称特性。我们将这一添加了对称蕴含关系的正向推理过程称为对称扩展的一元推导。

**定理1:** 如果存在文字蕴含关系 $(a \rightarrow b)$ , 则其对称的文字蕴含关系 $(\sim b \rightarrow \sim a)$ 必然成立。

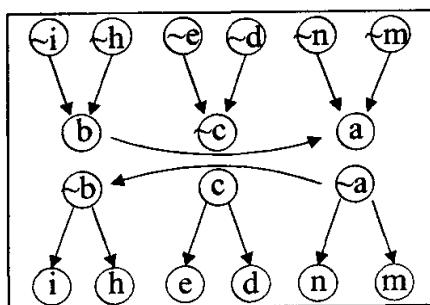
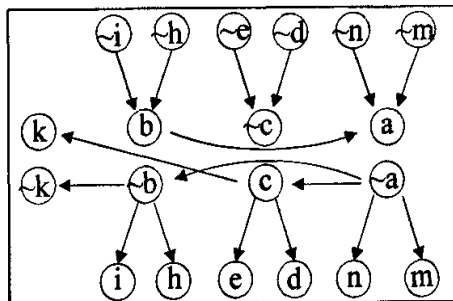


图 4-1 二元子句蕴含图

图 4-2 一元推导 $\sim a, c$ 和 $\sim b$ 后的蕴含图

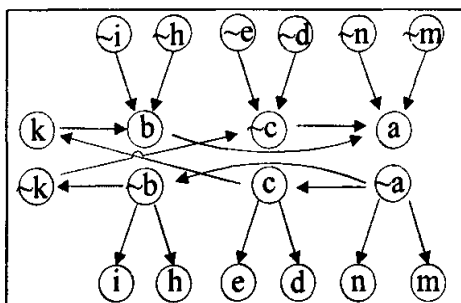


图 4-3 对称扩展一元推导之后的蕴涵图

值得注意的是这些新产生的对称的蕴含关系是不能够被传统的·一元化简推理得到的，因此必须把这些对称的蕴含关系一一地添加到蕴含图当中。图4-3即显示了经过了对称扩展的一元推导后的蕴含图。其中新添的蕴涵关系( $k \rightarrow b$ ), ( $\sim k \rightarrow \sim c$ ), ( $\sim c \rightarrow a$ )分别是原蕴涵关系( $\sim b \rightarrow \sim k$ ), ( $c \rightarrow k$ ), ( $\sim a \rightarrow c$ )的对称蕴涵关系，将这些对称的蕴涵关系添加至蕴含图之后可以发现图4-3所示的蕴含图中文字 $\sim a$ 可以由两条蕴含路径间接推导出文字 $a$ ，即( $\sim a \rightarrow \sim b \rightarrow \sim k \rightarrow \sim c \rightarrow a$ )和( $\sim a \rightarrow c \rightarrow k \rightarrow b \rightarrow a$ )这意味着文字 $\sim a$ 是失败性文字而对应的 $a$ 是强制性文字， $\sim a$ 必须赋值为0并可以进行一元推导化简。而对于图4-1和图4-2所示的蕴涵图来讲它们均无法判定文字 $\sim a$ 为失败性文字。

对称蕴含的预处理算法除了能发现深层次的失败性文字(或强制性文字)外，其同样还能发现用以往等价性检查法，如子句模式匹配法，二元子句蕴含图法，双相一元推导法难以发现的深层次的文字间的等价关系，进而可以获得对原始SAT问题的更大层度的化简。在对比说明对称蕴含预处理方法与其它等价性检查方法的性能差异之前，先简要列举一下已有的等价性检查算法。

**1、子句模式匹配法：**该方法是通过查找子句数据库找到匹配的子句组，进而进行等价变量化简。例如，在子句数据库中发现有这样两个二元子句存在： $(a + b)$ 和 $(\sim a + \sim b)$ ，则可判定等价关系 $a = \sim b$ ，进行化简。这个方法需要对子句库进行大量的检索，时间花费较大，而且不能发现深层次的文字等价关系。

**2、二元子句蕴含图法：**该方法是通过构建二元子句蕴含图，然后找到强连通子图的手段判定文字间的等价关系的。举例说明： $(a + b)(\sim a + c)(d + \sim b)(\sim d + \sim c)$ ，其所对应的二元子句蕴含图如图4-4所示。从图4-4可以看到该蕴含图中包含两个强连通子图，从而对应可以发现等价关系：“ $\sim a = b = \sim c = d$ ”或“ $a = \sim b = c = \sim d$ ”

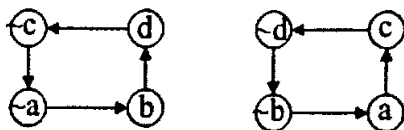


图 4-4 二元子句蕴含图示意图

该方法较子句模式匹配法能进一步发现较深层次的等价关系,并且目前已有搜索强连通子图的高效算法因而花费的时间较短,效率较高。但缺点是由于该方法仅利用二元子句构建蕴含图,许多非二元子句的信息无法利用,因此更深层次的文字间的等价关系无法用该方法得到。

3、双相一元推导法:该方法通过对变量的正反两个相位进行一元推导来发现等价文字关系。例如,  $BCP(a) \rightarrow b$ ;  $BCP(\neg a) \rightarrow \neg b$  即可判定  $a=b$ 。该方法能较前面两种方法发现更深层次的文字等价关系。

我们提出的对称蕴含的预处理技术结合了二元子句蕴含图法和双相一元推导法的优点,在构建蕴含图时,我们的方法不仅利用了二元子句,并且通过使用一元推导的方法利用了众多非二元子句的信息来构建蕴含图。并且在一元推导中额外引入了对称的蕴含关系,这样使得更多更深层次的文字蕴含关系被添加至所构建的蕴含图中,比用单纯的二元子句蕴含图或者双相一元推导法能发现更多的等价关系,这样我们就能更层次地化简原来的SAT问题规模。

接下来以一个包含七个子句的CNF表达式为例,对比说明本文提出的对称蕴含预处理方法相对于前面几种算法的优势。

$$(a + \neg y)(f + m)(f + n)(\neg m + \neg n + \neg d)(\neg f + \neg x)(\neg a + d)(\neg a + x + y)$$

可以看到,用传统的子句模式匹配法无法从以上CNF表达式中找到任何匹配的子句模式,如  $(a + b)$  和  $(\neg a + \neg b)$ 。因而该方法无法从以上CNF表达式中发现任何文字间的等价关系。

用二元子句蕴含图法,可构建二元子句蕴含图如下:

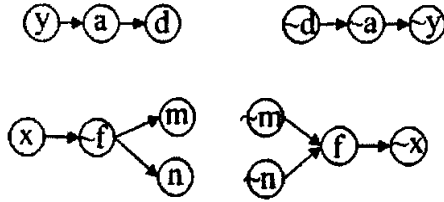


图 4-5 二元子句蕴含图法举例

从图4-5可以看到蕴含图中没有任何强连通子图，因而二元子句蕴含图法也无法发现CNF表达式中的等价文字。

用双相一元推导方法，对CNF式中每个变量进行双相一元推导，结果如下：

$\text{BCP}(a) \rightarrow d; \text{BCP}(\neg a) \rightarrow \neg y$

$\text{BCP}(y) \rightarrow a, d; \text{BCP}(\neg y) \rightarrow \text{NULL}$

$\text{BCP}(f) \rightarrow \neg x; \text{BCP}(\neg f) \rightarrow m, n, \neg d, \neg a, \neg y$

$\text{BCP}(m) \rightarrow \text{NULL}; \text{BCP}(\neg m) \rightarrow f, \neg x$

$\text{BCP}(n) \rightarrow \text{NULL}; \text{BCP}(\neg n) \rightarrow f, \neg x$

$\text{BCP}(d) \rightarrow \text{NULL}; \text{BCP}(\neg d) \rightarrow \neg a, \neg y$

$\text{BCP}(x) \rightarrow \neg f, m, n, \neg d, \neg a, \neg y; \text{BCP}(\neg x) \rightarrow \text{NULL}$

然而从以上双相一元推导的结果上看，同样得不出任何文字间的等价关系。到目前为止，我们可以看到以上三种等价性检查方法的局限性，实际上从以上包含七个子句的CNF表达式中是可以通过对称蕴含预处理方法发现文字间的等价关系的。具体处理流程如下图所示：

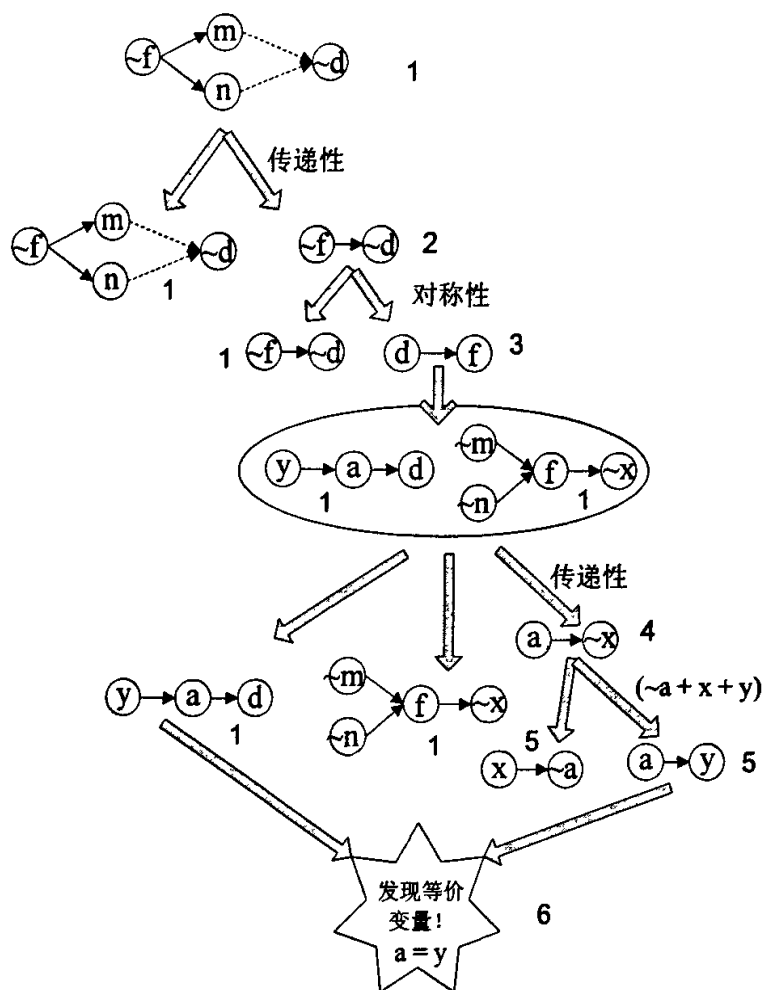


图 4-6 对称蕴含预处理算法举例

图4-6中的有向箭头“ $\rightarrow$ ”代表一元推导，如“ $\sim f \rightarrow m$ ”即表示文字“ $\sim f$ ”取值为“1”后通过一元推导可以得到文字“ $m$ ”的值为“1”。可以看到通过对称蕴含预处理方法最终可以推出： $a \rightarrow y$ ,  $\sim a \rightarrow \sim y$ ，因此可以发现等价关系：“ $a=y$ ”

## 4.2 推理闭包的实现

**定义 2:** 直接后代文字节点

在蕴含图中文字节点 $L$ 的直接后代文字节点是指那些可以由文字 $L$ 通过一步推导出来的文字节点，即在蕴含图中表示的是从 $L$ 出发的有向边所指的那些文字节点。

**定义 3:** 推理闭包



给定一个蕴含图G, 如果其中任何一个文字节点L的后代文字节点集S满足:

$$S = \text{UPL}(L) \cup \text{UPL}(L \text{ 的后代文字节点})$$

则我们称这个蕴含图G达到了推理闭包状态。在此状态下我们无法再通过一元推导G中任何一个文字节点以及其相应的后代文字节点来产生新的文字蕴涵关系。

实现推理闭包是我们预处理化简的最终目标。通过反复运用对称扩展后的一元推导并结合等价化简直至分析完所有的文字再不能产生新的蕴含关系, 此时即达到了推理闭包的状态。实际上在我们的方法中实现推理闭包的过程需要两个阶段。在第一阶段里我们对每一个未赋值的文字进行对称扩展的一元推导, 并将相应的直接后代文字节点记录下来构建出一个初始的蕴含图。在整个预处理过程中蕴含图的对称性始终得到保持, 即每当新添加一个蕴含关系如 $(a \rightarrow b)$ 时, 与其对称的蕴含关系 $(\neg b \rightarrow \neg a)$ 也会立即添加至蕴含图中。此外如果在构建蕴含图的过程中发现失败性文字或等价文字时我们将分别应用一元推导和等价化简进行化简处理。比如当同时发现 $(a \rightarrow b)$ 和 $(a \rightarrow \neg b)$ 时, 可判断文字a是失败性文字, 对应的 $\neg a$ 是强制性文字, 我们会立即对 $\neg a$ 以及 $\neg a$ 的所有后代节点进行一元推导化简。另外, 当同时发现 $(a \rightarrow b)$ 和 $(b \rightarrow a)$ , 即发现存在蕴涵环路的情况时, 则可得出等价关系 $a = b$ , 这时可以利用等价化简技术去掉冗余文字和子句从而化简原问题。其中等价化简主要包含三个步骤: 1、将原问题里所有文字b(或 $\neg b$ )都用文字a(或 $\neg a$ )代替; 2、从原问题中删去那些既含有文字a又含有文字 $\neg a$ 的子句; 3、在剩余的包含文字a(或 $\neg a$ )的子句中去掉那些重复冗余的a(或 $\neg a$ )。显然, 以上的一元推导化简和等价化简将会产生更多新的二元子句, 因此还必须要不断地将这些二元子句所包含的蕴涵关系添加到蕴含图中。当第一阶段完成后, 即建立了一个初始的蕴涵图。在第二阶段里进一步分析并完善蕴涵图, 直至达到推理闭包状态。第二阶段的处理与第一阶段处理十分类似, 也是反复的应用对称扩展的一元推导和等价化简。唯一与第一阶段不同的是我们不断地在一个文字节点与其间接的后代文字节点之间添加直接的蕴涵关系, 这样使文字间的蕴涵关系更加直接, 从而能进一步分析得出更多的文字间的蕴涵关系。比如, 蕴涵图中如果已存在两个文字蕴涵关系 $(a \rightarrow b)$ 和 $(b \rightarrow c)$ , 我们将添加新的蕴涵关系 $(a \rightarrow c)$ 到蕴涵图中, 这样当以后分析文字a, 即对a和所有a的直接后代节点进行一元推导时就能更快的发现

失败性或等价的文字。

### 4.3 预处理器 Snowball 算法

Snowball的基本预处理算法伪代码如表4-1所示。这个预处理算法通过构建和分析蕴涵图来化简原SAT问题的规模。

表 4-1 Snowball 预处理算法

```

1  一元推导所有一元子句
第一阶段: 构建初始蕴涵图
2  for each L from sorted list of literal{
3    if(L is removed) continue;
4    children_set = UPL(L & L's current children);
5    for each l in children_set {
6      if(l→L) {equalityReduction(l=L); Reconsideration();}
7      else if(l→¬L) {UP(¬L); Reconsideration();}
8      else add new implications: (L→l) & (¬l→¬L); }}
第二阶段: 分析并进一步完善蕴涵图
9  for each L from sorted list of literal {
10   if(L is removed) continue;
11   collect all L's current children to newchildren_set;
12   do{
13     for each l in newchildren_set{
14       for each l's child z {
15         if(z=L) {equalityReduction(l=L); Reconsideration();}
16         else if(z=¬L) { UP(¬L and ¬L's children); Reconsideration();}
17         else add new implications, (L→z) & (¬z→¬L);}}
18   newchildren_set = UPL(L & L's children)- newchildren_set;
19   }while(newchildren_set isn't empty);}
END

```

算法首先将原CNF表达式中的一元子句收集起来进行一元推导化简,使得处理后的CNF表达式仅含有二元或二元以上的子句。接下来我们对所有未赋值的变量按照其在二元子句中的数目进行排序,目的是为第一阶段预处理做准备,使得最活跃的文字优先得到处理,这样能较快的发现强制性文字和等价文字。在第一阶段处理过程中,正如第二节所提到的那样,通过应用对称扩展的一元推导并结合等价化简技术处理所有未赋值的文字节点直至不能推出新的文字间蕴涵关系,这时算法就完成了第一阶段的预处理工作得到了一个过渡状态的蕴涵图。伪代码里第2行至第8行描述了对每个文字的具体操作。其中对每一个未被删除的文字L,程序首先对它和它当前的直接后代文字节点进行一元推导,如果在这个操作过程当中发现冲突,则可判断文字L为强制性文字必须立即对它和它的所有直接后代节点一起进行一元推导化简,并把这些已赋值的文字从未赋值文字列表里删除。如果以上的操作没有发现冲突,则对一元推导所得到L的后代节点集进行分析,如

第6行至第8行所示。当在这一过程中发现强制性文字或者等价文字时则立即作相应的化简操作。如此访问完所有未赋值的文字后，第一阶段预处理操作结束。第二阶段的操作与第一阶段类似，不同的是增加了文字与其间接后代节点之间的蕴涵关系如第17行所示。需要指出的是从所示算法中还可以发现Reconsideration()函数，它的功能是重新分析考察以前曾被分析过的文字。需要这个函数的原因在于当程序进行了一元推导化简和等价化简之后可能会使原来的多元子句蜕变成二元子句，比如原来的三元子句(a, b, c)在经过一元推导化简文字 $\sim c$ 后，文字c被赋值为0，这样原来的三元子句就变成了二元子句(a, b)。从这个新产生的二元子句中可以推出两个互为对称的蕴涵关系( $\sim a \rightarrow b$ )和( $\sim b \rightarrow a$ )，这意味着文字 $\sim a$ 和 $\sim b$ 以及它们的父辈文字节点现在将很有可能通过对称的一元推导能得到更多新的文字蕴涵关系，因此所有的这些文字均需要进行重新分析和考察。

## 4.4 实验结果和分析

应用对称性扩展的一元推导技术以及等价化简技术，本文设计实现了一个预处理器Snowball。本节将列出实验结果以评估该预处理器的有效性。本文中具体的实验环境如下：Sun Blade1000工作站，900MHz UltraSPARC-III CPU，2GB内存，Solaris 8操作系统。

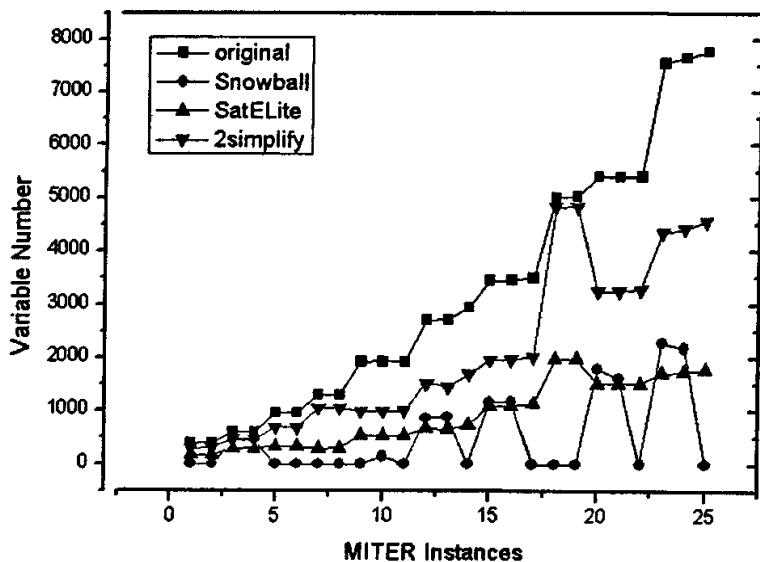


图 4-7 变量化简效果比较。图中 original 指代原 SAT 问题的变量数；而其它曲线代表经过各自预处理化简后 SAT 问题的变量数

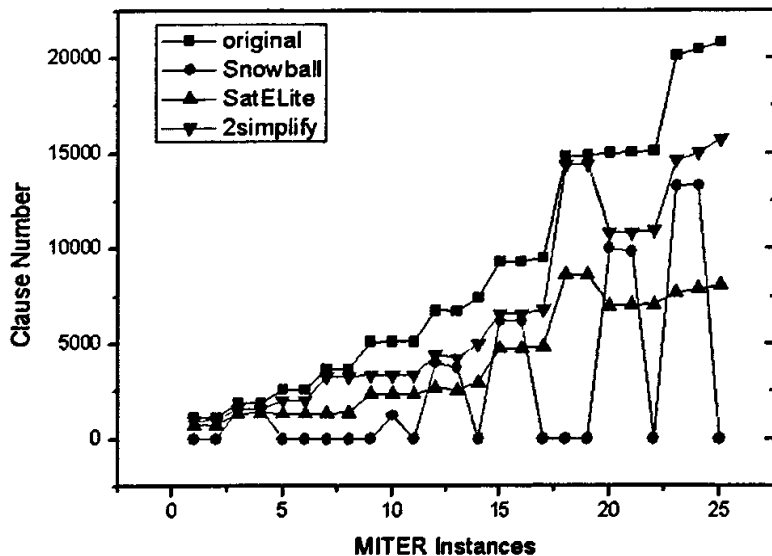


图 4-8 子句化简效果比较。图中 original 指代原 SAT 问题的子句数；而其它曲线代表经过各自预处理化简后 SAT 问题的变量数

图4-7和图4-8显示了预处理器SatELite, 2Simplify和Snowball的预处理效果比较结果。该实验中所选的实验例子来自MITER问题族。图4-7和图4-8中纵坐标表征的是SAT问题的规模（在图4-7中指的是变量数，在图4-8中指的是子句数），横坐标则指的是MITER族中各个SAT问题的序号。方点线表示的是原始SAT问题的规模，圆点线表示的是经过Snowball化简后的SAT问题的规模，而其它三角点线表示的是用2Simplify和SatELite化简后的SAT问题规模。从图4-7和图4-8中可以看到在化简效果方面，Snowball预处理器在MITER族所有例子上均优于2Simplify。尽管在时间上2Simplify的预处理花费较少，它的化简程度比Snowball小很多。在所有25个例子当中Snowball能够完全独立地解决其中的14个不可满足的例子，而2Simplify和SatELite却无法确定任何一个MITER族里的例子的可满足性。

表 4-2 MITER 族中两个乘法器 SAT 实例的运行时间比较

Instance	zChaff (s)	Minisat (s)	2Simplify +Minisat (s)	SatELite +Minisat (s)	Snowball (s)
c6288	>10,000	>10,000	>10,000	>10,000	0.59
c6288-s	>10,000	>10,000	>10,000	>10,000	0.57

最佳的展示Snowball预处理器能力的例子来自于MITER族里的c6288和c6288-s这两个乘法器的例子。表4-2中显示了SAT预处理器，zChaff解决器和Minisat解决器在这两个例子上所花费的运行时间。从表4-2中可以看到，虽然这两个例子似乎很难，以至于无论用zChaff还是Minisat解决器均无法在10,000秒

的运行时间之内解决，本文的预处理器Snowball却在不到2秒钟的时间内成功地解决了这两个例子。进一步我们还将这两个例子用2Simplify和SatELite预处理器简化后再让Minisat解决器进行处理，然而仍然均无法在10,000秒的时间内解决。从这里可以明显地看到在预处理器Snowball中应用的对称蕴含推理和等价化简技术相结合的预处理技术不同于2Simplify和SatELite中所用到的预处理技术，Snowball预处理算法能够对于那些较难的SAT问题如乘法器实例表现出非常好的解决效率。此外，我们发现C6288和C6288-s两个SAT实例是从乘法器电路转换过来的，其中包含有许多异或门结构，因而这两个SAT实例中必然包含有很多等价文字，这给Snowball预处理器提供了很大的化简空间。在前面4.1节的理论分析中就已经表明对称蕴含推理与等价化简相结合能够非常有效地发现深层次的文字间的等价关系，进而可以实现对于乘法器实例的有效化简和解决。以上解释与文献[13]中的观察发现相吻合，在[13]中作者也发现对于含有较多异或门结构的“异或链”SAT实例用传统的分支界定（如DPLL）算法很难获得高效的求解，然而结合了等价化简技术的其它处理方法可以很有效地解决这类问题。尽管2Simplify和SatELite预处理器中运用了好几种简化技术比如蕴涵推导，包含消除，纯文字化简以及等价化简等，但从图4-7，图4-8以及表4-2的数据可以看出它们对SAT问题的化简效果不如Snowball，结合了等价化简技术的对称蕴含推理算法比以上这些预处理技术在化简效果上更好，正向推导能力更强。

表 4-3 “预处理器+zChaff 解决器”在 rand\_net 实例上的运行时间比较

Instance	zChaff (s)	2Simplify+zChaff(s)		SatELite+zChaff(s)		Snowball+zChaff(s)	
		2Simplify	zChaff	SatELite	zChaff	Snowball	zChaff
rand_net40-25-10 shuffled cnf	>900	1.95	484.09	0.06	735.22	28.22	0.0
rand_net40-25-5 shuffled cnf	416.85	2.13	240.87	0.05	84.5	38.17	0.0
rand_net40-30-10 shuffled cnf	267.13	5.85	109.78	0.08	305.99	75.79	0.0
rand_net40-30-5 shuffled cnf	283.57	7.89	50.36	0.08	52.5	96.57	0.0
rand_net50-25-10 shuffled cnf	>900	2.5	>900	0.08	>900	51.08	0.0
rand_net50-25-5 shuffled cnf	>900	3.01	683.35	0.09	>900	63.76	0.0
rand_net50-30-10 shuffled cnf	>900	5.05	>900	0.10	>900	105.09	0.0
rand_net50-30-5 shuffled cnf	>900	3.85	>900	0.10	>900	93.06	0.0
rand_net50-40-5 shuffled cnf	>900	30.3	>900	0.13	>900	327.39	0.0
rand_net50-60-10 shuffled cnf	>900	33.13	>900	0.20	>900	200.03	70.37
rand_net60-25-10 shuffled cnf	>900	2.53	>900	0.10	>900	45.68	0.0
rand_net60-25-5 shuffled cnf	>900	2.71	>900	0.09	>900	54.11	0.0

表 4-4 “预处理器+Minisat 解决器”在 rand\_net 实例上的运行时间比较

Instance	Minisat (s)	2Simplify+Minisat (s)		SatELite+Minisat (s)		Snowball+Minisat (s)	
		2Simplify	Minisat	SatELite	Minisat	Snowball	Minisat
rand_net40-25-10 shuffled.cnf	12.19	1.95	4.19	0.06	7.88	28.22	0.0
rand_net40-25-5 shuffled.cnf	6.02	2.13	3.14	0.05	2.7	38.17	0.0
rand_net40-30-10 shuffled.cnf	12.94	5.85	8.08	0.08	7.87	75.79	0.0
rand_net40-30-5 shuffled.cnf	9.55	7.89	3.58	0.08	2.39	96.57	0.0
rand_net50-25-10 shuffled.cnf	73.99	2.5	35.21	0.08	33.66	51.08	0.0
rand_net50-25-5 shuffled.cnf	37.5	3.01	9.07	0.09	16.49	63.76	0.0
rand_net50-30-10 shuffled.cnf	148.07	5.05	89.36	0.1	78.57	105.09	0.0
rand_net50-30-5 shuffled.cnf	115.04	3.85	82.72	0.1	74.43	93.06	0.0
rand_net50-40-5 shuffled.cnf	63.96	30.3	44.98	0.13	20.7	327.39	0.0
rand_net50-60-10 shuffled.cnf	1801.1	33.13	486.26	0.2	1414.3	200.03	6.8
rand_net60-25-10 shuffled.cnf	261.46	2.53	82.6	0.1	53.26	45.68	0.0
rand_net60-25-5 shuffled.cnf	527.36	2.71	47.14	0.09	79.97	54.11	0.0

在实验当中，我们还发现使用不同的SAT解决器与各种SAT预处理器相结合能够对SAT问题整体解决效率产生较大的影响。在实验中我们使用了两种SAT解决器：Minisat2.0和zChaff。Minisat2.0是2006年国际SAT竞赛的冠军，被公认为是一个非常高效的SAT解决器。而zChaff解决器我们使用的是2001年的版本，尽管同样是较高效的SAT解决器但相对于Minisat2.0性能上要逊色一些。表4-3展示了各种不同预处理器结合了zChaff解决器后在解决SAT问题的时间上的对比。从表4-3中我们可以观察到Snowball预处理器对于给定的SAT实例表现得很好。从表4-3中的结果可以看到在经过预处理器2Simplify和SatELite化简之后的SAT问题对于zChaff解决器来讲还是非常难解，zChaff解决器甚至在900秒以内都无法解决许多2Simplify和SatELite化简后的SAT实例。而对比Snowball预处理器，绝大多数SAT实例都能够仅在Snowball预处理阶段就获得解决，剩下的被简化过的SAT实例也能够很短的时间内通过zChaff获得解决。当用Minisat解决器取代zChaff解决器后，见表4-4，可以发现2Simplify+Minisat模式和SatELite+Minisat模式的解决效率获得了提高，由于预处理的工作没有任何改变，这种效率上的提高主要归功于Minisat解决器的高效率。而对比Snowball+Minisat模式，由于大部分SAT处理工作是由于Snowball预处理器完成的，因此在解决SAT问题过程中并没有利用上Minisat的强大的处理能力，因此Snowball+Minisat模式整体解决效率相对于2Simplify+Minisat模式和SatELite+Minisat模式有所降低。从以上实验中我们可以

提出这样一个问题,对于一般的SAT实例如何在SAT预处理器和SAT解决器之间找到一个最优的平衡点,或者换句话说,如何找到SAT预处理化简工作和SAT回溯搜索工作的最佳结合点,以使得SAT问题整体的解决效率最高。我们认为以上问题的解答取决于所使用的SAT解决器以及SAT问题本身的难度。对于那些小规模比较容易解的SAT问题,使用轻量级的SAT预处理器如2Simplify和SatELite等并结合高效的SAT解决器是一个比较好的解决方案。但是对于某些大规模的或者较难的SAT实例,使用重量级的预处理器如Snowball更能够提高SAT问题整体解决效率。表4-5即展示了不同预处理器与解决器组合在解决某些较难的SAT问题上的时间对比。实验中的这些SAT实例均来自2003年的国际SAT竞赛。从表4-5中可以看到, Snowball+Minisat模式的解决问题的总时间要比其它组合以及单纯Minisat的解决问题的总时间要小很多。实验结果清晰地表明重量级的SAT预处理技术如对称蕴含推理对于解决许多较难的SAT问题非常有用, Snowball预处理器可以在总体上减少求解SAT问题的时间,提高SAT问题的解决效率。

表 4-5 “预处理器+Minisat 解决器”在一些较难实例上的运行时间比较

Instances	Minisat (s)	2Simplify +Minisat (s)	SatELite +Minisat (s)	Snowball +Minisat (s)
grupper11.shuffled.cnf	8388.86	>10,000	9884.77	3042.12
hanoi5u.shuffled.cnf	937.55	477.23	865.19	492.97
hanoi6.shuffled.cnf	270.3	1144.1	114.77	561.31
pyhala-braun-sat-40-4-03.shuffled.cnf	3708.05	496.06	1067.61	567.69
2000009987fw.shuffled.cnf	60.7	52.61	103.06	49.29
c7552mul.miter.shuffled.cnf	21.22	21.03	21.42	20.93
rotmul.miter.shuffled.cnf	396.57	262.15	232.6	334.1
term1mul.miter.shuffled.cnf	1368.19	1176.79	3421.14	821.84
frg2mul.miter.shuffled.cnf	396.77	572.02	721.63	647.45
Total Time (s)	15548.21	>14201.99	16432.19	6537.70

## 第5章 双变量决策 DPLL 算法

在本章里将详细阐述了一种采用双变量决策策略的SAT-Solver算法以及其完整的实现方式描述。该SAT-Solver软件工程的是以Minisat2.0为蓝本的，在其较完善的SAT-Solver整体框架的基础上我们对其中的各功能模块进行了修改或重新设计，将原先的单变量决策策略修改成双变量决策策略。使得改造后的SAT-Solver首次具有双变量决策功能，并使得各主要的软件模块：变量决策，propagation模块，冲突分析和回溯模块相互配合，协调一致。该工程的目标是加速SAT-Solver的求解速度。

### 5.1 算法目的

近十年来，国际上对可满足性问题（SAT problem）的研究已取得了相当大的进展。SAT求解器（SAT-Solver）的性能已由过去的仅能处理几百个变量的SAT问题发展到现在可以在较短的时间内解决几十万甚至上百万个变量的SAT问题[34]。尤其是近几年来从SAT-Solver竞赛中脱颖而出的一些SAT-Solver，如zChaff，Minisat等更是成为了人们借鉴，学习，比较的标准，这也极大地促进了SAT问题的研究。纵观当前这些著名的SAT-Solver，可以发现他们均集成了以往SAT问题的研究成果，如子句学习，非同步回溯，重启动，先进的变量决策策略等，使得SAT-Solver的整体性能有了很大地提升。因此，当前的SAT问题研究也应该是在吸收综合已有的研究成果的基础上进行进一步的探索和研究，这样才能使SAT-Solver的性能有更进一步的提高，在现有基础上所做出的新的算法或新思路才更容易体现出价值。基于这个理念，我们选择了Minisat解决器[22]（近两年来的SAT竞赛的冠军）作为我们的研究蓝本，在它的基础上进行进一步的改进和创新。

尽管当前先进的SAT-Solver已非常完整地集成了以往的一些优秀的研究成果，但是我们发现还有一些研究成果特别是正向推理技术并没有能很好地集成进当前最先进的SAT-Solve中。这其中最主要的原因是这些较复杂的正向推理技术并不能很好地与已有的算法框架，数据结构相兼容。比如Hyper-Resolution技术[28]，理论和实验均证明其具有很高的逻辑推理能力，能很好地降低决策数的规



模。但是实验也表明在现有的DPLL算法框架内动态地调用Hyper-Resolution技术不仅在数据存储空间上耗费较大，而且所花费地计算时间也相当高昂，这使得其本身具有的优势表现得并不明显，甚至不如单纯的BCP推导高效。因此，SAT-Solver研究不应当仅看作是算法方面的研究，我们更应该把其当作一个软件工程来看待。其中的算法和数据结构设计需要综合考虑，特别要重视各个部分之间的相互配合优化，以达到整体上的最优。

本文首次提出了一种双变量决策策略，并将其应用于当前较先进的SAT-Solver的算法框架内。以往的SAT问题求解算法在决策部分均是采用计算各文字得分值，然后选取一个最大值的文字作为决策点进行推导。每次决策仅取一个文字进行推导，效率比较低。本文提出在有二元子句存在的条件下，同时对二元子句中的两个文字进行推导，即在决策时选取两个文字进行赋值决策，这样决策效率会得到提高。并且双变量决策策略配合等价化简技术在冲突分析回溯时也会带来效率上的提高。基于以上思路，我们在Minisat2.0的基础上重新设计了各个程序模块，第一次实现了具有双变量决策功能的SAT-Solver。该算法同样是基于DPLL算法框架，如下图所示：

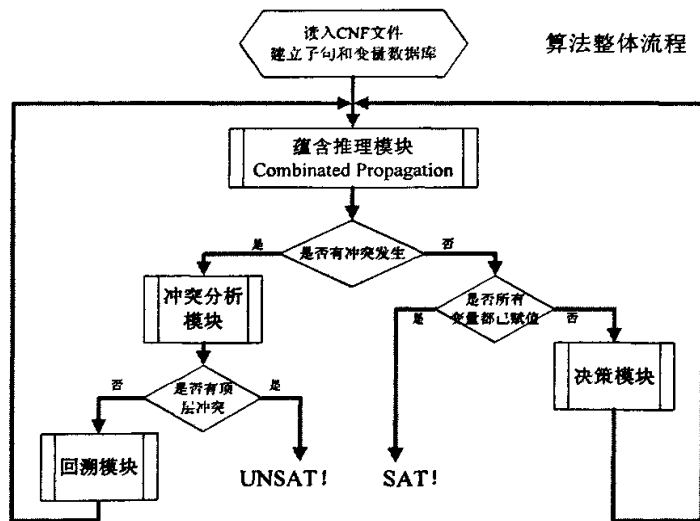


图 5-1 DPLL 算法整体流程图

在接下来的章节中将分别介绍每个模块的所用到的算法和实现方式，以及为支持双变量决策DPLL算法而设计的能有效处理等价文字的数据结构。在本章的最后将给出实验结果以及相应的算法性能分析总结。

## 5.2 决策模块

本章介绍的双变量决策DPLL算法与以往的SAT DPLL算法最大的不同体现在变量决策策略上的不同。传统的变量决策策略是在每次决策层级上仅选取一个未赋值的变量做决策,其决策效率不高,从而使得搜索和冲突回溯的工作量较大。而本章所提出来的双变量决策策略是在决策时选取两个未赋值的变量同时赋值,这样可以提高决策效率,从而可以极大地降低搜索的工作量。双变量决策的工作原理可以解释如下:

对于一个二元子句如  $(a + b)$ , 我们发现要使其满足,可能的文字赋值方式无非有三种: 1、 $a=1, b=1$ ; 2、 $a=0, b=1$ ; 3、 $a=1, b=0$ ; 其中后两种文字赋值方式可以统一用“ $a \sim b$ ”来表示。这样即可以将满足一个二元子句的赋值方式归纳为两种模式:“ $a=b=1$ ”和“ $a \sim b$ ”。因此在搜索中如果我们发现推理其中一个赋值模式时遇到冲突,则可以明确至少在当前文字赋值的条件下,必须遵循另外一种文字赋值模式才能使子句满足。在现阶段所做的双变量决策模块中,我们在每次决策时首先搜索“ $a=b=1$ ”赋值模式,即使二元子句中的两个文字同时赋值为1,并进行推理。若在接下来的推理中发现冲突,则说明“ $a=b=1$ ”的赋值模式无法满足子句,因而明确了另一种文字赋值模式的必要性,即发现了文字等价关系“ $a \sim b$ ”。图5-2给出了传统的单变量决策策略与本文提出的双变量决策策略的比较。可以看到对于同样的搜索路径,采用双变量决策策略可以减少决策级数,进而大大地降低问题搜索空间。

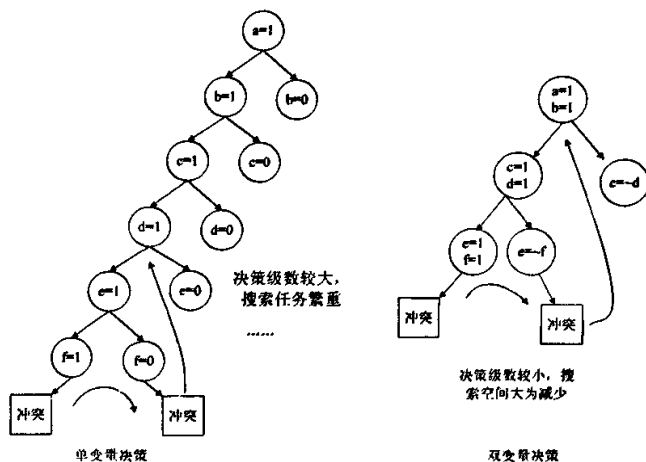


图 5-2 单变量决策和双变量决策

在双变量决策DPLL算法设计中, 我们对于决策变量的选取策略是在Minisat的决策启发式算法的基础上改进而成的。在Minisat解决器中, 其决策模块采用的是随机选取变量和基于活跃性选取变量相结合的方法来进行的。其算法流程如下: 首先, 随机选取一个变量, 如果该变量满足决策点的条件, 则确定该随机变量为下一个决策点, 其决策相位有四种情况, 分别是正相位, 负相位, 随机相位, 用户自定义相位。相位决策的方法取决于用户输入参数。如果所选取的随机变量不满足决策点条件, 比如其已被赋值, 则接下来通过基于活跃性的方法选出一个满足决策点条件并有最大活跃性的变量作为下一个决策点变量。其中变量活跃性的大小取决于该变量在学习子句中出现的次数。每纪录一个学习子句, 其中的变量的活跃值就会增加, 这样在学习子句中出现次数越多的变量, 其活跃值越高, 即越容易优先成为决策点。从整体上观察, 以上的决策方法在SAT搜索的初始阶段倾向于采用随机搜索的方法, 而随着决策深度逐渐增加, 未赋值的变量逐渐减少, 学习子句逐渐增加, 该方法逐渐以基于活跃性的变量选取为主。在搜索初期以随机策略为主选取变量策略对于重启动策略来讲是必需的, 也是至关重要, 它避免了重启动后将搜索方向引向以前已走过的路径上。而之后以活跃性为主的决策策略增加了SAT搜索的效率。

在以上变量选取策略的基础上, 修改后的决策模块的主要思想是首先通过以上方法找到一个未赋值的变量, 然后再试图找到含有该变量的一个未满足的二元子句, 如果能找到这样一个二元子句, 则同时对该子句的两个文字赋值为“1”, 此即所谓的双决策点策略。如果在一定范围内找不到二元子句, 则采用传统的单变量赋值策略进行赋值。在算法实现中, 程序设置了一个二元子句搜索次数限制, 若超过此限制次数还不能找到二元子句, 则采用单决策变量决策策略。一般给定的工业界的SAT实例中往往包含有许多的二元子句, 并且随着蕴含推理的进行又会不断地产生二元子句, 因此在决策时常能够采用双变量决策策略。在查找含有某个变量的二元子句时, 我们采用的是双观察指针的方法 (见第三章), 这样能高效地发现二元子句。

### 5.3 蕴含推理模块

从上一节介绍的双变量决策策略可以发现, 在文字推理的过程当中除了传统

的布尔约束推导之外还必须增加动态地等价变量化简和蕴含推理工作。由于SAT Solver计算所花费的时间大部分（大约80%以上）是花在蕴含推理模块上的，因此如何高效地进行等价化简和蕴含推理是双变量决策算法成功的一个关键。在程序设计中为了便于将来回溯的方便和快捷，我们采取了等价文字松耦合的指导思想，即在处理等价变量的时候不对子句库中的含有等价文字的子句作任何化简操作，只是记录下各文字间的等价关系用以将来的蕴含推理，并充分利用已有的蕴含推理算法进行文字推理。为此我们总结了几种情况进行等价化简推理。

以发现等价关系 $a=b$ ,  $\sim a=\sim b$ 为例（“ $\underline{a}$ ”代表文字 $a$ 是观察变量）：

- a. 对于那些同时含有观察变量 $a, b$ 的子句，如 $(\underline{a}, \underline{b}, \dots)$ 或 $(\sim \underline{a}, \sim \underline{b}, \dots)$ ，（值得注意的是在本程序中，一个子句“ $c$ ”里的两个观察文字始终处于该子句的头两个文字的位置）找到该类子句，并检查该子句中是否还有其他未赋值的非等价的变量，若找到其他非等价的未赋值变量，则将其中的一个观察变量与所找到的变量交换位置，使所找到的变量成为观察变量。若该子句中没有其他独立未赋值的变量，则发现当前子句为一元子句（由于变量等价引起的），立即将相应文字加入赋值队列，在此例中即为 $a$ （或 $\sim a$ ），并且把所有与 $a$ （或 $\sim a$ ）等价的文字一并赋值。
- b. 对于那些仅含有等价变量中的一个作为观察变量的子句，比如子句 $(\underline{a}, \underline{x}, \dots)$ 当给其他变量“ $x$ ”赋值推理时，若在给 $c$ 赋值为0后文字 $b$ （或 $a$ ）成为另一个观察变量，并且该子句中再没有其他未赋值的非等价变量，则发现了一元子句，立即将所有与文字 $a$ 有等价关系的文字加入赋值队列，一并赋值。

## 5.4 冲突分析和回溯模块

双变量决策算法的冲突分析模块相对于单变量决策的冲突分析模块作了一定扩展和改动，其主要也是采用第一蕴含点（First Unit Implication Point）的冲突分析算法对于冲突子句进行分析，收集理由产生学习子句。然而与传统的单变量决策算法不同的是，双变量决策算法进一步添加了文字等价关系的理由收集工作。如何高效地收集文字间等价关系的理由，即如何高效地找到最小的文字赋值组合使得给定的文字间产生等价关系，是双变量决策冲突分析模块设计中的难点，这也是决定双变量决策算法是否高效的另一个关键所在。为解决这个问题，初步的想法是在每次发现文字等价关系时就记录相应的导致该等价关系的原因

子句，正如同传统的记录每个变量的赋值的原因为子句一样。举例说明，对于等价关系“ $a=b$ ”，其可以由两个子句产生，如  $(a + \sim b + X)$  和  $(\sim a + b + Y)$ ，当  $X=0$ ， $Y=0$  时，等价关系“ $a=b$ ”即可成立，换句话说“ $X=0$ ， $Y=0$ ”是导致“ $a=b$ ”的直接原因，需要在添加  $a$ ， $b$  间等价关系的同时相应地记录下来。这样做将会使得在冲突分析中不仅可以对于 BCP 蕴含文字进行冲突分析而且还能够对于等价文字进行冲突分析，找到导致该冲突的根源。在第三章中我们已经提到，子句学习和非同步回溯技术是使得当前 SAT 解决器性能提高的最重要的技术因素之一。之所以进行冲突分析，目的也就是为了获得学习子句，并进行相应的非同步回溯。从前文的分析可以看出，由于可以对等价文字进行类似的冲突分析，双变量决策算法同样可以继承以往的子句学习和非同步回溯技术。举例说明：

$(A + B + R1)(\sim A + \sim B + R2) \xRightarrow{R1=R2=0} A = \sim B$ ，则该等价关系成立的原因是  $R1=0$  并且  $R2=0$ ，换句话说，在将来在冲突分析过程中，如果在向前推理时发现“ $A = \sim B$ ”等价关系不成立，则可以学习到一个新的子句： $(R1 + R2)$ ，即文字  $R1$  和  $R2$  不能同时赋值为 0。如果“ $A = \sim B$ ”等价关系保持成立，没有遇到冲突，并且当以后由于  $(A + \sim B + X) \xRightarrow{X=0} A = 1$  时，则“ $A=1$ ”的原因子句即为  $(A + R1 + R2 + X)$ ，即  $A=1$  成立的原因是由于文字“ $R1$ ， $R2$ ， $X$ ”的值均为 0。进一步如果将来出现的冲突的原因均是由于  $A=1$  引起的话，或则说， $A=1$  是唯一蕴含点，那么由冲突学习到的学习子句应为  $(\sim A + R1 + R2 + X)$ ，而非同步回溯的决策级数即是“ $R1$ ， $R2$ ， $X$ ”三个文字中所具有的最大的决策级数。

在程序中回溯模块的功能即是把当前的状态恢复到指定的某一决策层，并消除某些变量间的等价关系。当然对于回溯后可以立即产生一元子句的情况，则立即进行例行的一元推导。若回溯后无法产生一元子句则说明冲突是由于双决策点共同造成的，需要分两种情况讨论。如果仅回溯到当前决策层的上一层，则说明原先的双决策变量的同时为“1”的赋值导致冲突，必须进行等价文字化简（发现双决策变量文字的等价关系）。如果回溯到更高的决策层，则需要回溯到相应层后对学习子句中的两个未赋值的文字进行双变量决策。

## 5.5 数据结构

双变量决策算法相比较于传统的单变量决策算法添加了动态的等价变量的记录 and 推理工作。为此需要设计额外的数据结构来记录和管理动态的等价变量。为了高效地支持本文所提出来的双变量决策算法，我们提出了以下用以处理等价化简推理的数据结构的具体要求：

- 1、对于给定两个文字，必须能高效的判断出是它们否为等价文字。
- 2、对于文字间的等价关系必须要按照决策级数有序的存放，并支持动态地添加和回溯操作。
- 3、支持访问某一给定文字的所有等价文字。

基于以上想法我们认为在DPLL算法框架内实现等价化简推理，需要给每个变量增设等价文字组存储单元，并且还要在已有的数据结构的基础上，再添加一个专门的等价文字数据库。具体设计的数据结构形式如图5-3所示。

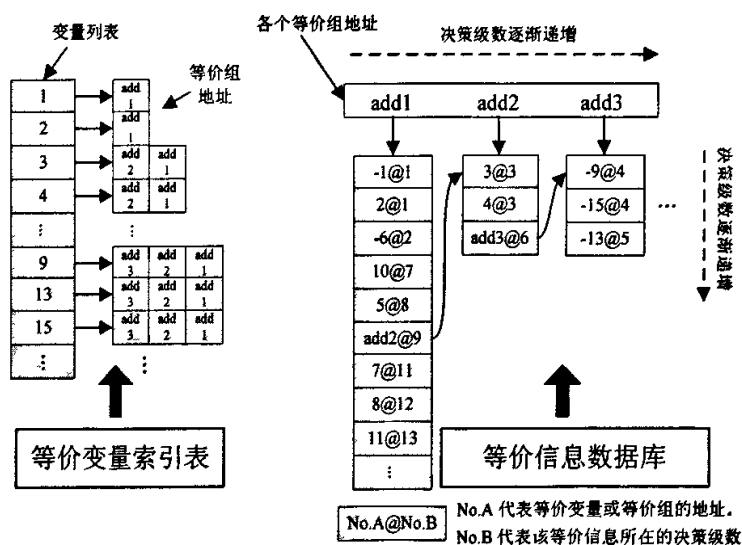


图 5-3 用于处理等价化简推理的数据结构

如图5-3所示，在以上数据结构中，我们给每个变量设置了一个动态可变长的向量存储空间，用来存储等价变量的信息。变量所指的向量存储空间中存储的是该变量所在的等价文字组的地址，其用来确定各文字间的等价关系。其中具体的地址的结构是由32位整数结构组成，32位的最后一位用来表示等价文字的相

位, 取值为{0, 1}。而前面31位数记录的是等价组所在的位置。其中最后一位取“0”代表相应等价组存储的是该变量的正相位文字, 取“1”则代表等价组存储的是该变量的反相文字。每次检查文字间的等价关系的时候, 均是以向量存储空间最后一个存储的地址为准。当有两个不同的等价组之间新产生了等价关系后, 如组A和组B, 则决策层数较大的等价组B中的等价变量对应所指的向量存储空间要新添加入A组的地址, 这样所有B组中的变量又可以归为A组的等价变量。

而对于等价文字数据库的设计, 我们也采用了动态二维数组的结构形式来存储等价文字的信息。其中内层的数组存储的是等价文字结构体。每个文字结构体是由三个元素构成, 其中的两个元素是两个整形数, 分别存储等价变量的文字号和决策级数。另外一个元素是子句地址的集合, 存储的是等价文字被添加入等价数据库时的子句理由, 即一些子句地址的集合。其中对于存储“文字号”的32位整形数的结构由两部分组成, 32位中的最低位为指示位, 当为“0”时, 表示高31位存储的是“文字索引号码”, 而当指示位为“1”时, 高31位存储的是“等价组索引地址”。这样一个32位的整形数就能同时表示“文字索引号码”和“等价组索引地址”的信息。

## 5.6 实验结果

表 5-1 双变量决策算法实验数据

SAT 实例	变量数	子句数	决策数	冲突数	重启动数	解决时间
rand4-1.cnf	400	1600	4100347	2882688	24	160.94 s
rand4-2.cnf	400	1600	1331860	937205	21	46.78 s
rand4-3.cnf	400	1600	1578107	1128194	22	58.35 s
rand4-4.cnf	400	1600	1131978	799663	21	40.01 s
rand4-5.cnf	400	1600	931930	659501	20	31.93 s
rand4-6.cnf	400	1600	846818	601286	20	28.88 s
rand4-7.cnf	400	1600	57590	40027	14	1.57 s
rand35-1.cnf	350	1400	1124552	838551	21	38.67 s
rand35-2.cnf	350	1400	13290	9402	10	0.32 s
rand35-3.cnf	350	1400	63017	46950	14	1.75s

表5-1展示了双变量决策算法的实验结果, 其中实验所用变量均是随机产生的SAT实例。实验结果初步验证了双变量决策算法的正确性。然而从实验中也发现了双变量决策算法中的一些缺陷和不足。首先从决策数目和冲突数目上看其数

量还是较大, 没有看到想象中双变量决策算法中决策数目和冲突数目减少的情况出现。通过仔细分析双变量决策算法我们发现在双变量决策算法中非回溯后并不能往往不能立即如单变量决策算法那样可以进行一元推理, 与单变量决策算法不同的是由于学习到的子句在回溯后往往成为二元子句, 因而无法进行一元推理, 只能进行新的决策才行。这样反而使得在搜索解的过程中决策次数会增加很多。其次, 我们还发现双变量决策算法中的等价推理的时间耗费与学习子句的数目有很强的正相关关系, 随着子句数目的增长, 等价推理的效率下降了。



## 第6章 总结与展望

本章将对本文所做的工作做系统的总结和回顾, 并对进一步值得研究的课题进行一些展望。

### 6.1 论文工作总结

尽管当前已知的所有解决SAT问题的算法的最坏时间复杂度均是指数的, 并没有多项式时间复杂度算法发现, 但无可否认可满足性问题算法在最近的十多年来还是取得了突飞猛进的发展, 随着各种启发式算法的提出以及高效的数据结构和算法实现使得可满足性问题解决器的性能越来越强, 速度越来越快, 能求解的SAT问题的规模也越来越大。这可以从一年一度的国际SAT解决器竞赛的结果中看出来。许多在前次的SAT解决器竞赛中无法得到解决的可满足性问题, 往往在下一次的竞赛中获得了解决。1997年Selman, Kautz, 和 McAllester所发表那篇著名的论文中所列出的十大有待研究解决的关于可满足性问题的课题[36], 现在已经有不少以获得解决或者是初步地解决。然而科学也是没有止境的, 一个问题的解决往往又会产生出更多新的课题。当前需要求解的可满足性SAT问题的规模越来越大, 这一方面是SAT算法研究的需要, 另一方面更多的是SAT应用领域的需要, 由于看到当前SAT求解算法高效性, 许多在工业界其他计算领域中的问题均倾向于转化成SAT问题以获得求解, 比如最优决策, 规划问题等等, 并且SAT传统的应用领域如, 形式验证, 人工智能等领域也都不断地对SAT求解器提出更高的要求。这使得当前的SAT算法研究也还面临着巨大的挑战。

本文的工作即是进一步探索新的SAT算法, 改进现有的可满足性问题求解技术, 以期解决更大规模的可满足性问题。本文的贡献主要可以归纳为以下两点。

首先, 本文提出了一种新的可满足性问题正向推理技术, 对称扩展的一元推导。基于这项新的推理技术并结合等价化简我们设计并实现了一个高效的SAT预处理器Snowball。正如前文所述通过保持一元推导的对称性质可以使传统的一元推导方法推导出更多的文字蕴涵关系, 因此对称扩展的一元推导能够很大程度上增强一元扩展的剪枝能力。实验结果验证了本文所提出的预处理算法的有效性并清晰地表明了这项对称扩展一元推导技术的强大正向推理能力。与其他复杂的正

向推理技术如利用子句分布的变量消除技术和超级分解技术等相比, 本文的对称扩展方法更简单直接, 更容易集成到DPLL算法框架内。我们相信如果这项正向推理技术能得到进一步的开发并集成到SAT解决器当中, 它将能够发挥出更大的作用, 进一步减小求解SAT问题的时间。

其次, 本文首次提出了双变量决策DPLL算法, 即在变量决策时选取两个未赋值的变量同时进行决策赋值而取代传统的单变量决策策略, 这样不仅可以提高变量决策的效率, 在理论上可大大减少SAT问题的搜索空间, 并且还能有效地把变量等价化简技术结合到DPLL算法框架内。本文较详细地阐述了双变量决策算法各个模块的设计思想, 并且还给出了针对有效解决等价文字信息的管理工作的数据结构。从算法上看, 双变量决策算法可以较好的集成已有的各项技术, 如冲突分析, 子句学习, 非同步回溯, 有效的变量选取启发式算法, 观察指针等, 因此其能很好与传统的DPLL算法相兼容。

## 6.2 对后续工作的展望

本文所包含的工作仍然是十分有限的, 并且存在有一些缺点和值得继续研究改进的地方。比如, 本文所提出的对称蕴含预处理算法从结果上看并没有对于所有的类型的SAT问题都表现得较为高效, 而是仅对于某些类型的SAT实例表现得较好。因此研究更加通用型的预处理算法和对于SAT问题普遍有效的简化技术是值得进一步改进和发展的方向。还有对于双变量决策DPLL算法, 尽管从理论上可以说明其能够集成已有的各种技术, 但是在算法实现过程中发现在双决策变量的选取规则方面还是基于传统的单变量选取规则, 并没有设计与双变量决策策略配合较好的变量选取规则, 因而导致变量选取效率不高。其次由于需要对等价关系进行冲突分析, 往往使得生成的学习子句长度较长, 因而回溯效率不如单变量决策算法高。因此如何有效地解决针对等价文字的子句学习和非同步回溯问题是将来有待改进的问题。此外, 当前国际上对于SAT实例所包含的结构信息的研究正逐渐引起理论学界的关注, 比如结构对称性[37], 最小不满足子集等。在将来的可满足性算法研究中, 可以考虑如何充分发现和利用这些SAT问题本身的结构信息, 来指导SAT问题的求解。最后值得提到的是除了SAT问题算法本身的研究之外, 广阔的SAT应用层面的研究也是值得进一步发掘的课题, 如电路等价性验

证，有界模型检查等。

当前形式验证在国内尚处于起步阶段，本文针对集成电路验证的瓶颈问题，对形式验证中的关键技术—SAT算法进行了研究，具有前沿性和广泛应用性。由于SAT问题的特殊性和重要性，除了在验证方面的应用之外，它还在EDA领域的其它方面，如布局布线，逻辑优化方面有着非常广泛应用。可以展望不断构建和开发更高效的SAT问题算法将会对EDA领域的发展起到巨大的推动作用。

## 参考文献

- [1] International Technology Roadmap for Semiconductors, Design part, 2004 update.  
<http://public.itrs.org>
- [2] 吴为民, 数字系统的形式化验证, 计算机世界报, 第 37 期, 2005
- [3] M. Davis and H. Putnam.: A Computing procedure for quantification theory. In: J. of the ACM, 7 (1960)
- [4] M. Davis, G. Logemann, D. Loveland. "A machine program for theorem-proving", Communications of ACM, Vol. 5, No. 7, pp. 394-397, 1962
- [5] 丁敏, 可满足性问题算法研究以及在时序电路等价验证中的应用, 博士论文, 复旦大学, 信息与工程学院微电子系, 2005
- [6] Inês Lynce, Propositional Satisfiability: Techniques, Algorithms and Applications, Ph.D. Dissertation, Technical University of Lisbon, Portugal, 2004
- [7] M.R. Garey, D.S. Johnson, 张立昂 (译), 《计算机和难解性: NP 完全性理论导引》, 科学出版社, 1987 年 1 月
- [8] J. Gu, P.W. Purdom, J. Franco, "Algorithms for the Satisfiability (SAT) Problem: A Survey", DIMACS Series on Discrete Mathematics and Theoretical Computer Science 35, pp. 19-151, 1997
- [9] J. Gu, "Local Search for Satisfiability (SAT) Problem", IEEE Trans. On Systems, Man, and Cybernetics Vol. 23, No. 4, pp. 1108-1129, 1993
- [10] J. Frank, "Weighting for Godot: Learning Heuristic for GSAT", Proc. Of National Conference on Artificial Intelligence, pp. 1996
- [11] Y. Shang, B.W. Wah, "A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems", Journal of Global Optimization, Vol. 12, pp. 61-99, 1998
- [12] B. Selman, H.A. Kautz, B. Cohen. "Noise strategies for improving local search", Proc. of the 12th National Conference on Artificial Intelligence, pp. 337~343, 1994
- [13] L. Simon, D. Le Berre, and E. Hirsch.: The SAT2002 Competition. Accepted for publication in Annals of Mathematics and Artificial Intelligence (AMAI), 43, pp. 343~378, 2005
- [14] R. Puri, J. Gu, "A BDD SAT Solver for Satisfiability Testing: An Industrial Case Study", Annals of Mathematics and Artificial Intelligence, Vol.17, No.3-4, pp.315-337, 1996
- [15] M. Sheeran, G. Stålmarck. A tutorial on Stålmarck's proof procedure [C]. Proc. Of Formal Methods in Computer-Aided Design, 1998
- [16] J.G. Groote, J.P. Warners. The Propositional Formula Checker Heerhugo [R]. Netherlands: Stichting Mathematisch Centrum (SMC), 1999
- [17] J.P. Marques-Silva, K.A. Sakallah, "GRASP: A Search Algorithm for Propositional

- Satisfiability", IEEE Trans. on Computers, Vol 48, pp. 506–521, 1999
- [18] R.J. Bayardo, R.C. Schrag, "Using CSP Look-back Techniques to Solve Real world SAT Instances", Proc. Of 14th National Conference on Artificial Intelligence, pp. 203-208, 1997
- [19] H. Zhang, "SATO: An Efficient Propositional Prover", Proc. Of 14th International Conference on Automated Deduction, Vol. 1249 of Lecture Notes in Computer Science, pp. 272-275, 1997
- [20] L. Zhang, C. Madigan, M. Moskewicz, et al. "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver", Proc. Of International Conference on Computer-Aided Design, pp. 279-285, 2001
- [21] E. Goldberg, Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver", Proc. Of Design Automation and Test in Europe, pp. 142-149, 2002
- [22] Niklas Een and Niklas Sorensson: An Extensible SAT-solver. In: Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT), pp. 502–518, 2003.
- [23] L. Drake, A. Frisch, I. Lynce, J. Marques-Silva and T. Walsh.: Comparing SAT preprocessing techniques. In: 9th Workshop on Automated Reasoning, April 2002.
- [24] R. I. Brafman.: A simplifier for propositional formulas with many binary clauses. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 515–522, 2001
- [25] S.Subbarayan and D.Pradhan.: NiVER: Non Increasing Variable Elimination Resolution for Preprocessing SAT instances. In: Prel. Proc. SAT'04.
- [26] Niklas Een and Armin Biere.: Effective preprocessing in SAT through variable and clause elimination. In: SAT'05, pp. 61–75, 2005.
- [27] Chu Min Li.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Proceedings of AAAI-2000, pp. 291-296, Austin, Texas, USA, July 2000.
- [28] F. Bachhus and J. Winter.: Effective preprocessing with Hyper-Resolution and Equality Reduction. In: SAT'03, pp. 341-355, 2003.
- [29] Roman Gershman and Ofer Strichman.: Cost-Effective Hyper-Resolution for Preprocessing CNF Formulas. In: SAT'05, pp. 423-429, 2005.
- [30] R. Jeroslow, J. Wang, "Solving Propositional Satisfiability Problem", Annals of Mathematics and AI, pp. 167-187, 1990
- [31] O. Dubois and G. Dequen: A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01), pp.248-253, Seattle, Morgan Kaufmann 2001

- [32] 荆名娥, 周电, 唐璞山, 启发式极性决策算法解 SAT 问题, 中国科学, 已录用, 编号: 112006-841
- [33] Inês Lynce, J.P. Marques-Silva.: Efficient Data Structures for Fast SAT Solvers. Technical Report, RT/05/2001, Cadence European Lab. 2001
- [34] L. Zhang, S. Malik.: The Quest for Efficient Boolean Satisfiability Solvers. In: Proc. Of 14th International Conference on Computer-Aided Verification, Vol. 2404 of Lecture Notes in Computer Science, pp. 200-215, 2002
- [35] M. Velev.: Efficient translation of boolean formulas to CNF in formal verification of microprocessors. In: Proc. ASP-DAC'04.
- [36] B. Selman, H.A. Kautz, D.A. McAllester, Ten challenges in propositional reasoning and search, In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), 1997, pp. 50–54.
- [37] F. Aloul, A. Ramani, I. Markov, and K. Sakallah.: Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetry. In: IEEE Transactions on Computer Aided Design, 22(9), pp. 1117-1137, Sept. 2003.

## 硕士期间撰写或参与撰写的论文

1. 熊伟, 唐璞山 “一种新的 SAT 问题预处理算法”, 《微电子学与计算机》, 已录用, 2008 年第一期发表

## 致 谢

在本论文工作即将完成之际,我首先衷心感谢我的导师唐璞山教授在我攻读硕士三年期间对我的悉心指导和启发。唐老师扎实的学术功底,积极创新的开拓精神,严谨治学的科学态度和永不言倦的工作热情为我和所有的学生都树立了一个光辉的榜样。从本论文课题的确立,研究工作的进行,研究成果的撰写与发表,一直到最终论文的完成,其中每一步都离不开唐老师的指导和帮助。从唐老师那里学到的研究方法和技能,更重要的是他的那种严谨的治学精神和高尚的品德将会成为我今后的学习和工作中不可缺少的宝贵财富。

同时我还要感谢复旦大学的荆明娥老师向我提供了需多有价值的资料以及讨论交流机会,并在 SAT 方面给与了我许多有建设性的意见及建议。

此外我还要感谢复旦大学的丁敏、柯宪明、吴洋、张忠林、黄伟等学长对我的大力帮助和指导,感谢他们对于我研究生涯提出了许多很好的建议,并给我树立了很好的榜样。同时感谢童家榕教授、曾璇教授、黄均鼐教授、赵文庆教授、谭道珍老师等其他 CAD 实验室老师的对于我在生活上的帮助和关怀。

当然,我还要感谢所有的好朋友一直给予我的鼓励和支持。

最后,谨以此文献给我亲爱的父母。



作者：[熊伟](#)  
学位授予单位：[复旦大学](#)

## 相似文献(3条)

### 1. 期刊论文 [张忠林, 唐璞山, ZHANG Zhonglin, TANG Pushan](#) 一个基于可满足性算法的时序深度计算方法 -计算机工程2006, 32(2)

有界模型校验(Bounded Model Checking)由于验证的不完备性而经常受到验证人员的指责. 为了解决这个问题, 计算时序深度的算法被提出. 该文算法基于可满足性算法引擎, 与其它基于可满足性算法引擎的算法不同, 为了减少可满足性算法引擎的负担, 采用了状态空间显式存储的方法. ISCAS' 89的实例很好证明了该算法的有效性.

### 2. 会议论文 [马海涛, 郝忠孝](#) 一种检验Active XML文档树模式查询可满足性算法 2008

检验查询可满足性是XML文档查询的一个重要问题. Active XML (AXML) 文档在XML文档中引入嵌入式Web服务, 增强了文档的动态性和灵活性, 同时也为现有文档查询可满足性问题的解决方法提出了新的要求和挑战. 研究了模式约束下的AXML文档查询可满足性问题, 给出了AXML查询可满足性问题的形式化定义, 基于树自动机理论, 针对XPath树模式查询片段  $\{"/, //, []^*\}$ , 提出了一种多项式时间的AXML文档查询可满足性检验算法. 实验数据表明, 提出的算法在查询过程中以较小的代价显著地节省查询处理时间.

### 3. 学位论文 [丁敏](#) 可满足性问题算法研究以及在时序电路等价验证中的应用 2005

本文基于形式验证作为传统模拟验证的补充越来越受到重视这一背景, 进行了可满足性算法的研究和时序电路等价验证的研究. 提出了具有动态删除策略的改进了的失败性文字检查 (FLD) 过程. 减小了实际的DPLL算法的搜索空间, 因而提高了DPLL算法的总体运行速度; 提出了单独使用可满足性算法作为引擎的基于时间帧展开的时序电路等价验证算法. 该算法的特点有: 不存在内存增长过快的问题. 提高了引擎在迭代使用时候的效率; 算法在结合了数学归纳法、不可满足子集提取和结构不动点计算这三种已有的技术基础后, 更适合用于时序深度较深的逻辑层时序电路的等价验证. 而且, 在整合三种技术后, 又提出了一些改进措施. 从实验分析表明, 这些改进措施在有效降低可满足性算法引擎的计算时间同时, 也进一步减缓了内存的增长, 优于单独使用数学归纳法的时序等价验证.

本文链接: [http://d.g.wanfangdata.com.cn/Thesis\\_Y1168766.aspx](http://d.g.wanfangdata.com.cn/Thesis_Y1168766.aspx)

授权使用: 中山大学(zsdx), 授权号: 15ff723e-5335-4853-9e40-9e2b0011237b

下载时间: 2010年11月11日