软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(8):3229−3256 [doi: 10.13328/j.cnki.jos.007489] [CSTR: 32375.14.jos.007489]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

增量 SAT/SMT 问题的求解与应用综述*

李博涵[1,2], 蔡少伟[1,2]

[1](基础软件与系统重点实验室 (中国科学院 软件研究所), 北京 100190)

[2](中国科学院大学 计算机学院, 北京 100049)

通信作者: 蔡少伟, E-mail: caisw@ios.ac.cn

摘　要: 命题可满足性问题 (propositional satisfiability problem, SAT) 和可满足性模理论问题 (satisfiability modulo theories problem, SMT) 是重要的计算机科学基础问题, 其在电路设计, 软件分析验证等领域都有着重要应用, 并且目前已有大量工作对其求解技术进行研究. 在实际应用场景中, SAT/SMT 求解器通常需要求解一系列互相紧密联系的公式. 相比于每次都调用独立的求解器重新求解, 增量求解技术可以复用之前搜索得到的信息, 包括之前的求解结果以及学习子句等, 从而有效提高了求解效率. 目前, 增量 SAT/SMT 求解已经受到广泛重视与研究, 并成功应用于有界模型检测, 符号执行, 最大可满足性问题等领域中. 对增量 SAT/SMT 的求解技术进行详细综述与梳理, 涵盖了完备与非完备算法. 此外, 详细总结增量 SAT/SMT 求解技术在实际场景中的主要应用. 最后, 对该领域的发展方向进行总结和展望.

关键词: 命题可满足性问题; 可满足性模理论问题; 增量求解

中图法分类号: TP301

中文引用格式: 李博涵, 蔡少伟. 增量SAT/SMT问题的求解与应用综述. 软件学报, 2026, 37(8): 3229–3256. http://www.jos.org.cn/1000-9825/7489.htm

英文引用格式: Li BH, Cai SW. Review on Solving Techniques and Applications for Incremental SAT/SMT Problems. Ruan Jian Xue Bao/Journal of Software, 2026, 37(8): 3229–3256 (in Chinese). http://www.jos.org.cn/1000-9825/7489.htm

Review on Solving Techniques and Applications for Incremental SAT/SMT Problems

LI Bo-Han[1,2], CAI Shao-Wei[1,2]

[1](Key Laboratory of Systems Software (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: The propositional satisfiability problem (SAT) and the satisfiability modulo theories problem (SMT) are fundamental problems in computer science, with significant applications in circuit design, software analysis and verification, and other fields. At present, extensive research has been conducted on their solving techniques. In practical applications, SAT/SMT solvers often need to solve a series of closely related formulas. Compared to solving each problem from scratch using an independent solver, incremental solving techniques can reuse previously obtained search information, including previous solutions and learned clauses, thus effectively improving solving efficiency. Currently, incremental SAT/SMT solving has received extensive attention and research, and has been successfully applied in fields such as bounded model checking, symbolic execution, and the maximum satisfiability problem (MaxSAT). This study provides a detailed review and categorization of incremental SAT/SMT solving techniques, covering both complete and incomplete algorithms. In addition, the applications of incremental SAT/SMT solving techniques in practical scenarios are comprehensively summarized. Finally, the development directions in this field are summarized and discussed.

Key words: propositional satisfiability problem (SAT); satisfiability modulo theories problem (SMT); incremental solving

* 基金项目: 国家重点研发计划 (2023YFA1009500)
收稿时间: 2025-01-09; 修改时间: 2025-04-21; 采用时间: 2025-06-11; jos 在线出版时间: 2025-11-05
CNKI 网络首发时间: 2025-11-06

命题可满足性问题 (propositional satisfiability problem, SAT) 是用于判断布尔逻辑公式的可满足性问题, 该问题是重要的计算机科学基础问题, 也是最早被判定为 NP Complete 的问题. 该问题在 EDA 设计, 密码学等领域都有着重要应用[1]. 可满足性模理论问题 (satisfiability modulo theories problem, SMT) 将 SAT 问题拓展到了特定理论背景下的一阶逻辑公式中, 具有更强的表达能力. 其在软件分析, 形式化验证领域都起到了重要应用[2,3]. 由于 SAT/SMT 问题的重要应用, 目前已经有大量研究投入 SAT/SMT 的求解技术开发, 主要可以被分为完备算法和不完备算法. 完备算法是基于回溯和约束传播的系统搜索, 其会利用预处理技术有效化简公式, 并在搜索过程中通过冲突分析得到学习子句用于指导搜索; 不完备算法基于局部搜索等技术, 可以在大规模随机实例中高效找到可满足解.

在实际应用场景中, SAT/SMT 求解器通常会被反复多次调用, 用于求解一系列互相接近的公式. 如果每次都重新调用新的求解器单独求解, 则会丢失之前求解得到的信息, 例如上一次搜索过程中得到的学习子句, 历史解信息等, 从而导致求解效率下降. 因此, 增量 SAT/SMT 会尝试复用这些信息来提升求解效率. 目前增量 SAT/SMT 的求解技术已经得到广泛关注, 主流的 SAT/SMT 求解器均支持增量求解. 每年的 SMT 国际比赛 (https://smt-comp.github.io/) 中都设置了专门的增量求解赛道. 在 2015–2020 年间, SAT 国际比赛 (https://satcompetition.github.io/) 也多次设置了专门的增量求解赛道. 此外, 最近的对比研究[4]显示, 非增量 SAT 求解领域所提出的多项重要技术对于增量实例的求解并没有显著帮助, 为此 SAT 社区近年来提出: “对 SAT 求解器的评估应考虑其在增量求解方面的性能改进”, 这也更印证了此类增量问题的重要意义.

增量 SAT/SMT 求解技术也可以被分为完备算法和不完备算法. 完备算法是在传统的非增量 SAT/SMT 算法的基础上衍生而来的. 它主要面临如何在删除公式后复用之前的学习子句的问题. 针对该问题, 主要的解决方案包括: 记录下学习子句的归结来源, 以及添加假设文字. 基于假设的方式更为轻量化, 也被主流的增量 SAT 完备求解器所广泛采用. 而不完备算法主要集中在对增量 SAT 问题的求解, 并基于以下直觉: 对公式增量修改后, 大部分约束仍然可满足. 因此, 不完备算法关注如何充分利用上一轮的求解结果, 并通过对上一轮求解的结果进行微调, 从而得到新公式的解. 目前, 增量 SAT/SMT 求解技术的研究重点在于: 如何将化简技术和增量求解有效结合. 针对该问题, 主要的解决方案包括: 冻结部分变量, 利用前瞻信息, 将假设作为单元子句, 以及恢复化简操作等. 在广泛采用的增量 SAT/SMT 测试集上, 我们对当前主流的领先 SAT/SMT 增量求解器进行了对比实验.

增量 SAT 问题被广泛应用于电路设计[5−7]和模型检查[8,9]等实际问题中, 此外, 基于假设的增量 SAT 求解方法也被作为基础能力, 用于支持不可满足集合提取[10,11]和最大可满足性[12]问题等其他逻辑问题. 增量 SMT 主要被用在软件分析验证等领域, 包括符号执行[13]、神经网络验证[14]等.

目前国内外关于增量 SAT/SMT 问题的研究综述很少. 本文详细综述增量 SAT/SMT 问题的研究现状, 梳理该类问题的各种求解算法, 并总结其实际应用场景. 第 1 节介绍 SAT/SMT 的背景知识, 包括其定义和求解技术概述. 第 2 节介绍增量 SAT 问题的求解技术, 包括完备算法和不完备算法. 第 3 节介绍增量 SMT 问题的求解技术. 第 4 节介绍增量 SAT/SMT 的实际应用场景. 第 5 节总结并给出对该方向研究的展望.

1 背景知识

1.1 SAT 问题定义

SAT 问题包含一系列命题变量 (propositional variable), 这些命题变量可以被唯一赋值为真 ($\top$) 或假 ($\bot$). 一个命题文字 (literal) 可以是一个命题变量, 或该变量的非 (形如 $\neg p$). 一个子句 (clause) 是一组命题文字的析取 (形如 $l_1 \vee l_2 \vee \ldots \vee l_k$ 或记作 $(l_1, l_2, \ldots, l_k)$), 当且仅当其中至少有一个文字被满足时子句满足. 空子句为不含任何文字的子句, 不可满足, 一般用于表示冲突. 合取范式 (conjunctive normal form, CNF) 公式是一系列子句的合取 (形如 $c_1 \wedge c_2 \wedge \ldots \wedge c_m$), CNF 公式满足当且仅当其中所有子句都满足. 给定一组命题变量 $V = \{p_1, p_2, \ldots, p_n\}$ 组赋值 (truth assignment) 是一组 $V \to \{\top, \bot\}$ 的映射. 给定一组 CNF 公式 ϕ, 如果存在一组赋值 α 能使得 ϕ 满足, 则 ϕ 是可满足的, 且 α 是 ϕ 的解 (model), 若不存在一组赋值使得 ϕ 满足, 则 ϕ 是不可满足的. SAT 问题是要判断给定逻辑公式是否可满足.

给定一组 CNF 公式 $\phi = c_1 \wedge c_2$, 其中 $c_1 = (p_1 \vee p_2)$, $c_2 = (\neg p_1 \vee \neg p_3)$, 存在一组赋值 $\alpha = \{p_1 = \top, p_2 = \bot, p_3 = \bot\}$ 使得 ϕ 满足, 因而 ϕ 是可满足的.

1.2 SMT 问题定义

SMT 问题将 SAT 问题拓展到了支持特定背景理论 (background theory) 下的一阶逻辑领域. 相比于 SAT 问题, SMT 问题有更强的表达能力. 在 SMT 公式中, 除了命题变量, 还包含理论变量. 理论原子公式 (theory atomic formula) 由理论变量组成, 例如在算数运算中, 理论原子公式是算数变量组成的不等式. 一个原子公式 (atomic formula) 可以是一个理论原子公式或一个命题变量. SMT 中的文字是原子公式或它的非. SMT 公式也会以 CNF 形式表示. SMT 公式的赋值是一组对命题变量和理论变量到其值域的映射. SMT 问题是要找一组赋值使得公式满足, 如果不存在则判定该 SMT 公式不可满足.

例 1: 给定算术理论的 SMT 公式 $\phi = c_1 \wedge c_2$, 其中 $c_1 = (p_1 \vee p_2)$, $c_1 = (\neg p_1 \vee (x - y \geqslant 0))$, 其中 p_1, p_2 是命题变量, x, y 是理论变量 (算术变量), $(x - y \geqslant 0)$ 是理论原子公式. 赋值 $\alpha = \{p_1 = \top, p_2 = \top, x = 0, y = 0\}$ 是公式的一组可满足赋值, 则 ϕ 是可满足的.

1.3 增量 SAT/SMT 问题定义

增量 SAT/SMT 问题是要处理一系列相关的 SAT/SMT 公式 $\{\phi_1, \ldots, \phi_k\}$, 需要判断其中每个公式是否可满足. 这些连续的公式之间满足以下关系:

$$\phi_{i+1} = (\phi_i - \rho_i) \cup \Delta_i,$$

其中, ρ_i 和 Δ_i 分别代表将公式 ϕ_i 转化为公式 ϕ_{i+1} 时, 需要向公式中删除和添加的子句集合.

1.4 SAT 问题主流算法

完备算法: DPLL 算法[15]是 SAT 求解的经典算法, 基于回溯搜索和单元传播: 算法会选择一个未赋值的命题变量并给它决策赋值 (decide), 每个决策变量会记录下其对应的决策层, 然后通过单元传播 (unit propagation) 进行推理: 若存在某子句中只有一个未赋值的文字且其他文字均为假, 则该文字必须为真. 如果在某次决策之后, 由单元传播推导出了冲突 (某子句中的所有文字均为假), 就会尝试当前决策变量的另一个赋值, 当该决策变量的两种赋值都尝试过, 且都会导致冲突时, 就向前回溯 (backtracking), 找到上一个没有尝试过两种赋值的决策变量, 改变该变量的赋值, 再继续单元传播; 如果决策赋值并进行单元传播后, 没有冲突且有未赋值的变量, 则会选择另一个未赋值的变量对其进行决策赋值. 当所有子句都满足时则返回可满足, 当没有未尝试决策的变量且仍未找到解时返回不可满足.

冲突驱动的子句学习 (conflict driven clause learning, CDCL) 算法[16,17]在 DPLL 的基础上引入了冲突分析和非时序回溯机制, 优化了搜索过程. 具体而言, 当算法遇到冲突时, 会进行冲突分析, 并通过归结方式得到学习子句. 学习子句是由原公式蕴含的, 可以用来帮助搜索进行剪枝并避免陷入同样的冲突. 利用冲突分析, CDCL 可以回溯到更早的变量决策点 (即非时序回溯), 从而加速搜索. 此外, CDCL 算法还引入了重启策略、变量活跃度以及文字块距离等启发式策略来指导搜索. 目前, 主流的现代 SAT 求解器[18−21]均基于 CDCL 算法.

不完备算法: 该方法主要基于局部搜索技术[22−24], 它会从一个完整的初始赋值开始, 通过对命题变量迭代修改来寻找一个可以使所有子句都满足的解. 由于该方法一般不存储历史搜索信息, 无法保证完整遍历搜索空间, 所以不能证明公式的不可满足性. 不完备算法在大规模随机实例中效果较好.

化简技术: 主要的化简技术包括有界变量消除 (bounded variable elimination)、包含 (subsumption)、自包含 (self-subsumption)[25]和阻塞子句删除 (blocked clause elimination)[26]等. 化简技术可以通过删除命题变量或子句来有效削减 SAT 公式规模. 以下文中涉及的有界变量消除技术为例: 如果关于变量 v 做归结操作产生的归结项数量不超过原公式中包含 v 的子句数量, 则将原公式中包含 v 的子句会被替换为归结项. 例如公式 $\phi = (p_1 \vee p_2) \wedge (\neg p_1 \vee p_3)$, 通过变量消除化简技术消除变量 p_1, 可以得到化简后的公式为 $\phi = (p_2 \vee p_3)$.

有关 SAT 求解的详细介绍可以参考文献 [27,28].

1.5 SMT 问题主流算法

SMT 求解的方法通常依赖于 SAT 求解器的能力. 根据对理论原子公式的编码方式不同, SMT 求解的方式可以被分为惰性方法和积极方法: 惰性方法[29,30]又被称为 DPLL(T) 方法[31], 该方法是 SMT 求解领域的主要进展, 它会将理论原子公式抽象为新的命题变量, 从而将 SMT 公式变为一个 SAT 公式. SAT 求解器会被用于对抽象后的布尔公式进行推理和求解, 然后理论求解器 (theory solver) 会接受由 SAT 求解器确定的原子公式取值, 并执行决策过程来求解这些原子公式的合取, 包括一致性检测和理论推理等. 目前该方法被主流 SMT 求解器广泛采用. 在积极方法[32,33]中, 根据理论变量之间的关系, SMT 公式会被转化为一个可满足性等价的布尔公式, 并直接交由 SAT 求解器进行求解.

局部搜索算法也被应用于 SMT 的求解之中, 该方法直接对命题变量和理论变量进行修改, 并在算术理论[34−37]和位向量理论[38,39]的 SMT 问题中展示出优秀的求解效果.

有关 SMT 求解技术的详细介绍可以参考文献 [40,41].

2 增量 SAT 问题的求解技术综述

本节将首先介绍针对增量 SAT 问题的完备算法, 该方向主要关注删除子句后如何管理学习子句. 接着, 介绍增量 SAT 问题的不完备算法, 该方向主要关注如何充分利用历史解信息, 从而在当前解附近寻找新公式的解. 然后, 介绍如何将化简技术[42]有效适配到增量求解中, 该方向也是增量 SAT 求解领域的研究重点. 随后, 将介绍增量 SAT 问题的其他形式以及针对这些问题特性设计的算法. 最后, 将依据最近的 SAT 国际比赛, 展示各主流领先增量 SAT 求解器的对比情况.

2.1 增量 SAT 的完备算法

增量 SAT 问题的完备算法最早是由 Hooker 在文献 [43] 中提出的, 该方法是基于经典的 DPLL 算法[15,44−46], 它只支持每次添加一个新子句 C 到公式集合 ϕ 中, 而不支持子句的删除. 文献 [47] 拓展了该方法, 允许单次添加多个子句.

然而, 基于 DPLL 的完备算法较少. 随着 20 世纪末 CDCL 算法的兴起, SAT 求解器的能力得到了迅速的发展, 目前的主流完备求解器均采用该方法. 基于 CDCL 算法的增量 SAT 求解器也相继被设计出来, 它们会尝试在后续的搜索中复用之前的求解信息, 包括学习子句和各类搜索启发式信息等. 本节将主要介绍基于 CDCL 算法的增量 SAT 求解技术.

基于 CDCL 的完备求解技术将主要围绕以下问题展开: 如何在删除子句时管理学习子句?

由于冲突分析会基于原公式中的子句归结得到学习子句, 删除子句后, 由被删除子句归结得到的学习子句会失效, 因为它无法再通过该删除子句归结得到, 因此它不能在后续求解中被复用.

例 2: 设原公式 ϕ 包含子句 c_1, c_2, c_3, 通过冲突分析得到了学习子句 c_l, 其中 c_l 是由 c_1 和 c_2 归结得到. 当删除子句 c_1 后, 相应的学习子句 c_l 无法再从 c_1 中归结得到, 因此 c_l 会失效, 无法在后续求解中被复用.

基于 CDCL 的增量 SAT 求解技术将主要围绕该问题展开. 在删除子句后管理学习子句的主要手段可以被分为以下 2 种.

(1) 维护学习子句归结来源.

(2) 基于假设的增量 SAT 求解.

前者需要动态维护子句之间的归结关系, 因此会带来较大的计算开销. 后者会通过引入新变量 (即假设) 来管理子句删除, 更加轻量化, 也被目前的主流增量 SAT 求解器所广泛采用. 本节最后我们会介绍如何在实际增量场景中提升基于假设的增量求解效率.

2.1.1 维护学习子句归结来源

该方法会在冲突分析的过程中确定学习子句和原公式中子句之间的归结关系. 由于冲突分析是根据归结原理进行的, 因此可以追踪这些学习子句的归结来源, 并将其记录为一个有向无环图, 其中的节点为子句. 如果在冲突

分析中由子句 α_j 和 α_k 归结得到子句 α_i, 则会有从节点 α_j 和 α_k 到节点 α_i 的边. 如果在后续的增量求解过程中要将一条子句移除, 则由该子句归结得到的所有学习子句都要被相应地移除, 即归结来源图上该子句的所有后继节点对应的学习子句都要被删除.

例 3: 例 3 的归结来源示意图如图 1 所示, 在之前的求解过程中, 通过冲突分析得到了学习子句 c_l, 其中 c_l 是由 c_1 和 c_2 归结得到. 则会记录下该归结来源, 在删除子句 c_1 时, 由 c_1 归结得到的学习子句也会被相应的删除, 即 c_l 会被删除. 如果在其他冲突分析过程中, c_l 还参与了其他学习子句 (如 c_l') 的产生, 则在删除 c_1 时, c_l' 也会被相应删除.

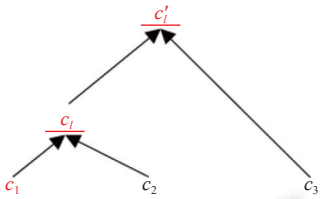


图 1 例 3 的归结来源图

该方法最早在文献 [48] 中被提出, 用于求解“基于栈的增量 SAT 问题” (见第 2.4.1 节), 并被著名的 CDCL 求解器 SATIRE[49]所采用. SATIRE 是基于 CDCL 求解器 GRASP[16]开发的增量 SAT 求解器, 它利用了冲突分析的技术. 它在求解新的公式 ϕ_{i+1} 时, 会从 ϕ_i 找到的解的基础上开始搜索, 并且会利用在历史求解过程中产生的学习子句. 实验证明, 在针对电子自动化设计 (electronic design automation, EDA) 场景的有界模型检查 (bounded model checking, BMC) 场景下, SATIRE 相比于每次重新计算的非增量求解, 平均求解速度提升 1 倍以上.

2.1.2 基于假设 (assumption) 的增量 SAT 求解

然而, 充分维护学习子句的归结来源信息会带来较大开销, 对于性能有很大的影响. 因此, 现代 SAT 求解一般不允许直接删除子句, 而是使用添加“假设”文字的方式管理子句的删除, 并在删除子句后复用学习子句.

考虑一个子句 c_i, 如果在增量 SAT 求解器的多轮调用中会添加或者删除 c_i, 那么实际添加到 SAT 求解器数据结构中的子句是: $(c_i \vee \neg s_i)$, 其中 s_i 为“假设”变量. 用户可以通过设置假设变量的取值, 从而条件性地编码子句的激活状态: 如果该子句需要被考虑则设置 $s_i = \top$, 若该子句需要被删除则设置 $s_i = \bot$. 这些假设会被以文字列表的形式传递给 SAT 求解器, 在实际搜索过程开始之前被首先决策赋值.

使用假设的好处是, 由于添加的假设文字在公式中均以 $\neg s_i$ 的形式出现, 因此在冲突分析过程中产生的学习子句会保留所有用来产生它们的原始子句对应的假设. 移除子句只需要将相应的假设设为 $\bot$ 即可, 相应的学习子句也会失效. 因此所有学习子句都可以被保留, 而不需要记录下原始子句与学习子句之间的关系.

例 4: 考虑 SAT 公式 $\phi = c_1 \wedge c_2 \wedge c_3$, 其中 c_1 是可能被删除的子句, 则交给 SAT 求解器的公式形式是 $\phi = (c_1 \vee \neg s_1) \wedge c_2 \wedge c_3$. 在求解 ϕ_1 时需要考虑 c_1, 因此交给 SAT 求解器的假设列表为 $\{s_1\}$, 并且在求解 ϕ 的过程中产生了由 c_1, c_2 归结得到的学习子句 c_l, 其中 c_l 中会包含文字 $\neg s_1$. 在下一轮求解中如果要删除 c_1, 则只需要交给 SAT 求解器假设文字列表 $\{\neg s_1\}$, 求解器会将 s_1 决策为 $\bot$, 则子句 (c_1) 已经被删除了, 因为 $(c_1 \vee \neg s_1)$ 已经被满足了, 同样, 含文字 $\neg s_1$ 的学习子句 c_l 会失效, 因为其已经被满足了.

对于那些被永久停用的子句, 一种永久删除该子句的简单解决办法是添加一个单元子句 $(\neg s_i)$. 相似地, 如果该子句在未来的所有公式中都会被使用, 则可以加上一个单元子句 (s_i).

基于假设的增量式求解方法最早在文献 [8] 中被提出, 并在 MiniSAT 求解器[18]中被实现, 自从 21 世纪初期以来, 该方法被广泛应用于大多数 CDCL 增量 SAT 求解器中. MiniSAT 在接受假设文字列表后, 会首先将其中的文字决策赋值为真, 并为每个假设变量分配一个决策层级. 在求解过程中, 如果通过冲突分析, 发现需要回退到假设文字的决策层, 并需要对假设列表中赋值的文字进行修改, 则说明在当前假设的前提下, 公式是不可满足的. 在一轮求解完成, 即找到解或者发现不可满足后, 这些假设赋值会被取消, 使得求解器重新回到一个可用的状态, 在下轮求解时, MiniSAT 会保留之前的学习子句, 获取新的假设文字列表并重新求解.

在实际增量求解场景中, 假设的数量可能会很多, 例如在提取最小不满足集合 (minimum unsat set, MUS) 的应

用场景中, 需要为每个原始子句加上假设文字, 则假设数量等同于子句的数量. 在这类情况下, 较多的假设会导致求解性能下降, 文献 [50] 中讨论了解决此类问题的优化手段, 并有效提升了增量求解在提取 MUS 问题实例上的效率.

首先, 较多的假设会使得指导搜索的启发式指标失效. 例如用于学习子句管理的启发式指标: 文字块距离 (literal block distance, LBD)[19], 该指标会衡量一个学习子句中变量的决策层数量. 有着较高 LBD 的冲突子句在未来的冲突中相关的概率较小, 因此这些子句会被考虑删除, 而较低 LBD 的子句则倾向于被保留. 由于每个假设都有自己的决策层, 学习子句的 LBD 指标会严重受到其中假设数量的影响. 因此为了保持搜索启发式指标的有效性, 在通过遍历子句来计算 LBD 时会忽略假设文字.

其次, 冲突分析的过程中产生的学习子句会包含较多假设, 过长的学习子句会导致子句遍历的速度变慢, 从而降低求解效率. 相应的数据结构被设计用来提升此类学习子句中的遍历效率: 学习子句中非假设文字的数量会被记录下来, 并且假设文字都会被放在学习子句的末尾, 在遍历子句时只需要遍历非假设文字. 此外, 如果已知某假设在后续求解中都为假, 则该假设所在的学习子句都可以被删除. 例如在 MUS 提取过程中, 已知某子句不在 MUS 中, 则其相应的假设在后续求解中都会被设为假的, 则由它归结得到的学习子句都可以被删除.

2.2 增量 SAT 的不完备算法

局部搜索等不完备算法可以较好适配增量化 SAT 求解场景, 这是基于以下直觉: 在实际增量求解中, 本次求解的公式和下次求解的公式之间会共享大部分子句, 改变部分的比例是相对较小的, 因此可能只需要对上一轮求解结果稍加修改, 便可以得到新的解. 对于增量变化的 SAT 公式, 可以利用不完备算法对上一轮求解的完整赋值进行迭代修改, 在之前的解附近寻找满足新公式的解, 从而节省求解时间. 针对增量 SAT 问题, 不完备求解技术的关键在于如何充分利用历史解, 进一步地, 可以使用“划分公式”的方法降低问题规模并提升复用历史解的效率.

2.2.1 利用历史解

Hoos 等人在文献 [51] 中首次提出使用局部搜索手段来求解增量 SAT 问题, 探究了利用之前的求解结果是否能指导局部搜索更高效地求解增量 SAT 问题. 具体而言, 他们对比了以下 2 种局部搜索求解增量 SAT 问题的基础手段.

(1) 随机重启: 将增量 SAT 问题视作一系列独立公式, 不利用之前求解的结果, 而是直接利用局部搜索求解器重新计算.

(2) 轨迹延续: 在每次公式增量变化之后, 从当前的解开始, 作为下一轮搜索的初始赋值, 使用局部搜索求解器继续搜索.

该研究对 SATLIB 中的随机 3-SAT 问题和图着色编码的结构化实例进行增量编码并测试, 采用已有的 WalkSAT 系列局部搜索 SAT 算法[22,23,52]. 实验表明: 在随机实例中, 轨迹延续方式整体比随机重启方法更为高效; 但是在图着色编码的结构化实例中, 单纯只使用轨迹延续的效果不如随机重启, 这是因为在某些结构化实例中, 之前的求解结果可能对改变后的公式而言是一个较差的搜索起点. 因此该研究采用了软重启的策略, 即如果未满足子句数量在指定迭代次数内没有减少, 就直接重启局部搜索. 实验证明, 轨迹延续+软重启的求解策略效率明显优于只使用随机重启策略.

2.2.2 划分公式

更进一步地, Mouhoub[53]针对增量 SAT 问题提出一种对新子句集合和原公式进行划分的方法, 对这些划分后的子句集合分别用不完备算法求解. 这样可以充分利用之前的求解结果并降低公式规模.

具体而言, 给定待求解的一连串公式 $\phi_0,\ldots,\phi_i,\phi_{i+1},\ldots,\phi_n$. 由于该研究只考虑添加子句集合的情况, 求解的公式可以被形式化地表示为 $\phi_{i+1}=\phi_i\cup\Delta$, 其中 ϕ_i 是当前已经求解完, 并且已经得到可满足解的 SAT 公式, 而 ϕ_{i+1} 是在加入新子句集合 Δ 之后的新公式. 需要判断新公式 ϕ_{i+1} 是否仍然可满足.

该方法的关键是将新子句集合 Δ 和原公式 ϕ_i 分别划分为 2 个子句集合. 划分的基本思想是: 找到 Δ 中与 ϕ_i 中互相影响的子句集合, 剩余的子句集合不会影响到其他子句集合的可满足性, 因此可以单独求解.

具体划分方式如下: $\Delta=\Delta_1\cup\Delta_2$, 其中 Δ_1 是包含出现在 ϕ_i 中变量的子句传递闭包, 其可满足性受到 ϕ_i 中变量

赋值的影响; 而 Δ_2 子句集合中不包含任何 ϕ_i 或 Δ_1 中出现的变量, 因此 ϕ_i 中变量的赋值不会影响 Δ_2 中子句的可满足性, Δ_2 可以被单独求解; 类似地, $\phi_i = \phi_i^1 \cup \phi_i^2$, 其中 ϕ_i^1 是包含出现在 Δ 中变量的子句传递闭包, 而 ϕ_i^2 中的子句不包含任何出现在 Δ 或 ϕ_i^1 中子句的变量. ϕ_i^2 在求解时可以被忽略, 因为新子句集合 Δ 中的赋值不会影响 ϕ_i^2 中子句的可满足性, 只需要保持其中变量的原赋值即可.

例 5: 给定原公式 $\phi_i = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_4)$ 以及新子句集合 $\Delta = (x_3 \vee x_5) \wedge (x_5 \vee x_6) \wedge (x_7)$, 则可以将 Δ 拆分为 $\Delta_1 = (x_3 \vee x_5) \wedge (x_5 \vee x_6)$ 和 $\Delta_2 = (x_7)$. 其中 Δ_1 是包含出现在 ϕ_i 中变量的子句传递闭包, 而 Δ_2 不受 ϕ_i 中变量赋值的影响, 可以单独求解. 类似地, ϕ_i 也可以被拆分为 $\phi_i^1 = (x_1 \vee x_2) \wedge (x_2 \vee x_3)$ 和 $\phi_i^2 = (x_4)$.

在完成划分之后的求解步骤分为以下 4 步.

1) 将 ϕ_i 中得到的可满足解赋值给 Δ_1, 如果 Δ_1 是可满足的则转步骤 4).

2) 对只出现在 Δ_1 而不出现在 ϕ_i^1 中的变量进行迭代修改, 如果 Δ_1 可以满足则转步骤 4).

3) 求解 $\phi_i^1 \cup \Delta_1$, 如果不能找到解, 则返回 Unknown.

4) 求解 Δ_2, 如果找不到解则返回 Unknown, 有解则返回 SAT.

其中步骤 1) 和 2) 是为了在不修改 ϕ_i^1 中变量赋值的前提下使 Δ_1 满足. 如果无法满足, 则会在步骤 3) 中尝试对 $\phi_i^1 \cup \Delta_1$ 中的变量同时修改, 从而让 ϕ_i^1 和 Δ_1 同时被满足, 该步骤会在步骤 2) 中找到的未满足子句最少的赋值基础上继续搜索. 最后步骤 4) 会对剩余的 Δ_2 进行求解, 由于 Δ_2 与其他子句集合没有共同变量, 因此不发生关联, 可以单独求解.

本划分方式是一种普遍适用的方式, 因此在步骤 2)–4) 中对各公式的求解过程中, 除了在文献 [53,54] 中采用的 GSAT[55]局部搜索算法之外, 还可以采用其他不完备算法求解. 例如文献 [54,56,57] 中提出使用遗传算法[58]来进行求解, 在文献 [59] 中还采用了名为极值优化 (extremal optimization, EO)[60]的局部搜索启发式策略来求解.

2.2.3 增量 SAT 不完备算法的局限性

目前针对增量 SAT 问题的不完备求解算法研究还较为初步, 并且存在以下 3 点主要局限性.

(1) 上述 2 种方法均仅考虑了添加子句集合的情况, 并未涉及在增量求解过程中删除子句集合的情况, 缺乏系统的回滚机制. 在实际应用中, 删除子句集合的情况较为常见, 而增量 SAT 不完备算法难以应对此类场景.

(2) 不完备算法相较于完备算法存在以下 2 点固有不足: 不完备算法不能证明公式的不可满足性; 不完备算法的逻辑推理能力不足, 对于带有复杂结构特征的布尔公式, 其求解性能往往较差. 而实际应用场景中, 例如 EDA、模型检查等, 增量公式往往带有明显的结构特征, 且需要求解器判断不可满足性并给出证明. 因此, 仅使用不完备算法并不能应对这些实际场景.

(3) 公式的增量变化会导致不完备算法求解性能不稳定: 不完备算法的性能往往对于参数设置较为敏感, 在增量求解的过程中, 原有的参数设置会因为公式结构变化而不再适用, 因此需要实时调整; 此外, 局部搜索通常需要结合重启策略以跳出局部最优, 但增量模式下难以判断何时重启: 频繁重启会丢失已探索区域的历史解信息, 而重启不足则可能会导致算法长期停滞.

因此, 对于增量 SAT 的不完备算法, 推荐的求解策略是将其与完备算法进行结合: 在公式增量变化时, 用不完备算法辅助完备算法快速找到可满足解, 加速搜索过程. 并且, 在增量求解过程中, 需要通过自适应机制调整参数与重启策略, 使不完备算法在公式增量变化后迅速收敛.

2.3 适配增量求解的化简技术

化简技术可以通过删除子句或变量的方式, 很大程度上减小问题规模, 从而有效提升求解效率. 然而增量求解与化简技术并不能有效结合, 主要基于以下原因.

1) 对假设文字的化简会导致求解不可靠.

例 6: 公式 $\phi = (\neg s_1 \vee a \vee b) \wedge (\neg a) \wedge (\neg b)$, 在假设 $\{s_1\}$ 下是不可满足的, 但是如果对假设 s_1 做化简删除, 会得到新公式 $\phi' = (\neg a) \wedge (\neg b)$, 这是可满足的, 所以求解不可靠.

2) 当化简删除的变量被重引入到公式时, 增量求解会变得不可靠.

例 7: $\phi=(p_1\vee p_2)\wedge(\neg p_1\vee p_3)$, 通过变量消除化简技术消除变量 p_1, 可以将公式化简为 $\phi=(p_2\vee p_3)$, 在加入新子句集合 $\Delta=(\neg p_1)\wedge(\neg p_2)$ 后得到新的公式为 $\phi'=(p_2\vee p_3)\wedge(\neg p_1)\wedge(\neg p_2)$. ϕ' 是可满足的, 并有解为 $\alpha=\{p_1=\bot, p_2=\bot, p_3=\top\}$. 但是很明显 Δ 与原公式中的 $(p_1\vee p_2)$ 是冲突的, 因此 $\phi\cup\Delta$ 是不可满足的. 所以此时求解变得不可靠.

围绕这 2 个问题, 本节将详细综述如何为增量求解适配化简技术, 该方向也是增量 SAT 问题求解的主要研究重点.

2.3.1 冻结变量

为解决上述问题, 一种直观的手段是“冻结变量”, 即在化简过程中不允许对某些特定的变量化简删除.

为了解决上述第 2.3 节 1) 中提到的“对假设文字的化简会导致求解不可靠”问题, 在对公式化简时, 文献 [8,61] 中会冻结假设文字, 即不允许对假设文字进行变量消除. 这样虽然保证了求解的可靠性, 但是由于限制了化简技术应用的范围, 化简的有效性也会下降.

为了解决上述第 2.3 节 2) 中提到的“当化简删除的变量被重引入到公式时, 增量求解会变得不可靠”问题, 文献 [25,61] 中提出了“前瞻技术 (look ahead)”来冻结在未来公式中可能出现的变量. 具体而言, 该方法会提前预知后续所有的待求解公式, 对于那些在未来公式中会出现的变量, 不对其使用变量消除. 然而该方法并不总是实用, 因为在实际求解场景中, 并不总是能够知道未来待求解的公式. 并且该操作同样也会限制化简技术的应用范围和效果.

文献 [62] 中对“前瞻技术”进行了改进, 提出了“部分前瞻”方法: 求解器会预知 k 步之内需要求解的公式, 将其合取作为新的公式一起交给求解器, 并通过假设文字列表来控制不同的待求解公式. 化简时只需要对假设文字冻结即可, 而不用考虑被删除的变量被重新引入, 因为所有后续可能出现的变量都已经出现在了合取的公式中. 在完成这 k 轮求解后, 会重新调用一个 SAT 求解器, 再次将后 k 轮的公式合取以及之前的学习子句交给其进行求解. 其具体流程如例 8 所示.

例 8: 预知的后 3 轮中待求解的公式为 $\phi_1=c_1$, $\phi_2=c_1\wedge c_2$, $\phi_3=c_1\wedge c_3$, 算法会为每个子句 c_i 都添加上假设文字 s_i, 并将其合取成为一个新的公式 $\phi'=(c_1\vee\neg s_1)\wedge(c_2\vee\neg s_2)\wedge(c_3\vee\neg s_3)$. 3 次交给增量求解器的假设文字列表分别为 $\{s_1,\neg s_2,\neg s_3\}$, $\{s_1,s_2,\neg s_3\}$, $\{s_1,\neg s_2,s_3\}$. 在对 ϕ' 化简时, 会冻结所有假设文字 s_i, 而允许对其他变量化简删除. 由于 3 次公式中的所有变量都出现在 ϕ', 所以对其化简不会导致重引入的问题.

2.3.2 将假设作为单元子句

虽然冻结假设文字可以解决第 2.3 节中所提到的因为对假设文字化简删除而导致的不可靠问题, 但是这会缩减化简技术的作用范围, 从而明显降低化简能力. 为了充分发挥预处理能力[25], 一种技术路线是将假设文字编码为单元子句, 而非将其作为决策文字. 具体而言, 相比于 MiniSAT[18]中每次会交给求解器假设文字列表 $\{l_1,l_2,\ldots,l_n\}$ 并在求解前对假设文字进行决策的方法, 在每轮求解时本方法会交给求解器公式 $\phi\cup A$, 其中 A 是单元子句集合 $\{(l_1),(l_2),\ldots,(l_n)\}$.

这样能够充分发挥预处理的功能, 因为预处理的化简功能允许对假设变量进行化简, 可以根据假设单元子句进行传播并删除冗余子句, 从而更进一步降低问题规模. 如例 7 中的情况, 交给求解器的公式为 $\phi\cup(s_1)$, 即 $(\neg s_1\vee a\vee b)\wedge(\neg a)\wedge(\neg b)\wedge(s_1)$. 对 s_1 化简消元后会得到新公式 $(a\vee b)\wedge(\neg a)\wedge(\neg b)$, 该公式不可满足, 可以保持求解可靠性.

然而相对于将假设作为决策文字的做法, 该方法存在以下缺点: 由于各轮求解中假设不同, 因此由假设作为单元子句归结得到的学习子句并不能在后续求解中被复用.

例 9: 给定公式 $\phi=(\neg s_1\vee c\vee d)\wedge(\neg s_2\vee\neg d)\wedge(\neg c\vee e)\wedge(\neg c\vee\neg e)$, 并给定假设文字列表为 $\{s_1,s_2\}$, 本方法会将假设文字作为单元子句添加到公式 ϕ 中交给 SAT 求解器, 即待求解的公式为 $\phi\wedge(s_1)\wedge(s_2)$. 求解过程中的归结来源如图 2 所示 (带下划线节点表示依赖于假设的子句), 单元子句 (s_1) 和 (s_2) 可以参与传播和化简. 但是部分学习子句会依赖于假设单元子句, 因此不能在后续的求解中被复用, 如子句 $\alpha_7=c\vee d$ 是通过子句 (s_1) 和子句 $(\neg s_1\vee c\vee d)$ 归结得到的. 如果在后续求解中不再假设 s_1 为真, 则单元子句 (s_1) 会被删除, α_7 也会失效.

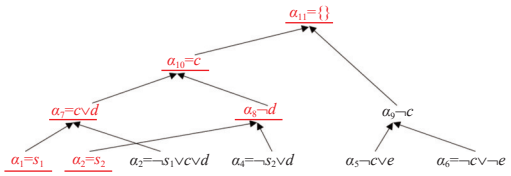


图 2　例 10 对应的归结来源图. 其中假设为 $\alpha_1 = s_1$ 和 $\alpha_2 = s_2$

文献 [6] 最早采用了将假设编码为单元子句的方法, 它在每次求解修改后的公式时, 都会分别调用一个单独的 SAT 求解器, 并将假设作为单元子句加入 SAT 公式分别进行求解, 因此被称为“多实例法”. 该方法会复用那些与假设无关的学习子句. 在该方法中, 需要记录下学习子句的归结来源, 如果其归结来源中包含假设单元子句, 则该学习子句不允许在后续的求解中被复用.

文献 [63] 改进了该方法, 除了将假设编码为单元子句外, 它还提出了一种名为 T2P (temporary to pervasive) 的后处理技术, 用于将那些依赖于假设的学习子句 (即归结来源包含假设, 记作 temporary) 转换为独立于假设的 (记作 pervasive), 从而使其可以在后面的求解中被复用, 该方法被称为“假设传播”. 具体而言, 本方法会记录下各个学习子句的归结来源. 对于一个学习子句 β, $T(\beta)$ 表示从节点 β 反向搜索可达的假设单元子句. 通过将这些假设取反并加入该子句中, 可以将该学习子句转化为不依赖于假设的学习子句. 这种转换表示, 仅当学习子句 β 的归结来源中的假设都为真时, 该学习子句才会生效. 如在例 10 的情况下, 通过进行转换后, $\alpha_7 = c \vee d \vee \neg s_1$, $\alpha_8 = \neg d \vee \neg s_2$, $\alpha_{10} = c \vee \neg s_1 \vee \neg s_2$, $\alpha_{11} = \neg s_1 \vee \neg s_2$. 其中学习子句 α_7 仅当假设 s_1 成立时才生效.

文献 [63] 中将 T2P 技术用于提升上述的“部分前瞻”方法中, 它会将 k 轮求解中的共同假设作为单元子句, 用于化简, 并冻结剩余的假设. 在 k 轮求解中用不同的假设列表进行求解. k 轮求解之后, 会重新调用一个 SAT 求解器进行求解, 并将依赖于共同假设单元子句的学习子句进行转换, 添加到新的 SAT 求解器中.

例 10: 如例 9 中的情况, 由于 3 个公式中均有共同假设 s_1, 因此可以将 (s_1) 作为单元子句进行化简, 并冻结其余假设文字 s_2, s_3. 在 3 轮求解中, 交给增量 SAT 求解器的假设列表为 $\{\neg s_2, \neg s_3\}$, $\{s_2, \neg s_3\}$, $\{\neg s_2, s_3\}$. 在对这 3 个公式求解完成后, 会将归结来源包含 (s_1) 的学习子句进行 T2P 转换, 将其保留到后续的求解中复用.

文献 [64] 中进一步改进了上述 T2P 后处理转换, 提出了“增量 T2P”技术. 它仅对归结来源是失效假设的学习子句进行转换: 若上一轮求解的假设文字列表被记作 A_{i-1}, 而新一轮求解时的假设文字列表记作 A_i, 则在新一轮的增量求解时, 仅有 $A_{i-1} \setminus A_i$ 部分的假设会变得失效, 其余部分的假设仍然有效. 因此只需要对失效假设归结得到的学习子句进行转换即可. 其具体算法流程如算法 1 所示. 由失效假设 $A_{i-1} \setminus A_i$ 开始遍历归结来源, 寻找由失效假设归结得到的学习子句, 然后将对其进行转换.

算法 1. 消除失效假设影响.

For each 子句 c, $A(c) = \{\}$; // $A(c)$ 中记录着子句 c 的归结来源中是失效假设的部分
For each $a \in A_{i-1} \setminus A_i$: //遍历失效假设
　For each 归结图上 a 的后继子句 c:
　　$A(c) = A(c) \cup \{a\}$; //将失效假设记录在由其归结得到的学习子句中
For each $A(c) \neq \{\}$ 的子句 c:
　将 c 替代为 $(V_{a \in A(c)} \neg a) \vee (c)$; //对失效假设归结得到的子句进行转换

例 11: 如例 10 中的公式, 本次求解的假设列表为 $\{s_1, s_2\}$, 如果下一次的假设列表为 $\{\neg s_1, s_2\}$, 则 2 轮中将假设作为单元子句交给求解器的公式分别为 $\phi \wedge (s_1) \wedge (s_2)$ 和 $\phi \wedge (\neg s_1) \wedge (s_2)$. 在第 2 轮求解时, 子句 (s_1) 会被删除, 而子句 (s_2) 仍然存在, 因此仅需要对由 (s_1) 归结得到的学习子句进行转换. 具体而言, 进行转换后, 会得到 $\alpha_7 = c \vee d \vee \neg s_1$, $\alpha_{10} = c \vee \neg s_1$, $\alpha_{11} = \neg s_1$.

2.3.3　增量预处理

虽然利用“前瞻技术”[25,61]在化简时冻结未来可能再次出现的变量, 可以解决第 2.3 节中所提及的因重引入被

删除变量而导致增量求解不可靠的问题, 但是“前瞻技术”需要预知未来待求解的实例, 而这一信息在实际场景中并不总是可以获取的. 并且, “前瞻技术”在每轮求解完成之后都会将公式恢复, 并在添加子句后重新进行完整的化简过程[61]. 这样会重复大量的化简工作.

文献 [65] 中提出了“增量预处理”的技术用于解决上述问题, 该技术只对新加入的子句进行预处理, 而不需要重复之前所做的预处理工作. 并且, “增量预处理”不需要提前预知未来的实例, 即允许用化简技术删除任意非假设变量.

增量预处理依赖如下附加数据: *ElimVarQ* 是一个队列, 其中包含已经被删除的变量. 对于变量 v, ϕ_v 和 $\phi_{\bar{v}}$ 分别表示当前公式中包含文字 v 和 $\neg v$ 的子句. 此外, 还会记录因为删除变量 v 而被删除的包含文字 v 和 $\neg v$ 的子句集合, 分别记作 S_v 和 $S_{\bar{v}}$.

在加入新子句集合 Δ 并得到新公式 ϕ 后, 首先要处理那些已经通过化简被删除的变量, 即在 *ElimVarQ* 中的变量. 对于此类变量, 如果其在新公式 ϕ 中重新出现了, 则需要考虑对其进行“重删除”或“重引入”操作. 文献 [65] 中证明了若每次增量求解时, 变量被重删除或重引入的顺序保持一致, 可靠性就得以保证. 具体流程如下.

“重引入”会将变量 v 重新加入公式中, 具体做法是将之前被删除的子句集合, 即 S_v 和 $S_{\bar{v}}$, 都重新加回到公式中去, 并将该变量从 *ElimVarQ* 中移除. “重删除”相当于在重引入变量 v 之后, 又重新对 v 做变量消除化简, 相应的附加数据也需要被更新: $S_v = S_v \cup \phi_v$ 和 $S_{\bar{v}} = S_{\bar{v}} \cup \phi_{\bar{v}}$. 具体是选择重删除变量还是重引入变量, 取决于执行哪种操作之后得到的公式更小.

对于那些在之前没有被化简删除的变量, 对其的处理方式和普通的预处理相同, 只是需要在删除此类变量时更新附加数据: $S_v = \phi_v^i$, $S_{\bar{v}} = \phi_{\bar{v}}^i$, 并将 v 加到 *ElimVarQ* 中.

例 12: 如例 8 中的情况, 公式 $\phi = (p_1 \vee p_2) \wedge (\neg p_1 \vee p_3)$, 将变量 p_1 化简删除后, 得到化简后的公式: $(p_2 \vee p_3)$, 在 *ElimVarQ* 中记录下变量 p_1, 并记录下删除 p_1 时包含文字 p_1 和 $\neg p_1$ 的子句, 分别是 $S_{p_1} = \{(p_1 \vee p_2)\}$ 和 $S_{\overline{p_1}} = \{(\neg p_1 \vee p_3)\}$. 当加入新子句集合 $\Delta = (\neg p_1) \wedge (\neg p_2)$ 后, 得到新的公式为 $\phi = (p_2 \vee p_3) \wedge (\neg p_1) \wedge (\neg p_2)$. 由于被删除的变量 p_1 又出现了, 因此便需要考虑重删除或重引入 p_1. 如果重引入变量 p_1, 则会将 S_{p_1} 和 $S_{\overline{p_1}}$ 加回公式中, 得到 $\phi_1 = (p_2 \vee p_3) \wedge (\neg p_1) \wedge (\neg p_2) \wedge (p_1 \vee p_2) \wedge (\neg p_1 \vee p_3)$. 如果重删除变量 p_1, 则相当于在 ϕ_1 的基础上对 p_1 做变量消除化简, 得到公式 $\phi_2 = (p_2 \vee p_3) \wedge (p_2) \wedge (\neg p_2)$. ϕ_1, ϕ_2 均不可满足, 因此保持了可靠性. 又因为 ϕ_2 更小, 所以会选择对 p_1 重删除.

2.3.4 结合手段

上述的“将假设作为单元子句”和“增量预处理”技术分别针对假设文字和非假设变量处理. 文献 [64] 中提出将这两种方法进行有机结合, 并命名为“超增量 (ultimate incremental)”方法, 它结合了两种方法的优势. 具体而言, 该方法会将所有假设文字都作为单元子句交给求解器. 本轮和上轮的假设列表分别记作 A_i 和 A_{i-1}. 在新一轮求解开始之前, 会先通过“增量 T2P”的方式对由失效假设 $A_{i-1} \setminus A_i$ 对应的单元子句归结得到的学习子句进行转换, 使其可以在后续求解中使用. 然后将新加入的假设 $A_i \setminus A_{i-1}$ 作为单元子句作为新子句集合加入公式中. 最后调用增量预处理对公式进行化简.

2.3.5 增量过程中处理

文献 [66] 中提出了一种更普适的增量化简方法, 将现有的“过程中处理 (inprocessing)”方法应用到增量 SAT 求解中. 它会识别并自动恢复与新加入的子句集合不相容的之前化简步骤. 除了第 2.3 节中提到的变量删除化简技术, 本技术支持更为普适的化简方法, 拓展了文献 [67] 中的“过程中处理”框架, 允许将更多的化简方法应用到增量求解中.

具体而言, 过程中处理会将化简的效果以“证据标记子句 (witness labeled clause)”, 形如 $(w : D)$ 的形式记录在重构栈中. 其中 D 是被化简删除的子句, 而 w 是文字集合, 记作证据 (witness). 证据可被视为是一个部分赋值, 当移除一个子句后不能保持原来的解时, 重构栈上的证据会指示对简化后公式的解进行重构: 自顶向下遍历重构栈, 如果被移除的子句不可满足, 则将证据文字作为赋值, 使重构后的解也能满足该子句. 具体如例 13 所示.

例 13: 公式 $\phi = (a \vee b) \wedge (\neg a \vee c)$, 对变量 a 进行变量消除化简后, 会消除其所在的子句, 并记录在重构栈中: $\{(a : (a \vee b)), (\neg a : (\neg a \vee c))\}$. 化简后的公式为 $(b \vee c)$, 对其求解后得到解为 $\{b = \bot, c = \top\}$. 在重构时会先设被删除的

变量 a 赋值为 $\bot$, 并自顶向下遍历重构栈, 在遍历到 $(a:(a\vee b))$ 时发现子句不可满足, 则要将证据文字 a 作为赋值, 即得到解为 $\{a=\top,b=\bot,c=\top\}$.

利用该重构栈, 可以识别出因为增量添加的子句而导致无效的化简步骤, 然后仅恢复这些有问题的化简步骤, 从而使得化简方法变得增量. 本研究提出了“子句干净”的概念: 对于一个证据标记子句 $(w:D)$, 如果子句 C 满足: $\neg C$ 和 w 没有交集, 则称该子句 C 关于 $(w:D)$ 是干净的. 例如子句 (a) 相对于 $(a:(a\vee b))$ 是干净的. 基于干净的概念提出以下 2 条规则.

(1) 添加子句规则: 当添加一个子句集合时, 要求其中的每一条子句都要相对于重构栈中的所有证据标记子句是干净的.

(2) 恢复化简规则: 当将 $(w:D)$ 从重构栈中删除, 并将被删除的子句 D 添加回公式时, 要求子句 D 相对于重构栈中位于 $(w:D)$ 之后的所有证据标记子句干净.

文献 [66] 中证明了, 根据上述 2 条规则恢复重构栈并加入新子句集合增量求解时, 之前的学习子句可以被复用, 且公式可满足性得以保证, 并且还可以利用重构栈来重构解. 因此要做的就是在加入新子句集合之前, 对重构栈处理, 将其中的化简步骤按“恢复化简规则”恢复, 从而保证重构栈相对于新加入子句集合是干净的, 即满足“添加子句规则”. 其具体实现方式如算法 2 所示. 自底向上遍历重构栈, 并检查 Δ (新加入的以及已经从重构栈中被恢复的子句集) 中是否存在子句相对于 $(w_i:D_i)$ 是不干净的, 如果有, 则将子句 D_i 恢复到 Δ 中, 并将其从重构栈中移除.

算法 2. 恢复 (新子句集合 Δ, 重构栈 θ).

$(w_1:D_1)\ldots(w_n:D_n):=\theta$
For $i=1,2,\ldots,n$ do
  If $\exists \ell\in w_i,\neg\ell$ 出现在 Δ 中 then //判断 Δ 中是否有子句相对于 $(w_i:D_i)$ 不干净
    $\Delta:=\Delta\cup D_i$, $\theta:=\theta\setminus(w_i:D_i)$ //将 $(w_i:D_i)$ 从重构栈中删除, 并将 D_i 移到新子句集合中
Return $\langle\Delta,\sigma\rangle$

例 14: 如例 13 中所示, 当前重构栈为 $\{(a:(a\vee b)),(\neg a:(\neg a\vee c))\}$, 公式为 $(b\vee c)$. 当加入新子句集合 $\Delta=\{(a)\}$ 时, 会从底向上遍历重构栈: $(a:(a\vee b))$ 相对于 Δ 是干净的, 不进行恢复, $(\neg a:(\neg a\vee c))$ 相对于 Δ 是不干净的, 则会将其从重构栈中移除, 并将子句 $(\neg a\vee c)$ 添加到原公式中. 得到新的待求解公式为 $(b\vee c)\wedge(\neg a\vee c)\wedge(a)$ 以及新的重构栈 $\{(a:(a\vee b))\}$.

Kiesl-Reiter 等人在文献 [68] 中提出在“过程中处理”的基础上加以改进, 使 SAT 求解器在增量求解时能够生成可靠的不可满足性证明 (proof). 该方法将“恢复”步骤拓展到了标准形式的不可满足证明中, 并在后处理阶段将这些“恢复”步骤删除, 从而产生标准的不可满足证明.

2.3.6 增量求解中的化简技术总结

后文表 1 总结了本节中提到的各种化简技术在增量 SAT 求解中的应用, 按以下几个维度进行对比.

(1) 求解器数量: 在求解各轮增量 SAT 问题时, 调用单个还是多个求解器.

(2) 对假设归结得到的学习子句处理方式: 包括保留、抛弃以及对其进行转换.

(3) 支持的化简方式: 全量预处理 (每轮增量求解前重新预处理), 增量预处理, 增量过程中处理.

(4) 是否使用前瞻信息: 预知全部或部分未来的求解实例, 或完全不使用前瞻信息.

(5) 对假设的处理: 是将其作为文字列表在求解开始前进行决策, 还是将其作为单元子句.

2.4 增量 SAT 的其他形式

本节中我们将补充增量 SAT 问题的其他表现形式, 这些形式可以被视为增量 SAT 的特殊情况. 根据其特殊性质, 可以制定相应的优化方式来提升其求解效率.

表 1 增量 SAT 求解中化简技术总结

文献来源	求解器数量	对假设归结得到的学习子句处理方式	支持的化简方式	是否使用前瞻信息	对假设的处理
MiniSAT[8]	单个	保留全部	不支持	否	假设文字列表
前瞻技术[25,61]	单个	保留全部	全量预处理	预知全部未来公式	假设文字列表
部分前瞻[62]	多个	保留全部	全量预处理	预知部分未来公式	假设文字列表
多实例求解[6]	多个	抛弃	不支持	否	单元子句
假设传播[63]	多个	T2P	全量预处理	预知部分未来公式	共同假设作为单元子句, 其余为假设文字列表
增量预处理[65]	单个	保留全部	增量预处理	否	假设文字列表
ultimate incremental[64]	单个	增量T2P	增量预处理	否	单元子句
增量过程中处理[66]	单个	保留全部	增量过程中处理	否	假设文字列表

2.4.1 基于栈的增量 SAT

该类问题是增量 SAT 的一种特殊情况, 只允许按先进后出的顺序添加 (push) 和删除 (pop) 子句集合. 如图 3 所示, 该问题可以被建模为一个问题树, 其中每个节点都表示一个 SAT 公式, 边 $\phi_i \to \phi_k$ 表示公式 $\phi_k = \phi_i \cup C_k$, 即向公式 ϕ_i 中添加子句集合 C_k 得到公式 ϕ_k. 求解过程会按深度优先搜索的顺序遍历该问题树, 并判断每个节点对应的公式的可满足性.

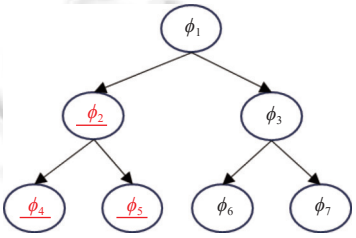


图 3 基于栈的增量求解对应的问题树

文献 [48] 中针对该问题提出了求解方法: 当求解公式 ϕ_i 时, 如果它是可满足的, 则可以直接向其中添加子句集合, 并且从 ϕ_i 的当前解开始继续搜索. 该算法提出了一种名为缓存 (caching) 的方法来记录搜索状态: 对于可满足的实例, 算法会记录下其中的变量决策情况, 如果通过删除子句再回退到该公式时, 根据这些决策变量, 可以重新构建可满足解.

如果公式 ϕ_i 是不可满足的, 则其后继节点都不需要被求解了, 因为任何向 ϕ_i 中添加子句后的公式都不满足. 算法会删除 (pop) 子句集合, 并回退到父节点, 即上次可满足的公式, 恢复缓存的解, 并继续添加子句集合进行搜索. 如图 3 所示, 带下划线的节点表示不可满足的实例, 即 ϕ_2 不可满足之后, 后继节点对应的公式便不需要求解了. 此时搜索会删除子句集合 C_2, 回退到可满足的公式 ϕ_1, 并继续添加公式集合 C_3, 即求解 ϕ_3. 在文献 [48] 中会记录学习子句的归结来源, 在删除子句后, 会将其相关的学习子句也连带删除.

2.4.2 同步 SAT (simultaneous SAT, SSAT) 问题

在给定一个 CNF 公式的前提下, SSAT 会接受一系列的证明目标文字 (proof object, PO), 并对于每个 PO, 判断本公式是否存在一个解使其赋值为假 (falsified 状态), 还是该 PO 的赋值在本公式的所有解中均为真 (valid 状态). 该类问题在有界模型检查 (bounded model checking, BMC) 领域有重要应用, 因为需要在同一模型 (对应同一组 SAT 公式) 上证明一系列的属性 (即 PO) 是否永真.

例 15: 给定一组公式 $F = \{c_1, c_2, c_3, c_4\}$, 并附加上 3 个证明目标 $PO_1 = \neg x_1$, $PO_2 = x_5$, $PO_3 = x_2$. SSAT 相当于同时判断以下 3 个公式 $F_i = F \cup \neg PO_i$ 是否可满足: $F_1 = \{c_1, c_2, c_3, c_4, x_1\}$, $F_2 = \{c_1, c_2, c_3, c_4, \neg x_5\}$, $F_3 = \{c_1, c_2, c_3, c_4, \neg x_2\}$. 如果 F_i 可满足, 则说明 F 存在解使得 PO_i 为假, 若 F_i 不可满足, 则说明在 F 的所有解中 PO_i 均为真. 这 3 个公式可以利用添加/删除子句的方式增量求解.

SSAT 问题可以被转化为一系列相关的 SAT 问题, 并用增量 SAT 求解器求解. 但是 SSAT 与增量 SAT 存在本质区别: SSAT 不关心公式的先后顺序, 而是同时考察所有证明目标的状态. 因此, 除了通过多次迭代调用增量 SAT 求解器, 文献 [69] 中还提出了一种改进的 CDCL 过程, 仅通过一次调用来求解 SSAT 问题, 即同时求解关于统一 SAT 实例的所有证明目标, 判断其为永真或为假. 具体算法流程如下所示.

SSAT 算法会在一开始将所有 PO 的状态都设为未定义的 (undefined), 然后在搜索过程中, 记录下一个当前关注的证明目标 (currently watched proof object, CWPO), 并且将 CWPO 的赋值决策为假. 然后算法会在 CWPO 赋值为假的情况下进行搜索, 直到如下情况之一.

(1) 找到了一个使公式可满足的解.

(2) 通过冲突分析, 得到仅包含 CWPO 的单元子句冲突子句.

在情况 (1) 下, 可以将 CWPO 的状态标记为 falsified, 并将当前解中未赋值或赋值为假的 PO 都标记为 falsified, 因为这些 PO 取假也可以使得公式满足. 在情况 (2) 下, 说明原公式中蕴含 CWPO 为真, 因此可以将 CWPO 标记为 valid. 以上 2 种情况下, 都需要中止搜索并将所有文字的赋值取消, 然后将 CWPO 切换到其他未定义状态 (undefined) 的 PO 上, 将新的 CWPO 赋值决策为假, 并继续搜索. 直到所有 PO 都标记为 valid 或 falsified, 或者原公式被证明 unsat (证明所有 PO 都是 valid). 由于单次调用的特点, 该算法比逐个增量求解更高效, 并且允许在搜索全部过程中复用所有的学习子句. 文献 [62] 中将 PO 的定义进一步扩展到了等价性验证中, 即 PO 不仅是一个文字, 也可以是形如 ($a \leftrightarrow b$) 的等价关系.

2.4.3 并行化增量 SAT 求解

Wieringa 等人[70]设计了一个名为 Tarmo 的多核增量 SAT 求解器, 它提供了一种用于增量 SAT 求解器的异步接口, 这是对传统求解器接口的自然扩展, 允许构建高效的特定于应用程序的并行化. 通过该异步接口, 可以对多个增量 SAT 公式进行同时求解. 这是基于在文献 [71] 中的发现: 在有界模型检测的应用场景中, 对于一个增量 SAT 求解序列, 如果只关注最后一个实例的可满足性, 则有时候直接对最后一个实例求解的时间会比逐个增量求解的时间总和更短, 而这与搜索过程中决策变量的活跃度有关. 基于该观察, Tarmo 会利用这些信息在并行搜索过程中来预测是否要采用更为冒险的策略, 即直接对序列中的最后一个实例求解.

2.5 增量 SAT 求解器对比

本节中, 我们根据最近一届带有增量 SAT 赛道的 SAT 国际比赛 (SAT Competition 2020)[72]中提供的测试集, 对 9 个目前主流领先的增量 SAT 求解器进行了实验对比. 其中包括以下求解器: RLNT (版本 2020)[4]、Riss (版本 7.1.2)[73]、abcdSAT (版本 i20)[74]、CaDiCaL (版本 2.1.0)[75]、CryptoMiniSat 5 (版本 5.6.8)[76]、Glucose (版本 4.1)[77]、Lingeling (版本 1.0.0)[78]、MiniSAT (版本 2.2)[18]、PicoSAT (版本 9.6.1)[79]. 其中, CaDiCaL[80]正在逐步取代 MiniSAT, 成为增量 SAT 问题的主流求解器, 其对增量场景进行了专门优化, 实现了第 2.3 节中所介绍的各种适配增量求解的预处理方法, 并支持高效的库接口.

由于各类增量 SAT 求解器的接口各不相同, 因而较难对其进行统一对比. 在文献 [81] 中, Balyo 等人提出了一种名为 IPASIR (re-entrant incremental solver API) 的统一增量 SAT 接口 (https://github.com/biotomas/ipasir). 国际 SAT 比赛中的增量 SAT 赛道采用该接口进行统一对比, 并且参加该赛道的增量求解器都需要支持 IPASIR 接口.

根据 SAT 国际比赛, 测试集包含以下 6 个应用场景: (1) 骨架变量计算 (Bones); (2) 关键变量计算 (Essential); (3) 最长简单路径 (LSP); (4) 最大可满足性问题 (MaxSAT); (5) 带量词布尔公式 (QBF); (6) 规划问题 (PSAS). 测试时对其中每个应用场景选择了 50 个实例进行对比.

实验的环境为 AMD EPYC 7763 64-Core 2.45 GHz 的处理器, 内存为 2 048 GB RAM, 系统为 Ubuntu 20.04. 根据 SAT 比赛规则, 每个例子被分配 2 000 s 的时间限制. 评判标准根据 PAR-2 分数计算: 对于一个例子, 如果求解成功则运行时间记作分数, 如果运行失败则以时间限制的 2 倍作为分数, 分数低者优胜.

这些增量 SAT 求解器的对比情况如表 2 所示, 我们展示各求解器在每种应用中的平均 PAR-2 分数和成功求解的个数 (括号内数值). 在每种应用场景中 PAR-2 分数最小的被加粗强调.

表 2 各类增量 SAT 求解器的性能对比

应用名	RLNT	Riss	abcdSAT	CaDiCaL	CryptoMiniSat 5	Glucose	Lingeling	MiniSAT	PicoSAT
Bones	**299 (47)**	895 (40)	398 (47)	462 (45)	348 (47)	380 (46)	387 (46)	572 (45)	554 (45)
Essential	1 124 (38)	1 231 (36)	1 237 (36)	1 055 (38)	1 230 (36)	**1 034 (38)**	1 063 (38)	1 096 (38)	1 317 (35)
LSP	2 059 (26)	**1 699 (30)**	2 555 (20)	1 889 (27)	1 756 (29)	1 833 (28)	1 805 (29)	1 922 (27)	1 998 (26)
MaxSAT	2 083 (25)	2 016 (25)	**1 847 (28)**	1 969 (26)	1 975 (26)	2 019 (25)	2 262 (22)	2 095 (24)	2 225 (23)
QBF	3 140 (11)	3 138 (11)	3 098 (12)	3 005 (13)	3 057 (12)	3 084 (12)	**2 939 (14)**	3 076 (12)	2 996 (13)
PSAS	3 172 (16)	3 029 (18)	3 344 (11)	3 775 (4)	3 826 (3)	3 479 (9)	3 177 (15)	**2 525 (27)**	3 796 (3)

3 增量 SMT 问题求解技术综述

增量 SMT 问题将增量 SAT 问题从命题逻辑提升到了特定理论背景的一阶逻辑公式中.

在应用场景方面, 相较于增量 SAT 在硬件验证, 模型检测等纯布尔结构实例中的广泛应用, 增量 SMT 凭借其丰富的表达能力, 在软件验证、程序分析等涉及多种背景理论的场景中有着重要的应用, 具体内容在第 4 节中详细介绍.

在子句管理方式方面, 除了同增量 SAT 求解器一样提供基于假设的增量接口外, 主流的 SMT 求解器, 如 Z3 和 CVC5 等, 还提供了 push/pop 增量接口命令来管理断言栈[82], 具体内容在第 3.1 节中介绍.

相比于增量 SAT 求解, 增量 SMT 求解技术发展较晚, 并且目前没有非完备求解技术. 与增量 SAT 求解一样, 目前对增量 SMT 求解的研究重点在于如何将预处理技术有效结合进增量求解中, 其采用的方式为恢复化简步骤, 具体内容在第 3.2 节中介绍.

在第 3.3 节中, 我们对各种具体背景理论下的增量 SMT 求解技术进行了介绍. 最后, 在第 3.4 节中, 我们在权威测试集 SMTLIB 上对当前的主流增量 SMT 求解器进行了对比实验.

3.1 完备算法中的学习子句管理

SMT 领域中大部分求解器都支持增量求解, 其子句集合会按照先进后出的方式被添加 (push) 和删除 (pop) 到断言栈中, 类似于第 2.4.1 节中“基于栈的增量 SAT 求解”. 其中对学习子句的管理方式如下.

每次添加 (push) 子句集合后, SMT 求解器会保留之前求解得到的所有学习子句, 尝试基于先前的求解结果继续求解, 并且添加 (push) 操作会创建学习子句“回退点”. 在删除 (pop) 子句集合后, 搜索会回退到最近的“回退点”, 并删除该回退点之后得到的全部学习子句.

这种对学习子句的管理方式相比于文献 [48] 中提到的“基于栈的增量 SAT”的方法更为轻量化, 因为不再需要记录下学习子句的归结来源. 不过该操作也会删除更多学习子句, 因为即使某学习子句与被删除的子句无关, 但只要该学习子句是在最近的回退点之后得到的, 也会被删除.

3.2 适配增量求解的化简技术

在 SMT 领域中, 增量求解的研究重点也在于如何将预处理技术应用到增量求解当中. 针对现有的化简技术, 文献 [83] 中提出了普适的增量预处理技术: 识别出与新加入的子句集合不相容的之前的化简步骤, 通过将其应用到新子句集合中或将其恢复到原公式中, 保证新子句集合可以可靠地加入.

对于 SMT 公式的一系列预处理过程可以被记录为一系列“替代操作”组成的化简序列. 这些替代操作可以被记作 $<x \leftarrow t;\Psi>^{B}$, 表示将变量 x 替代为表达式 t, 并由被删除的子句集合 Ψ 证明. 标签 $B \in \{\bot, \top\}$ 表示该替代是否会改变原公式的解集合. 例如 $<x \leftarrow y+1;(x=y+1)>^{\bot}$ 表示被删除的子句 $(x=y+1)$ 可以证明将变量 x 替代为 $y+1$. $\bot$ 表示将该替代应用到任何带有 $x=y+1$ 的公式中都不会改变相应的解集合.

为了支持增量预处理, 该策略拓展了文献 [66] 中的“子句干净”概念: 对于新子句集合 Δ 和替代操作 $<x_i \leftarrow t_i;\Psi_i>^{B_i}$, 若被替代的变量 x_i 不出现在 Δ 中, 则 Δ 相对于该替代是干净的; 或者 x_i 虽出现在 Δ 中, 但该替代操作可保留解集合不变, 即 $B_i=\bot$, 则可将该替代应用到 Δ 中, 得到的子句集合 $\Delta[t_i/x_i]$ 表示 Δ 中的 x_i 被 t_i 替代了, 其

相对于该替代也是干净的. 基于“子句干净”的概念, 提出了以下规则.

(1) 添加规则: 当添加新子句集合时, 要求新子句集合要相对于化简序列中的每个替代都是干净的.

(2) 恢复规则: 当恢复替代 $< x_i \leftarrow t_i; \Psi_i >^{B_i}$ 时, 即将其从化简序列中移除, 并将子句 Ψ_i 添加回公式中, 要求 Ψ_i 相对于化简序列中本替代后的所有替代都干净.

当有新的子句集合 Δ 加入时, 需要对化简序列 θ 进行处理: 考虑将原替代操作应用到新公式中, 还是将相应的替代操作用“恢复规则”从 θ 中移除并将被删除的公式恢复到新公式中, 从而保证处理完后的新子句集合关于化简序列中的各替代都是干净的, 即满足“添加规则”.

该过程被称为重放 (replay), 具体算法流程如算法 3 中所示. 遍历化简步骤序列, 如果替代操作 $< x_i \leftarrow t_i; \Psi_i >^{B_i}$ 中被替代的变量未出现在 Δ (新子句和被恢复的子句) 中, 则 Δ 相对于该替代是干净的. 否则要判断本替代是否是可保持解集合不变, 若保持不变, 即 $B_i = \bot$, 则将该替代应用到新的公式中; 若不可保持解集合, 即 $B_i = \top$, 那么将替代应用到新公式中并不可靠, Δ 相对于该替代也不干净, 则需要将该替代移除化简序列, 并将被删除的公式 Ψ_i 重新引入公式中.

算法 3. 重放 (新子句集合 Δ, 化简序列 θ).

$< x_i \leftarrow t_i; \Psi_i >^{B_i}, \ldots, < x_n \leftarrow t_n; \Psi_n >^{B_n} := \theta$
For i = 1, 2,…, n do:
  If $x_i \in \Delta$ then: // x_i 出现在 Δ 中
    If $B_i = \bot$ then: //替代操作不会改变解集合
      $\Delta = \Delta[t_i/x_i]$; //将替代操作应用到新的子句集合中
    Else: //替代操作会改变解集合
      $\Delta = \Delta \cup \Psi_i$; //重引入被删除的子句集合
      $\theta \setminus < x_i \leftarrow t_i; \Psi_i >^{B_i}$ //从化简序列中删除该替代操作
Return $< \Delta, \theta >$ //返回要添加的子句 (包括被恢复的子句) 和处理后的化简序列

3.3 具体理论 SMT 问题增量求解

上述的适配增量求解的预处理技术适用于普遍的 SMT 求解框架, 本节将针对具体理论的增量 SMT 求解技术进行分析. 如第 1.5 节所述, 根据编码方式的不同, SMT 求解技术可以被主要分为惰性方法和积极方法. 我们将对这 2 种方法适用的典型理论进行概述, 分别是线性算术理论与位向量理论.

3.3.1 线性算术理论 SMT 增量求解

线性算术理论 SMT 问题增量求解的主流方法为惰性方法, 即 DPLL(T) 方法[31]. 该方法会调用 SAT 求解器对布尔骨架进行逻辑推理, 并用理论求解器判断由 SAT 求解器确定的理论原子公式合取是否可满足. 线性算术理论中的算术原子公式为线性等式或不等式, 因此在该问题中, 理论求解器可被视作一个无优化目标的线性规划求解器. 线性算术理论 SMT 增量求解过程中, 理论求解器接受的算术原子公式合取会增量变化. 对该问题的研究重点在于: 如何为该增量化场景设计高效的理论求解器.

根据算术变量的取值是否为整数, 线性算术理论可以被分为线性整数算术理论 (linear integer arithmetic, LIA) 和线性实数算术理论 (linear real arithmetic, LRA). 它们对应的理论求解器均依赖于增量的单纯形算法, 以下将分别对它们展开介绍.

(1) LRA: Dutertre 等人[84]在 2006 年提出了一种单纯形算法的高效变种作为 LRA 理论求解器, 该方法可用于处理不带目标函数的线性规划问题, 它的优势在于能高效地处理增量操作, 即可以方便地添加和删除约束. 其专为 DPLL(T) 框架设计了快速回退、理论传播与严格不等式处理等增量接口, 极大提高了求解效率. 在 2013 年, King 等人[85]通过最小化约束的不可满足部分之和, 改进了基于单纯形法的理论求解器, 并将其融合至 CVC5 求解

器中.

(2) LIA: Dutertre 等人[86]于 2006 年首次提出一种基于分支定界的判定框架作为 LIA 理论求解器, 该框架结合了割平面法和单纯形法. 在 2009 年, Dillig 等人[87]进一步改进了该方法, 提出通过不可满足证明来计算割平面的方法, 并将其结合进求解器 Mistral 中. 在 2012 年, Griggio[88]提出了一个利用分层策略和多种启发式方法的 LIA 理论求解器, 并将其结合进 MathSAT5 求解器中. 在 2016 年, Bromberger 等人[89]设计了一个名为 SPASS-IQ 的理论求解器, 提出了结合快速可行性检测 (unit cube test) 来高效处理无界 LIA 问题, 从而实现对 LIA 理论的高效增量求解, 并将其结合进 SMT(LIA) 求解器 SPASS-SATT 中.

3.3.2 位向量理论 SMT 增量求解

位向量理论 SMT 问题 (SMT(BV)) 增量求解的主流方法为积极方法, 即基于位展开 (bit-blasting) 的方法. 该方法会将原 SMT(BV) 公式转化为可满足性等价的 SAT 公式, 并交给增量 SAT 求解器求解. 该方法最早由 Ganesh 等人[90]提出, 并被实现于 SMT(BV) 求解器 STP 中. STP 支持断言栈的 push/pop 操作, 从而实现增量求解. 其后的 SMT(BV) 求解器均沿用该技术路线. 其中代表性的求解器包括: Boolector[91]实现了断言栈的上下文管理, 用于支持多轮查询与冲突学习, 其利用 PicoSAT 作为后端求解器. Bitwuzla[92]作为 Boolector 的后继, 提供了高效的重写、预处理、可增量的求解核心以及完善的增量接口, 支持断言的动态添加与删除, 并可复用先前检查中获得的理论引理与模型信息, 从而显著提升多轮查询效率; CVC4[93]针对增量 SMT(BV) 进行了优化配置, 采用增量 SAT 求解器 CaDiCaL 作为后端求解器, 并整合了交互式输入的断言栈管理来提升增量查询效率. CVC5[94]作为 CVC4 的后继, 针对增量 SMT(BV) 求解进一步改进, 提出词级推理与位展开的组合策略, 以及集成浮点数库的增量调用接口.

近年来也有相关工作对基于位展开的积极方法进行改进, 其中包括 Zohar 等人[95]提出整数展开 (int-blasting) 方法, 将位向量公式转化为整数算术公式以取代纯位展开, 并依赖按需增量转换来降低位展开开销. Yao 等人[96]提出通过收集词级信息 (如数据依赖关系与变量区间) 来指导布尔搜索, 并结合增量位向量子问题的局部简化和位展开策略, 取得了显著加速. Hadarean 等人[97]比较了位展开方法与 DPLL(T) 两种路线, 并提出模块化的 LBV 架构, 使多种子求解器 (例如不等式子求解器与位展开子求解器) 能够协同工作、支持增量推理.

3.4 增量 SMT 求解器对比

本文对当前主流支持增量求解的 8 种 SMT 求解器进行了实验对比, 包括 Z3 (版本 4.13.4)[98]、CVC5 (版本 1.2.0)[94]、Yices2 (版本 2.6.5)[99]、MathSAT (版本 5.6.11)[100]、OpenSMT (版本 2.8.0)[101]、STP (版本 2.3.4)[90]、Bitwuzla (版本 0.7.0)[92]、SMTInterpol (版本 2.5)[102]. 这些求解器的可执行文件均从其官网下载. 实验的环境为 AMD EPYC 7763 64-Core 2.45 GHz 的处理器, 内存为 2 048 GB RAM, 系统为 Ubuntu 20.04.4. 采用与 SMT 国际比赛相同的规则, 即为每个实例分配的时间限制为 1 200 s, 由于每个实例中会多次增删子句集合并进行求解, 我们会汇报成功求解的总数. 根据 SMT 国际比赛 SMT-COMP 2024 (https://smt-comp.github.io/2024) 的结果, 选取了其中未成功求解次数超过 10 个的较难理论进行测试. 包括 9 种主要理论: QF_ABV、QF_ABVFP、QF_BV、QF_BVLRA、QF_LIA、QF_LRA、QF_NIA、QF_UFLIA、QF_AUFLIA. 这些实例均从被广泛采用的数据集合 SMTLIB (https://smt-lib.org/benchmarks.shtml) 中下载. 依据 SMT-COMP 的衡量标准, 表 3 中展示了各增量 SMT 求解器在 SMTLIB 中的求解个数以及求解时间. 其中求解时间被显示在括号中, 其单位为 s. 若求解器不支持相应的理论, 则求解个数为 0, 求解时间为 NA. 每个理论中求解个数最多的求解器被用粗体强调. 最后统计了各个求解器获胜的理论个数, 其中 Z3 在各求解器中获胜的理论数量最多.

需要注意的是, 为节省空间, 我们没有汇报各增量 SMT 求解器的内存消耗情况. 因为在实际应用中, 内存消耗并非衡量 SMT 求解器的关键指标. 在软件分析验证等场景中, 通常会为 SMT 求解器分配充足的内存资源, 只要内存消耗不超过预设上限即可. 根据 SMT-COMP 规则, 为增量 SMT 求解器分配的最大内存为 60 GB, 且比赛结果仅汇报是否超过该限制. 实验证明, 各求解器在运行过程中的内存开销均未超过该限制.

表 3 各类增量 SMT 求解器在 SMTLIB 中的求解情况对比

理论	总数	Z3	CVC5	OpenSMT	Yices2	SMTInterpol	MathSAT	STP	Bitwuzla
QF_ABV	3296	2884 (349017 s)	2450 (496228 s)	0 (NA)	3244 (36260 s)	2046 (287063 s)	3121 (51819 s)	0 (NA)	**3249** (**85208** s)
QF_ABVFP	550088	541438 (1625626 s)	534637 (1662272 s)	0 (NA)	0 (NA)	153819 (241087 s)	382179 (219287 s)	156758 (45445 s)	**550036** (**69285** s)
QF_BVLRA	32654	**32654** (**680** s)	32536 (28077 s)	0 (NA)	32652 (3060 s)	16442 (12559 s)	32636 (8000 s)	0 (NA)	0 (NA)
QF_BV	53593	52042 (105406 s)	52127 (117700 s)	0 (NA)	52528 (46413 s)	43920 (751817 s)	**52582** (**71259** s)	52280 (38345 s)	52492 (85600 s)
QF_LIA	20041214	20039289 (44377 s)	2991533 (73436 s)	1068117 (88158 s)	**20040405** (**54220** s)	17826975 (32590 s)	12145216 (73249 s)	0 (NA)	0 (NA)
QF_LRA	1515	675 (12000 s)	665 (12000 s)	981 (8153 s)	959 (12000 s)	430 (12000 s)	**1056** (**10021** s)	0 (NA)	0 (NA)
QF_NIA	4219213	**4181658** (**2991** s)	2614278 (13550 s)	0 (NA)	253256 (14403 s)	4181657 (3439 s)	409731 (2631 s)	0 (NA)	0 (NA)
QF_UFLIA	787670	**776571** (**189399** s)	507415 (242791 s)	99182 (196301 s)	768234 (179728 s)	764725 (229242 s)	367844 (230997 s)	0 (NA)	0 (NA)
QF_AUFLIA	4699864	**4699864** (**1876** s)	4051214 (12272 s)	420846 (32303 s)	3050626 (1817 s)	3906444 (6914 s)	3259140 (6124 s)	0 (NA)	0 (NA)
获胜次数	—	**4**	0	0	1	0	2	0	2

4 增量 SAT/SMT 问题的应用

4.1 增量 SAT 的应用

增量 SAT 求解被广泛应用于电路设计、模型检查和组合优化等实际问题中, 此外, 基于假设的增量 SAT 求解方法也被作为基础能力, 用于支持不可满足集合提取和最大可满足性问题等其他逻辑问题.

4.1.1 电子自动化设计问题 (EDA)

- 电路的组合等价性验证

在电路等价性验证中, 对于两个多输出的电路, 需要检验是否存在一组输入使得两者的输出不同. 具体的做法是如图 4 所示. 将电路从输入到输出的逻辑关系进行编码, 然后将两者的输出值用异或关系表示, 编码成 SAT 公式. 如果 SAT 公式能找到可满足解, 则说明两者不等价. 文献 [5] 中提出对电路的输出逐个进行验证, 由于在不同的输出函数中存在电路的逻辑共享, 因而对于不同的输出验证等价性时, 各次求解的 SAT 公式中会存在重叠部分, 可以按电路输出将相应的公式逐个加入增量求解器中进行验证. 在后一次的求解可以从前一次求解时冲突分析得到的学习子句中受益.

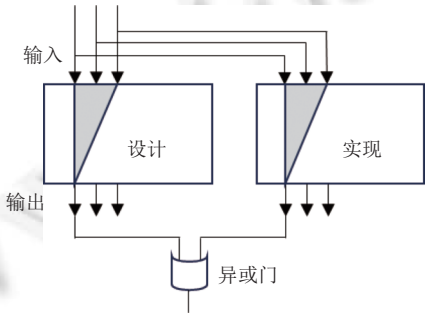


图 4 等价性验证示例

● 自动测试模式生成 (automatic test pattern generation, ATPG)

自动测试模式生成主要用于集成电路设计中的故障检测, 确保电路在生产过程中能够可靠运行. 电路逻辑和待检测的错误可以被建模为 SAT 公式. 针对同一个电路, 在对不同错误的检测过程中, SAT 公式中与电路结构和函数功能相关的子句会复用, 只有与目标错误相关的子句会在不同的检测之间有所差异. 因此, 可以利用增量 SAT 求解技术, 复用与各个测试公式中共同子句相关的学习子句[6,7].

● 电路综合

精确综合是逻辑综合中根据给定布尔函数生成最优电路的一项关键技术. 文献 [103] 中提出了一种基于增量式 SAT 的精确综合方法 (IncSyn). 通过定义辅助函数和相应的启发式算法, IncSyn 可以利用之前计算的最优实现的拓扑信息和最优子电路. 实验表明, IncSyn 在限定时间内实现了 12 个输入函数的规模提升目标, 并且在测试基准上达到了 15 倍的加速. 它还在库构建和动态重写的实际应用中显示了有效性, 而且执行时间要显著减少.

● 自动电路纠错

Alizadeh 等人在文献 [104] 中提出了一种基于增量 SAT 求解技术的方法, 用于自动修复门级电路中的多个漏洞. 该方法在修复阶段的每轮迭代中, 会根据当前的测试模式, 用正确电路与待修复电路分别生成 2 个解. 如果这 2 个解不相同, 则会产生新的测试向量来表示该差异, 并将该测试向量增量添加到测试模式集合中, 用来在下轮迭代中消除错误的解. 该方法通过多轮迭代确保电路最终被完全修复, 即修复后的电路与正确电路的行为完全一致. 随后, 他们在文献 [105] 中对该方法做了进一步改进, 提出充分复用在前一轮迭代中被部分修复的电路, 用于更快找到最终解.

● 量子电路映射

在量子计算机上执行量子电路需要电路映射, 该过程涉及放置初始量子比特, 并使用量子 SWAP 操作重新定位非相邻量子比特. 减少电路映射过程中的 SWAP 数量对于提高量子电路执行的成功率至关重要, 因为 SWAP 操作成本高且易出错. Yang 等人[106]基于增量 SAT 求解技术, 提出了一种针对电路映射问题的创新 SAT 编码方式, 并在编译时间与编译质量间保持了平衡, 使 SWAP 数量减少了 26%.

● 自动布线

在集成电路的自动布线过程中, 引脚的可接入分析是其中的关键步骤. 随着设计规则与引脚形状的日益复杂, 高效准确的引脚可接入分析变得越发重要. Wang 等人在文献 [107] 中提出了一种名为 FastPass 的引脚接入分析框架. 该框架首先为每个引脚生成符合设计规则检查的引脚接入路径候选方案, 预先计算路由路径的不兼容对, 随后采用增量 SAT 求解技术来寻找最优化的引脚接入方案.

4.1.2 模型检查

● 有界模型检查 (bounded model checking, BMC)

有界模型检查中会使用增量 SAT 求解来检验系统的时序属性 (例如不变式, 或 LTL 公式等)[8,108−113], BMC 实例会被转为一个 SAT 问题, 它会将系统的转移关系展开 k 步, 来寻找该属性的反例. 如果在步长为 k 时没有找到反例来反驳该属性后, 步长会增加并重新使用 BMC 过程来验证 $k+1$ 步时的属性. 具体而言, 会删除描述步骤 k 中用于属性检查的子句, 并添加 $k+1$ 步时的属性检查子句. 由于表示展开转移关系 $k' > k$ 步时的 SAT 公式会共享大部分展开 k 步时的 SAT 公式, 因此 BMC 会从增量求解技术中受益.

● 归纳推理

IC3[114,115]使用归纳推理来构建系统的安全性不变式, 验证系统是否满足某个性质 (如安全性、可达性等). 通过从一个初始状态开始, IC3 逐步验证系统是否能够达到某个不安全的状态 (或违反某个安全性质). 如果可以达到不安全的状态, IC3 会构造一个反例并尝试加强不变式. 增量求解主要用于不断增强不变式, 并通过增量的方式验证每个状态是否符合不变式.

PDR[116]旨在通过递归的方式验证系统的性质, 通过逐步构建可达性约束, 基于归纳推理将不变式从初始状态逐步推导到更深的时间步, 验证每个时间步的状态是否满足不变式. 在 PDR 中, 增量求解用于递归地构建可达性约束, 并验证系统是否违反了某个给定的不变式. Blankestijn 等人[117]提出了增量式 PDR 算法 (IPDR), 实现了对参

数化硬件/软件系统的分步验证. 该算法基于增量式 SAT 求解器的概念, 通过复用先前已验证系统实例的验证结果来加速验证过程. 其支持增量式交替收紧与放松约束, 即既可以基于过约束实例逐步放宽约束条件, 也能从基础约束逐步增强约束条件进行验证.

4.1.3 组合优化问题

SAT 问题作为首个被证明为 NP 完全的组合问题, 由于其近年来的飞速发展, 已经被广泛应用于其他组合优化问题的求解中. 通过 CNF 编码, 其他 NP 难组合优化问题可以被转化为 SAT 问题进行求解, 从而利用 SAT 求解器强大的求解能力. 而实际应用中, 直接编码产生的公式规模往往过大, 导致求解很困难. 因此, 常用的做法是首先仅将部分约束编码为 SAT 公式, 然后验证求解得到的结果是否满足所有约束. 若满足则得到了原问题的解, 否则就增强约束, 即编码更多子句并将其添加到原 SAT 公式中, 再调用增量 SAT 求解器. 以下通过 2 个典型的组合优化问题为例来说明该过程.

(1) 图着色问题

图着色问题是求解在一个拓扑图上最少要用几种颜色为顶点着色, 才能使所有顶点都与其相邻的顶点颜色不同. 该问题是一类重要的 NP 难组合优化问题. Glorian 等人在文献 [118] 中提出了一种 CNF 编码方式: 给定拓扑图与整数 k, 当且仅当着色数不低于 k 时, 相应的 CNF 公式是可满足的. 由于着色点对间的传递性约束需要编码大量的子句 (子句数量是顶点数量的立方数量级), 直接编码将难以应对大规模实例. 为此, 他们设计了一种基于反例引导抽象细化的方法, 该方法只对原问题的部分约束进行编码, 如果求解后发现结果违反部分约束, 则以增量方式添加相应的缺失约束, 直到找到 k 种着色数的有效解或证明问题的不可满足性 (即意味着图的色数大于 k). 在增量添加约束的过程中, 会反复调用增量 SAT 求解器.

(2) 实时出租车共享服务问题

该问题用于规划和调度一组出租车的路径, 要求在特定时间范围内将一组乘客从他们的上车地点运送到下车地点, 目标是使运送所有乘客的总时间最短. 该问题的约束可被编码为 CNF 公式, 包括“顺序约束”“截止时间约束”“容量约束”“环路约束”. 其中, “截止时间约束”和“容量约束”的编码会产生大量子句并引入大量辅助布尔变量, 因为需要表示所载乘客数量及消耗时间的所有可能组合. 为了提升求解效率, Zha 等人[119]提出不直接编码该类约束, 并用 SAT 求解器求解, 如果求解结果违反了相应的约束, 则增量地编码这些约束并将其加入原公式中, 再继续增量求解, 直到找到满足所有约束的解或证明原问题不可满足.

4.1.4 其他逻辑问题

(1) 最小不可满足集合提取

在实际应用场景中, 人们通常会对理解不可满足的原因感兴趣. 一种对常见的不可满足原因的表示是不可满足集合 (unsatisfiable set), 这代表了一个在原公式中不可同时满足的子句子集. 该集合 UC. 可以是从原公式中的所有子句中提取, 即 UC 中子句本身构成的公式就是不可满足的; UC 也可以是从原公式 ϕ 中一部分子句集合 $S \subseteq \phi$ 中提取, 并且和剩余部分的子句 $\phi \setminus S$ 合取构成不可满足公式.

对于带有假设接口的增量 CDCL SAT 求解器, 不可满足集合可以通过检查保留在最终学习子句中的假设来计算. 具体而言, 对于不可满足的公式, 如果不使用假设, 一个 SAT 求解器最终总会通过冲突分析得到空子句, 用于证明该公式是不可满足的. 而如果使用了假设文字, 则任何由带假设的子句归结得到的学习子句中, 都会保留有相应的假设文字. 结果就是, 对于一个不可满足公式, SAT 求解器会最终学到一个完全由假设文字的非构成的子句, 其中的假设对应的子句就是不可满足集合.

例 16: 考虑一个公式 $\phi = (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg s_1) \wedge (\neg b \vee \neg c \vee \neg s_2) \wedge (a \vee \neg c \vee \neg s_3)$. 其中 s_1、s_2、s_3 是假设. 在给定假设文字列表为 $\{s_1, s_2, s_3\}$ 的情况下, 通过冲突分析最终得到的学习子句为 $(\neg s_1 \vee \neg s_2)$. 这意味着假设 s_1 和 s_2 不能同时为真, 即它们分别对应的子句: (a) 和 $(\bar{b} \vee \bar{c})$ 在当前公式下不能同时满足, 也就是说子句集合 $\{(a), (\bar{b} \vee \bar{c})\}$ 是当前公式的不可满足集合.

上述过程中找到的不可满足集合不一定是最小的, 这意味着在从中删除子句后的集合仍然可能是不可满足

的. 在形式化验证问题[120,121]中, 通常会需要寻找最小的不可满足集合 (minimal unsatisfiable set, MUS), 即该集合中的子句不可同时满足, 并且在删除其中任何一个子句后便会变得可满足.

为了寻找最小的不可满足集合, 会逐个尝试删除不可满足集合中的子句, 判断删除子句之后的子句集合是否仍然是不可满足的. 在该过程中, 会依赖增量求解器迭代进行求解[10,50,122], 并寻找不可满足集合. 具体而言, 算法会首先为所有受关注的子句添加假设, 并将所有假设列为真交给 SAT 求解器求解, 从而得到一个不可满足子句集合 UC. 然后算法会逐个尝试删除 UC 中的子句, 并将删除后 UC 中剩余子句对应的假设列为真, 其余假设列为假, 交给增量 SAT 求解器求解. 如果删除子句后该子句集合仍然保持不可满足, 则说明该子句可以被删除. 通过本次求解, 还会从中找到一个包含于当前子句集合的更小不可满足集合, 可将其作为新的不可满足集合继续求解. 如果尝试删除某子句后公式变得可满足, 则说明删除该子句会破坏当前子句集合是不可满足的性质, 则要将其保留在不可满足集合中. 算法会迭代执行上述过程, 直到当前子句集合中没有未被尝试删除过的子句.

(2) 最大可满足性问题 (MaxSAT)

MaxSAT 问题[123]是 SAT 问题的拓展, 其中的子句会被分为软子句 F_{soft} 和硬子句 F_{hard}, MaxSAT 会在满足所有硬子句的可行解中, 寻找使得未满足软子句数量 (记作代价) 尽量少的解. 目前的主流 MaxSAT 算法, 包括“基于不满足集合指导”算法 (core-guided) 和“集合命中”算法 (implicit hitting set) 等, 均会为所有软子句添加假设 (转化后的公式为 F'), 并依赖增量 SAT 求解器寻找不可满足集合提取来指导搜索.

基于核的 MaxSAT 算法[124,125]会迭代调用增量 SAT 求解器来寻找不可满足核, 并通过不断松弛基数约束来找到 MaxSAT 公式的解. 具体而言, 算法会维护出现在不可满足集合中的子句集合 *inCard* (初始化为空) 和需要被满足的软子句对应的假设列表 *assumption* (所有软子句对应的假设都初始化为真, 即一开始假设所有软子句都要被满足). 在每轮迭代中, 增量 SAT 求解器会在当前假设列表 *assumption* 的前提下求解 $F' \cup card$, 其中 *card* 表示将“*inCard* 集合中未满足的软子句数量 $\leqslant$ *cost*”这一基数约束编码的 CNF 公式. 如果在本轮求解中没有找到可满足解, 则会返回不可满足集合 UC. 在下一轮的迭代中, 需要将 UC 中软子句对应的假设在 *assumption* 中列为假 (即不再假设该软子句必须被满足), 并将 UC 中软子句加到 *inCard* 中. 每轮求解时 *cost* 从 0 开始会逐个增量, 直到找到可满足的解, 即未满足软子句数量最小的解. 文献 [126] 中显示通过使用基数约束的总计器编码[127], 可以通过仅向 SAT 求解器添加新子句来实现 *cost* 的增加, 因此可以进行增量求解.

集合命中算法[128−131]同样也高度依赖增量求解及其对不可满足核的提取能力. 具体而言, 算法会在不同的假设下, 迭代增量求解 F' 多次, 并会维护各次求解中所累积的不可满足集合, 将其记录在 K 中. 命中集 HS 是一个子句集合, 要求使得 K 中的每个不可满足集合都至少有一个子句在 HS 中. 由于在硬子句满足时, 每个不可满足集合中至少有一个子句不可满足, 所以对于任何可行解, 其中未满足的软子句集合都是一个命中集. 根据当前的 K, 最小代价的 HS (minimal cost hitting set, MCHS) 是代价的下限. 在每轮求解中, 会根据当前的 K 计算 MCHS, 假设 MCHS 以外的子句都满足, 并进行增量求解. 如果不可满足, 则会得到一个新的不可满足集合 UC, 并将 UC 添加到 K 中. 如果可满足, 则找到了最小代价的解, 求解终止. 由于每轮求解时都是在 F' 的基础上假设 MCHS 以外的子句为可满足的, 因此会反复调用增量求解器.

4.2 增量 SMT 的应用

增量 SMT 主要被用在软件分析验证等领域, 我们列举了其主要应用方向, 包括符号执行、重写理论 SMT 以及神经网络验证等.

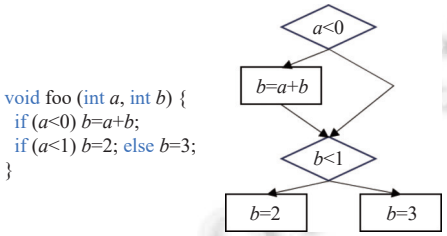
(1) 符号执行

符号执行可用于测试用例生成等场景, 它会将所有输入作为符号值, 它会将每条路径编码为基于输入变量符号值的路径条件. 路径条件会被定义为一系列表达式的合取 $pc = \Lambda b_i$, 其中每个 b_i 表示在当前执行路径过程中进行的分支决策. 在遇到分支时, 符号执行需要调用 SMT 求解器来求解路径条件, 从而决策选择哪个控制流分支.

由于路径条件在同一条控制流路径上是累加的, 即同一条路径上后续的路径条件会包含之前的, 因此可以利用增量求解来复用之前的计算. 文献 [13] 中提出了基于增量 SMT 的 push/pop 接口的求解方式. 具体而言, 符号执

行会按深度优先搜索顺序遍历控制流图, 在到达一个新的分支处时, 向 SMT 求解器的断言栈添加 (push) 分支条件, 直到离开该分支时会将其从求解器的断言栈中删除 (pop). 实验证明, 在大规模 C 程序实例以及随机生成实例上, 相比于基线版本, 该方法可以提升求解速度 2–5 倍.

例 17: 如图 5(a) 中的示例代码所示, 按深度优先搜索遍历图 5(b) 的控制流图时. 符号执行调用 SMT 求解器的公式依次为 $(a<0)$, $(a<0)\wedge(a+b<1)$, $(a<0)\wedge\neg(a+b<1)$, $\neg(a<0)$, $\neg(a<0)\wedge(b<1)$, $\neg(a<0)\wedge\neg(b<1)$. 因此, 使用 push/pop 接口时, 调用的流程如下: (push); (assert a<0); (check-sat); (push); (assert a+b<1); (check-sat); (pop); (push); (assert $\neg(a+b<1)$); (check-sat); (pop); (pop); (push); (assert $\neg(a<0)$); (check-sat); (push); (assert $(b<1)$; (check-sat); (pop); (push); (assert $\neg(b<1)$); (check-sat).

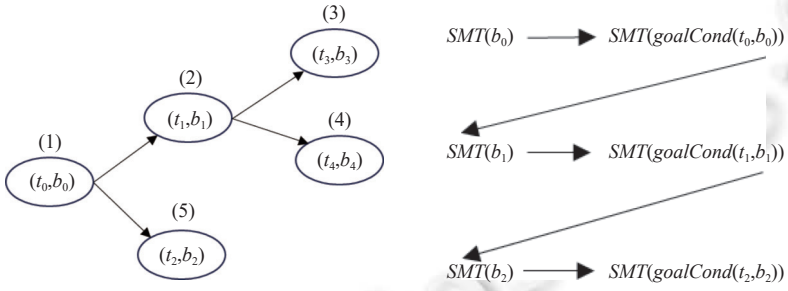


(a) 示例代码 (b) 示例控制流图

图 5 示例代码和示例控制流图

(2) 重写理论 SMT

重写模理论 SMT[132]被用于网络安全验证[133]和算法验证[134]. 在重写模理论 SMT 中定义了符号状态 (t,b), 以及重写规则: $(t,b)\rightarrow(t',b')$, 其中 t 和 t' 是包含变量的项, b 是条件, 重写规则表示如果满足条件 b', 即可从状态 (t,b) 转移到状态 (t',b'). 验证问题会被建模为对符号状态空间的可达性搜索问题, 即判断是否存在从状态 (t_0,b_0) 可达的状态 (t',b'), 使得目标条件 $goalCond(t',b')$ 成立. 如图 6(a) 中所示, 对符号状态空间按广度优先搜索 (breath first search, BFS) 时, 对 SMT 的调用如图 6(b) 中所示. 对于某个状态 (t,b), 会先用 SMT 求解条件 b, 判断该状态是否可达, 然后用 SMT 求解 $goalCond(t,b)$ 判断目标条件是否可满足.



(a) BFS 进行搜索时的顺序 (b) SMT 求解器的调用顺序

图 6 用 BFS 进行搜索时的顺序和 SMT 求解器的调用顺序

文献 [135] 中提出了将增量 SMT 求解应用于重写理论 SMT 中, 其中识别了一系列普适的可用于增量求解的增量重写规则: 形如 $(t,b)\rightarrow(t',b\wedge b')$. 此类规则只允许向条件中添加新的约束. 深度优先搜索 (depth first search, DFS) 可以利用 SMT 求解器的增量求解能力. 例如图 6(a) 中, 若在用 SMT 求解器求解完 b_1 之后再求解 $b_3=b_1\wedge b_{1,3}$ 时, 则可以直接添加 (push) 约束 $b_{1,3}$, 将之前求解时的学习子句复用. 相比之下, BFS 并不能复用历史求解信息, 例如求解 b_1 后求解 b_2 时无法复用学习子句, 因为 b_1 不是 b_2 的子公式, 需要先删除 b_1. 但 BFS 有利于提高搜索的覆盖率, 因为它会在搜索树的不同分支中切换. 该工作中提出将 DFS 和 BFS 混合使用, 即指定 DFS 的搜索深度, 当达到指定深度时便切换回 BFS. 实验证明, 混合求解的速度相比于只应用 BFS 求解要快 10 倍以上.

(3) 神经网络验证

深度神经网络在进行微调或修复等调整后, 通常需要进行全面的重新验证以确保新网络维持所需的特性. 然而, 传统验证方法在面对大规模复杂的神经网络时, 重新验证过程往往耗时且效率低下. 因此需要探索更高效的验证手段. 增量验证技术通过保留首次验证时的关键信息, 显著提高了后续验证的效率.

DeepInc[14]工具实现了一个基于增量 SMT 求解器的框架, 用于对深度神经网络进行增量验证. 该框架在之前的验证过程中记录了由于分支选择产生的搜索树, 以及叶子节点 (已经证明状态为 UNSAT 或 SAT 的节点) 的证明格局 (如分支决策情况等). 由于网络的修复或调整通常只涉及微小的改动, 原先验证中已经确定状态的叶节点在网络权重的微小变化后很可能保持状态不变. 在增量验证过程中, DeepInc 会直接访问这些叶子节点, 并尝试使用相同的证明格局进行验证. 如果验证成功, 工具将转向下一个叶子节点; 若失败, 则在当前状态下继续搜索. 这种方法避免了重新构建搜索树, 从而实现了验证加速.

5 总结与展望

本文对增量 SAT/SMT 问题的求解技术与应用进行了综述. 目前, 增量 SAT/SMT 问题的完备算法已经得到了比较深入的研究, 并且目前已经成功应用于模型检测, 符号执行等领域. 其关注的重点在于如何在后续的求解中复用之前得到的学习子句, 以及如何将增量求解技术和化简技术相结合. 然而, 对于增量 SAT/SMT 问题的不完备算法研究还比较初步. 在增量 SAT 问题中, 不完备算法仅支持添加子句集合的操作, 而不能支持删除子句, 并且对于历史信息的利用仅限于上一轮的求解结果, 其他搜索信息, 例如子句权重等, 都未被充分利用. 此外, 在增量 SMT 领域, 不完备算法的研究尚属空白.

综合以上分析, 下面给出增量 SAT/SMT 领域的研究提出若干展望.

(1) 为增量 SMT 问题设计不完备算法: 如果对公式的修改较小, 可能只需要对当前解进行少数迭代修改, 便可以得到新公式的可满足解. 因此可以充分发挥不完备算法对解空间的采样能力, 快速找到新的可满足解.

(2) 为增量 SAT 问题设计不完备算法, 支持更丰富的操作, 利用更丰富的历史信息指导搜索.

(3) 将不完备算法和完备算法进行混合增量求解: 在非增量的 SAT/SMT 求解领域中, 混合求解技术已经得到了较多的探索[136,137], 并且展现出优秀的求解效果. 然而其在增量 SAT/SMT 领域的应用尚属空白. 可以尝试在加入新子句时, 首先利用局部搜索等不完备算法对解空间进行快速采样, 尝试在当前解附近寻找新的解, 如果不能找到新的可满足解, 再将不完备搜索时得到的信息用于指导完备搜索.

References

[1] Biere A, Fleury M, Froleyks N, Heule MJH. The SAT museum. In: Proc. of the 14th Int'l Workshop on Pragmatics of SAT Co-located with the 26th Int'l Conf. on Theory and Applications of Satisfiability Testing. Alghero: SAT, 2023. 72–87.

[2] Rungta N. A billion SMT queries a day (invited paper). In: Proc. of the 34th Int'l Conf. on Computer Aided Verification. Haifa: Springer, 2022. 3–18. [doi: 10.1007/978-3-031-13185-1_1]

[3] Cai SW, Chen ZB, Wang J, Zhan BH, Zhao YW. Preface to the special topic on constraint solving and theorem proving. Ruan Jian Xue Bao/Journal of Software, 2023, 34(8): 3465–3466 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6871.htm [doi: 10.13328/j.cnki.jos.006871]

[4] Kochemazov S, Ignatiev A, Marques-Silva J. Assessing progress in SAT solvers through the lens of incremental SAT. In: Proc. of the 24th Int'l Conf. on Theory and Applications of Satisfiability Testing. Barcelona: Springer, 2021. 280–298. [doi: 10.1007/978-3-030-80223-3_20]

[5] Disch S, Scholl C. Combinational equivalence checking using incremental SAT solving, output ordering, and resets. In: Proc. of the 2007 Asia and South Pacific Design Automation Conf. Yokohama: IEEE, 2007. 938–943. [doi: 10.1109/ASPDAC.2007.358110]

[6] Silva JOM, Sakallah KA. Robust search algorithms for test pattern generation. In: Proc. of the 27th IEEE Int'l Symp. on Fault Tolerant Computing. Seattle: IEEE, 1997. 152–161. [doi: 10.1109/FTCS.1997.614088]

[7] de Ita G, Contreras M, Bello P. Applying an incremental satisfiability algorithm to automatic test pattern generation. Research in Computing Science, 2016, 123(1): 39–49. [doi: 10.13053/rcs-123-1-3]

[8] Eén N, Sörensson N. Temporal induction by incremental SAT solving. Electronic Notes in Theoretical Computer Science, 2003, 89(4): 543–560. [doi: 10.1016/S1571-0661(05)82542-3]

[9] Bradley AR. IC3 and beyond: Incremental, inductive verification. In: Proc. of the 24th Int'l Conf. on Computer Aided Verification. Berkeley: Springer, 2012. 4. [doi: 10.1007/978-3-642-31424-7_4]

[10] Nadel A. Boosting minimal unsatisfiable core extraction. In: Proc. of the 2010 Conf. on Formal Methods in Computer Aided Design. Lugano: IEEE, 2010. 221–229.

[11] Oh Y, Mneimneh MN, Andraus ZS, Sakallah KA, Markov IL. AMUSE: A minimally-unsatisfiable subformula extractor. In: Proc. of the 41st Annual Design Automation Conf. San Diego: IEEE, 2004. 518–523.

[12] Fu ZH, Malik S. On solving the partial MAX-SAT problem. In: Proc. of the 9th Int'l Conf. on Theory and Applications of Satisfiability Testing. Seattle: Springer, 2006. 252–265. [doi: 10.1007/11814948_25]

[13] Liu TH, Araújo M, D'Amorim M, Taghdiri M. A comparative study of incremental constraint solving approaches in symbolic execution. In: Proc. of the 10th Int'l Haifa Verification Conf. on Hardware and Software: Verification and Testing. Haifa: Springer, 2014. 284–299. [doi: 10.1007/978-3-319-13338-6_21]

[14] Yang PF, Chi ZM, Liu ZX, Zhao MY, Huang CC, Cai SW, Zhang LJ. Incremental satisfiability modulo theory for verification of deep neural networks. arXiv:2302.06455, 2023.

[15] Davis M, Logemann G, Loveland D. A machine program for theorem-proving. Communications of the ACM, 1962, 5(7): 394–397. [doi: 10.1145/368273.368557]

[16] Marques-Silva JP, Sakallah KA. GRASP: A search algorithm for propositional satisfiability. IEEE Trans. on Computers, 1999, 48(5): 506–521. [doi: 10.1109/12.769433]

[17] Marques-Silva J, Lynce I, Malik S. Conflict-driven clause learning SAT solvers. In: Biere A, Heule M, van Maaren H, Walsh T, eds. Handbook of Satisfiability. Amsterdam: IOS Press, 2021. 133–182. [doi: 10.3233/978-1-58603-929-5-131]

[18] Eén N, Sörensson N. An extensible SAT-solver. In: Proc. of the 6th Int'l Conf. on Theory and Applications of Satisfiability Testing. Santa Margherita Ligure: Springer, 2004. 502–518. [doi: 10.1007/978-3-540-24605-3_37]

[19] Audemard G, Simon L. Predicting learnt clauses quality in modern SAT solvers. In: Proc. of the 21st Int'l Joint Conf. on Artificial Intelligence. Pasadena: ACM, 2009. 399–404.

[20] Biere A. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT Competition 2018. 2017. https://fmv.jku.at/papers/Biere-SAT-Competition-2018-solvers.pdf

[21] Queue SD. CaDiCaL at the SAT Race 2019. 2019. https://cca.informatik.uni-freiburg.de/papers/Biere-SAT-Race-2019-solvers.pdf

[22] Selman B, Kautz HA, Cohen B. Noise strategies for improving local search. In: Proc. of the 12th National Conf. on Artificial Intelligence. Seattle: AAAI, 1994. 337–343.

[23] Hoos HH. On the run-time behaviour of stochastic local search algorithms for SAT. In: Proc. of the 16th National Conf. on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conf. Innovative Applications of Artificial Intelligence. Orlando: AAAI, 1999. 661–666.

[24] Cai SW, Luo C, Su KL. CCAnr: A configuration checking based local search solver for non-random satisfiability. In: Proc. of the 18th Int'l Conf. on Theory and Applications of Satisfiability Testing. Austin: Springer, 2015. 1–8. [doi: 10.1007/978-3-319-24318-4_1]

[25] Eén N, Biere A. Effective preprocessing in SAT through variable and clause elimination. In: Proc. of the 8th Int'l Conf. on Theory and Applications of Satisfiability Testing. St. Andrews: Springer, 2005. 61–75. [doi: 10.1007/11499107_5]

[26] Järvisalo M, Biere A, Heule M. Blocked clause elimination. In: Proc. of the 16th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Paphos: Springer, 2010. 129–144. [doi: 10.1007/978-3-642-12002-2_10]

[27] Wang JX, Guan LN, Jiang GH. A survey of algorithms for SAT problem. Computing Technology and Automation, 2009, 28(4): 138–143 (in Chinese with English abstract). [doi: 10.3969/j.issn.1003-6199.2009.04.036]

[28] Wang XF, Pang LC, Mo CH, Yang Y, Zhao XY, Yang L. A review of structural characteristics of satisfiability problems. Journal of Zhengzhou University (Engineering Science), 2023, 44(6): 40–47 (in Chinese with English abstract). [doi: 10.13705/j.issn.1671-6833.2023.03.013]

[29] Barrett C, Tinelli C. Satisfiability modulo theories. In: Clarke EM, Henzinger TA, Veith H, Bloem R, eds. Handbook of Model Checking. Cham: Springer, 2018. 305–343. [doi: 10.1007/978-3-319-10575-8_11]

[30] Sebastiani R. Lazy satisfiability modulo theories. Journal on Satisfiability, Boolean Modeling and Computation, 2007, 3(3–4): 141–224. [doi: 10.3233/SAT190034]

[31] Nieuwenhuis R, Oliveras A, Tinelli C. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(*T*). Journal of the ACM, 2006, 53(6): 937–977. [doi: 10.1145/1217856.1217859]

[32] Pnueli A, Rodeh Y, Strichman O, Siegel M. The small model property: How small can it be? Information and Computation, 2002, 178(1): 279–293. [doi: 10.1006/inco.2002.3175]

[33] Talupur M, Sinha N, Strichman O, Pnueli A. Range allocation for separation logic. In: Proc. of the 16th Int'l Conf. on Computer Aided Verification. Boston: Springer, 2004. 148–161. [doi: 10.1007/978-3-540-27813-9_12]

[34] Cai SW, Li BH, Zhang XD. Local search for SMT on linear integer arithmetic. In: Proc. of the 34th Int'l Conf. on Computer Aided Verification. Haifa: Springer, 2022. 227–248. [doi: 10.1007/978-3-031-13188-2_12]

[35] Cai SW, Li BH, Zhang XD. Local search for satisfiability modulo integer arithmetic theories. ACM Trans. on Computational Logic, 2023, 24(4): 32. [doi: 10.1145/3597495]

[36] Li HK, Xia BC, Zhao TQ. Local search for solving satisfiability of polynomial formulas. In: Proc. of the 35th Int'l Conf. on Computer Aided Verification. Paris: Springer, 2023. 87–109. [doi: 10.1007/978-3-031-37703-7_5]

[37] Li BH, Cai SW. Local search for SMT on linear and multilinear real arithmetic. In: Proc. of the 23rd Conf. on Formal Methods in Computer-aided Design. Vienna: TU Wien Academic Press, 2023. 168–177. [doi: 10.34727/2023/isbn.978-3-85448-060-0_25]

[38] Fröhlich A, Biere A, Wintersteiger C, Hamadi Y. Stochastic local search for satisfiability modulo theories. In: Proc. of the 29th AAAI Conf. on Artificial Intelligence. Austin: AAAI, 2015. [doi: 10.1609/aaai.v29i1.9372]

[39] Niemetz A, Preiner M, Biere A. Precise and complete propagation based local search for satisfiability modulo theories. In: Proc. of the 28th Int'l Conf. on Computer Aided Verification. Toronto: Springer, 2016. 199–217. [doi: 10.1007/978-3-319-41528-4_11]

[40] Tang A, Wang XF, He F. A survey of satisfiability modulo theories. Computer Engineering & Science, 2024, 46(3): 400–415 (in Chinese with English abstract). [doi: 10.3969/j.issn.1007-130X.2024.03.003]

[41] Wang C, Lü YR, Chen L, Wang XL, Wang YJ. Survey on development of solving methods and state-of-the-art applications of satisfiability modulo theories. Journal of Computer Research and Development, 2017, 54(7): 1405–1425 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2017.20160303]

[42] Biere A, Järvisalo M, Kiesl B. Preprocessing in SAT solving. In: Biere A, Heule M, van Maaren H, Walsh T, eds. Handbook of Satisfiability. Amsterdam: IOS Press, 2021. 391–435. [doi: 10.3233/FAIA200992]

[43] Hooker JN. Solving the incremental satisfiability problem. The Journal of Logic Programming, 1993, 15(1–2): 177–186. [doi: 10.1016/0743-1066(93)90018-C]

[44] Davis M, Putnam H. A computing procedure for quantification theory. Journal of the ACM, 1960, 7(3): 201–215. [doi: 10.1145/321033.321034]

[45] Hooker JN. A quantitative approach to logical inference. Decision Support Systems, 1988, 4(1): 45–69. [doi: 10.1016/0167-9236(88)90097-8]

[46] Loveland DW. Automated Theorem Proving: A Logical Basis. New York: North-Holland, 1978.

[47] Bennaceur H, Gouachi I, Plateau G. An incremental branch-and-bound method for the satisfiability problem. INFORMS Journal on Computing, 1998, 10(3): 301–308. [doi: 10.1287/ijoc.10.3.301]

[48] Kim J, Whittemore J, Sakallah K. On solving stack-based incremental satisfiability problems. In: Proc. of the 2000 Int'l Conf. on Computer Design. Austin: IEEE, 2000. 379–382. [doi: 10.1109/ICCD.2000.878311]

[49] Whittemore J, Kim J, Sakallah K. SATIRE: A new incremental satisfiability engine. In: Proc. of the 38th Design Automation Conf. Las Vegas: IEEE, 2001. 542–545. [doi: 10.1145/378239.379019]

[50] Audemard G, Lagniez JM, Simon L. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: Proc. of the 16th Int'l Conf. on Theory and Applications of Satisfiability Testing. Helsinki: Springer, 2013. 309–317. [doi: 10.1007/978-3-642-39071-5_23]

[51] Hoos HH, O'Neill K. Stochastic local search methods for dynamic SAT—An initial investigation. Technical Report, Vancouver: University of British Columbia, 2000. 22–26.

[52] McAllester D, Selman B, Kautz H. Evidence for invariants in local search. In: Proc. of the 14th National Conf. on Artificial Intelligence and the 9th Conf. on Innovative Applications of Artificial Intelligence. Rhode Island: AAAI, 1997. 321–326.

[53] Mouhoub M. Stochastic local search for incremental SAT. Int'l Journal of Knowledge-based and Intelligent Engineering Systems, 2005, 9(3): 191–195. [doi: 10.3233/KES-2005-9303]

[54] Mouhoub M, Sadaoui S. Solving incremental satisfiability. Int'l Journal on Artificial Intelligence Tools, 2007, 16(1): 139–147. [doi: 10.1142/S0218213007003254]

[55] Selman B, Kautz H. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In: Proc. of the 13th Int'l Joint Conf. on Artifical Intelligence. Chambery: ACM, 1993. 290–295.

[56] Mouhoub M. Systematic versus local search and GA techniques for incremental SAT. Int'l Journal of Computational Intelligence and

Applications, 2008, 7(1): 77–96. [doi: 10.1142/S1469026808002193]

[57] Mouhoub M, Sadaoui S, Feng XK. A new branch and bound method for incremental satisfiability problem. In: Proc. of the 2004 Int'l Conf. on Computational Intelligence. Istanbul: ICCI, 2004. 424–427.

[58] Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Heidelberg: Springer, 1992.

[59] El Bachir Menaï M. An evolutionary local search method for incremental satisfiability. In: Proc. of the 7th Int'l Conf. on Artificial Intelligence and Symbolic Computation. Linz: Springer, 2004. 143–156. [doi: 10.1007/978-3-540-30210-0_13]

[60] Boettcher S, Percus A. Nature's way of optimizing. Artificial Intelligence, 2000, 119(1–2): 275–286. [doi: 10.1016/S0004-3702(00)00007-2]

[61] Kupferschmid S, Lewis M, Schubert T, Becker B. Incremental preprocessing methods for use in BMC. Formal Methods in System Design, 2011, 39(2): 185–204. [doi: 10.1007/s10703-011-0122-4]

[62] Khasidashvili Z, Nadel A. Implicative simultaneous satisfiability and applications. In: Proc. of the 7th Int'l Haifa Verification Conf. on Hardware and Software: Verification and Testing. Haifa: Springer, 2013. 66–79. [doi: 10.1007/978-3-642-34188-5_9]

[63] Nadel A, Ryvchin V. Efficient SAT solving under assumptions. In: Proc. of the 15th Int'l Conf. on Theory and Applications of Satisfiability Testing. Trento: Springer, 2012. 242–255. [doi: 10.1007/978-3-642-31612-8_19]

[64] Nadel A, Ryvchin V, Strichman O. Ultimately incremental SAT. In: Proc. of the 17th Int'l Conf. on Theory and Applications of Satisfiability Testing. Vienna: Springer, 2014. 206–218. [doi: 10.1007/978-3-319-09284-3_16]

[65] Nadel A, Ryvchin V, Strichman O. Preprocessing in incremental SAT. In: Proc. of the 15th Int'l Conf. on Theory and Applications of Satisfiability Testing. Trento: Springer, 2012. 256–269. [doi: 10.1007/978-3-642-31612-8_20]

[66] Fazekas K, Biere A, Scholl C. Incremental inprocessing in SAT solving. In: Proc. of the 22nd Int'l Conf. on Theory and Applications of Satisfiability Testing. Lisbon: Springer, 2019. 136–154. [doi: 10.1007/978-3-030-24258-9_9]

[67] Järvisalo M, Heule MJH, Biere A. Inprocessing rules. In: Proc. of the 6th Int'l Joint Conf. on Automated Reasoning. Manchester: Springer, 2012. 355–370. [doi: 10.1007/978-3-642-31365-3_28]

[68] Kiesl-Reiter B, Whalen MW. Proofs for incremental SAT with inprocessing. In: Proc. of the 2023 Conf. on Formal Methods in Computer-aided Design. Ames: IEEE, 2023. 132–140. [doi: 10.34727/2023/isbn.978-3-85448-060-0_21]

[69] Khasidashvili Z, Nadel A, Palti A, Hanna Z. Simultaneous SAT-based model checking of safety properties. In: Proc. of the 1st Int'l Haifa Verification Conf. on Hardware and Software, Verification and Testing. Haifa: Springer, 2006. 56–75. [doi: 10.1007/11678779_5]

[70] Wieringa S, Heljanko K. Asynchronous multi-core incremental SAT solving. In: Proc. of the 19th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Rome: Springer, 2013. 139–153. [doi: 10.1007/978-3-642-36742-7_10]

[71] Wieringa S. On incremental satisfiability and bounded model checking. In: Proc. of the 1st Int'l Workshop on Design and Implementation of Formal Tools and Systems. Austin: DIFTS@FMCAD, 2011.

[72] Froleyks N, Heule M, Iser M, Järvisalo M, Suda M. SAT Competition 2020. Artificial Intelligence, 2021, 301: 103572. [doi: 10.1016/j.artint.2021.103572]

[73] Manthey N. Riss 7.1 at SAT Race 2019. 2019. https://core.ac.uk/download/pdf/245128292.pdf#page=37

[74] Chen J. MiniSAT BCD and abcdSAT: Solvers based on blocked clause decomposition. 2015. https://paperzz.com/doc/7057786/minisat-bcd-and-abcdsat--solvers-based-on-blocked-clause

[75] Biere A, Fazekas K, Fleury M, Heisinger M. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Proc. of the SAT Competition 2020—Solver and Benchmark Descriptions. Helsinki: University of Helsinki, 2020. 50–53.

[76] Soos M. The CryptoMiniSat 5 set of solvers at SAT Competition 2016. 2016. https://helda.helsinki.fi/bitstreams/643324a8-982d-4510-a310-2d73d7ba6fef/download#page=28

[77] Audemard G, Simon L. On the Glucose SAT solver. Int'l Journal on Artificial Intelligence Tools, 2018, 27(1): 1840001. [doi: 10.1142/S0218213018400018]

[78] Biere A. Lingeling, Plingeling and Treengeling entering the SAT Competition 2013. In: Proc. of the SAT Competition 2013. Helsinki: University of Helsinki, 2013.

[79] Biere A. PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation, 2008, 4(2–4): 75–97. [doi: 10.3233/SAT190039]

[80] Biere A, Faller T, Fazekas K, Fleury M, Froleyks N, Pollitt F. CaDiCaL 2.0. In: Proc. of the 36th Int'l Conf. on Computer Aided Verification. Montreal: Springer, 2024. 133–152. [doi: 10.1007/978-3-031-65627-9_7]

[81] Balyo T, Biere A, Iser M, Sinz C. SAT race 2015. Artificial Intelligence, 2016, 241: 45–65. [doi: 10.1016/j.artint.2016.08.007]

[82] Preiner M, Schurr HJ, Barrett C, Fontaine P, Niemetz A, Tinelli C. SMT-LIB release 2024 (incremental benchmarks). 2024. https://zenodo.org/records/11186591 [doi: 10.5281/zenodo.11061219]

[83] Bjørner N, Fazekas K. On incremental pre-processing for SMT. In: Proc. of the 29th Int'l Conf. on Automated Deduction. Rome: Springer, 2023. 41–60. [doi: 10.1007/978-3-031-38499-8_3]

[84] Dutertre B, de Moura L. A fast linear-arithmetic solver for DPLL(*T*). In: Proc. of the 18th Int'l Conf. on Computer Aided Verification. Seattle: Springer, 2006. 81–94. [doi: 10.1007/11817963_11]

[85] King T, Barrett C, Dutertre B. Simplex with sum of infeasibilities for SMT. In: Proc. of the 2013 Formal Methods in Computer-aided Design. Portland: IEEE, 2013. 189–196. [doi: 10.1109/FMCAD.2013.6679409]

[86] Dutertre B, de Moura L. Integrating simplex with DPLL(*T*). Technical Report, SRI-CSL-06–01, Menlo Park: SRI Int'l, 2006.

[87] Dillig I, Dillig T, Aiken A. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In: Proc. of the 21st Int'l Conf. on Computer Aided Verification. Grenoble: Springer, 2009. 233–247. [doi: 10.1007/978-3-642-02658-4_20]

[88] Griggio A. A practical approach to satisfiability modulo linear integer arithmetic. Journal on Satisfiability, Boolean Modeling and Computation, 2012, 8(1–2): 1–27. [doi: 10.3233/SAT190086]

[89] Bromberger M, Weidenbach C. Fast cube tests for LIA constraint solving. In: Proc. of the 8th Int'l Joint Conf. on Automated Reasoning. Coimbra: Springer, 2016. 116–132. [doi: 10.1007/978-3-319-40229-1_9]

[90] Ganesh V, Dill DL. A decision procedure for bit-vectors and arrays. In: Proc. of the 19th Int'l Conf. on Computer Aided Verification. Berlin: Springer, 2007. 519–531. [doi: 10.1007/978-3-540-73368-3_52]

[91] Brummayer R, Biere A. Boolector: An efficient SMT solver for bit-vectors and arrays. In: Proc. of the 15th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems. York: Springer, 2009. 174–177. [doi: 10.1007/978-3-642-00768-2_16]

[92] Niemetz A, Preiner M. Bitwuzla. In: Proc. of the 35th Int'l Conf. on Computer Aided Verification. Paris: Springer, 2023. 3–17. [doi: 10.1007/978-3-031-37703-7_1]

[93] Barrett C, Barbosa H, Brain M, Ibeling D, King T, Meng P, *et al*. CVC4 at the SMT Competition 2019. 2019. https://cs.stanford.edu/~preiner/publications/2019/CVC4-SMT-Competition-2019.pdf

[94] Barbosa H, Barrett C, Brain M, Kremer G, Lachnitt H, Mann M, Mohamed A, Mohamed M, Niemetz A, Nötzli A, Ozdemir A, Preiner M, Reynolds A, Sheng Y, Tinelli C, Yoni Zohar Y. CVC5: A versatile and industrial-strength SMT solver. In: Proc. of the 28th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Munich: Springer, 2022. 415–442. [doi: 10.1007/978-3-030-99524-9_24]

[95] Zohar Y, Irfan A, Mann M, Niemetz A, Nötzli A, Preiner M, Reynolds A, Barrett C, Cesare Tinelli C. Bit-precise reasoning via int-blasting. In: Proc. of the 23rd Int'l Conf. on Verification, Model Checking, and Abstract Interpretation. Philadelphia: Springer, 2022. 496–518. [doi: 10.1007/978-3-030-94583-1_24]

[96] Yao PS, Shi QK, Huang HQ, Zhang C. Fast bit-vector satisfiability. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 38–50. [doi: 10.1145/3395363.3397378]

[97] Hadarean L, Bansal K, Jovanović D, Barrett C, Tinelli C. A tale of two solvers: Eager and lazy approaches to bit-vectors. In: Proc. of the 26th Int'l Conf. on Computer Aided Verification. Vienna: Springer, 2014. 680–695. [doi: 10.1007/978-3-319-08867-9_45]

[98] de Moura L, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Budapest: Springer, 2008. 337–340. [doi: 10.1007/978-3-540-78800-3_24]

[99] Dutertre B. Yices 2.2. In: Proc. of the 26th Int'l Conf. on Computer Aided Verification. Vienna: Springer, 2014. 737–744. [doi: 10.1007/978-3-319-08867-9_49]

[100] Cimatti A, Griggio A, Schaafsma BJ, Sebastiani R. The MathSAT5 SMT solver. In: Proc. of the 19th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Rome: Springer, 2013. 93–107. [doi: 10.1007/978-3-642-36742-7_7]

[101] Bruttomesso R, Pek E, Sharygina N, Tsitovich A. The openSMT solver. In: Proc. of the 16th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Paphos: Springer, 2010. 150–153. [doi: 10.1007/978-3-642-12002-2_12]

[102] Christ J, Hoenicke J, Nutz A. SMTInterpol: An interpolating SMT solver. In: Proc. of the 19th Int'l SPIN Workshop on Model Checking of Software. Oxford: Springer, 2012. 248–254. [doi: 10.1007/978-3-642-31759-0_19]

[103] Zou SN, Zhang JX, Luo GJ. Incremental SAT-based exact synthesis. In: Proc. of the 2024 Great Lakes Symp. on VLSI 2024. Clearwater: ACM, 2024. 158–163. [doi: 10.1145/3649476.3658739]

[104] Alizadeh B, Sharafinejad SR. Incremental SAT-based accurate auto-correction of sequential circuits through automatic test pattern generation. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2019, 38(2): 245–252. [doi: 10.1109/TCAD.2018.2812123]

[105] Alizadeh B, Abadi Y. Incremental SAT-based correction of gate level circuits by reusing partially corrected circuits. IEEE Trans. on Circuits and Systems II: Express Briefs, 2020, 67(12): 3063–3067. [doi: 10.1109/TCSII.2020.2997212]

[106] Yang J, Kharkov YA, Shi YN, Heule MJH, Dutertre B. Quantum circuit mapping based on incremental and parallel SAT solving. In:

Proc. of the 27th Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT 2024). Pune: SAT, 2024. 29. [doi: 10.4230/LIPIcs.SAT.2024.29]

[107] Wang FZ, Liu JW, Young EFY. FastPass: Fast pin access analysis with incremental SAT solving. In: Proc. of the 2023 Int'l Symp. on Physical Design. ACM, 2023. 9–16. [doi: 10.1145/3569052.3571879]

[108] Shtrichman O. Pruning techniques for the SAT-based bounded model checking problem. In: Proc. of the 11th Int'l Conf. on Correct Hardware Design and Verification Methods. Scotland: Springer, 2001. 58–70. [doi: 10.1007/3-540-44798-9_4]

[109] Schrammel P, Kroening D, Brain M, Martins R, Teige T, Bienmüller T. Successful use of incremental BMC in the automotive industry. In: Proc. of the 20th Int'l Workshop on Formal Methods for Industrial Critical Systems. Oslo: Springer, 2015. 62–77. [doi: 10.1007/978-3-319-19458-5_5]

[110] Biere A, Cimatti A, Clarke E, Zhu YS. Symbolic model checking without BDDs. In: Proc. of the 5th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Amsterdam: Springer, 1999. 193–207. [doi: 10.1007/3-540-49059-0_14]

[111] Biere A, Cimatti A, Clarke EM, Fujita M, Zhu Y. Symbolic model checking using SAT procedures instead of BDDs. In: Proc. of the 36th Annual ACM/IEEE Design Automation Conf. New Orleans: ACM, 1999. 317–320. [doi: 10.1145/309847.309942]

[112] Shtrichman O. Tuning SAT checkers for bounded model checking. In: Proc. of the 12th Int'l Conf. on Computer Aided Verification. Chicago: Springer, 2000. 480–494. [doi: 10.1007/10722167_36]

[113] Biere A. Bounded model checking. In: Biere A, Heule M, van Maaren H, Walsh T, eds. Handbook of Satisfiability. Amsterdam: IOS press, 2021. 739–764.

[114] Bradley AR. SAT-based model checking without unrolling. In: Proc. of the 12th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation. Austin: Springer, 2011. 70–87. [doi: 10.1007/978-3-642-18275-4_7]

[115] Biere A, Kröning D. SAT-based model checking. In: Clarke EM, Henzinger TA, Veith H, Bloem R, eds. Handbook of Model Checking. Cham: Springer, 2018. 277–303. [doi: 10.1007/978-3-319-10575-8_10]

[116] Eén N, Mishchenko A, Brayton R. Efficient implementation of property directed reachability. In: Proc. of the 2011 Formal Methods in Computer-aided Design (FMCAD). Austin: IEEE, 2011. 125–134.

[117] Blankestijn M, Laarman A. Incremental property directed reachability. In: Proc. of the 24th Int'l Conf. on Formal Engineering Methods. Brisbane: Springer, 2023. 208–227. [doi: 10.1007/978-981-99-7584-6_13]

[118] Glorian G, Lagniez JM, Montmirail V, Szczepanski N. An incremental SAT-based approach to the graph colouring problem. In: Proc. of the 25th Int'l Conf. on Principles and Practice of Constraint Programming. Stamford: Springer, 2019. 213–231. [doi: 10.1007/978-3-030-30048-7_13]

[119] Zha A, Chang Q, Noda I. An incremental SAT-based approach for solving the real-time taxi-sharing service problem. Discrete Applied Mathematics, 2023, 335: 131–145. [doi: 10.1016/j.dam.2022.08.008]

[120] Huang SY, Cheng KT. Formal Equivalence Checking and Design Debugging. New York: Springer, 2012.

[121] Cohen O, Gordon M, Lifshits M, Nadel A, Ryvchin V. Designers work less with quality formal equivalence checking. 2010. https://dvcon-proceedings.org/wp-content/uploads/designers-work-less-with-quality-formal-equivalence-checking.pdf

[122] Grégoire É, Mazure B, Piette C. Extracting MUSes. In: Proc. of the 17th European Conf. on Artificial Intelligence. Amsterdam: IOS Press, 2006. 387–391.

[123] Bacchus F, Järvisalo M, Martins R. Maximum satisfiability. In: Biere A, Heule M, van Maaren H, Walsh T, eds. Handbook of Satisfiability. Amsterdam: IOS Press, 2021. 929–991.

[124] Marques-Silva J, Planes J. On using unsatisfiability for solving maximum satisfiability. arXiv:0712.1097, 2007.

[125] Martins R, Manquinho V, Lynce I. Open-WBO: A modular MaxSAT solver. In: Proc. of the 17th Int'l Conf. on Theory and Applications of Satisfiability Testing. Vienna: Springer, 2014. 438–445. [doi: 10.1007/978-3-319-09284-3_33]

[126] Martins R, Joshi S, Manquinho V, Lynce I. Incremental cardinality constraints for MaxSAT. In: Proc. of the 20th Int'l Conf. on Principles and Practice of Constraint Programming. Lyon: Springer, 2014. 531–548. [doi: 10.1007/978-3-319-10428-7_39]

[127] Bailleux O, Boufkhad Y. Efficient CNF encoding of boolean cardinality constraints. In: Proc. of the 9th Int'l Conf. on Principles and Practice of Constraint Programming. Kinsale: Springer, 2003. 108–122. [doi: 10.1007/978-3-540-45193-8_8]

[128] Davies J, Bacchus F. Solving MAXSAT by solving a sequence of simpler SAT instances. In: Proc. of the 17th Int'l Conf. on Principles and Practice of Constraint Programming. Perugia: Springer, 2011. 225–239. [doi: 10.1007/978-3-642-23786-7_19]

[129] Davies J, Bacchus F. Exploiting the power of MIP solvers in MaxSAT. In: Proc. of the 16th Int'l Conf. on Theory and Applications of Satisfiability Testing. Helsinki: Springer, 2013. 166–181. [doi: 10.1007/978-3-642-39071-5_13]

[130] Davies J, Bacchus F. Postponing optimization to speed up MaxSAT solving. In: Proc. of the 19th Int'l Conf. on Principles and Practice of Constraint Programing. Uppsala: Springer, 2013. 247–262. [doi: 10.1007/978-3-642-40627-0_21]

[131] Davies J. Solving MaxSAT by decoupling optimization and satisfaction [Ph.D. Thesis]. Toronto: University of Toronto, 2013.
[132] Rocha C, Meseguer J, Muñoz C. Rewriting modulo SMT and open system analysis. Journal of Logical and Algebraic Methods in Programming, 2017, 86(1): 269–297. [doi: 10.1016/j.jlamp.2016.10.001]
[133] Nigam V, Talcott C. Automating safety proofs about cyber-physical systems using rewriting modulo SMT. In: Proc. of the 14th Int'l Workshop on Rewriting Logic and Its Applications. Munich: Springer, 2022. 212–229. [doi: 10.1007/978-3-031-12441-9_11]
[134] Bae K, Rocha C. Symbolic state space reduction with guarded terms for rewriting modulo SMT. Science of Computer Programming, 2019, 178: 20–42. [doi: 10.1016/j.scico.2019.03.006]
[135] Whitters G, Nigam V, Talcott C. Incremental rewriting modulo SMT. In: Proc. of the 29th Int'l Conf. on Automated Deduction. Rome: Springer, 2023. 560–576. [doi: 10.1007/978-3-031-38499-8_32]
[136] Cai SW, Zhang XD. Deep cooperation of CDCL and local search for SAT. In: Proc. of the 24th Int'l Conf. Theory and Applications of Satisfiability Testing. Barcelona: Springer, 2021. 64–81. [doi: 10.1007/978-3-030-80223-3_6]
[137] Zhang XD, Li BH, Cai SW. Deep combination of CDCL(T) and local search for satisfiability modulo non-linear integer arithmetic theory. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: IEEE, 2024. 1534–1546. [doi: 10.1145/3597503.3639105]

附中文参考文献

[3] 蔡少伟, 陈振邦, 王戟, 赵永望. 约束求解与定理证明专题前言. 软件学报, 2023, 34(8): 3465–3466. http://www.jos.org.cn/1000-9825/6871.htm [doi: 10.13328/j.cnki.jos.006871]
[27] 王建新, 管利娜, 江国红. 可满足性问题的研究综述. 计算技术与自动化, 2009, 28(4): 138–143. [doi: 10.3969/j.issn.1003-6199.2009.04.036]
[28] 王晓峰, 庞立超, 莫淳惠, 杨易, 赵星宇, 杨澜. 可满足性问题的结构特征进展综述. 郑州大学学报 (工学版), 2023, 44(6): 40–47. [doi: 10.13705/j.issn.1671-6833.2023.03.013]
[40] 唐傲, 王晓峰, 何飞. 可满足性模理论综述. 计算机工程与科学, 2024, 46(3): 400–415. [doi: 10.3969/j.issn.1007-130X.2024.03.003]
[41] 王翀, 吕荫润, 陈力, 王秀利, 王永吉. SMT 求解技术的发展及最新应用研究综述. 计算机研究与发展, 2017, 54(7): 1405–1425. [doi: 10.7544/issn1000-1239.2017.20160303]

作者简介

李博涵, 博士, 特别研究助理, CCF 专业会员, 主要研究领域为可满足模理论的求解与应用.
蔡少伟, 博士, 研究员, CCF 杰出会员, 主要研究领域为约束求解, EDA 算法.