西安电子科技大学 XIDIAN UNIVERSITY | 广州研究院 Guangzhou institute of technology

LaM4Inv：LLM Meets Bounded Model Checking: Neuro-symbolic Loop Invariant Inference

大模型协同BMC：神经符号融合循环不变量推理

——ASE'24　南京大学

汇 报 人：胡 俊 杰

2025 年 1 月 3 日

相关名词介绍

一、循环不变量(Loop Invariant):

循环不变量是程序验证和软件工程中的一个基本概念。它是指在循环的每次迭代开始时都成立的一个条件或命题。如果一个条件在循环开始之前成立，并且在每次循环迭代后仍然成立，那么这个条件就是循环不变式。

二、神经符号(Neuro-symbolic)：

它集成了神经和符号人工智能架构，以解决各自的弱点，从而提供具有推理、学习和认知建模能力的强大人工智能(维基百科)。

它结合了基于人工神经网络的机器学习方法（如深度学习）与计算和人工智能（AI）的符号方法。这种方法的核心在于将神经网络的数据处理和模式识别能力与符号推理的逻辑性和可解释性相结合，以解决传统方法在处理不确定性、抽象推理和智能学习方面的局限性(GPT)。

确定的符号计算与非确定的概率计算相融合的智能化方案——人工基于逻辑设计的算法+机器学习数据训练的神经网络模型(南京大学李宣东——《神经符号融合软件的可信理论与方法》)。

背景知识：循环不变量的推理

$$\frac{P \Rightarrow I \qquad \{I \wedge B\}S\{I\} \qquad (I \wedge \neg B) \Rightarrow Q}{\{P\} \text{ while } B \text{ do } S\{Q\}} \qquad (1)$$

对于给定的循环 while B do S，循环不变量的推理旨在识别一个循环不变量 I 满足(1)式，其中P为前置条件，Q为后置条件，S为循环体，B为循环条件。本质上，(1)式满足以下三个要求，即(2)式。

可达性：指在循环开始之前，循环不变式所表达的条件必须成立。

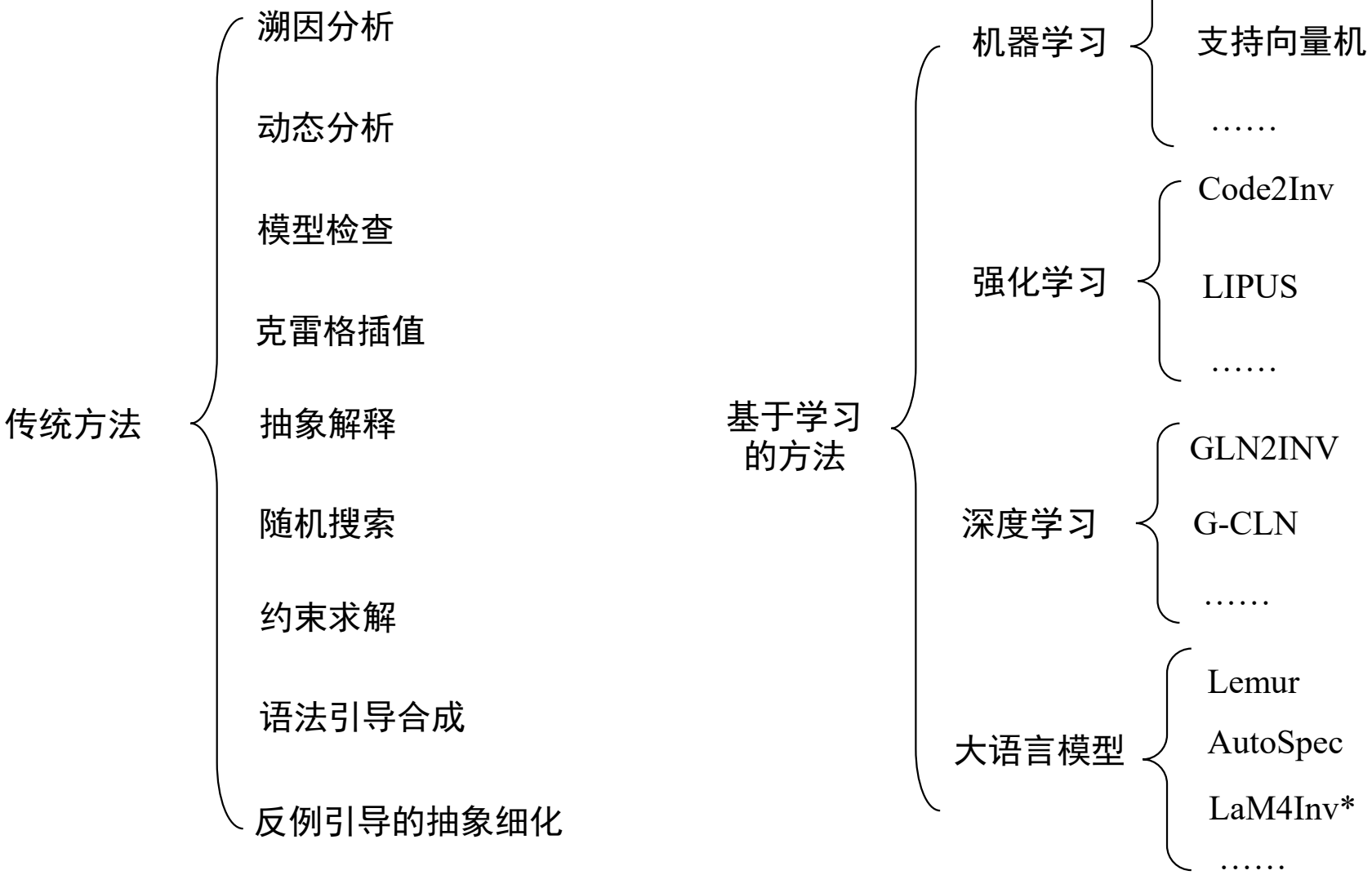
归纳性：对于循环不变量中的每个程序状态，在每次循环执行导致新的程序状态之后，这个新状态也必须保留在循环不变量中。

可证明性：在满足循环退出条件的情况下，循环不变量内的状态必须满足我们要验证的属性。

$$\neg(P \Rightarrow I) \vee \neg\,(I \wedge B \wedge S \Rightarrow I) \vee \neg(I \wedge \neg B \Rightarrow Q). \qquad (2)$$

如果SMT求解器返回“sat”(可满足的)，我们将得到一个反例，该反例表明违反了三个条件之一。否则，即求解器返回“unsat”(不可满足的)，我们将得到一个有效的循环不变量。

相关工作介绍——循环不变量推理



本方法核心思想：查询—过滤—重组

```
1.  int main() {
2.      //variable declarations
3.      int lo, mid, hi;
4.      //pre-condition
5.      assume(mid > 0 && lo == 0 && hi == 2 * mid);
6.      //loop-body
7.      while(mid > 0) {
8.          lo = lo + 1;
9.          hi = hi - 1;
10.         mid = mid - 1;
11.     }
12.     //post-condition
13.     assert(lo == hi);
14.}
```

```
Valid loop invariant:
   assert((hi − lo == 2 * mid) && (mid >= 0));
GPT's answer:
   assert((lo + hi == 2 * mid + lo) && (mid >= 0));
   assert((lo + hi == 2 * mid));
   assert((lo + mid == hi) && (mid >= 0));
   assert((hi − lo == 2 * mid));
The red part is the correct predicate.
```

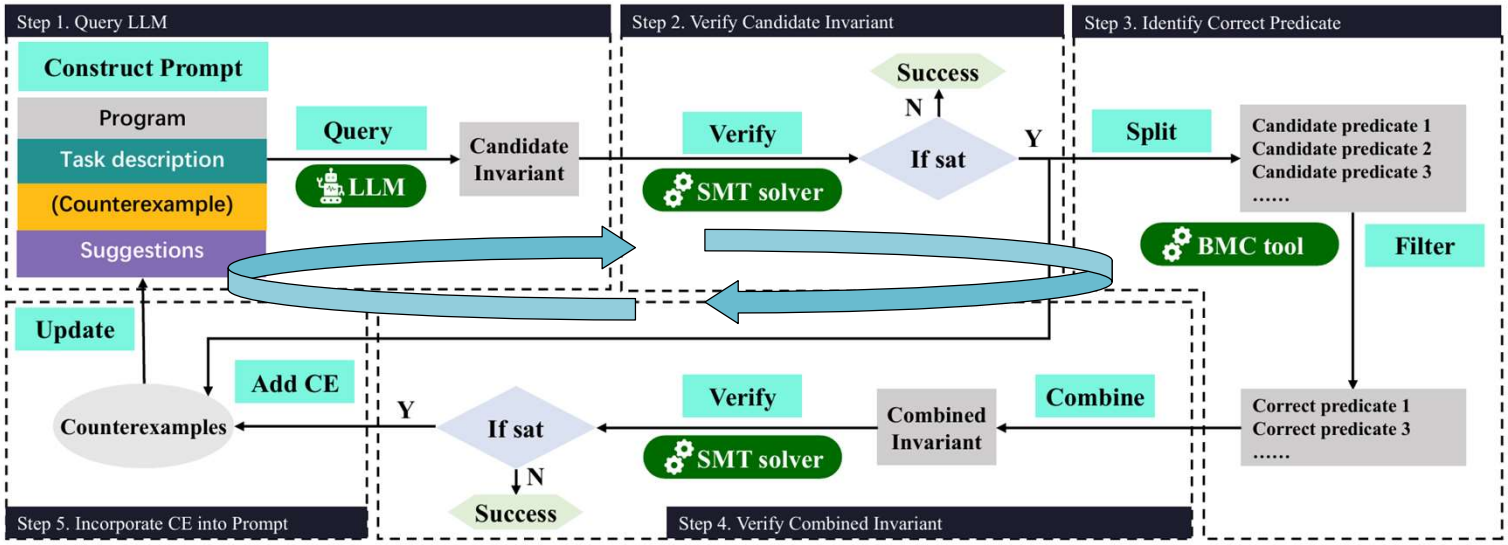
查询：利用两种类别的Prompt模板（初始提示词和带有反馈的提示词）向LLM提问，获得多个候选循环不变量。

过滤：将验证失败的循环不变量拆分为单个谓词，利用有界模型检查BMC检查单个谓词是否满足循环不变量的性质。

$$\neg(P \Rightarrow I) \vee \neg\,(I \wedge B \wedge S \Rightarrow I) \vee \neg(I \wedge \neg B \Rightarrow Q). \qquad (2)$$

重组：将过滤后剩下的有效不变量组装成新的候选循环不变量，通过反馈进一步验证。

*LaM4Inv*工作流程



Step1

- 为LLM准备了一个包含程序信息、任务描述和建议的提示，要求生成循环不变量。输出是一组候选不变量，形式为“assert(…)”

Step2

- 使用SMT求解器验证从LLM返回的每个候选不变量。如果该不变量满足(1)式中的三个条件，则返回该不变量，并终止该过程;否则，SMT求解器返回候选不变量的反例。

Step3

- 将每个失败的候选不变量拆分为单独的谓词，并使用有界模型检查工具检查每个谓词是否在输入程序中成立。经过验证的谓词形成正确的谓词集。

Step4

- 将验证的谓词组合成一个新的循环不变量，并将其提交给SMT求解器进行验证。类似地，如果组合不变式通过了SMT检查，则过程终止，否则返回反例。

Step5

- 将步骤2或步骤4中生成的反例合并为一个新的提示符，查询LLM以获得一个新的循环不变量，然后返回步骤2。

重复自动生成过程，直到获得所需的循环不变量或耗尽资源预算。

*LaM4Inv*实现细节

一、谓词过滤

Algorithm 1: Predicate Filtering Algorithm

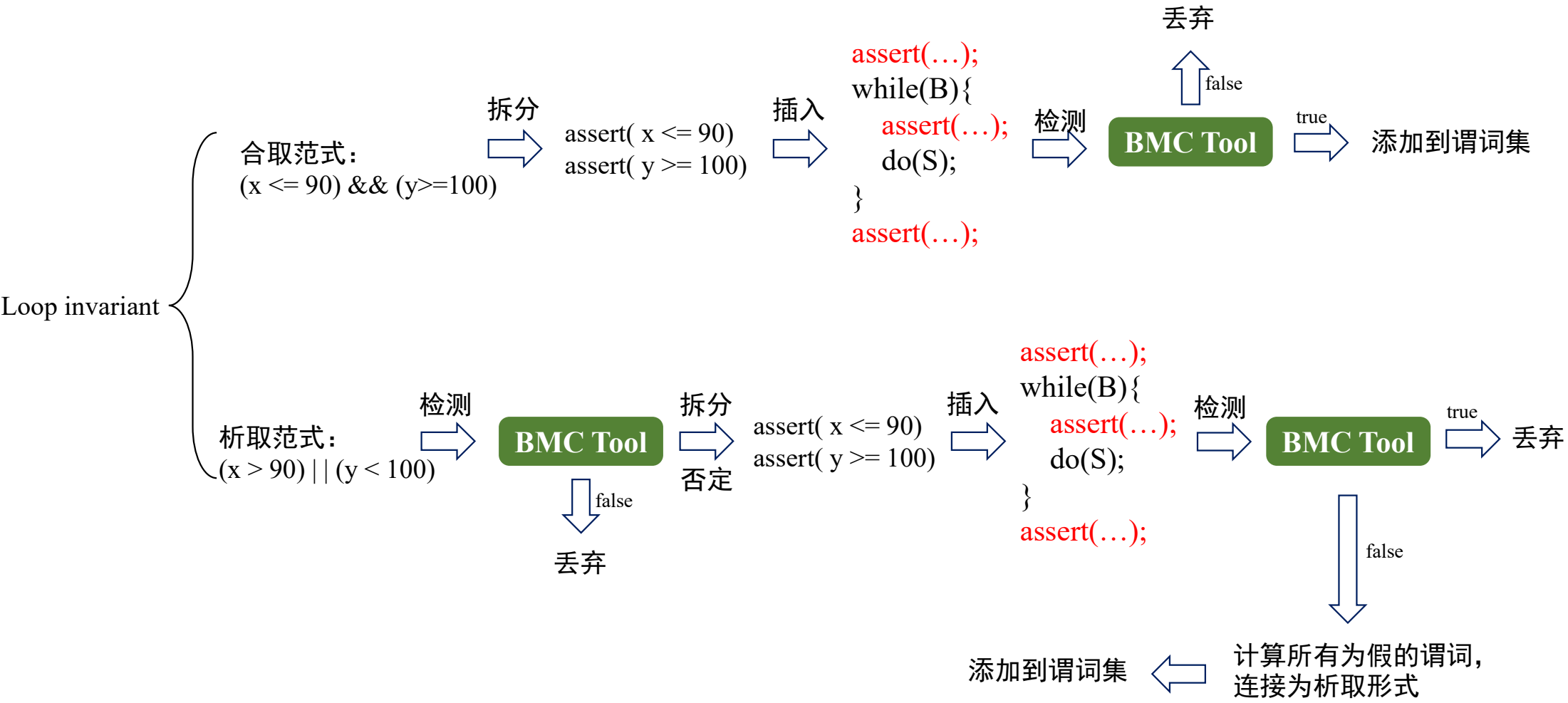
Input: A program p, current candidate invariant set CIs, correct predicate set CPs.
Output: Updated correct predicate set CPs.

```
Function BMCFilter(p, CIs, CPs):
    foreach can_I in CIs do
        if can_I is in conjunctive normal form then
            foreach can_P in can_I.split() do
                if BMC.verify(p, can_P) is not Failure then
                    CPs.add(can_P);
        else
            if BMC.verify(p, can_I) is not Failure then
                foreach can_P in can_I.split() do
                    if BMC.verify(p, ¬(can_P)) is not
                      Failure then
                        can_I.remove(can_P);
                CPs.add(can_I);
    return CPs
```

简单来说，LLM难以做到生成一个完整、准确、pass@1的循环不变量，他有一点顾头不顾尾的感觉。

这里要做的就是寻找当前生成的不变量中是否有可用的部分，将可用的部分保留下来，尝试着组合出一个完整、可用的循环不变量。

*LaM4Inv*实现细节



*LaM4Inv*实现细节

二、Prompt语句：初始提示和中间提示（反馈）两种类型

使用SMT求解器验证每个生成的候选循环不变量(要么直接从LLM获得，要么从经过验证的谓词重新组合)。如果验证失败，SMT求解器可能会提供关键的反馈，我们的中间提示被设计用来处理这种情况。

为了使反馈的效果最大化，本文既考虑了所提供的反例，也考虑了反例的具体类型(即，违反了(2)式中的哪个条件)。为了获得这种类型，本文使用SMT求解器来求解(2)式中的三个子句，如果有任何子句产生解，则记录该类型。根据当前候选不变量无法满足的三个条件(即可达性、归纳性和可证明性)，设计了不同的中间提示。

$$\neg(P \Rightarrow I) \vee \neg\,(I \wedge B \wedge S \Rightarrow I) \vee \neg(I \wedge \neg B \Rightarrow Q). \qquad (2)$$

*LaM4Inv*实现细节

Prompt 2: Intermediate prompt with reachability violation.

[p]

Print loop invariants as valid C assertions that help prove the assertion. Your previous answer [CI] is too strict and not reachable. The reachability of the loop invariant means that the loop invariant I can be derived based on the pre-condition P, i.e. $P \Rightarrow I$.

The following is a counterexample given by z3: [CE]

In order to get a correct answer, you may want to consider the initial situation where the program won't enter the loop. Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'

Prompt 4: Intermediate prompt with inductiveness violation.

[p]

Print loop invariants as valid C assertions that help prove the assertion. Your previous answer [CI] is not inductive. The Inductiveness of the loop invariant means that if the program state satisfies loop condition B, the new state obtained after the loop execution S still satisfies, i.e. $\{I \wedge B\}\ S\ \{I\}$.

The following is a counterexample given by z3: [CE]

In order to get a correct answer, you may want to consider the special case of the program executing to the end of the loop. Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'

Prompt 5: Intermediate prompt with provability violation.

[p]

Print loop invariants as valid C assertions that help prove the assertion. Your previous answer [CI] is too weak and not provable. The Provability of the loop invariant means that after unsatisfying loop condition B, we can prove the post-condition Q, i.e. $(I \wedge \neg B) \Rightarrow Q$.

The following is a counterexample given by z3: [CE]

In order to get a correct answer, you may want to consider the special case of the program executing to the end of the loop. If some of the preconditions are also loop invariant, you need to add them to your answer as well. Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'

以可达性(Reachability)为例。相应的提示符在Prompt 2中给出。

中间提示用于指导LLM根据反例(表示为[CE])的反馈修改候选不变量(表示为[CI])。它通常由以下部分组成:当前程序P、任务描述、上一个答案、上一个答案的失败原因、SMT解算器提供的反例以及生成建议。具体来说，当候选不变量不能满足可达性时，我们通知LLM之前的答案过于严格，使得无法从前置条件中推导出候选不变量。

本质上，如果可证明性不满足，我们表明之前的答案过于松散，推导出的不变量不足以推断后置条件;如果归纳性不满足，我们解释之前的答案对每次循环迭代都不成立，可能需要考虑循环执行期间的特殊情况。

实验设置

基准数据集：

作者构建了一个包含316个基准问题的数据集。基准包含由Code2Inv收集的133个问题，这些问题通常由以前的工作评估。此外，作者还手工制作了2019年SyGuS竞赛中的84个问题和2024年SV-COMP基准测试中的99个问题。

对比基线方法：

方法	描述	类别
LoopInvGen	枚举合成	符号方法
CVC5	SMT求解器，配备用于循环不变量的语法引导合成引擎	语法引导合成
Code2Inv	使用强化学习来推断循环不变量，图神经网络来捕获程序的特征。	强化学习
LIPUS	基于Code2Inv构建。它通过二维奖励改进强化学习	强化学习
CLN2INV	设计新的神经网络模型来拟合逻辑表达式	深度学习
G-CLN	基于程序执行跟踪和进一步扩展CLN2INV	深度学习
ESBMC	结合了BMC, k归纳，抽象解释，SMT和约束编程求解器，是用于验证程序的上下文绑定模型检查器	符号方法
LEMUR	将程序验证任务转换为一系列由LLMs提出并由自动推理器验证的演绎步骤，协助ESBMC验证程序。	大语言模型

实验评估

RQ1：与最先进的方法相比，LaM4Inv在推断循环不变量方面的效果如何?

采用以下性能指标，第一个指标与有效性有关，后两个指标与效率有关。

1)成功生成循环不变量的数量；2)在推理过程中提出的候选不变量的数量；3)推断不变量所消耗的时间。

八种方法都能求解　八种方法都失败　只有这种方法能解　（Code2Inv/SyGuS/SV-COMP 2024)

Methods	# Solved Benchmarks				# Avg. Proposals	Avg. Time (s)
	All	None	Only	Total (133/84/99)		
LoopInvGen [38]			0	218 (107/49/62)	5.6	17.8
CVC5 [3]			1	207 (107/46/54)	13.9	5.5
Code2Inv [46]			0	210 (110/47/53)	10.9	137.6
LIPUS [54]			0	159 (124/18/17)	3.7	48.4
CLN2INV [42]	43	4	0	211 (124/35/52)	23.9	0.6
G-CLN [52]			0	219 (116/45/58)	20.1	2.6
ESBMC [21]			0	126 (70/23/33)	0	0.2
LEMUR [50]			0	177 (81/43/53)	1.5	9.8
LaM4Inv			20	309 (133/81/95)	3.7	35.7

如表所示，LaM4Inv成功地推断出316个基准问题中的309个循环不变量，这比竞争对手至少多90个。此外，LaM4Inv为20个问题生成了正确的循环不变量，而现有的竞争对手都无法解决这些问题。这样的结果证明了所提出方法的优越性能。

实验评估（消融实验）

Lemur所采用的基线prompt

Prompt 3: Baseline prompt.

[p]

Print loop invariants as valid C assertions that help prove the assertion. Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'

RQ2：不同的LLM、提示设计和谓词过滤机制对LaM4Inv的性能有什么影响?

LLMs	# Solved Benchmarks 共316个			# Avg. Proposals			# Avg. Time (s)		
	Baseline Prompt	No BMC	Full	Baseline Prompt	No BMC	Full	Baseline Prompt	No BMC	Full
Llama-3-8B	231	214	233	8.1	10.8	7.0	20.7	10.4	18.8
GPT-3.5-Turbo	265	248	271	6.1	8.9	5.4	22.0	21.2	28.9
GPT-4	241	275	306	3.8	6.3	5.0	27.1	26.5	46.3
GPT-4-Turbo	246	275	309	3.5	5.5	3.7	28.2	27.0	35.7

1、不同LLM的影响

GPT-4和GPT-4- turbo通常在“无BMC”和“FULL”列下都表现出更好的性能，GPT-3.5-Turbo在基线提示下的表现优于其他两种。即使是表现最差的法学硕士也表现出与前面提到的最先进的方法相当的有效性。

2、反馈信息的影响

将“Baseline Prompt”列的结果与“FULL”列的结果进行比较。在解决基准问题的数量方面，完整版的LaM4Inv在所有情况下都更好，特别是对于GPT-4系列。

3、谓词过滤的影响

比较了“无BMC”列和“FULL”列的结果。当使用GPT-4-Turbo时，完整版LaM4Inv平均解决了309个基准问题，并提出了3.7个建议;当排除谓词过滤机制时，它解决的基准问题少了34个，同时平均需要多1.8个提案。这种改进的代价是在使用BMC工具过滤谓词时需要额外的时间。

LaM4Inv的局限性

1、SMT求解能力有限

SMT求解器经常在处理乘法、除法和幂运算时遇到困难。LLM可以生成“pow(a, b)”形式的候选不变量，但由于性能的原因，SMT无法验证这些不变量。

2、有限的数据集

尽管本文将基准测试集扩展到316个问题，但与实际应用程序相比，这些基准测试仍然非常基本，类似于玩具问题。数据集的有限大小也对微调LLM提出了挑战，因为它没有提供足够的数据来捕捉现实世界程序的复杂性。

3、数据污染

不能保证研究中使用的benchmarks不是LLM预训练数据的一部分。

4、依赖于BMC的性能

可以学习的地方

1、任务分解思想——在AutoSpec的基础上继续细分

2、将invariant直接当作assert插入，用于反馈或修复

《A Framework for Debugging Automated Program Verification Proofs via Proof Actions》 ——CAV'24 CMU

3、数据集的设计

LaM4Inv/Benchmarks at main · SoftWiser-group/LaM4Inv (github.com)

总结

本文提出了一种新的方法LaM4Inv，将LLM与有界模型检查相结合，自动生成循环不变量。LaM4Inv的关键是“查询-过滤器-重组”过程，该过程在(神经)LLM(用于生成候选变量)和(符号)有界模型检查(用于过滤掉错误谓词)之间形成闭环协同作用。对包含316个问题的扩展数据集的评估证实了LaM4Inv在循环不变量生成方面优于一系列最先进的方法。