BALI：结合大语言模型的分支感知循环不变式推断

Mingxiu Wang[1]**,Jiawei Wang**[2]**,Xiao Cheng**[1]1麦考瑞大学2新南威尔士大学
mingxiu.wang@students.mq.edu.au, jiawei.wang6@unsw.edu.au,
xiao.cheng@mq.edu.au

arXiv:2601.00882v1 [cs.PL] 31 Dec 2025

摘要

循环不变式对于推理迭代算法的正确性至关重要。然而，推导合适的不变式仍是一项艰巨且通常需要手动完成的任务，尤其是对于复杂程序。本文介绍 BALI，这是一种分支感知框架，它集成了大语言模型（LLM），以增强循环不变式的推断与验证。我们的方法将自动化推理与分支感知静态程序分析相结合，从而提升精确性和可扩展性。具体而言，与以往仅依赖 LLM 的猜测与检查方法不同，BALI 首先使用可满足性模理论（SMT）验证分支序列级（路径级）子句，然后将它们组合成程序级不变式。我们概述其关键组件，展示初步结果，并探讨迈向全自动不变式发现的未来方向。

关键词： 循环不变式、SMT 求解器、大语言模型、自动推理

引言与动机

循环结构在编程中无处不在，是迭代推理的基础。在传统的C语言程序中，为 for 或 while 循环确定合适的循环不变量仍然十分困难，通常依赖启发式方法或人工洞察。人工智能智能体中也存在类似的迭代行为：它们反复观察周围环境，根据当前状态做出决策，并不断执行操作或遍历条件，直到实现特定目标（Evertz 2003；Wu等人 2022年；Mosqueira-Rey等人 2023年）。因此，确保这些程序级循环的正确性，对于此类系统的可靠运行至关重要。

一种成熟的循环正确性推理方法是通过循环不变量——即在每次循环迭代之前和之后都保持为真的性质（Furia、Meyer 和 Velder 2014；D'silva、Kroening 和 Weissenbacher 2008）。多年来，人们提出了多种自动生成循环不变量的方法。

传统方法——包括约束求解（Col´on、Sankaranarayanan和 Sipma，2003年；Hojjat和R¨ummer，2018年）、反例引导的细化（Clarke等人，2000年；Bogomolov等人，2017年）以及Craig插值（Jhala和McMillan，2006年；McMillan，2010年）——将不变式编码为逻辑约束，并通过符号推理对其进行迭代细化。尽管这些技术在小型程序上有效，但它们需要领域专业知识来设计模板或逻辑规则，并且通常难以扩展到现实世界的软件，因为每种新的数据结构都需要专门的推断模式。

基于机器学习的方法利用大型代码语料库学习句法和语义模式，并自动生成不变式（Garg 等人，2014、2016；Nguyen 等人，2017；Si 等人，2020年）。与符号方法相比，这些数据驱动技术通常具有更好的可扩展性，因为它们通过从数据中学习不变式模式，减少了对人工设计模板和规则的依赖。然而，这些方法往往会遇到过拟合、可解释性有限，以及过度依赖训练数据的质量和多样性等问题，导致其在未见过的程序或数据结构上的性能下降。

近来，基于大语言模型（LLM）的方法直接根据程序代码和自然语言提示推断循环不变式（Wu等人，2024年；Cao 等人，2025；Liu 等人，2026）。这类方法通常采用猜测并检查 范式：大语言模型生成候选不变式，随后使用可满足性模理论求解器或形式推理工具进行验证，并将反例反馈回来以供改进。这类方法通常更加灵活、适应性更强，能够利用大语言模型中蕴含的丰富知识和上下文理解能力。

然而，尽管当前的大语言模型能力出众，在为控制流复杂的程序（例如包含条件分支的程序）生成正确且精确的不变量方面仍面临困难。这一局限部分源于 Transformer 架构：它虽然擅长捕捉全局上下文，却缺乏用于细粒度结构推理和处理代码长程依赖的显式机制。

为应对这一挑战，我们提出了 BALI，这是一种新颖的方法，将能够捕捉分支相关性的程序分析精度与大语言模型的生成能力和上下文理解优势相结合。我们的方法旨在提高复杂循环不变式生成的精度和鲁棒性。这一方向有望推动程序验证和综合任务中的自动化推理。

版权所有 © 2026，美国人工智能促进协会（www.aaai.org）。保留所有权利。

背景与相关工作

循环不变式

循环不变式是一种逻辑断言 I，用于描述在循环执行过程中始终成立的性质。在霍尔三元组

$$\frac{P \Rightarrow I \quad \{I \wedge B\}\, S\, \{I\} \quad (I \wedge \neg B) \Rightarrow Q}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \{Q\}}$$

各组成部分定义如下：

- P：**前置条件**，描述循环开始前的程序初始状态。
- B：**循环守卫**，用于控制循环执行的布尔条件。
- S：**循环体**，当 B成立时执行的语句。
- I：**循环不变量**，它必须在每次迭代前后都成立。
- Q：**后置条件**，用于描述循环终止后的状态。

不变量 I必须满足三个基本条件：

1. **初始化：** 前置条件建立不变量（$P \Rightarrow I$）。 2. **保持性：** 如果 I 成立且守卫条件 B 为真，执行 S 会保持 I（$\{I \wedge B\}S\{I\}$）。 3. **终止性：** 如果 I 成立且守卫条件 B 为假，则可推出后置条件（（$I \wedge \neg B$） $\Rightarrow Q$）。

因此，证明循环正确性归结为找到一个循环不变量 I，它由 P建立，由 S保持，并且在 B 为假时足以推出 Q。

最先进的研究成果

近期的不变式生成研究开始探索利用大语言模型（LLM）自动推断逻辑不变式。该方向最具影响力的工作之一是 CLAUSE2INV（Cao 等人，2025），它以更系统的猜测–检查范式取代了传统的生成–组合–检查框架。

从总体上看，CLAUSE2INV 旨在生成满足循环正确性的三个基本性质——初始化、保持性和终止性——的不变式 I：利用 LLM 生成原子子句，由组合器将其组合成候选不变式，再通过 SMT 求解器对其进行验证。当其中任何性质未得到满足时，系统会作出相应响应：

- **如果初始化失败：** 生成的不变式 I并不由前置条件 P ($P \not\Rightarrow I$) 蕴含。在这种情况下，系统会提示 LLM 重新生成与程序初始状态更为一致的子句。
- **如果保持性失败：** 在 $I \wedge B$ 条件下执行循环体 S 无法维持 I（$\{I \wedge B\}S\{\neg I\}$）。此时，SMT 求解器会识别出反例——即 I 被破坏的特定程序状态。这些反例会被存储在反例集（ceSet）中，并反馈给大语言模型，以完善其下一次生成结果。
- **如果终止性失败：** 一旦循环守卫 B 变为 false（（$I \wedge \neg B$） $\not\Rightarrow Q$），不变式 I 就过弱，无法推出后置条件 Q。在这种情况下，系统会指示大语言模型生成更强的不变式，以确保所需的后置状态。

为支持这一过程，CLAUSE2INV 首先生成原子子句——例如 ($a > 1$) 这样的简单表达式，不包含 && 和 || 等逻辑连接词。这些子句会存储在 exprStore 中，专用的组合器组件则以系统化方式将其组合成更复杂的候选不变式。每个候选不变式都会使用 SMT 求解器进行形式化验证。经过验证且足够强的不变式将被接受；否则，将触发上述反馈循环。这一迭代过程会持续进行，直到找到有效不变式或计算超时。

在本工作中，我们采用 CLAUSE2INV（Cao 等人，2025）作为基线，并通过融入程序控制流图中的结构信息对其进行扩展。通过在生成阶段整合控制流关系，我们的方法引导大语言模型生成的不变式不仅在语法上正确，也与程序的执行结构在语义上相符。

方法

本节首先概述 BALI 框架，然后详细介绍该框架的实现方式。

概述

图 1 展示了 BALI 框架的概况，该框架包含以下三个主要阶段：

(a) 预处理。我们的方法首先使用 TREE-SITTER（Brunsfeld 等人，2025）将源代码解析为抽象语法树（AST），然后将其转换为控制流图（CFG），以捕捉程序中的结构关系和执行关系。随后遍历 CFG，提取所有执行路径，这些路径根据程序的层级结构和顺序结构编码分支序列（参见“路径提取算法”一节）。

(b) 路径摘要。 大语言模型会独立分析每条路径，以生成子句级不变式。对应于内部或较早控制结构的路径（例如最内层循环或最先遇到的分支）会被赋予更高优先级，以确保优先生成并验证这些路径的不变式。已验证的不变式会记录为摘要，供后续路径使用，使大语言模型能够在推理过程中利用上下文知识。由于每条路径仅对应一个较小的程序片段，这种基于路径的摘要方法提高了推理准确率和可扩展性。

（c）循环不变式检查。处理完所有路径后，将收集到的不变式和完整源代码提供给大语言模型，以推断更精确的程序级不变式。随后，将这些候选不变式合并，并通过 SMT 求解器进行验证。如果验证失败，则将反例反馈给大语言模型以指导修正——通过强化较弱的不变式或修订错误的不变式。

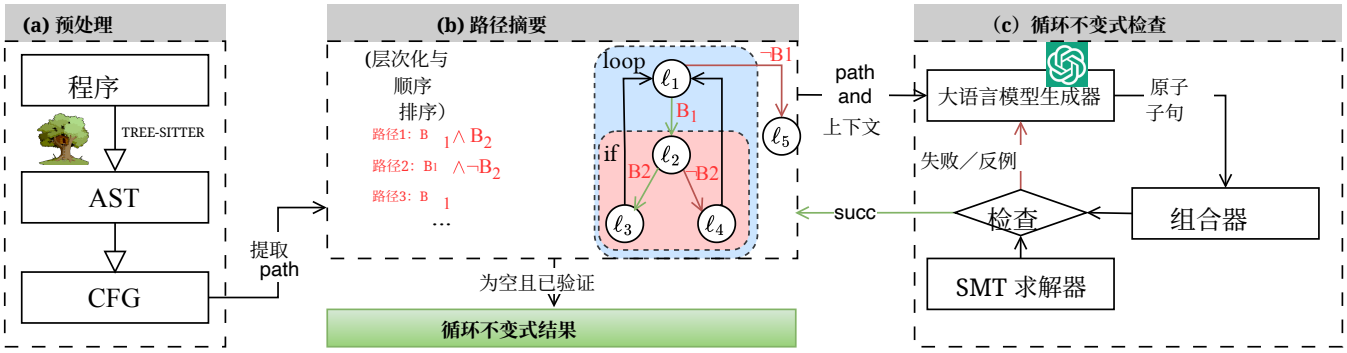

图 1：BALI 概览。

算法 1：在控制流图（CFG）中查找所有路径 要求：包含入口节点$CFG.entry$的控制流图 CFG 确保：所有执行路径的列表 $Paths$1： $Paths \leftarrow \varnothing$2： $node \leftarrow CFG.entry$3：当 $node \neq$ 不为 null 时执行 4：如果 $node.out\ degree > 1$则 5：如果 $node$ 是循环头节点，则 6：$loopCFG \leftarrow$ 提取循环控制流图($node$) 7：$loopPaths \leftarrow$查找所有路径($loopCFG$) 8：$Paths \leftarrow Paths \cup loopPaths$9：否则 10：$trueCFG \leftarrow$ 获取真分支控制流图($node$) 11：$falseCFG \leftarrow$获取假分支控制流图($node$) 12：$truePaths \leftarrow$查找所有路径($trueCFG$) 13：$falsePaths \leftarrow$查找所有路径($falseCFG$) 14：$branchPaths \leftarrow truePaths \cup falsePaths$15：$Paths \leftarrow Paths \cup branchPaths$16：结束如果 17：否则如果 $node.out\ degree = 1$ 则 18：将其添加到上下文($node$) 19：结束如果 20：$node \leftarrow node.successor$21：结束循环 22：返回 $Paths$

这一迭代式生成–验证–改进过程会持续进行，直到获得有效的不变式或计算超时。

路径提取算法

为了系统地提取程序中的所有执行路径，我们提出了算法 1，该算法在程序的控制流图（CFG）上运行。算法从入口节点开始遍历控制流图，并探索每个分支和循环，以枚举所有可能的执行路径。对于具有多条出边的节点，算法会区分循环头节点和一般分支节点。

对于循环头，算法会识别与循环体对应的子控制流图（该图已在根据抽象语法树构建控制流图时生成），并递归计算循环内所有可能的执行路径，将这些路径追加到整体路径列表中。对于一般的分支节点，算法会递归计算并合并沿真、假两个后继节点的路径。只有一个后继节点的节点会被添加到上下文中，随后将其作为大语言模型的输入。当入口节点出发的所有可达路径均已探索完毕时，算法终止。

基于已验证不变式的分层路径摘要

算法 2 将算法 1 得到的执行路径集合进一步细化，将循环体和条件分支等复合路径归纳为已验证不变式。其目标是用逻辑摘要替换复杂的控制流片段，从而能够在更高层次的抽象上推理正确性。

给定执行路径列表后，算法会逐条迭代处理。对于循环，它会识别循环体内的子路径，递归计算其不变式，并将循环片段替换为使用 SMT 求解器验证的循环不变式。同样，对于条件分支，算法会分别计算真分支和假分支的不变式，并用它们对应的逻辑析取替换相应路径。

我们维护一个全局前置条件，用于累积不同路径上的约束，同时使用独立的栈记录当前活动循环和分支结构。这些元素共同构成提供给大语言模型的上下文，使其能够生成更强且更准确的不变式。在算法 2 中，该上下文包括全局前置条件、循环栈、分支栈以及其他相关的程序信息。

早期结果与未来计划

我们在一部分基准程序上进行了初步实验，以评估我们提出的分支感知循环不变式生成方法在回答以下研究问题（RQ）方面的效果：“在准确率方面，BALI 能否优于现有基于大语言模型的循环不变式生成方法？”我们还总结了未来计划，以进一步改进和验证我们的方法。

初步结果

我们从广泛使用的 SV-COMP 套件中选取了基准程序，该套件包含一组多样化的程序，其中

算法 2：利用已验证不变式进行分层路径摘要 需要：执行路径列表 $Paths$**确保：总结所有路径的已验证不变式 1：** $pre\ cond \leftarrow \varnothing$ {**全局前置条件**}**2：** $loop\ stack \leftarrow$**空栈 3：** $branch\ stack \leftarrow$**空栈 4：对于每个** $path \in Paths$ **执行 5：如果** $path$ **包含循环，则 6：将循环上下文推入** $loop\ stack$ **7：** $loop\ body \leftarrow$ **提取循环内的子路径 8：** $loop\ inv \leftarrow$ **HierarchSummarize(**$loop\ body$**) 9：** $pre\ cond \leftarrow pre\ cond \wedge loop\ inv$**10：从** $loop\ stack$**弹出循环上下文 11：否则，如果** $path$ **包含条件分支，则 12：将分支上下文推入** $branch\ stack$ **13：** $true\ branch$**、** $false\ branch \leftarrow$ **提取分支 14：** $true_inv \leftarrow$ **HierarchSummarize(**$true\ branch$**) 15：** $false\ inv \leftarrow$ **HierarchSummarize(**$false\ branch$**) 16：** $pre\ cond \leftarrow pre\ cond \wedge$ **(**$true_inv \vee false\ inv$**) 17：从** $branch\ stack$**弹出分支上下文 18：否则 19：** $path_inv \leftarrow$ **ComputeInvariant(**$path$**) 20：** $pre\ cond \leftarrow pre\ cond \wedge path\ inv$**21：结束 if 22：结束 for 23：** $context \leftarrow$ **(**$loop\ stack, branch\ stack, pre\ cond$**) 24：** $invariants \leftarrow$**getInvariantsFromLLM(**$context$**) 25：返回** $invariants$

适用于复杂度和控制流结构各异的程序，例如并行循环和嵌套循环。首先，我们实现了最先进的基于大语言模型的循环不变式生成工具，例如 Clause2Inv 和 Lam4Inv，以评估它们能否为这些程序生成正确的不变式。结果表明，这些工具难以处理控制流复杂的程序，往往无法在合理的时间内生成正确的不变式。

接下来，我们从这些程序中提取不同的执行路径，并将其输入大语言模型以生成路径特定不变式。随后，我们使用 SMT 求解器验证生成的每个不变式是否正确。我们用这些已验证不变式替换相应路径，并重复这一过程，直至处理完所有路径。最后，我们将生成的所有不变式与源代码一并输入大语言模型，为整个程序生成最终不变式，并使用 SMT 求解器对其进行检查。

初步结果令人鼓舞。我们基于结构的方法成功为现有方法此前难以处理的程序生成了正确的不变式。具体而言，我们观察到准确率显著提升，我们的方法生成正确不变式的成功率更高。这一提升归功于路径特定分析所提供的聚焦推理，它使大语言模型能够更好地理解每条执行路径的上下文和约束，每次专注于程序的一小部分，从而降低推理任务的复杂度。

未来方向

总之，我们的未来工作将致力于扩展数据集、实现端到端工作流程自动化，并提升分支感知循环不变量生成方法的有效性和可扩展性。

首先，我们计划从 SV-COMP 套件中提取更多样化的基准程序，重点关注具有复杂控制结构、精细算术运算和数组操作的程序。这将使我们能够更好地评估我们的方法在不同程序类型上的泛化能力，并确定需要改进的领域。

其次，我们旨在实现方法部分所述的算法，以实现完全自动化的端到端工作流。通过整合路径提取、分层路径摘要、基于大语言模型的循环不变式生成以及基于可满足性模理论的验证，我们可以最大限度地减少人工干预，并简化整个循环不变式生成过程。

最后，我们计划在大规模和工业基准上，将我们的方法与最先进的基于大语言模型的循环不变式生成工具进行全面评估。此次评估将考察准确率、效率和可扩展性。此外，我们还将探索以下增强措施：

- 上下文感知的大语言模型提示策略，以提高不变式的精确性。
- 结合符号推理与大语言模型预测的混合技术。
- 扩展该方法，以处理并发、递归及其他高级结构。

通过这些努力，我们旨在开发一种稳健、可扩展且实用的分支感知循环不变式生成框架，使其优于现有方法。

结论

本文考察了当前基于大语言模型的不变式生成工具在处理具有复杂控制流的程序时的局限性，例如包含循环和嵌套分支的程序。为应对这些挑战，我们提出了 BALI，这是一种基于路径的框架，可生成经可满足性模理论求解器验证的路径特定不变式。通过将程序分解为单独的执行路径，BALI 简化了大语言模型的推理任务，并提高了生成不变式的正确性。

基准程序的初步结果表明，BALI能够成功为现有方法难以处理的程序生成正确的不变式，展现出更高的准确率和良好的效率。未来的工作包括扩展基准集、实现端到端工作流的自动化，以及探索上下文感知和混合推理策略，以进一步提升性能和可扩展性。

参考文献

Bogomolov, S.; Frehse, G.; Giacobbe, M.; and Henzinger, T. A. 2017年。模板多面体的反例引导的细化。载于国际系统构造与分析工具及算法会议，589–606。Springer。Brunsfeld, M.; Qureshi, A.; Hlynskyi, A.; ObserverOfTime; Lillis, W.; Vera, J.; dundargoc; Turnbull, P.; Clem, T.; Creager, D.; Helwer, A.; Rix, R.; Kavolis, D.; Clason, C.; Davis, M.; Bruins, R.; Delpeuch, A.; Ika; Ya, A.; Nguyen, T.-A.; bfredl; Brunk, S.; Massicotte, M.; Hasabnis, N.; McCoy, J.; Dong, M.; Moelius, S.; Kalt, S.; and Kolja。2025年。tree-sitter/tree-sitter：v0.25.10。Cao, W.; Wu, G.; Xu, T.; Yao, Y.; Wei, H.; Chen, T.; and Ma, X. 2025年。Clause2Inv：用于循环不变式推断的生成-组合-检查框架。*ACM*软件工程会议论文集，2(ISSTA)：1009–1030。Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000年。反例引导的抽象细化。载于国际计算机辅助验证会议，154–169。Springer。Col´on, M.; Sankaranarayanan, S.; and Sipma, H. B. 2003年。使用非线性约束求解生成线性不变式。载于国际计算机辅助验证会议。D'silva, V.; Kroening, D.; and Weissenbacher, G. 2008年。形式化软件验证自动化技术综述。*IEEE*集成电路与系统计算机辅助设计汇刊，27(7)：1165–1178。Evertz, H. G. 2003年。环路算法。《物理学进展》，52(1)：1–66。Furia, C. A.; Meyer, B.; and Velder, S. 2014年。循环不变式：分析、分类与示例。*ACM*计算调查（*CSUR*），46(3)：1–51。Garg, P.; L¨oding, C.; Madhusudan, P.; and Neider, D. 2014年。ICE：一种稳健的不变式学习框架。载于国际计算机辅助验证会议，69–87。Springer。Garg, P.; Neider, D.; Madhusudan, P.; and Roth, D. 2016年。使用决策树和蕴含反例学习不变式。*ACM SIGPLAN*通报，51(1)：499–512。Hojjat, H.; and R¨ummer, P. 2018年。ELDARICA Horn求解器。载于*2018*年计算机辅助设计形式化方法（*FMCAD*），1–7。Jhala, R.; and McMillan, K. L. 2006年。一种实用且完备的谓词细化方法。载于国际系统构造与分析工具及算法会议，459–473。Springer。Liu, R.; Chen, M.; Wu, L.-I.; Ke, J.; and Li, G. 2026年。利用大语言模型增强复杂程序的自动循环不变式生成。《计算机编程科学》，248：103387。McMillan, K. L. 2010年。用于程序测试和验证的惰性注释。载于国际计算机辅助验证会议，104–118。Springer。Mosqueira-Rey, E.; Hern´andez-Pereira, E.; Alonso-R´ıos, D.; ´Bobes-Bascar´an, J.; and Fern´andez-Leal, A. 2023年. 人在回路机器学习：现状。《人工智能评论》, 56(4): 3005–3054. Nguyen, T.; Antonopoulos, T.; Ruef, A.; and Hicks, M. 2017年. 用反例引导方法查找数值不变式。载于*2017*年第*11*届软件工程基础联合会议论文集, 605–615. Si, X.; Naik, A.; Dai, H.; Naik, M.; and Song, L. 2020年. Code2inv：用于程序验证的深度学习框架。载于计算机辅助验证国际会议, 151–164. Springer. Wu, G.; Cao, W.; Yao, Y.; Wei, H.; Chen, T.; and Ma, X. 2024年. 大语言模型与有界模型检验相遇：神经符号循环不变式推断。载于第*39*届*IEEE/ACM*自动化软件工程国际会议论文集, 406–417. Wu, X.; Xiao, L.; Sun, Y.; Zhang, J.; Ma, T.; and He, L. 2022年. 人在回路机器学习综述。《未来一代计算机系统》, 135: 364–381.