基于 ASP 的归纳一阶公式综合：不变式推断案例研究*

Ziyi Yang GeorgePîrlea IlyaSergey新加坡国立大学，新加坡 yangziyi@u.nus.edu gpirlea@u.nus.edu ilya@nus.edu.sg

我们提出了一个从示例中综合一阶逻辑（FOL）公式的框架，该框架统一并推进了用于推断迁移系统不变式的先进方法。为此，我们研究并分类了现有方法，以及它们通过答案集编程（ASP）进行公式综合时采用的编码技术。基于所得分类，我们提出了正交切片这一公式枚举新技术，将搜索空间划分为易于处理的块，从而支持两种增量式候选项剪枝方法。结合现有的一阶（FO）不变式综合技术，以及在我们的 FORCE 框架中实现的正交切片，我们显著加速了用于分布式系统不变式推断的先进算法。我们还展示了该方法如何促进不同不变式推断框架的组合，从而实现新的优化。

1 引言

一阶逻辑（FOL）已成功用于对复杂系统进行建模和验证，是一项基础工具。其应用遍及多个领域，从硬件设计 [5]到软件验证 [21]。这些成功很大程度上归功于先进框架的发展，这些框架支持自动化验证与综合，通常借助 Z3 [19] 和 cvc5 [1]等高性能证明器。

要在一阶逻辑中对复杂系统实现完全自动化的验证，通常需要综合能够刻画待验证系统不变式的公式（即始终成立的性质）。尽管这项任务本质上不可判定，近期许多研究仍取得了显著进展：它们通过示例（协议的采样轨迹或归纳反例）综合一阶公式，从而推断复杂分布式系统的归纳不变式（例如 Lamport 的 Paxos 共识协议 [16]）。现有方法由大量不同的框架实现 [27, 8], , 但目前仍缺少对这些框架之间联系的系统研究。这引发了一些重要问题：现有的综合方法是否从根本上互不重叠？为一种方法开发的技术能否经过调整后用于改进其他方法？解决这些问题不仅能加深我们对底层方法的理解，还能推动开发更出色的公式综合工具，并有望提升不同领域自动化验证工具的可扩展性。

本文提出一个统一框架，用于从示例——即公式所满足的一阶结构——综合有界一阶公式。我们的框架旨在对多种综合技术进行编码并结合 起来，从而能够与不同的高级算法无缝集成

*我们感谢 Andrew Cropper、Roland Kaminski 和 Jianan Yao 分别为 Popper、Clingo 和 DuoAI 提供帮助。我们也感谢 Roland Yap 对本文草稿提出的反馈。本研究部分得到了新加坡教育部（MoE）第三类资助“自动化程序修复”（MOE-MOET32021-0001）以及亚马逊研究奖“通过规约转换与综合扩展分布式协议的自动化验证”的支持。

M. Gebser、D.Inclezan、F. Ricca、M. Carro 和 M. Truszczynski：第 41 届国际逻辑编程会议（ICLP 2025） EPTCS 439，2026 年，第 511–527页，doi：10.4204/EPTCS.439.35

©Z.Yang 等本作品根据知识共享署名许可协议授权。

以进行不变式推断。为此，我们详细研究了九 种近期用于分布式系统（DS）不变式推断的方法，每种方法对一阶公式的归纳综合都有不同的思路。我们将研究结果概括为关于一阶公式归纳综合的两项主要观察方面，并指出最先进水平仍有改进空间：

O1 现有技术根据处理输入和结果的方式，可分为少数几个不同类别，例如，如何使用示例以及如何构造公式。我们对这些方法进行了统一分类。

O2 尽管综合技术种类繁多，但绝大多数都利用所采用的一阶理论的性质，归入两类之一：“冗余消除”和“增量剪枝”。

为构建一个能够涵盖 **O1** 的统一框架，我们采用答案集编程（ASP） [17], ，将基于枚举的一阶公式综合编码为约束求解，并将综合中的定制项和技术编码为知识表示。为改进现有技术，我们利用 **O2**，提出了正交切片这一一阶搜索空间的新方法（同样使用 ASP 实现），以便在归纳综合过程中高效剪枝候选公式。正交切片的核心思想是将搜索空间划分为有序切片，利用前面切片的综合结果，通过满足或不满足的公式剪枝后续切片。统一框架 FORCE (一阶**公式**综合器：基于**正**交切**片**）的实际优势，体现在它改进了两个最先进的分布式系统不变量综合器 DuoAI [27] 和Flyvy [8], ，且无需对其高级算法进行任何概念性修改。结果表明，我们的框架具有足够的表达能力和可扩展性，能够编码并组合现有的公式综合技术，从而推动分布式系统不变量综合领域的最先进水平。

2 概述

我们首先介绍一阶公式归纳综合的基础知识——这是现有分布式系统（DS）不变式推断框架中常用的子程序。随后，我们总结九种值得关注的现有分布式系统不变式推断方法（其中最早的方法发表于2019年），最后简要介绍我们基于答案集编程的框架，该框架用于捕获综合的各个方面及其特定实例——正交切片。

2.1 问题定义

一阶语言。 本文关注的是能够用带等号的一阶多排序逻辑表达的系统性质，并遵循常见的教科书定义。一个签名 $\Sigma = \langle C、R、F、S\rangle$由以下部分组成：一组常量符号 C、一组关系符号 （谓词）R、一组函数符号 F，以及一组用于变量、常量和函数符号的类型 S。在本文余下部分中，我们假设所有签名都是有限的，即，集合C、R、F和S 都是有限的。

逻辑项是递归定义的。项可以是常量 $c \in C$、变量 x，也可以是函数符号 $f \in F$ 作用于其他项的结果（例如，$f(x_1,x_2)$）。逻辑原子公式是将R 中的关系符号或等号应用于类型适当的项而形成的基本公式；文字则是原子公式或其否定（即， $\neg p(x)$）。公式通过对文字应用逻辑联结词（即，合取 $\wedge$和析取 $\vee$）以及量化（全称量词 $\forall$ 和存在量词 $\exists$）封闭构造而成。众所周知，任何一阶公式都可以转换为与之等价的前束范式公式，其中所有量词都置于开头。在这种结构中，前缀包含量词，而

矩阵由其余的布尔分量构成。例如，公式 $\forall x{:}\, s_1\, \exists y{:}\, s_2.\, p(x,y) \vee (\neg q(x) \wedge r(y))$ 属于前束范式，其前缀为 $\forall x{:}\, s_1\, \exists y{:}\, s_2$，矩阵为$p(x,y) \vee (\neg q(x) \wedge r(y))$。

定义 1（一阶综合问题）给定一组关于签名 Σ 的公式 Ω_0，以及一组一阶结构 $\sigma = \{M_1, \ldots, M_k\}$，其中每个 M_i 都是关于 Σ 的模型，求一组公式 $\Phi = \{\phi_1, \ldots, \phi_n\} \subseteq \Omega_0$，使得 $\forall \phi \in \Phi$:

1. $\forall M \in \sigma M \models \phi$，（满足所有输入的一阶结构）；*2.* 自由变量$(\phi) = /0$，（闭公式）；*3.* $\exists M \notin \sigma M \not\models \phi$，（非重言式）；*4.* $\forall \phi' \in \Phi. \phi \neq \phi' \wedge \phi \not\models \phi'$，（没有公式蕴含另一公式）。

换言之，该问题是在搜索空间 Ω_0 中寻找一组合取的良构公式 Φ，使其满足 σ中的所有给定一阶结构。这种合取描述了满足所有给定结构的“最精确”公式，因为不存在被任何 ϕ_i蕴含且又满足条件的公式。该问题可视为 [22] 在仅正例学习 [26]设定下的一般规格综合问题附录 A中给出了该问题的一个示例。

值得注意的是，我们对上述问题定义作出了若干假设和简化：（1）所有输入一阶结构的析取总是一个有效的（尽管是过拟合的）解，但在实践中，有意义的公式应当在受大小限制的搜索空间中寻找；（2）通过引入新文字，可以轻松处理包含函数符号和常量的搜索空间 [28, §4], 因此，我们将避免对其进行详细讨论；（3）我们假设用户希望综合前束析取范式公式，因为 DS 不变式通常以这种形式表示；我们的方法也可以扩展到其他一阶公式。

2.2 现有不变量推断算法的分类

现有的不变量推断方法采用不同的公式枚举技术，并运用多种优化手段来有效缩小搜索空间。我们总结了九种具有代表性的现有方法：I4 [18], FOL-IC3 [13], IC3PO [11], SWISS [12], DistAI [28], P-FOL-IC3 [14], DuoAI [27],Scimitar [25], 以及 Flyvy [8]（按其发表日期排序）。

这些类别基于五个详细的子方面。前两个子方面属于“系统级”，描述综合框架的高层设计；其余三个属于“算法级”：它们定义了应用于搜索空间穷举的剪枝技术。

2.2.1 系统级方面

推理模式。这一方面决定了整体推理过程如何使用一阶公式综合。具体而言，单轮 综合 [18, 28, 27] 会根据固定的输入示例（例如，分布式协议的采样轨迹），生成搜索空间中所有满足的公式；而多轮 综合[25, 8] 则基于逐步获取的示例（例如，当前不变式的反例），增量式地生成一组候选公式。组合方法 [13, 11, 12, 14] 使用系统轨迹或反例来引导多轮综合，并多次调用综合过程。

从历史上看，大多数多轮综合算法都可以视为 IC3 [2], 的扩展，而单轮综合则可以视为 Houdini [7] 的扩展。至关重要的是，这两种模式都采用了与我们所定义的合成问题类似的一阶综合问题作为基础，这意味着我们的 FORCE框架 可用于改进大多数现有方法。

语言限制。为了使合成问题易于处理，它通常被定义在一阶逻辑的某个子类中。最常见的是有效命题逻辑（EPR [23, 6]）片段，它是一阶逻辑的子集，其中的公式可以转换为等价的命题公式，从而使验证可被证明为可判定。

尽管现有的大多数方法都采用 EPR 及其扩展，并具有理论上的可判定性保证，但在实践中，综合方法往往还会施加其他额外的句法约束。例如，k-伪析取范式（在 P-FOL-IC3 [14], 中提出，并进一步应用于 [8]）是一种出于实际效率考虑的语法限制：它基于这样一种观察，即以这种形式书写的不变式公式比标准析取范式更小，从而缩小了需要探索的搜索空间。具体而言，k-伪析取范式公式的矩阵形式为 $c_1 \rightarrow (c_2 \vee \ldots \vee c_k)$，其中 c_i 是文字的合取。这本质上是一种搜索启发式方法，因为蕴含关系通常用于表达不变式，所以这种做法是合理的。

DistAI [28] 提出了子模板（进一步应用于 [27]）以提高效率，并利用了一阶逻辑与等号的以下性质：

$$\forall X_1, X_2 : T.mat(X_1, X_2) \equiv \forall X : T.mat(X, X) \wedge \forall X_1 \neq X_2 : T.mat(X_1, X_2) \tag{0}$$

利用这一性质，公式综合可以避免枚举 第 0 节 左侧的公式，因为它们等价于右侧两个公式的合取。因此，只需枚举右侧的两个子模板，即可综合出左侧满足条件的公式，从而加快枚举。

除了上述句法约束外，特定工具 [13, 12, 25, 8] 还会定义各自的句法定制项。然而，仔细审查现有的不变式推断实现后，我们发现，许多工具中的句法约束都无法扩展。例如，要通过添加子模板来扩展 k-伪析取范式，就需要修改 P-FOL-IC3中任何处理公式的算法。

2.2.2 算法层面

冗余消除。 最常用的剪枝方法是冗余消除，它广泛应用于各种综合任务。其思路是直接消除彼此等价的公式。通常会采用诸如“对称性剪枝”这样的近似方法（详见 第 3 节），通过识别量化下公式的对称性来剪除冗余公式。除基于句法的等价性外，冗余也可能由语义引入（例如，重言式和矛盾式），而 DistAI [28] 和 DuoAI [27]中使用了许多针对具体情况的规则。通过这种冗余消除的公式示例包括含有 $p(X) \wedge \neg p(X)$ 或 $p(X) \vee \neg p(X)$ 作为子表达式的公式。另一种处理冗余的方法是消除规范化；规范化由 Flyvy [8]实现：它也采用对称性破缺，但定义的是公式集合（而非单个公式）的偏序，以消除冗余。

这一类中一种更复杂但非常有效的剪枝策略在 DuoAI [27], 中提出，其中大量析取范式公式被证明可以通过其 分解而判定为冗余：$A \equiv B \wedge C$。与第 0 节类似，公式 A 可以分解为较小公式的合取，因此，如果所有较小公式都在搜索空间中，就可以剪枝掉原始公式。然而，Flyvy (参见 [8,附录 D]) 的作者发现，DuoAI 中的分解在某些情况下并不可靠（也就是说，它会导致过度剪枝），并提出了一个修订版本。我们发现，将 DuoAI 的实现切换为修订版本并不容易，因为它在整体综合循环中使用了原始分解的中间结果，这也使得公平比较两种方法的效率颇具挑战。

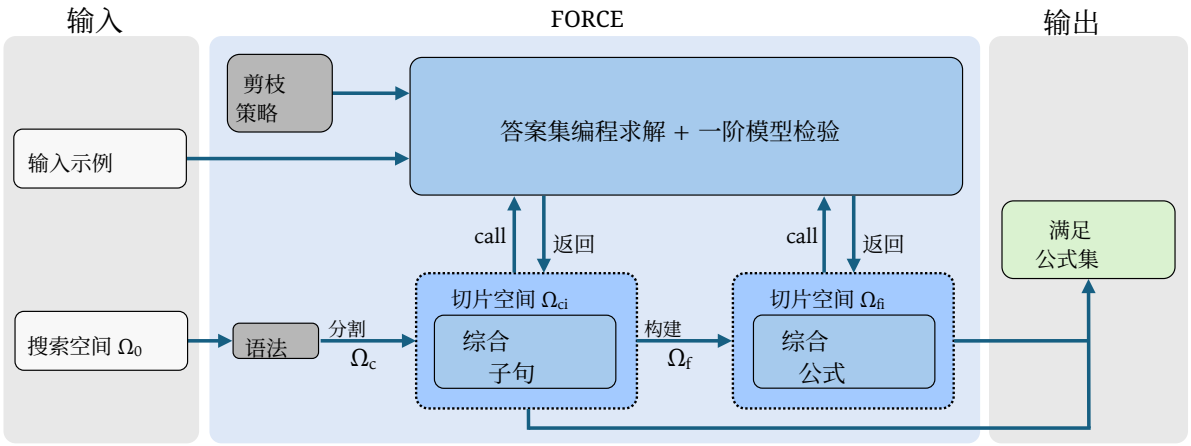

图 1：FORCE 的工作流程。

增量剪枝。 几乎所有高效的一阶逻辑公式综合方法都离不开增量剪枝。其思路是按照特定顺序，针对输入的一阶结构集合测试公式，并利用公式之间的蕴涵关系，免去对部分公式的测试。

第 4.1 节将详细介绍的蕴含图是 DuoAI [27] 在这一方面采用的一种有效技术。不仅如此，DuoAI 还识别出另一种基于共同蕴含的增量剪枝技术，该技术结合一阶模型检验的中间结果与冗余消除来剪枝搜索空间。例如，如果我们发现公式 $\forall x.P(x) \Rightarrow R(x)$ 满足所有示例，就不需要测试任何形如 prefix. $(P(x) \wedge R(x) \wedge F) \vee G$ 的公式，因为它们都被原公式和 prefix. $(P(x) \wedge F) \vee G$ 共同蕴含。

其他技术。现有的不变式推断工具中介绍了几种剪枝技术 [13,11, 25]。我们将在第 6 节讨论这些技术，因为它们并不普遍适用于所有一阶综合任务。

总的来说，对现有方法的分类表明，可以构建一个统一框架来涵盖现有的一阶综合技术并提出新技术。具体而言，我们发现，**语言**和**冗余**两个方面共同构成**静态**搜索空间，而**增量性**则定义了综合过程中对搜索空间进行动态剪枝。这些方面适用于任何一阶综合问题。

2.3 FORCE：基于答案集编程的综合 +正交切片

在介绍技术细节之前，我们先概述基于ASP的综合器 FORCE，其工作流程如图 1所示，它结合了剪枝策略与全新的正交切片技术。该框架由灰色部分定制：(1) 一阶语言（默认为析取范式）和 (2) 一组剪枝规则（预定义但可扩展）。给定初始搜索空间 Ω_0 和一组作为输入的一阶结构，FORCE 首先将子句的搜索空间 Ω_c 与 Ω_0分割。对于 Ω_c 的后续切片（称为Ω_{c_1}、 Ω_{c_2}.....），算法会生成其中的公式 Ω_{c_i} （通过 ASP 求解），在示例上测试这些公式（通过模型检查），并根据结果剪枝后续切片（$\Omega_{c_j},\ j > i$）。然后，利用 Ω_c的输出，算法进一步将公式搜索空间中除子句以外的部分 Ω_f构建为 $\Omega_{f_1}, \Omega_{f_2}, \ldots$，并重用 Ω_c 中的相同流程进行生成、测试和剪枝。最后，FORCE 输出满足输入条件的子句以及

```
1: tmp ← init()
2: 对所有 x ∈ Ω0 执行
3:     若 pruning1(x, tmp) 则
4:         继续
5:     否则若 pruning2(x, tmp) 则
6:         更新 2(tmp, x)
7:     else
8:         如果 满足(x)则
9:             tmp.update(x)
   返回 tmp.result
```

算法1 扩展定义 1 使用定制的剪枝规则

```
1: 函数 FOSYN(Ω0, σ, 剪枝)
2:     求解器 ← init(Ω0, 剪枝)
3:     当 solver.solve()时
4:         x ←求解器.get_current()
5:         如果 满足(x)则
6:             返回 x
```

图 2：综合循环（命令式编程与 ASP 求解）。

非子句公式。之所以称为正交，是因为 (1) 将 Ω_0 切片为 Ω_c 和Ω_f，以及 (2) 对 Ω_c和 Ω_f进行切片时，采用了不同的剪枝策略，并协同工作。

3 答案集编程中的一阶公式静态搜索空间

我们假设读者熟悉 ASP 的基础知识，例如规则、选择和聚合；更多详情请参阅相关文献 [9]。本节将展示 ASP 如何适用于一阶公式静态搜索空间的编码。

我们采用的高层方法遵循“生成并测试”工作流程，与其他不变量综合系统类似，但在搜索空间和剪枝技术的编码方式上有所不同。ASP 是一种适合在限制条件下进行穷举的范式（在我们的案例中，限制条件包括搜索空间定制项和剪枝技术）。如图 2所示，现有的命令式方法（左侧示例）需要手动整合剪枝步骤（第 3–6 行），由于依赖中间结果和求值顺序，代码可能变得脆弱且难以维护。相比之下，我们右侧基于 ASP 的方法将搜索空间和剪枝策略直接模块化到求解器中（第 2 行）。这种基于 ASP 的“生成并测试”循环通过简化新剪枝规则（作为新的领域知识）的整合以及自动处理依赖关系，提高了可扩展性和可维护性。本节余下部分将先展示如何对一阶公式的基本搜索进行编码，然后说明如何利用不同的剪枝知识定制搜索。

3.1 枚举的编码

下面说明如何使用答案集编程对一阶公式进行枚举。作为示例，我们采用 DuoAI 的 [27]用于综合不变式的一阶搜索空间配置，该配置针对`lockserv` 分布式协议。

```
变量：节点：n1、n2；锁：l1 关系：lock_msg：节点、锁；grant_msg：节
点、锁；unlock_msg：节点、锁；holds_lock：节点、锁；server_holds_
lock：锁 最大文字数：4 最大或数：3 最大与数：3 最大存在量词数：1
```

前两行“var”和“relations”共同指定了可用于公式的变量（属于 node 和 lock 类型）以及原子公式（由五种关系和有效变量构成）。请注意，在 EPR公式中，公式前缀中的变量顺序通常是固定的，也就是说，变量 l1 不能出现在任何节点变量（n1、n2）之前。随后，DNF 公式的搜索空间受到特定于问题的定制项限制：公式中文字的最大数量、

析取式的最大数量、立方项的最大数量（也就是说，文字合取），以及存在量词的最大数量。它们共同定义了 Ω_0 定义 1 中的定义 1。

得益于一阶公式前束范式的简洁性，枚举可以轻松地用答案集编程编码。公式的枚举需要在搜索空间的限制范围内生成公式的两个部分：前缀和矩阵。基本搜索空间的答案集编程编码如下所示：

```
var(node, n1). var(node, n2). var(lock,l1). %变量 0{exists(Var): var(Var, _)}1. %前缀受
max-exists限制 rel(lock_msg, (node, lock)). ... %关系 vars((node, lock), (n1, l1)). ... %变量元组
atom(Pred, Args) :- rel(Pred, Types),vars(Types, Args). %原子公式 pos(0..1). cube(1..3). %文字的符
号，max-or 0{lit_in_C(P,A,Pos,C):atom(P, A), pos(Pos)}3 :- cube(C). %矩阵受max-and限制 :- #
count{P,A,Pos,C: lit_in_C(P,A,Pos,C)} >= 4. %max-literal
```

在上述程序中，公式的前缀和矩阵是通过作用于 exists() 和 lit_in_C() 谓词的选择结构 0{...}n 生成的，并受到配置中参数的约束。最后一行则排除了对应公式中包含超过四个文字的答案集。例如，答案集 {lit_in_C(lock_msg,(n1,l1),0,1), lit_in_C(grant_msg,(n2,l1),1,1), lit_in_C(unlock_msg,(n1,l1),0,2), exists(l1)}cor- 对应于 $\forall$n1 n2，$\exists$l1。 (¬lock_msg(n1,l1) $\wedge$ grant_msg(n2,l1)) $\vee$¬unlock_msg(n1,l1)。还应注意，这里给出的编码旨在阐释枚举的基本思路；对于一阶逻辑公式的不同句法定制项（例如，前缀中的变量顺序），需要在编码中添加相应的谓词和规则。

3.2 通过冗余进行编码剪枝

对搜索空间的一种特定编码可能会输出许多答案集，而这些答案集对应的公式不一定满足示例，甚至不满足基本的良构性约束。例如，公式 $\forall n_1$.锁_msg(n_2,l_1) 应被忽略，因为 n_2 不在前缀中。再比如，前缀. 锁_msg(n_1,l_1) $\vee$锁_msg(n_2,l_1) 与前缀.锁_msg(n_2,l_1) $\vee$锁_msg(n_1,l_1) 等价，但可能作为两个不同的答案集重复出现。因此，下一步是在搜索空间上编码第 2.2 节中的冗余处理技术，从而只输出良构公式。

我们通过答案集编程说明基于对称性的规范化技术如何利用等价性；冗余消除的其余编码可在我们的实现中找到。这些规范化通过在公式上构建偏序来完成。也就是说，我们可以确保：(1) 立方项中的文字数量（在析取范式中从左到右）非递减；(2) 如果立方项中的文字数量相同，则立方项i 中字母顺序最靠前的文字小于立方项j $>i$ 中字母顺序最靠前的文字；(3) 如果同一类型的两个变量$v_i < vj$，那么v_i 出现的最小谓词应小于或等于vj 出现的最小谓词。以下答案集程序通过约束公式中的顺序来编码这些剪枝规则。

```
lit_no_in(No, C) :- lit_in_C(P,A,Pos,C), lit_no(P,A,Pos,No). num_lit(C, N) :- 立方体(C), #count{P,A,Pos: lit_
in_C(P,A,Pos,C)} =N. :- num_lit(C1,N1), num_lit(C2,N2), C1 < C2, N1 > N2. min_lit(C, Min) :- 立方体
(C), #min{No:lit_no_in(No,C)} = Min. :- num_lit(C1,N), num_lit(C2,N), min_lit(C1,Min1), min_
lit(C2,Min2),C1<C2, Min1>Min2. min_lit_var(V,Min) :- used_var(V), #min{P:lit_in_C(P,A,_,_), var_
in(V,A)} = Min. :- min_lit_var(V1,Min1), min_lit_var(V2,Min2), V1 < V2, Min1 > Min2.
```

$$\forall x.P(x) \vee R(x) \quad (1)$$
$$\forall x.P(x) \vee S(x) \quad (2)$$
$$\forall x。\ P(x) \vee (R(x) \wedge S(x)) \quad (3)$$

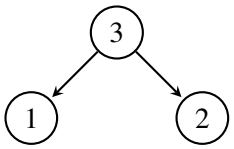

图3：左：公式示例。右：这些公式的蕴含图。

总而言之，本节表明，在一阶公式综合中，现有技术可以轻松地编码为答案集编程，并作为我们综合器的基础。答案集编程的表达能力使我们能够简洁地编码第 2.2 节中介绍的现有技术，并使它们高效协同工作；此外，我们还利用下一节将介绍的新剪枝技术进一步优化了这些技术。

4 通过正交切片实现动态搜索空间

借助上一节提出的基于ASP的公式枚举框架（该框架允许组合剪枝技术），我们提出了正交切片 来处理动态搜索空间。该方法（1）推广了最先进的增量剪枝技术蕴含图 （IG），（2）引入了一种新颖的互补剪枝方法，以解决 IG 的瓶颈。两者均通过将一阶搜索空间切分为更小的有序部分来实现，并且可以通过 ASP 的增量求解轻松实现。

4.1 蕴含图

在第 2.2 节提出的分布式系统不变量综合剪枝技术中，一阶公式的蕴含图是一种重要方法。 I G 是一种有向图，其中每个节点表示一个公式，从 A 到 B 的一条边表示 A 蕴含 B。剪枝通过动态地从搜索空间中移除已满足公式所蕴含的公式来进行，从而加快综合过程。也就是说，我们可以将其称为“按已满足公式剪枝”。基于 IG的工具DuoAI [27]能够为许多其他现有工具无法处理的复杂协议推断归纳不变式。

示例1（图3所示蕴含图的说明）公式图3）公式(3）同时蕴涵公式(1）和公式(2)。若公式(3）得到满足，则公式(1）和(2）也会自动得到满足，因此可以从搜索空间中剪枝。

4.2 从公式到搜索空间

正交切片的第一步是将公式之间的蕴涵关系抽象到搜索空间层面，然后按照搜索空间上的偏序综合公式，以实现增量剪枝。我们通过对候选项施加句法约束，并将可能的取值集作为参数，来表示一阶公式搜索空间。因此，搜索空间是这些集合的笛卡尔积，其中每个元素都描述一个子搜索空间。

定义 2（一阶公式的模板及其切片）一阶公式模板 T 定义为参数集的一个元组 $T = (P_1, \ldots, P_n)$，其中每个 P_i 表示一个参数的可能取值集合。由 T 定义的搜索空间 $\Omega(T)$ 是这些参数集的笛卡尔积。T 的一个有效切片（称为$\mathrm{SL}(T)$）定义为将 T 划分为切片模板 $\{T_1, \ldots, T_j\}$，

其中每个切片模板 $T_i = (P_{i1}, \ldots, P_{in})$ 对应于参数集的一个子集。于是，搜索空间 $\Omega(T)$ 被划分为子集 $\{\Omega(T_1). .., \Omega(T_j)\}$，其中：

$$\Omega(T) = \prod_{i=1}^{n} P_i. \qquad \bigcup_{i=1}^{j} \Omega(T_i) = \Omega(T), \quad and \quad \Omega_m \cap \Omega_n = \emptyset \quad for\ all\ m \neq n.$$

这确保整个搜索空间均被覆盖，切片之间互不重叠，并且每个切片 Ω_i对应一个切片模板 T_i。

示例 2（一个lockserv 在第 3.1 节中的模板） • 存在量词的数量 ne ，

• 元组 tv = （$nv_1, \ldots, nvi$ …）中每种类型所用变量的数量，

• 元组 tl = （$nl_1, \ldots, nli$ …）中每个立方体所含文字的数量。
以及模板的参数（在 第 3.2 节中的冗余剪枝之后）为：

• $P_{ne} = \{0, 1\}$

• $P_{tv} = \{(0, 1), (1, 0), (1, 1), (2, 0), (2, 1)\}$

• $P_{tl} =$ {（1）、（2）、（3）、（1，1）、（1，2）、（1，3）、（2，2）、（1，1，2）}

现在我们描述“按已满足公式剪枝”的切片过程，该过程由切片模板（及其参数）的偏序定义。

定义 3（模板参数的偏序） 给定模板 T =（P_1，……，P_i，……，P_n）中作为 P_i子集的两个参数 P_{ix} 和 P_{iy}，如果 P_{ix}小于或等于 P_{iy}（记作 $P_{ix} \preceq P_{iy}$；当 $P_{ix} \neq P_{iy}$时记作 $P_{ix} \prec P_{iy}$），相对于 SL（T），则称……

$$\forall T_1 = (P'_1, \ldots, P_{ix}, \ldots, P'_n),\ T_2 = (P'_1, \ldots, P_{iy}, \ldots, P'_n) \in \mathrm{SL}(T),$$
$$\forall \phi \in \Omega(T_1), \exists \phi' \in \Omega(T_2)\ such\ that\ \phi' \models \phi.$$

定义 4（切片模板的偏序） 给定从 $\mathrm{SL}(T)$中切片得到的两个切片模板 $T_i = (P_{i1}, \ldots, P_{in})$和 $T_j = (P_{j1}, \ldots, P_{jn})$，若$\forall k \in [1, n], P_{ik} \preceq P_{jk}$，则称 T_i小于或等于 T_j（记作 $T_i \preceq T_j$）；当且仅当对所有 $k \in [1, n]$，$P_{ik} = P_{jk}$时，等号成立。

根据上述两个定义，整个一阶公式搜索空间按照偏序进行切片（并排序）；我们将此过程称为 SplitTem。请注意，搜索空间的参数可能不像例 2中那样规整，但偏序的最坏情况恰好就是蕴含图：可能的公式候选项集合是唯一的参数，而偏序由一阶逻辑蕴含定义。这表明，始终可以通过句法约束对一阶搜索空间进行参数化。

示例 3（例 2 中模板的偏序） *lockserv* 的模板参数偏序定义如下：

• 对于 ne，它遵循整数序：$\{0\} \prec \{1\}$。

• 对于 tv，集合 {（nv_1，nv_2，……，nvi …）} ≺ {（nv_1，nv_2，……，$nvi + 1$，……）}

• 对于 tl，集合 $\{(nl_1、nl_2、\ldots、nl_i \cdots)\} \prec \{(nl_1、nl_2、\ldots、nl_i - 1、\ldots)\}$
和$\{(nl_1、nl_2、\ldots、nl_i \cdots)\} \prec \{(nl_1 + 1、nl_2、\ldots、nl_i、1)\}$

模板参数上的偏序定义与公式蕴涵关系一致：通过交换 $\forall$ 为 $\exists$，公式变得更加一般；通过将一个变量替换为一个新变量（连同公式分解，证明见 [28, §4]），公式变得更加一般；通过从立方体中删除一个文字或添加一个新立方体，公式变得更加一般。因此，通过获得搜索空间切片的偏序，即可自然地实现“按已满足公式剪枝”。

“按已满足公式剪枝”的 ASP 编码通过多轮求解实现 [10], ，这是在 ASP 中实现增量求解的标准方法。其编码概要如下：

```
#program inv(前缀,矩阵). :- output(Prefix, Matrix), pre_weaken(Prefix, 前缀), mat_
weaken(Matrix, 矩阵).
```

这表示，不应生成由 inv 所蕴涵的公式（即一个满足的公式）。蕴涵检查通过变量替换以及前缀和矩阵弱化实现（详见我们的实现）。

4.3 析取范式模子句切片

蕴含图（或一般意义上的“按已满足公式剪枝”）的问题在于其可扩展性：搜索从图的根节点开始，遍历至叶节点（图3中从上到下），但根节点的数量相对于搜索空间的大小仍呈指数级增长。为说明这一问题，我们以公式前缀 $(\text{lit}_{11} \wedge \text{lit}_{12} \wedge \text{lit}_{13}) \vee (\text{lit}_{21} \wedge \text{lit}_{22})$ 为例：由于蕴涵关系，它可以作为用于剪枝公式前缀 $\text{lit}_{1i} \vee \text{lit}_{2j}$ 的根节点。然而，检查所有根公式的复杂度为 $O(n^5)$（n 为文字数），而这番工作只能剪去占用 $O(n^2)$ 空间的公式。

直观而言，如果使用较大搜索空间的结果来剪枝较小搜索空间时，“按已满足公式剪枝”的代价很高，那么它的对偶版本——“通过未满足公式剪枝”——应该可以解决这一问题。然而，这个想法无法直接应用，原因有二：（1）这两种剪枝策略的方向不同（从一般到具体与从具体到一般），这意味着算法需要同时处理两种方向；（2）对于复杂问题，大多数公式都不满足（无法覆盖所有示例），这意味着反复缩减搜索空间会带来很大的性能开销。

我们对第一个问题的解决方案只是再进行一次切片，这与模板切片正交：首先从整个搜索空间中切片出子句（文字的析取式）并进行综合（称为 SPLITDNF），然后根据未满足子句剪枝析取范式综合的搜索。为进一步解决第二个问题，我们通过构建析取范式搜索空间，使其基于已满足子句，从而避免“按未满足子句剪枝”的高昂成本。下文给出的析取范式搜索空间构造定义描述了这种切片的剪枝。

定义 5（模子句可能满足的析取范式）给定一组满足（相对于输入示例）的子句 Φ_c，形如 $\mathit{prefix}.\ (\mathit{lit}_{11} \wedge \ldots \wedge \mathit{lit}^{1k_1}) \vee \ldots \vee (\mathit{lit}_{m1} \wedge \ldots \wedge \mathit{lit}^{mk_m})$ 的析取范式公式只有在以下情况下才可能满足

$$\forall l_i \in \bigcup_{i=1}^{m} \{\mathit{lit}_{i1}, \ldots, \mathit{lit}_{ik_i}\},\ \mathit{prefix}.\ l_1 \wedge \ldots \wedge l_m$$

相对于 Φ_c 满足。构造析取范式的函数记为 BUILDFROMCLAUSES。

通俗地说，只有当公式所蕴含的所有子句都满足所有输入的一阶结构时，析取范式中的公式才会被加入搜索空间。我们为何能够解决“按已满足公式剪枝”这一瓶颈，原因显而易见：子句的搜索空间关于立方体数量呈指数级增长，但在大多数情况下，它远小于析取范式的搜索空间（后者关于文字数量呈指数级增长）。此外，由于这种搜索空间构造也可以自然地用答案集编程编码，因此第 2.2 节 中针对公式的所有现有技术都可以直接应用于这个基于子句构建的搜索空间，以进一步对其进行细化。

综合	ASP	连接
参数化搜索空间	选择构造	集合笛卡尔积
切片	外部语句	集合划分
静态剪枝	完整性约束	集合构造式
动态剪枝	多轮求解	集合并、差集

表 1：综合与答案集编程之间的连接。

4.4 综合算法

给定图 1 中针对输入搜索空间的公式合成算法，以及两种切分搜索空间的方法，公式综合过程如算法2所示：首先通过切片模板综合子句（第 4 至 7 行），然后通过模子句的析取范式的切片模板综合其他公式（第 9 至 12 行），最后对满足的公式进行规范化并输出（第 13 和 14 行）。输入的剪枝规则可以自定义，但我们预先定义了第 2.2 节中已有的冗余规则和增量规则。该算法关于相对于正交切片（即，不会过度剪枝）的可靠性由以下事实保证：如果公式A 被剪枝，

算法 2 FORCE 的核心算法

1: **函数** FORCE(Ω_0, σ, 剪枝 $=$ *pre_def*)
2: Φ_c, $\Phi_f \leftarrow /0$
3: $\Omega_c \leftarrow$SPLITDNF(Ω_0)
4: **对于** $\Omega_{ci} \in$SPLITTEM(Ω_c) **执行**
5: $\Phi_i \leftarrow$ FOSYN(Ω_{ci}, σ, 剪枝)
6: $\Phi_c \leftarrow \Phi_c \cup \Phi_i$
7: 剪枝 $\leftarrow$*prunings.update*(Φ_i)
8: $\Omega_f \leftarrow$ BUILDFROMCLAUSES(Φ_c, Ω_0)
9: **对于** $\Omega_{fi} \in$ SPLITTEM(Ω_f) **执行**
10: $\Phi_i \leftarrow$FOSYN(Ωfi σ, 剪枝)
11: $\Phi_f \leftarrow \Phi_f \cup \Phi_i$
12: 剪枝 $\leftarrow$*prunings.update*(Φ_i)
13: $\Phi_c \leftarrow$ FILTERIMPLIED(Φ_c, Φ_f)
14: **返回** $\Phi_c \cup \Phi_f$

A 要么未满足，即因模子句的析取范式中的未满足子句而被剪枝；要么虽然已满足，但比在切片模板中剪枝A 的已满足公式更一般。

回顾整个综合过程，我们讨论过的静态和动态搜索空间本质上都是通过不同的集合运算实现的，这也是我们使用答案集编程的算法层面原因。在表 1中，我们将综合问题与答案集编程之间的联系统一表述为集合运算，希望这有助于综合或答案集编程领域的读者理解“为什么以及哪些综合任务适合使用答案集编程”。

5 实验评估

5.1 实现与设置

我们对 FORCE 的实现结合了答案集编程（用于枚举的代码为 250 行，用于剪枝的代码为 80 行，能够以极其精简的方式编码现有的各种剪枝技术及新技术），以及 C++来调用 Clingo [9] 的 API（综合器调用函数为 400 行，与其他系统集成的代码为 200 行）。基准测试在一台配备 8 核 M1 Pro 处理器和 16GB 随机存取存储器的 MacBook Pro 上运行。当前版本的 FORCE 可从以下地址获取：https://zenodo.org/records/15654427。

我们在FORCE 上运行了一组来自DuoAI [27], ——目前最高效的分布式系统不变量推理工具——相关研究中的基准测试。简而言之，DuoAI的算法采用基于枚举的归纳式合成，以协议的采样轨迹作为示例，随后使用经过优化的 Houdini [7]。由于公式枚举组件的运行时间不可忽略（详见下文），减少枚举部分的开销

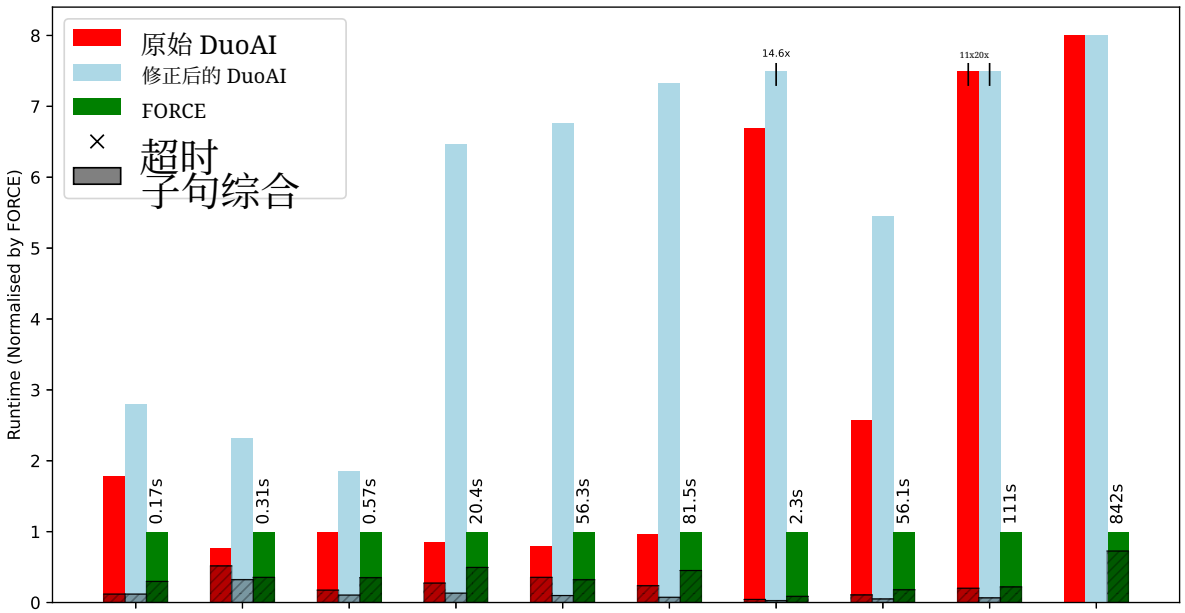

图 4：不变量综合。结果已针对 FORCE 运行时间进行了归一化。

可以显著提升整体性能。

请注意，DuoAI中至少有两项不健全的优化：第一项是过度剪枝，由 [8]发现并修正：DuoAI 会识别出可分解为更小公式的公式，然后不再测试原公式以缩小搜索空间；然而，其中一部分公式不应被分解，这会因遗漏可能满足的公式而导致不健全性。第二项不健全的优化是对公式使用受限量词；具体而言，DuoAI 的默认设置只允许在模板的最后三个变量上使用任意量词（例如，对于 $Q_1X_1.Q_2X_2.Q_3X_3.Q_4X_4,Q_1$ 只能是 $\forall$），这通常会导致搜索不完备，但在实践中适用于其基准测试。我们将第一个问题视为漏洞，并将第二项观察视为领域特定知识，它为搜索空间增加了一项约束。

5.2 结果与分析

我们的性能统计数据如图 4所示。我们从DuoAI 套件中选取了六种复杂的分布式协议，用于评估归纳不变式综合效果。FORCE 使用了与DuoAI相同的输入：作为输入示例的轨迹，以及包含归纳不变式的搜索空间配置。对于每种协议，我们使用两种配置进行基准测试：一种采用 DuoAI 中的量词限制，另一种则不采用。启用无限制量词设置时，图中会以 $*$ 标示。之所以只展示 10 个数据点而非 12 个，是因为对于前两种规模较小的协议，量词限制的差异不会影响较小的搜索空间。我们尽最大努力修复了DuoAI 中过度剪枝的错误（将修复后的结果称为“修复版 DuoAI”），以确保比较公平。除此之外，我们还提供了“原始版”DuoAI 的结果作为参照。

结果 结果表明，FORCE 在所有基准测试中均显著优于修正后的 DuoAI，并且 so

在大多数情况下优于原始（不健全的）DuoAI。不加量词限制时，性能差距更大。作为参考，原始 DuoAI 为 Paxos、FlexiblePaxos 和 MultiPaxos 综合不变式的运行时间分别为 60.4 秒、78.7 秒和 1,549 秒；这意味着 DuoAI 中的枚举是复杂协议的主要瓶颈，因此这一改进能有效缩短总体运行时间。

改进主要源于模子句的析取范式所实现的有效剪枝（第 4.3 节）：如条形图中的子句综合部分所示，FORCE 在子句综合上耗时（略）长于DuoAI，但通过缩小搜索空间，使剩余综合所需的时间大幅缩短。对于不进行量词限制的情况，巨大的差异也可由同样的原因解释：以 Paxos 为例，在FORCE中禁用该限制后，已满足子句的数量从 134 增加到 141，这意味着析取范式的搜索空间增加并不大（时间差为 36 秒）；但对于DuoAI，枚举会探索整个新的搜索空间（因此总时间相差 76 秒）。需要注意的是，如第 4.2 节所述，将公式抽象到搜索空间中，使得FORCE中的公式综合可以实现并行，这也提高了效率；但这具体取决于（1）并行性的实现方式，以及（2）Clingo中的多线程求解，而这些并非此处讨论的重点。在我们的系统中，禁用并行性所导致的总体额外时间从 50% 到 100% 不等，这表明通过改进并行性实现仍有提升空间。

另一个有趣的比较关于DuoAI 中的不健全性，如红色和蓝色柱状条之间的对比所示：修正后的版本需要更长时间来探索（如今更大的）搜索空间，但子句综合时间更短，因为额外发现的满足公式会剪除其蕴含的子句（这也说明原版DuoAI存在不健全性）。与我们的正交切片类似，他们利用“较大切片”的结果来剪除“较小切片”，因此效率低下，这也在意料之中（见第 4.3 节）。从宏观角度看，这一比较体现了综合任务中完备性与效率之间常见的权衡，但FORCE 通过花费少量额外时间（绿色柱状条）综合子句，相较于原版DuoAI 实现了完整搜索，效率甚至更高。

5.3 FORCE 的可组合性

除了正交切片带来的性能优势外，FORCE 的“求解器辅助”特性也很有前景：它通过为一阶综合提供通用接口，作为一个通用框架来组合不同的方法。为说明这一点，我们分别为DuoAI 和Flyvy [8] 搭建了桥接模块（每个约 200 行代码），并使用子句输出（即，图 2 第 8 行之前的部分）来优化Flyvy。

Flyvy 的高层任务是Flyvy 输出最强的归纳不变式（其中包含一组一阶逻辑公式），输入为分布式系统协议和有界一阶语言。不涉及Flyvy算法的细节，显然，如果Flyvy 正在获取公式，那么这些公式可以由模子句的析取范式界定。我们使用DuoAI 获取的轨迹以及有界语言来综合已满足子句，并将它们作为答案集程序输出，再由FORCE调用该答案集程序，筛除不必要的公式，从而精简Flyvy 中的公式。结果表明，对于Flyvy能够综合的两个最复杂协议——Paxos 和 FlexiblePaxos，输出不变式的规模平均分别减少了约 13%（165/1260）和 7%（61/816）（详见附录 B），且仍然是最强的，因为在Flyvy的中间步骤中移除了不必要的公式。我们相信，结合现有方法的不同方案与FORCE的强大能力进行探索，是一个很有前景的方向。

6 相关工作与讨论

归纳逻辑编程。 我们的工作与归纳逻辑编程领域密切相关 [20, 3]，该领域专注于归纳学习逻辑程序。具体而言，当前最先进的

归纳逻辑编程系统Popper [4] 与 FORCE采用了相似的工作流程：在“生成—测试—剪枝”循环中，两个系统都使用答案集编程来生成候选项并剪枝搜索。 FORCE 和 Popper之间主要有两点差异：(1) 在归纳逻辑编程中，由于同时给出了正例和反例，因此在所有正确解中，优先选择的候选项是最一般的程序；而在FORCE中，输出是满足所有正例的最具体公式的合取，因为在没有反例的情况下，“逻辑真”始终是一个有效的一般公式。(2) 与霍恩子句相比，一阶公式的性质揭示了更多可由答案集编程编码、用于剪枝的领域知识。

不变式推断的其他方法。 在 第 2.3 节所示的技术中，三个工具（归入 **其他技术** 类别）的技术尚未被编码到 FORCE 中。其中两种工具在改进推断过程的方式上有相似之处： FOL-IC3 [13]是一种基于一阶逻辑可分性 求解器构建的推断框架； IC3PO [11] 是一种不变式推断工具，它基于已满足的一阶结构的对称性 与满足一阶逻辑中公式所需的量化 之间的关系。两者都基于 PDR/IC3 [2], ，并根据一小组示例（通常少于 10 个）对一阶结构进行泛化；相比之下，DuoAI、SWISS和 FORCE则是在数千个示例中进行泛化。从技术角度来看，由于两者均基于布尔可满足性，因此可以将它们的技术编码到 FORCE中。然而，这些技术的可扩展性有限，因此通常不适用于大型示例集；我们将在后文讨论这一点。

Scimitar [25], 不同于大多数相关工作中推断“全局”归纳不变式的方法，它是一种为分布式系统协议中的每个迁移状态推断“局部”不变式的工具。它构建了一个归纳证明图 来抽象协议，并为图中的每个节点局部综合不变式，其中局部合成问题是我们一阶综合的一个实例。

关于归纳泛化的讨论。 最小一般泛化 [24] 被认为是自底向上的归纳逻辑编程领域中的基础性成果，其中给定的程序（底部）被泛化为一般程序（顶部）。相比之下，FORCE 是一种通过“生成并测试”实现的自顶向下方法。值得注意的是，在分布式系统不变式推断领域，SWISS 工具的作者 [12] 描述了他们使用约束求解进行自底向上泛化的“失败尝试”（但未作详细解释）。这与我们的观察一致：与自顶向下方法相比，自底向上的泛化（如 FOL-IC3 和 IC3PO）无法扩展到包含大量示例的情形。我们对此给出自己的解释：给定整个搜索空间，任何示例都可以被视为用于剪枝搜索的约束；虽然约束的成本可视为与示例数量呈线性关系，但随着约束数量增加，剪枝收益会降低。这为进一步结合自顶向下和自底向上方法提供了一种可能性，即使用示例子集来利用自底向上的泛化。

在上述讨论的领域之外， [15] 是一项来自自动机理论的重要工作，研究有限变元逻辑中公式的可学习性（关于泛化）。该工作并不关注实践中的具体算法，但展现了将 FORCE 扩展到比一阶逻辑更一般的逻辑的潜力。

7 结论

在本研究中，我们提出了一个统一的基于 ASP 的框架 FORCE，用于从示例中综合一阶逻辑公式。为此，我们将 ASP 用作归纳式公式合成的实现框架，通过约束求解对搜索进行编码，并利用基于规则的知识剪枝搜索空间。借助基于 ASP 的编码，我们提出了正交切片——一种能够显著加速公式综合的新技术。最后，我们证明，以声明式方式在 ASP 中刻画不同公式综合方法的核心，有助于实现更高效且更易组合的解决方案。

参考文献

[1] Haniel Barbosa、Clark W. Barrett、Martin Brain、Gereon Kremer、Hanna Lachnitt、Makai Mann、Abdalrhman Mohamed、Mudathir Mohamed、Aina Niemetz、Andres Nötzli、Alex Ozdemir、Mathias Preiner、Andrew Reynolds、Ying Sheng、Cesare Tinelli 和 Yoni Zohar（2022年）：cvc5：一种用途广泛、具备工业级能力的 SMT 求解器。载于：TACAS，LNCS 13243，Springer，第 415–442 页，doi:10.1007/978-3-030-99524-9_24.[2] Aaron R. Bradley（2011年）：无需展开的基于 SAT 的模型检查。载于：VMCAI，LNCS 6538，Springer，第 70–87 页，doi:10.1007/978-3-642-18275-4_7.[3] Andrew Cropper 和 Sebastijan Dumancic（2022年）：归纳逻辑编程三十年：全新导论。《人工智能研究杂志》74，第 765–850 页，doi:10.1613/JAIR.1.13507.[4] Andrew Cropper 和 Rolf Morel（2021年）：通过从失败中学习来学习程序。《机器学习》110(4)，第 801–856 页，doi:10.1007/S10994-020-05934-Z.[5] Niklas Eén、Alan Mishchenko 和 Robert K. Brayton（2011年）：属性导向可达性的高效实现。载于：FMCAD，FMCAD Inc.，第 125–134 页。[6] Moshe Emmer、Zurab Khasidashvili、Konstantin Korovin 和 Andrei Voronkov（2010年）：将工业硬件验证问题编码为有效命题逻辑。载于：FMCAD，IEEE，第 137–144 页。[7] Cormac Flanagan 和 K. Rustan M. Leino（2001年）：Houdini：ESC/Java 的注释助手。载于：FME，LNCS 2021，Springer，第 500–517 页，doi:10.1007/3-540-45251-6_29.[8] Eden Frenkel、Tej Chajed、Oded Padon 和 Sharon Shoham（2024年）：带量词的一阶公式抽象域的高效实现。Co RR abs/2405.10308，doi:10.48550/ARXIV.2405.10308。arXiv:2405.10308。[9] Martin Gebser、Roland Kaminski、Benjamin Kaufmann 和 Torsten Schaub（2012年）：实践中的答案集求解。《人工智能与机器学习综合讲座》，Morgan & Claypool Publishers，doi:10.2200/S00457ED1V01Y201211AIM019。[10] Martin Gebser、Roland Kaminski、Benjamin Kaufmann 和 Torsten Schaub（2019年）：使用 clingo 进行多轮 ASP 求解。《理论与实践逻辑编程》19(1)，第 27–82 页，doi:10.1017/S1471068418000054。[11] Aman Goel 和 Karem A. Sakallah（2021年）：关于对称性与量化：验证分布式协议的新方法。载于：NASA 形式化方法，LNCS 12673，Springer，doi:10.1007/978-3-030-76384- 8 9。_[12] Travis Hance、Marijn Heule、Ruben Martins 和 Bryan Parno（2021年）：寻找分布式系统的不变式：小世界终究（足够）小。载于：NSDI，USENIX 协会，第 115–131 页。可访问：https://www.usenix.org/conference/nsdi21/presentation/hance。[13] Jason R. Koenig、Oded Padon、Neil Immerman 和 Alex Aiken（2020年）：一阶量词分隔符。载于：PLDI，ACM，第 703–717 页，doi:10.1145/3385412.3386018。[14] Jason R. Koenig、Oded Padon、Sharon Shoham 和 Alex Aiken（2022年）：通过量词交替推断不变式：驾驭搜索空间爆炸。载于：TACAS，LNCS 13243，Springer，第 338–356 页，doi:10.1007/978-3-030-99524-9 18。_[15] Paul Krogmeier 和 P. Madhusudan（2022年）：学习有限变量逻辑中的公式。《ACM 程序语言学报》6(POPL)，第 1–28 页，doi:10.1145/3498671。[16] Leslie Lamport（1998年）：兼职议会。《ACM 计算机系统汇刊》16(2)，第 133–169 页，doi:10.1145/279227.279229。[17] Vladimir Lifschitz（2002年）：答案集编程与计划生成。《人工智能》138(1-2)，第 39–54 页，doi:10.1016/S0004-3702(02)00186-8。[18] Haojun Ma、Aman Goel、Jean-Baptiste Jeannin、Manos Kapritsos、Baris Kasikci 和 Karem A. Sakallah（2019年）：I4：用于验证分布式协议的归纳不变式增量推理。载于：SOSP，ACM，第 370–384 页，doi:10.1145/3341301.3359651。

[19] Leonardo Mendonça de Moura 与 Nikolaj S. Bjørner（2008）：Z3：一种高效的 SMT 求解器。载于：TACAS，LNCS 4963，Springer，第 337–340 页，doi:10.1007/978-3-540-78800-3_24.[20] Stephen H. Muggleton 与 Luc De Raedt（1994）：归纳逻辑编程：理论与方法。《逻辑程序》19/20，第 629–679 页，doi:10.1016/0743-1066(94)90035-3.[21] Oded Padon、Kenneth L. McMillan、Aurojit Panda、Mooly Sagiv 与 Sharon Shoham（2016）：Ivy：通过交互式泛化进行安全性验证。载于：PLDI，ACM，第 614–630 页，doi:10.1145/2908080.2908118.[22] Kanghee Park、Loris D'Antoni 与 Thomas W. Reps（2023）：规格综合。《ACM 程序语言学报》7(OOPSLA2)，第 1787–1816 页，doi:10.1145/3622861.[23] Ruzica Piskac、Leonardo Mendonça de Moura 与 Nikolaj S. Bjørner（2010）：使用 DPLL 和替换集有效判定命题逻辑。《自动推理期刊》44(4)，第 401–424 页，doi:10.1007/S10817- 009-9161-6.[24] Gordon D Plotkin（1970）：关于归纳泛化的札记。机器智能 5(1)，第 153–163 页。[25]William Schultz、Edward Ashton、Heidi Howard 与 Stavros Tripakis（2024）：通过归纳证明切片实现可扩展、可解释的分布式协议验证。CoRR abs/2404.18048，doi:10.48550/ARXIV.2404.18048。arXiv:2404.18048。[26] Ziyi Yang 与 Ilya Sergey（2025）：归纳堆谓词的归纳综合。《ACM 程序语言学报》9(OOPSLA1)，第 169–195 页，doi:10.1145/3720420。[27] Jianan Yao、Runzhou Tao、Ronghui Gu 与 Jason Nieh（2022）：DuoAI：用于验证分布式协议的快速自动化归纳不变式推断。载于：OSDI，USENIX 协会，第 485–501 页。可访问：https://www.usenix.org/conference/osdi22/presentation/yao。[28] Jianan Yao、Runzhou Tao、Ronghui Gu、Jason Nieh、Suman Jana 与 Gabriel Ryan（2021）：DistAI：面向分布式协议的数据驱动自动化不变式学习。载于：OSDI，USENIX 协会，第 405–421 页。可访问：https://www.usenix.org/conference/osdi21/presentation/yao。

一个定义 1中的第 2.1 节第 2.1 节

本节给出一阶综合问题的一个示例。

示例 4（一个小例子）考虑一个一阶逻辑签名 $\Sigma = \langle C,R,F,S\rangle$，其中 C 和 F 为空，$R = \{p, q, r\}$，$S = \{X\}$。该综合问题由以下输入定义：

- 一阶公式集合被限制为以下形式：$\forall X:$ 文字$_1$ 或 $\forall X:$ 文字$_1 \vee$文字$_2$，即带有一个全称量词的一个或两个文字的析取。不进行剪枝时，Ω_0 总共有 $(2*3) + (2*3)^2 = 42$个公式。

- 一阶结构集合为 $\sigma = \{M_1, M_2\}$，其中 M_1 和 M_2 是共享论域 $\Sigma x_0,x_1, x_2\} \in X\{$的两个模型，其对 p、q 和 r 的解释为：
  1. M_1: $[p^{M_1} = \{x_0, x_1\}, q^{M_1} = \{x_1, x_2\}, r^{M_1} = \emptyset]$
  2. M_2: $[p^{M_2} = \{x_0, x_1\}, q^{M_2} = \{x_2\}, r^{M_2} = \{x_1\}]$

基于定义 *1*的合成问题的输出是一组公式

$$\Phi = \{\forall X.p(X) \vee q(X), \forall X.p(X) \vee \neg r(X)\}.$$

作为切片模板的一个简单示例，我们的算法会先在 $\forall X: lit_1$中找到满足的公式（但会失败），然后再检查 $\forall X: lit_1 \vee lit_2$ 中的满足公式（成功找到这两个公式）。FORCE的输入、有界一阶逻辑搜索空间以及分布式系统协议的真实轨迹要复杂得多。感兴趣的读者可以参阅 `configs/` 和 `traces/` 文件夹，访问https://github.com/verse-lab/FORCE 查看真实实例。

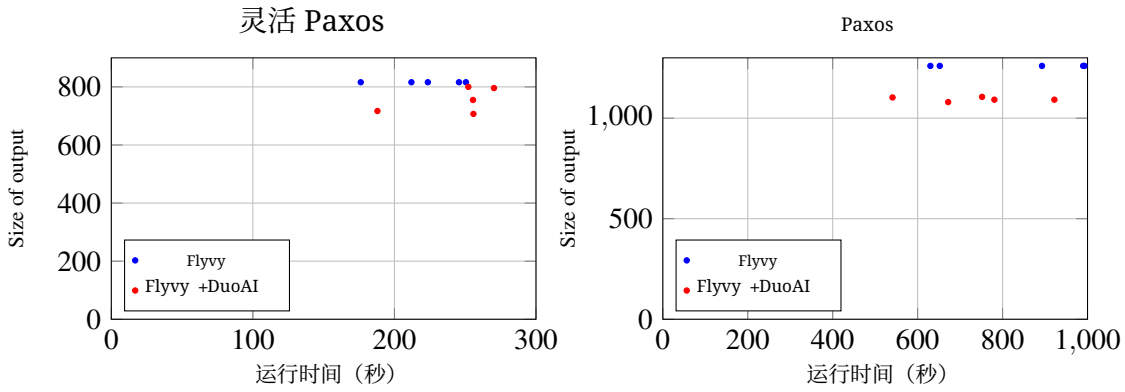

图 5：通过 DuoAI 的输出优化 Flyvy。

B 补充统计数据第 5.3 节

如第 5.3 节所述，Flyvy的基准测试中，只有两个具有量词交替的非平凡协议，我们可以为其生成可比较的结果。对于这两个协议，我们分别运行原版Flyvy 和优化后的 Flyvy 各 5 次，并在图 5中展示（前文已讨论的）输出大小和运行时间。从性能上看，我们未能显著缩短Flyvy的运行时间，这并不意外，因为Flyvy 的瓶颈并非综合；相反，Flyvy 中的 SMT 求解占据了大部分运行时间，并导致运行时间不稳定。