基于大语言模型的循环不变量的自动生成

马修齐

2024 年 12 月 12 日

摘要

循环不变量是程序验证中的关键概念。由于其多样性和复杂性，以及程序本身的复杂性，循环不变量的自动生成成为实现自动程序验证的瓶颈之一。本文着眼于近年来新涌现的一类自动生成循环不变量的方法——基于大语言模型的先猜后证 (guess and check) 模式，并着重介绍 Wu et al. 的一种将大语言模型与有界模型检查相结合的具体生成模型——LaM4Inv[6]。

1 引言

循环不变量是一个每次在循环迭代开始时都为真的断言。它浓缩了循环的过程，描述了循环的核心性质。循环不变量的引入使得人们在验证带循环的程序时，无需将循环展开，而可以通过验证循环不变量的性质，进而推出循环的部分正确性，大大提高了验证效率。因此，寻找合适的循环不变量是验证带循环的程序的关键步骤。

然而，如何用自动化工具高效生成合适的循环不变量一直是困扰学术界的难题。一个较为经典的方法是模版化约束求解 (template-based constraint-solving)[4]。它通过预设带未知数的循环不变式的模版（如线性表达式），再依靠约束求解器根据归纳性算出未知数。然而该方法高度依赖于模版选择的正确性，存在较明显的局限性。

近年来，随着机器学习的技术的发展，通过数据训练机器生成循环不变量的方法进入研究者的视野 [3][5][7]。该类方法的验证模式可以概括为先猜后证，即先用训练过的模型根据新的程序生成候选循环不变量，再使用相关求解器 (如 Z3) 筛选出正确的循环不变量。尽管此类方法本质上是启发性的，仍不具有完备性保证，但它有理解复杂程序、生成超越传统模型的循环不变量的潜力，进而具有广阔研究前景。

大语言模型作为市场上较为成熟的机器学习模型，有较强的程序、文本理解和学习能力以及较低的使用成本，被许多研究者青睐。使用大语言模型自动生成循环不变量的最关键的步骤在于如何处理大语言模型给出的候选循环不变量。Saikat Chakraborty et al. 设计了将候选不变量排序的函数 [1]，Adharsh Kamath et al. 则引入了 Houdini 算法及修复步骤 [2] 等等。但这些方法对大语言模型生成完全正确的循环不变量的依赖性依旧很高，限制了生成成功率。为了解决这个难点，Wu et al. 观察到尽管大语言模型有时候无法生成完全正确的循环不变量，组合成正确的循环不变量的各个谓词往往分布在大语言模型给出的多个候选循环不变量之中。于是他们设计了拆分、过滤、重组循环不变量的算法，最终大大提升了获得正确的循环不变量的概率。下面本文将详细介绍 Wu et al. 的处理方法 (LaM4Inv)——一个将大语言模型与有界模型检查相结合的生成方案。

2 背景知识——循环不变量

对于一个循环 while B do S 这个循环的循环不变量 I 可以通过霍尔逻辑表示为

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

其中 P 为循环的前置条件，Q 为循环的后置条件。

根据以上表示可以知道，一个正确的循环不变量应满足三个性质：

(1) 可及性 (Reachability)，即循环不变量所表示的程序状态的集合应包含前置条件所表示的程序状态的集合。

(2) 递归性 (Inductiveness)，即对于每一组在循环不变量表示范围内的程序状态，经过一次循环后，新的程序状态仍在循环不变量的表示范围内。

(3) 可推性 (Provability)，即当一组循环不变量表示范围内的程序状态满足循环终止条件时，它应该落在后置条件所表示的程序状态之内。

3 LaM4Inv 的具体实现方法

3.1 总体流程

1. 生成　将带循环的且注明前置条件和后置条件的程序发送给大语言模型，让其生成多个候选循环不变量。

2. 检验　将每一个生成的候选循环不变量交给 SMT 求解器检验。若返回“不可满足”，则已找到正确的循环不变量，流程终止；否则获得一组反例。

3. 拆分与过滤　将每一个未通过验证的候选循环不变量拆分成单个谓词，用有界模型检查工具检查各个谓词在循环中是否一直成立，未通过检验的谓词应删去，留下来的谓词构成一正确谓词集。

4. 重组　将正确谓词集里的谓词相交，得到一个新的循环不变量，带入步骤 2.

5. 循环　将步骤 2 和 4 中验证失败的循环不变量和反例返回给大语言模型，重复上述过程，直至达到时间上限。

Figure 1: Code

```
int main() {
    int i=1, j=9;
    //pre-condition
    assert(i=1 && j=9);
    //loop body
    while (j > i)
        j = j - 3;
    //post-condition
    assert(j==0);
}
```

Figure 2: Prompt

Print loop invariants as valid C assertions that help prove the assertion. In order to get a correct answer, you may want to consider both the situation of not entering the loop and the situation of jumping out of the loop. Use ‘&&’ or ‘||’ if necessary. Do not explain. Your answer should be ‘assert(...);’

Figure 3: Reply

A possibly correct invariant:

assert(i == 1 && (j % 3 == 0) && j >= i - 1);

GPT4’s answers:

assert(i == 1 && j >= 0 && (j % 3 == 0) && j > i);

assert(i == 1 && (j % 3 == 0) && j >= i);

assert((j - i) % 3 == 0 && j > i);

assert(i == 1 && (j % 3 == 0) && j >= i - 1);

3.2 技术细节与说明

生成　提示词的设计如上图 (2), 图 (1)(2)(3) 展示了一个用大语言模型生成最初的候选不变量的例子。

检验　SMT 求解器可以验证一个逻辑公式 α 是否是可满足的（是否存在一组变量的赋值使 α 为真）。这与我们验证候选循环不变量是否满足 2.1 节中三个性质的需求契合。由于要求无论怎么取值，循环不变量的三个性质均成立，可以将它们取反并相并，得到如下逻辑公式交给 SMT 求解器检验

$$\neg(P \Rightarrow I) \vee \neg(I \wedge B \wedge S \Rightarrow I) \vee \neg(I \wedge \neg B \Rightarrow Q)$$

若 SMT 求解器返回“不可满足”，这个候选循环不变量就是一个真正的循环不变量；如果 SMT 求解器返回“可满足”，这个候选循环不变量就是一个假的循环不变量，求解器返回的状态就是一组反例。

拆分与过滤　简单来说，大语言模型生成的候选不变量有两种形式，A && B && C... 和 A || B || C... 对于第一种形式，我们将其拆分为 A，B，C... 这样单独的谓词，并使用有界模型检查工具检查每一个谓词在循环前、每一次循环的开头及结尾是否一直成立。如果返回的是不成立，说明这个谓词不满足循环不变量的可达性或递归性，我们将其删去。如果返回的是成立，将该谓词放进正确的谓词集。对于第二种形式，我们对整个候选不变量用相同方法检查。如果返回的是不成立，说明这个最松的逻辑公式也不满足可达性或递归性，我们将其整个放弃。如果返回的是成立，我们再将其拆分为单独的谓词，将它们分别取反，再进行检查。如果返回的是成立，说明取反之前的谓词在循环中一直不成立，即对循环不变量没有任何贡献，我们将去舍去。最后，将所有留下的谓词相并得到的大谓词放进正确的谓词集。

重组　正确的谓词集里的谓词均通过了有界模型检验，它们自然满足可达性和递归性，只是它们的可推性未能得到保证，即它们可能过松，从而无法证出我们所需的结论。于是我们将谓词集里的谓词相交，得到一个最紧的谓词。它的可达性和递归性不会被破坏，并且最有可能满足可推性，所以成为了新的候选循环不变量，带回用 SMT 求解器检验。

一个自然的问题是，过滤步骤本质是对谓词可达性和递归性的检测，为什么不直接和检验步骤一样使用 SMT 求解器对这两个性质进行推导，而要用低效率的有界模型检查展开每一次循环，检查真实的程序状态呢? 我们可以再看图（1）的例子。假设我们要检验谓词 $j >= i - 1$，SMT 求解器实际上会返回一个反例，如 $\{i : 1,\ j : 2\}$(此时下一个状态是 $\{i : 1,\ j : -1\}$, 不满

足 $j >= i - 1$)，进而认为这个谓词不满足递归性。事实上，我们发现这个状态不发生在真实的循环过程中，$j >= i - 1$ 实际满足递归性，因此 SMT 返回的是个假反例。假反例的生成原因可以概括为，在得到的谓词不够紧的情况下，SMT 求解的状态空间大于实际程序执行到循环时到状态空间，假反例便存在于多出来的那部分状态空间中。

循环　该步骤的设计充分利用了大语言模型的学习、理解能力，进而创造出更多的候选循环不变量，提高正确的循环不变量的获得率。对于分别违背三个不同性质的反例，研究者设计了三种具体的提示词，告知了大语言模型三个性质需满足的逻辑公式。由于篇幅较长，便不在此展示。

4　性能评估

研究者对 LaM4Inv 进行了 316 个程序的基准测试 [6]。这些基准程序来源于经典的 Code2Inv 数据集、2019 年的 SyGuS 竞赛，以及 2024 年的 SV-COMP 竞赛。最终，LaM4Inv 成功生成了 309 个循环不变量。作为对比，CLN2Inv（基于机器学习）[3]，Code2Inv（基于强化学习）[5]，LEMUR（基于 LLM 的直接生成）[7] 等目前较为先进的循环不变量自动生成模型均只成功生成了不超过 220 个。在有效性上，LaM4Inv 领先其他所有模型。

但与此同时，LaM4Inv 对每个成功解决的问题的平均用时为 35.7s，长于传统方法。这主要是因为有界模型检查的效率相对较低。

5　局限性

LaM4Inv 主要存在两点不足。一是有界模型检查需要将循环展开，这使得该方案在应对特定程序时效率会大幅降低。二是在应对形如 $A \mid\mid B$ 的未通过有界模型检查的候选不变量时，出于效率方面的考量，该方案没有对 A 和 B 内部再进行过滤，所以会错失一些生成正确不变量的机会。

6　总结

本文介绍了一个基于大语言模型自动生成循环不变量的方法 LaM4Inv。可以看出，大语言模型的生成，以及对其生成候选进行拆分重组的优化，的确能够解决更多复杂循环不变量的自动生成问题，具有投入实践的前景。

参考文献

[1] Saikat Chakraborty, Shuvendu K Lahiri, Sarah Fakhoury, Akash Lal, Madanlal Musuvathi, Aseem Rastogi, Aditya Senthilnathan, Rahul Sharma, and Nikhil Swamy. Ranking llm-generated loop invariants for program verification. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

[2] Adharsh Kamath, Aditya Senthilnathan, Saikat Chakraborty, Pantazis Deligiannis, Shuvendu K Lahiri, Akash Lal, Aseem Rastogi, Subhajit Roy, and Rahul Sharma. Finding inductive loop invariants using large language models. *arXiv preprint arXiv:2311.07948*, 2023.

[3] Gabriel Ryan, Justin Wong, Jianan Yao, Ronghui Gu, and Suman Jana. CLN2INV: Learning Loop Invariants with Continuous Logic Networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[4] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constraint-based linear-relations analysis. In Roberto Giacobazzi, editor, *Static Analysis*, pages 53–68, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[5] Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *Advances in Neural Information Processing Systems(NeurIPS) 31(2018)*, 2020.

[6] Guangyuan Wu, Weining Cao, Yuan Yao, Hengfeng Wei, Taolue Chen, and Xiaoxing Ma. LLM Meets Bounded Model Checking: Neuro-symbolic Loop Invariant Inference. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, China, 2024. State Key Laboratory for Novel Software Technology, Nanjing University.

[7] Haoze Wu, Clark Barrett, and Nina Narodytska. Lemur: Integrating large language models in automated program verification. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.