第46卷　第10期　　　　　计　算　机　学　报　　　　　Vol. 46 No. 10
2023年10月　　　　　CHINESE JOURNAL OF COMPUTERS　　　　　Oct. 2023

# 基于同位替换的深度程序生成模型测试及修复方法

孙泽宇[1),2)]　张　洁[3)]　熊英飞[2)]　郝　丹[2)]　张　路[2)]

[1)](中关村实验室　北京　100871)

[2)](北京大学计算机学院高可信软件技术教育部重点实验室　北京　100871)

[3)](伦敦国王学院　伦敦　英国)

**摘　要**　程序的编写是软件开发中的主要活动. 提高程序编写的效率一直是软件工程研究关注的重要问题. 基于深度学习的程序生成是提高程序编写效率的重要途径. 该类方法借鉴了自然语言处理中的基于深度神经网络的机器翻译方法,试图将输入的自然语言描述自动转换为对应程序. 然而,现有程序生成方法的生成效果很难让人满意. 在这类方法中,对输入自然语言描述的微小改动可能使得输出的代码发生巨大改变. 这种变化会给开发者带来理解上的困难. 为了解决这个问题,本文提出了一种感知上下文语境的测试和修复算法(COTE). COTE将变异和蜕变测试相结合以实现程序生成系统中相应问题的自动测试并在测试的基础上实现自动修复. 本文在常用程序生成工具CodeGPT上对COTE进行验证. 实验结果表明:在COTE的测试下,CodeGPT大约有39%的输入存在问题;同时,COTE可以自动修复其中33%~42%的问题.

**关键词**　程序生成;程序测试;程序修复;神经网络;软件工程

**中图法分类号**　TP311　　**DOI号**　10.11897/SP.J.1016.2023.02025

# An Isotopic-Replacement-Based Approach for Testing and Improving Code Generation Systems

SUN Ze-Yu[1),2)]　ZHANG Jie[3)]　XIONG Ying-Fei[2)]　HAO Dan[2)]　ZHANG Lu[2)]

[1)](*Zhongguancun Laboratory, Beijing* 100871)

[2)](*Key Lab of HCST (PKU), MOE; SCS, Peking University, Beijing* 100871)

[3)](*King's College London, London, UK*)

**Abstract**　Programming is a fundamental aspect of software development, serving as the building block for creating robust, functional, and efficient applications. In today's technology-driven world, the demand for software solutions is growing at an unprecedented rate. As a result, there is an increasing need for streamlined programming processes that can keep pace with this rapid growth. To address this need, many researchers have turned their attention towards code generation as a means to automate and expedite the coding process, making it more accessible and efficient for developers. Code generation systems are designed to accept an input in the form of a natural language description and then automatically generate the target program. This approach

---

收稿日期:2022-09-26;在线发布日期:2023-04-21. 本课题得到国家重点研发计划课题(编号:2022YFB4501902)和中兴通讯-北京大学基础软件联合实验室项目的支持. **孙泽宇**(通信作者),博士,助理研究员,中国计算机学会(CCF)专业会员,主要研究领域为软件测试、程序生成. E-mail: szy_@pku.edu.cn. **张　洁**,博士,助理教授,主要研究领域为软件测试、机器学习可信度、机器学习公平性、程序生成和程序分析. **熊英飞**,博士,长聘副教授,国家自然科学基金优秀青年科学基金获得者,中国计算机学会(CCF)专业会员,主要研究领域为软件工程、程序设计语言. **郝　丹**,长江学者,博士,教授,中国计算机学会(CCF)杰出会员,主要研究领域为软件测试与排错. **张　路**,博士,教授,长江学者以及国家杰出青年科学基金入选者,中国计算机学会(CCF)杰出会员,主要研究领域为软件分析、软件测试、软件维护与演化.

has the potential to revolutionize the programming landscape by reducing the time and effort required to develop software solutions. However, despite the potential benefits of code generation, existing approaches are not without their limitations. One of the most significant challenges faced by code generation systems is their sensitivity to changes in the input sentence. Even minor modifications to the input can lead to substantial and undesirable changes in the output, ultimately impacting the reliability and effectiveness of code generation systems in real-world applications. This sensitivity poses a significant obstacle to the widespread adoption of code generation technologies, as developers need to have confidence in the consistency and accuracy of the generated code. In order to tackle this challenge, we propose a COntext-aware code generation TEsting and Repair approach (COTE). COTE integrates mutation and metamorphic testing techniques. This method utilizes context-similar mutations to generate mutated sentences, which serve as test inputs for the code generation system being evaluated. If a context-similar mutation causes a disruption exceeding the predetermined threshold in the code generation system of the non-mutated portion, the approach identifies and reports it as a bug. Once a bug is reported, COTE further leverages its black-box/grey-box repair capabilities to automatically repair these bugs, thereby enhancing the overall quality and reliability of the generated code. To assess the effectiveness of COTE, we conducted a comprehensive series of experiments using CodeGPT, a state-of-the-art code generation system. Our experimental results provide valuable insights into the performance of COTE in a real-world setting. With the implementation of COTE, we found that bugs were detected in approximately 39% of CodeGPT's input. This demonstrates COTE's capacity to identify a substantial number of issues within the generated code that might have otherwise gone unnoticed. Furthermore, COTE showcases a remarkable ability to automatically repair the detected bugs, further emphasizing its potential as a powerful tool for improving code generation systems. Our experiments reveal that COTE successfully repaired between 33%~42% of the identified bugs, which underlines its capacity to enhance the overall reliability and quality of the generated code.

**Keywords** code generation; software testing; program repair; neural network; software engineering

# 1 引 言

随着人类社会的发展,信息化逐渐成为了社会发展的关键词之一,软件也逐渐成为了信息化服务设施的基本构建元素[1-2]. 在信息化的过程中,软件所涉及的功能越来越复杂,其程序对应的代码规模也日益增长. 而软件开发主要依赖于开发者的人工劳动付出,如何有效地提高开发者的开发效率、减轻开发者的开发负担也成为了软件工程中相关研究领域所关注的问题.

长期以来,许多研究人员通过研究辅助软件开发的方法来提高软件开发的自动化水平. 其中具有代表性的技术便是程序生成方法. 给定一段自然语言描述,程序生成方法可以帮助使用者自动生成该描述所对应的目标代码. 举例来说,给定一段自然语言"打开A文件",程序生成方法可以生成一段指定语言的代码"f = open(A)"(Python).

随着人工智能技术的发展,Ling等人[3]将自然语言翻译中的基于序列到序列(sequence-tosequence)的深度学习框架与程序生成相结合. 该工作把程序当作一种新的自然语言并基于程序分词后所得程序词序列来生成代码. 给定一段自然语言描述,一个用数据训练好的神经网络模型可以自动生成对应的程序. 在此工作的基础上,研究者们针对程序生成提出了一系列方法[4-9].

然而,现有程序生成方法在很多情况下的生成效果很难让人满意. 如表1所示,程序生成方法CodeGPT在生成自然语言"returns the first child

element of a node which matches the given tag name.”的输出时，其得到 Java 代码“T function（T arg0，String arg1）for（T loc0：children（））if（arg0. getTagName（）. equals（arg1））return loc0；return null；”. 然而，当我们把输入中的“which”修改为具有同样含义的“that”时，其输出的代码发生了巨大改变，即生成了代码“T function（Document arg0，String arg1）return（T）arg0. getFirstChild（arg1）；”，这种生成的变化往往会给开发者尤其是初学者带来显著的理解上的困难.

**表1 程序生成模型的错误示例**

| 输入自然语言描述 | 输出代码 |
|---|---|
| returns the first child element of a node ***which*** matches the given tag name. | T function (T arg0, String arg1) {<br>for (T loc0: children()) {<br>if (arg0. getTagName(). equals(arg1)) {<br>return loc0;<br>}<br>}<br>return null;<br>} |
| returns the first child element of a node ***that*** matches the given tag name. | T function (Document arg0, String arg1) {<br>return (T) arg0. getFirstChild(arg1);<br>} |

现有的一些相关的机器翻译测试技术[10-11]提出用蜕变测试对机器翻译的效果进行测试，然而，其并不是为程序生成所设计的. 同时，即使测试出问题，该类机器翻译测试工作无法自动修复所测试出来程序生成模型中的问题，进而难以达到自动提升程序生成效果的目的. 因此，程序生成系统需要一个自动提升生成效果的技术以解决以上问题.

本文认为程序生成系统应该满足以下扰动约束：输入的少量修改理应对输出的程序代码造成微小的影响（本文称之为一致性）. 如果输入的少量修改对程序生成系统所生成的程序造成了巨大的影响，本文则认为其是一个约束不一致性问题.

在这个关系的基础上，本文提出了一种可以感知上下文语境的测试和修复算法（COntext-aware code generation TEsting and repair approach，COTE）. 该算法的目标是用以实现全自动测试和提升程序生成系统的效果. COTE将变异与蜕变测试相结合以实现自动测试程序生成系统中的不一致性问题的功能. 然后，COTE采用一种多样本组合的方法对测试出的问题进行自动修改. 该技术可以生成大量具有相似输出的程序，利用输入输出一致性性质对原本的输入所对应生成的程序以多样本组合的方式进行修改.

本文在常用的程序生成工具上对COTE进行验证. COTE的自动测试方法发现：在验证模型上，大约有39%的输入存在约束不一致性问题. 而COTE的黑盒修复平均为验证修复了42%的问题. 灰盒修复平均为验证修复了33%的问题.

## 2 相关工作

本文关注于测试和修复程序生成模型的方法. 因而，本章描述了与本文研究领域相关的程序生成相关方法和与本文方法相关的机器翻译测试方法.

### 2.1 程序生成

早期的程序生成方法主要基于人工预定义的模板来生成程序[12-15]. 该类方法往往需要人类根据数据集的不同来总结不同的模板以满足程序生成的需要. 其有较高的人工开销，同时具有较差的可移植性.

Ling 等人[3]首次应用了基于深度学习的序列到序列（sequence-to-sequence）框架并把程序当作一种新的自然语言并基于程序分词后所得程序令牌序列来生成代码. 在该方法的基础上一系列方法进一步引入了抽象语法树，实现语法制导的程序生成[4-5, 16-19]. 同时，也有一部分研究者着力于使用大规模预训练的方法实现程序生成[8].

### 2.2 机器翻译测试

现有的机器翻译测试主要分为两类：(1)基于翻

译有效性指标的机器翻译鲁棒性测试;(2)基于蜕变关系的机器翻译测试.

(1)基于翻译有效性指标的机器翻译鲁棒性测试

为了测试翻译的鲁棒性,研究人员探索了测试输入上的干扰如何影响翻译.Heigold等人[20]研究了三种类型的字符级噪声测试输入,这些输入是通过字符交换,单词加扰和字符翻转生成的.他们发现,机器翻译系统对不会对人类构成挑战的具有微小扰动的句子非常敏感.Belinkov和Bisk[21]得出了类似的结论,在上述人工合成的噪声输入的基础上,进一步融入了自然噪声,(自然产生的错误,如错别字和拼写错误)对机器翻译的鲁棒性进行测试.为了拥有更多用于鲁棒性测试的测试用例,Zhao等人[22]提出了使用生成对抗网络(GANs).他们将输入的句子映射到一个潜在的空间,然后将其用于搜索靠近输入的句子作为测试输入.

这些工作针对鲁棒性测试.测试输入是人工合成的错误或自然发生的错误.这些测试的测试预言通常是用于机器翻译的有效性指标(如:BLEU分数).

(2)基于蜕变关系的机器翻译测试

不同于鲁棒性测试,基于蜕变关系的机器翻译测试不依赖于BLEU指标转而依赖一些预定义的蜕变关系.在这类工作当中,不同的工作会使用不同的蜕变关系假设.其主要包括:基于交叉引用的测试模型和基于输入生成的测试模型.

① 基于交叉引用的测试模型

Pesu等人[23]假定同一句子的直接翻译(从源语言到目标语言)和间接翻译(从源语言到中间语言,然后从中间语言到目标语言)应该相同.否则的话,则该方法将其汇报为测试到一个错误.Cao等人[24]则提出了类似的想法,其通过一个测试输入去测试不同翻译系统之间的翻译差别,其中,一个句子在多个翻译器上预期具有相似的翻译.

② 基于输入生成的测试模型

主流方法则通过输入和生成的变异体输入之间的关联关系来对机器翻译系统进行测试[25].他们采用在输入的源语言中,对输入进行变异修改从而生成新句子或短语作为输入的策略.具体来说,给定源语言中的一个输入句子,该类方法会生成一个与其相关的句子或短语,然后将原始的输入句子和其生成的句子(短语)输入到待测机器翻译系统当中.进而,该方法将两者的翻译结果进行比较,以测试是否发现了错误.特别的是,Purity[26]将输入句子分解为短语,并假设每个单独的短语(无上下文)的翻译应与整个句子所提供的上下文中该短语的翻译相似.

与Purity不相同的是,其他基于输入生成的测试模型的方法都采用单词替换的方式用以生成新的输入.换句话说,这些方法通过将原句中的一个词替换为另一个相关词来生成一个新句子作为输入.Gupta等人[27]通过将一个单词替换为另一个具有完全不同含义的单词来测试翻译错误,并期望被替换的单词应和原单词具有不同的翻译.Sun和Zhou[28]通过在“喜欢”或“讨厌”之前的人名替换生成测试输入,他们认为转换前后句子的翻译应该相似.Gupta等人[27]通过将一个单词转换为另一个含义完全不同的单词来测试翻译错误,并期望该转换可以产生不同的翻译.He等人[11]在输入句子中的单词转换为另一个单词的同时保持句子结构的不变,假设输入句子和转换所得句子的翻译也应具有相似的结构.

Sun等人[10,29]通过将一个单词转换为另一个含义相近的单词来测试并修复翻译错误.本文所提出的方法主要基于Sun等人所提出的TransRepair方法[10]和CAT方法[29].

# 3 COTE

本文的目标是测试模型中的约束不一致性问题.如果输入少量的修改,同时对程序生成模型输出的程序代码造成了较大的影响,那么我们认为其为一个约束不一致性问题.具体来说,输入允许将某个词通过同位替换的方式转为相近的词汇,其造成的语义变化应与将其输入到程序生成模型所得的程序语义变化相近.而由于同位替换所造成的语义变化较小,其对应模型输出中程序的语义变换理应较小.

为了解决该问题,本节将介绍所提出COTE方法的概述和其详细的步骤.

## 3.1 方法概述

COTE的大致流程如图1所示.COTE基于以下三个主要步骤自动测试和修复程序生成模型的约束不一致性问题(即,扰动约束).

### 3.1.1 约束测试输入生成

该步骤的目标是生成转换后的句子(测试输入)以用于约束不一致性测试.对于每个输入句子,COTE通过上下文相似的单词替换来进行句子变异,进而生成候选变异体并对候选变异体进行过

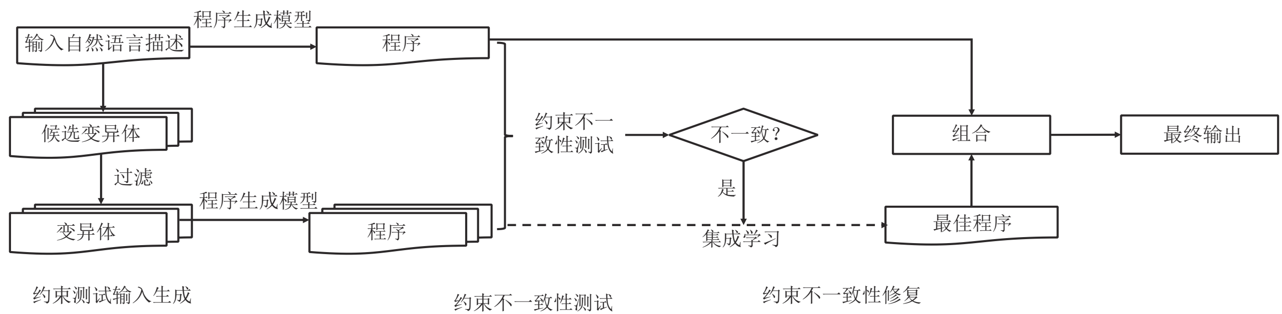


图1 COTE方法测试和修复程序生成系统的流程

滤.然后将过滤后留下的变异体作为被测程序生成系统的最终测试输入.其详细信息在第3.2节中介绍.

3.1.2 约束不一致性测试

该步骤的目标是引入自动化的不一致性测试.自然语言输入和程序输出之间的扰动约束可以被表示为:原始句子对应的程序及其上下文相似的变异体句子对应的程序在变异词未影响的程序部分的应该具有一定程度上的一致性.为了测试该扰动约束,该步骤使用了相似度度量的方法来衡量程序之间的一致性程度并将其作为测试结果.此步骤的详细信息在第3.3节中介绍.本文也探索了四个不同的相似度衡量指标,这些指标在第4.1.2节中进行了详细描述.

3.1.3 约束不一致性修复

该步骤的目标是自动修复约束不一致性问题.COTE应用黑盒和灰盒方法根据多个变异体中的最佳程序来转换为原始句子对应的程序.本文探索了两种选择最佳程序的方法,一种是使用程序生成系统预测概率进行选择的方法,另一种则是使用交叉引用进行选择的方法.此步骤的详细信息将在第3.4节中给出.

### 3.2 约束测试输入生成

为了自动生成约束测试的输入,本节提出了两种不同的方法:1)一种轻量级的相似词语替换方法(其对应的测试和修复方法记为COTE-L);2)一种基于神经网络的同位词语替换方法(需要依赖于高性能计算,其对应的测试和修复方法记为COTE-N).

3.2.1 轻量级的相似词语替换方法

本文将该词语替换方法记为COTE-L.COTE-L的测试输入生成包括以下两个步骤.

(1)语义相似的替换语料构建

要进行语义相似词替换,其关键步骤是找到一个可以替换为其他(相似词)不损害句子结构及含义的词.通过单词替换生成的新句子应该产生与原始句子一致的程序.

一个词可以通过基于上下文词语计算所得词向量表示含义[30].为了测量相似度,本文使用从文本语料库训练的词向量.在COTE-L中,两个词$w_1$和$w_2$之间的词相似度,用sim($w_1$,$w_2$)表示.其由以下公式计算,其中$\boldsymbol{v}_x$表示单词x的词向量.

$$sim(w_1,w_2)=\frac{v_{w_1}v_{w_2}}{|v_{w_1}||v_{w_2}|} \tag{1}$$

为了构建可靠的语义相似词语集,COTE-L采用两个常用的词向量模型并使用它们结果的交集作为语义相似词语集.第一个模型是GloVe[30],它是从Wikipedia 2014数据[31]和GigaWord 5数据[32]训练而来的.第二个模型是SpaCy[33],它是一个在OntoNotes[34]上训练的多任务卷积神经网络模型,其中训练数据包括从电话对话、新闻专线、新闻组、广播新闻、广播对话和博客中收集的数据.当两个词的相似度在这两个模型中都超过0.9时,COTE-L则认为这两个词是语义相似的,并将其放入语义相似词语集中.COTE-L使用这种方法总共收集了131,933个词对.

(2)输入句子变异

下面本文分别介绍词语替换和句子结构过滤.

① 词语替换:对于原始输入句子中的每个单词,COTE-L遍历语义相似词语集以确定是否存在匹配项.如果找到匹配项,COTE-L会将单词替换为其语义相似的单词,并生成一个新的变异输入句子.与原始句子相比,每个变异句子都包含一个替换词.为了减少产生含义不相近的变异体的可能性,该方法在替换的过程中只替换名词、形容词和数词.

② 句子结构过滤:生成的变异句子可能会产生与原来句子不同的含义,因为替换的单词可能不适合变异句子的上下文[10-11].例如,"one"和"another"是

相似的词，但是“a good one”和“a good another”并不具有相近的含义．为了过滤这种类型的变异句子，本文提出应用额外的约束来检查生成的变异体．在这里，COTE-L应用了基于Stanford解析器[35]的结构过滤技术．假设原句为$s=w_1,w_2,\cdots,w_i,\cdots,w_n$，变异的句子是$s'=w_1,w_2,\cdots,w'_i,\cdots,w_n$，其中$s$中的$w_i$替换为$s'$中的$w'_i$．对于每个句子，Stanford解析器输出$l(w_i)$是Penn Treebank项目[36]中的词性标记．如果$l(w_i)\neq l(w'_i)$，COTE-L则从候选变异体句子中删除s'，因为该变异会导致句法结构发生变化，从而造成句子含义发生改变．

### 3.2.2 基于神经网络的同位词语替换方法

基于神经网络的同位词语替换方法（记为COTE-N）和上述轻量级的方法一样，依旧遵循基于单词替换的方法的一般过程：给定输入句子$s$和$s$中的单词$w$，COTE-N识别一组单词（表示为$W_w$），其中每一个单词都可以用来替换$s$中的$w$．对于每个单词$w_f\in W_w$，COTE-N期望用句子$s$中的词语$w_f$替换词语w是同位替换（替换为一个对句子含义的变化产生微小影响的词）．实现同位替换的关键思想是在$s$的上下文中评估$w$和每个候选替换词$w_r\in C_w$之间的上下文感知的语义相似度．

实现同位替换需要两个阶段．在第一阶段，COTE-N为输入句子$s$中的每个单词$w$生成一组单词替换候选$C_w$．第二阶段，COTE-N从候选词$C_w$中识别出最终的词集$W_w$，通过评估$w_r\in C_w$中的每个单词与句子$s$中的单词$w$之间的上下文感知的语义相似度来实现同位替换．

（1）候选词生成

为了实现同位替换，COTE-N先为输入原句$s$中的每个词$w$生成一组候选词$C_w$，其中$W_w\subset C_w$．然后，COTE-N使用词语基于上下文进一步评估上下文感知的语义相似度．

给定一个输入句子$s$，COTE-N在确保$C_w$中的每个词也适合该句子的上下文的情况下选择候选词$C_w$．为了实现该想法，COTE-N使用Transformers（BERT）[37]的双向编码器表示来编码句子上下文并找出哪些单词适合上下文．

图2显示了COTE-N中上下文感知的单词替换模型．作为基于Transformer[38]的预训练模型，对于每一条输入句子BERT根据上下文为句子中的每个单词输出一个实值向量．在进行单词替换时，输入句子中待替换的单词先被特殊标记“[MASK]”所替换．例如，在图2中，输入句子“Within the past things...”中的单词“Within”被“[MASK]”所替换．然后将带有“[MASK]”单词的句子输入到BERT模型，该模型输出一组实值向量．为了进一步实现单词替换，BERT将“[MASK]”这个词的向量输入到一个预先训练好的线性分类器当中，得到一组具有不同预测概率的替换词列表．

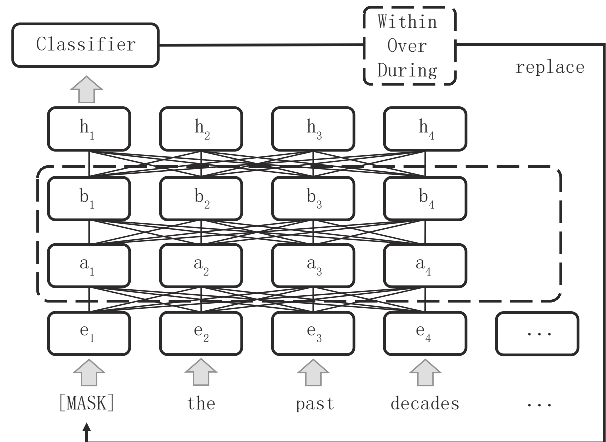


图2 基于BERT的上下文感知的词替换

给定一个单词序列$w_1,w_2,\cdots,w_N$，其中$N$是单词的总数，COTE-N依次使用“[MASK]”对句子中的单词进行替换（每次替换一个单词）并将每个含有“[MASK]”的句子输入到BERT．

BERT的输出是一组向量$\boldsymbol{h}_1,\boldsymbol{h}_2,\cdots,\boldsymbol{h}_N$，它们表示输入词的上下文感知向量表示．然后，一个预训练的线性分类器以“[MASK]”词$\mathrm{h}_{mask}$的向量为输入，输出一组初始候选词$\mathrm{C}_i$．其中的$\mathrm{C}_i$中每个候选词都有一个预测概率．

在预测中，所有候选词的概率之和为1.0．然后，COTE-N使用概率过滤器去除具有较低预测概率的候选词（本文将概率阈值设置为0.05以去除最不合适的候选词）．此外，如果该词与COTE-N所替换的原始词相同，该词也会被去除．最后，COTE-N将剩余的单词作为替换词$w_{mask}$的候选词$C_{wmask}$，其中每个词都可以用来填充输入句子中的“[MASK]”词进行单词替换．

请注意，上述单词生成过程是一种上下文感知的过程，因为BERT可以准确地根据句子的剩余部分含义来预测“[MASK]”位置的合适词语．

（2）语义验证

同位替换旨在为输入句子$s$中的每个单词$w$识别一组单词$W_w$，使得$W_w$中的每个单词仅与$s$上下

文中的$w$略有不同. 在候选词生成中，COTE-N为$s$中的词$w$生成一组候选词$C_w$. 然而，在某些情况下，BERT会预测一个与原始单词具有相似上下文但含义不同的单词. 因此，COTE-N接下来计算一种上下文感知的语义相似度，并用它来去除$C_w$中不合适的词，以识别最终的词集$W_w$.

为了实现COTE-N的目的，COTE-N使用了一种新的基于神经网络的机制来评估上下文感知语义相似性. 计算上下文感知语义相似度的关键步骤是测量原始词与其替换词之间的词向量相似度. COTE-N再次使用BERT来获得句中词对应词向量. 其中，BERT是近年来自然语言处理中使用最广泛的word2vec方法[37,39,40]. 与上一节中在候选词生成中使用BERT不同，COTE-N不给BERT提供含有"[MASK]"的句子，而是直接将整个句子作为其输入. 这样，BERT先将每一个词表示为从训练语料库训练出来的向量. 该向量表示该单个词的语义. BERT进一步考虑句子上下文并计算上下文感知语义向量表示. 这样，COTE-N就得到了句子中每个词的上下文感知的词向量，作为最终的词向量.

**算法1.** 语义验证的计算过程.

输入：输入句子$s$；输入句子$s$中的词语$w$；词语$w$对应的候选词集合$C_w$（候选词生成方法的输出结果）

输出：$W_w$：最后用以同位替换句子$s$中词语$w$的词集合

1. $W_w = \{\}$
2. $\boldsymbol{h}_a = BERT(s,w)$
3. FOR 每个候选词 $w_r \in C_w$ DO
4. $\quad s_r = Replace(s,w,w_r)$
5. $\quad \boldsymbol{h}_b = BERT(s_r,w_r)$
6. $\quad Similarity = CosSim(\boldsymbol{h}_a, \boldsymbol{h}_b)$
7. $\quad$ IF $Similarity >= SThreshold$ THEN
8. $\quad\quad W_w = W_w \cup w_r$
9. $\quad$ END
10. END
11. RETURN $W_w$

然后，COTE-N计算$s$中的单词$w$（记为$\boldsymbol{h}_a$）和$s_r$中的候选词$w_r \in C_w$之间的上下文感知的语义相似度. 通过将$s$中的$w$替换为变异句子中的$w_r$（记为$\boldsymbol{h}_b$），COTE-N进一步使用它们的词向量的余弦相似度CosSim（公式1）进行语义相似度验证.

COTE-N的语义验证的详细过程由算法1展示. 对于一个输入句子$s$，$s$中的一个词$w$，以及通过候选词生成所生成的候选词集$C_w$，COTE-N的目标是得到一个最终的词集$W_w$.

COTE-N先定义一个空集$W_w$用于记录通过验证后的单词（第1行）. 对于$s$中的单词$w$，COTE-N通过BERT进一步提取其词向量$\boldsymbol{h}_a$（第2行）. 接下来，对于每个候选词$w_r \in C_w$，COTE-N用它来替换$s$中的w并生成一个变异句子$s_r$（第3和第4行）. 对于$s_r$中的词$w_r$，COTE-N提取其词向量$\boldsymbol{h}_b$（第5行）. 然后，为了捕捉上述两个词的上下文感知语义相似度，COTE-N计算它们的词向量$h_a$和$h_b$的余弦相似度CosSim（第6行）. 如果相似度高于预定义的阈值SThreshold（根据文献[29]，本文将阈值设置为0.85），则认为其通过了语义验证（第7行）. 因此，COTE-N将单词$w_r$添加到集合$W_w$作为最终生成的单词替换（第8行）. 在COTE-N自动验证完所有单词之后，COTE-N为s中的单词$w$输出最终的替换单词集$W_w$（第11行）. 此过程可以尽可能确保在$s$中用$w_f \in W_w$替换$w$是同位替换.

最后，COTE-N依次使用单词$w_f \in W_w$替换$s$中的单词$w$，并对$s$中的每个单词重复此过程以生成最终的句子集$M_f$. COTE-N将这些最终生成的句子中的每一个都称为变异体[41-42]. 这组生成的变异体进一步用于自动测试（每个变异体与原始句子配对作为一个测试输入对）和修复.

COTE-L和COTE-N共用一套测试和修复方法COTE. 为了便于表示，在本文后续方法部分统一使用COTE来进行介绍. 同时，在本文后续实验中，将使用轻量级的相似词语替换方法的测试和修复方法统称为COTE-L，将使用基于神经网络的同位词语替换方法统称为COTE-N.

### 3.3　约束不一致性测试

为了进行约束不一致性测试，COTE需要对其生成的输入进行验证，即检查是否发现约束不一致性问题. COTE假设输入句子中的未更改部分保有了它们应有的语义充分性. 含义充分性是指生成的程序是否传达了相同的意思，是否有信息丢失、添加或扭曲.

设$t(s)$和$t(s')$是句子$s$和$s'$的对应程序，其中$s'$句子是由$s$句子通过上述单词替换方法替换单词$w$为$w'$得到. 为了实现想要的约束不一致性测试，COTE需要在忽略所替换单词$w$和$w'$对输出程序造成影响的情况下检查两个程序之间的相似性. 然而，要自动化忽略单词$w$和$w'$对输出程序造成的影响并不容易. 因为$w$和$w'$替换可能会改变整个程序，并且准确映射出输入文本中的单词$w$和$w'$所对

应的程序元素并不容易.

**算法2.** 一致性分数计算过程.

输入:$t(s)$:原始输入句子对应的程序;$t(s')$:变异体对应的程序

输出:*ConScore*:$t(s)$和$t(s')$之间的一致性分数

```
1. Bs,Bs' = Wdiff(t(s), t(s'))
2. T_o = {t(s)}
3. T_m = {t(s')}
4. FOR 每个子序列 b_s ∈ B_s DO
5.     r = DeleteSub(t(s), b_s)
6.     T_o = T_o ∪ {r}
7. END
8. FOR 每个子序列 b_s' ∈ B_s' DO
9.     r' = DeleteSub(t(s'), bs')
10.    T_m = T_m ∪ {r'}
11. END
12. ConScore = -1
13. FOR 每个序列 a ∈ T_o DO
14.    FOR 每个序列 b ∈ T_m DO
15.        Sim = ComputeSimilarity(a, b)
16.        ConScore = Max(ConScore, Sim)
17.    END
18. END
19. RETURN ConScore
```

为了绕过这个问题,COTE通过计算$t(s)$和$t(s')$中子句(程序序列的子序列)的相似度并利用多个子句相似度的最大值来近似估计$t(s)$和$t(s')$之间未受影响部分的一致性水平.算法2展示了这个过程.对于$t(s)$和$t(s')$,COTE先应用GNU Wdiff[43]来获得不同程序之间的差异切片(第1行).GNU Wdiff以词为基础比较程序,该方法对于比较只有少量修改的两个相似程序很有效[43].使用Wdiff,两个程序"A B C D F"和"B B C G H F"之间的差异切片分别表示为前一个程序中的"A"和"D"以及后一个程序中的"B"和"G H".

COTE将$t(s)$和$t(s')$的差异切片分别保存到集合$B_s$和$B_{s'}$之中.然后COTE依次从程序中删除一个差异切片的差异部分,每次只删除一个(第5行和第9行).通过这样的方式,COTE会得到一个程序的多个子序列.对于上面的例子,"A B C D F"将有两个子序列:"B C D F"(删除"A")和"A B C F"(删除"D").COTE将$t(s)/t(s')$的这些新的子序列添加到集合$T_o/T_m$中(第6行和第10行).

对于集合$T_o$中的每个子序列元素,COTE将其与集合$T_m$中的每个子序列元素两两之间计算相似性①(第15行).因此,COTE得到$|T_o|*|T_m|$个不同相似度分数,其中$|T_o|$和$|T_m|$分别是集合$T_o$和集合$T_m$的大小.然后,COTE使用其中最高的相似度值作为最终的一致性得分的结果(第16行).

这种策略减少了变异词对程序的影响,并有助于找出不一致性分数的上限.即使相似度最大的两个子序列包含所替换的词对应的程序修改部分,程序其他部分的相似度也比之会更差,因此这种情况不太可能因为含有替换词而导致产生偏差,从而导致误报.

### 3.4 约束不一致性修复

本节先介绍约束不一致性修复过程的整体情况,然后介绍两种不同的修复过程中程序排序策略(基于概率的程序排序和基于交叉引用的程序排序).

#### 3.4.1 整体的修复过程

COTE先修复原始句子对应的程序,然后修复变异体对应的程序,使得其通过COTE的程序不一致性测试.

算法3显示了整体的修复过程.对于已发现存在约束不一致性问题的程序$t(s)$,COTE生成一组变异体并得到它们的程序$t(s_1)$,$t(s_2)$,…,$t(s_n)$.这些变异体及其程序,连同原始句子及其程序,被放入字典T中(第1行).然后,COTE使用预测概率或交叉引用的方式按降序排列T中的元素,并将结果放入OrderedList(第2行).第3.4.2节和第3.4.3节给出了预测概率和交叉引用排名的详细信息.

接下来,COTE应用词对齐来获得$s$和$t(s)$之间的映射词并记为$a(s)$(第3行).词对齐是一种自然语言处理技术,当且仅当两边的词具有程序关系时,它才将两个词连接起来.特别的,COTE采用了基于全匹配的词对齐技术.在词对齐之后,COTE检查是否可以采用OrderedList中的句子对($s_r$, $t(s_r)$)来修复原始程序.COTE按照排名顺序,直到找到一种可以满足约束不一致修复要求的变异程序.

如果$s_r$是原句($s_r == s$),这意味着原句对应的程序被认为是比其他变异体对应的程序更好的选择,所以COTE不会对其程序进行任何修改(第6-8行).否则,COTE对$s_1$和$t(s_1)$进行与$s$和$t(s)$相同的对齐.变量$w_i$, $w^r_i$表示$s$, $s_r$中的替换词,COTE通过对齐得到其对应的程序部分$t(w_i)$,$t(w^r_i)$(第9-12行).

① 本文探讨了四种类型的相似性度量(其详细信息参见章节4.1.2).

由于词对齐不是百分之百准确的,同时由于程序的特殊性,大量相似的输入理应生成相同的程序. 因此,COTE使用了以下规则来完成程序映射:如果所替换的词并没有存在于词对齐列表中,我们默认其生成的程序不需要任何修改(第13-15行).

在修复变异体的程序时(第16行),COTE已知原句的修复结果(第17行),然后检查候选程序方案是否与修复后的原句对应程序一致(第18-20行). 如果不一致,COTE继续尝试选择其他候选者.

3.4.2 基于概率的程序排序

对于一个句子$s$及其变体$S=s_1, s_2, \cdots s_n$, $t(s)$表示为$s$对应的程序,$t(s_i)$表示为变异体$s_i$对应的程序. 该排序方法记录每个$t(s_i)$的程序生成概率,并选择概率最高的变异体作为程序映射候选者. 然后将相应变异体的程序映射回待修复程序,以使用单词对齐生成$s$对应的最终程序.

这是一种灰盒修复方法. 它既不需要训练数据也不需要训练算法的源代码,但需要程序生成系统提供的预测概率. 本文将此称为灰盒修复方法,因为实现者可能将此通常不能访问到的概率信息视为该方法的内部属性.

3.4.3 基于交叉引用的程序排序

对于一个句子$s$及其变体$S=s_1, s_2, \cdots s_n$, $t(s)$表示为$s$对应的程序,$t(s_i)$表示为变异体$s_i$对应的程序. 该方法计算$t(s)$, $t(s_1)$, $t(s_2)$, $\cdots t(s_n)$之间的相似度,并使用与其他程序最近(具有最大平均相似度分数)的程序映射回并修复原始输入对应的程序.

这是一种黑盒修复方法. 它只需要执行被测程序生成系统和该系统的程序输出.

**算法3.** 自动化修复过程.

输入:$s$:一个输入语句;$t(s)$:$s$在程序生成系统中对应的程序;$s_1, s_2, \cdots, s_n$:s通过自动输入生成所生成的变异体;$t(s_1), t(s_2), \cdots, t(s_n)$:变异体所得程序;

输出:*NewTrans*:$s$程序的修复结果

1. $T=\{(s, t(s)), (s_1, t(s_1)), (s_2, t(s_2)), \cdots, (s_n, t(s_n))\}$
2. $OrderedList = Rank(T)$
3. $a(s) = wordAlignment(s, t(s))$
4. $NewTrans = t(s)$
5. FOR 每条输入语句及其对应的输出程序 $s_r$, $t(s_r) \in OrderedList$ DO
6.     IF $s_r == s$ THEN
7.         BREAK
8.     END
9.     $a(s_r) = wordAlignment(s_r, t(s_r))$
10.     $w_i, w^r_i = getReplacedWord(s, s_r)$
11.     $t(w_i) = getTranslatedWord(w_i, a(s))$
12.     $t(w^r_i) = getTranslatedWord(w^r_i, a(sr))$
13.     $t^r(s_r) = mapBack(t(s), t(s_r), s, s_r, a(s), a(s_r))$
14.     IF $isinaligner(w^r_i)$ THEN
15.         $NewTrans = t^r(s_r)$
16.         $s_o, t(s_o) = getRepairedResult(s)$
17.         IF NOT isConsistent($t(s_o)$, $t^r(s_r)$) THEN
18.             CONTINUE
19.         END
20.     END
21.     IF $isTest(s)$ THEN
22.         $s_o, t(s_o) = getRepairedResult(s)$
23.         IF NOT isConsistent($t(s_o)$, $t^r(s_r)$) THEN
24.             CONTINUE
25.         END
26.     END
27.     $NewTrans = t^r(s_r)$
28.     BREAK
29. END
30. RETURN *NewTrans*

# 4 实验验证

## 4.1 实验设置

本节将介绍验证自动测试输入生成、自动不一致性测试和自动不一致性修复的实验设置.

4.1.1 研究问题

本文通过评估COTE生成可用于约束不一致性测试测试输入的能力来开始本文的研究. 因此,本实验首先关注:

**RQ1:COTE方法生成测试输入的准确率如何?** 本文通过随机抽样一些生成的测试输入对并(人工)检查①它们是否有效来回答这个研究问题. 这个问题的答案确保COTE确实生成了适合约束不一致性测试及修复的输入.

鉴于本问题发现了COTE可以生成有效测试输入对的证据,本文进一步将注意力转向COTE在测试约束不一致性问题方面的有效性问题. 因此本文提出了第二个研究问题:

**RQ2:COTE揭示约束不一致性问题的能力如何?** 为了回答RQ2,本文根据相似度指标计算一致

① 人工检查由论文前两个作者进行手动标注并结合讨论完成.

性分数以作为约束不一致性测试标准(用来测试是否存在不一致性问题).为了评估COTE的约束不一致性问题揭示能力,本文通过抽样的方式手动检查测试结果,并将手动检查结果与自动测试结果进行比较.

在研究了该问题之后,本文进一步评估了COTE的修复步骤,以了解它修复约束不一致性问题的能力.因此,本文提出:

**RQ3:COTE对约束不一致性问题的修复能力如何?** 为了回答这个问题,本文评估了COTE修复约束不一致性问题的数量及比例(通过一致性指标和手动检查进行评估).同时,本文还人工检查了由COTE修复的程序,并检查它们是否提高了程序的一致性和可接受性(人工评判的生成质量).

4.1.2 相似度衡量指标

本文探索了四种广泛采用的相似性指标来衡量生成程序的相似度.为了便于说明,本文用$t_1$来表示原始输入的对应的程序;本文使用$t_2$来表示变异体输入的对应的程序.

**基于LCS的相似度衡量指标**.它通过$t_1$和$t_2$之间最长公共子序列(Longest Common Subsequence,LCS)的长度来衡量相似度:

$$M_{LCS}=\frac{len(LCS(t_1,t_2))}{Max(len(t_1),\ len(t_2))} \quad (2)$$

在这个公式中,LCS是一个计算在$t_1$和$t_2$之间以相同顺序出现的最长公共子序列[44]的函数.例如,输入序列"A B C D G H"和"A E D F H R"的LCS是长度为3的"A D H".$len(t_1)$和$len(t_2)$则表示的是序列$t_1$和$t_2$的长度值.

**基于ED的相似度衡量指标**.这个指标的计算基于$t_1$和$t_2$之间的编辑距离(Edit Distance,ED).编辑距离是一种通过计算将一个字符串转换为另一个字符串所需的最小操作数来量化两个字符串的不同程度的方法[45].为了归一化编辑距离,本文使用了之前工作中也采用的以下公式[46-47].

$$M_{ED}=1-\frac{ED(t_1,t_2)}{Max(len(t_1),\ len(t_2))} \quad (3)$$

在这个公式中,ED是一个计算$t_1$和$t_2$之间编辑距离的函数.

**基于tf-idf的相似度衡量指标**.tf-idf(term frequency-inverse document frequency)可用于衡量词频方面的相似性.每个单词$w$都有一个权重因子,根据以下公式计算,其中$C$是文本语料库,$|C|$是句子数,$f_w$是$C$包含$w$的句子的数量.

$$w_{idf}=\log\left(\left(|C|+1\right)/\left(f_w+1\right)\right) \quad (4)$$

本文用词袋模型来表示每个程序[48],这是自然语言处理中常用的一种表示表示.在这个模型中,语法和顺序被忽略,只考虑程序元素出现的频次(即"A B C A"表示为"A":2,"B":1,"C":1,即向量中的[2,1]).向量的每个维度都乘以它的权重$w_{idf}$.本文计算$t_1$和$t_2$的加权向量的余弦相似度(公式1)作为它们最终的基于tf-idf的一致性分数.

**基于BLEU的相似度衡量指标**.BLEU(Bi-Lingual Evaluation Understudy)是一种通过检查机器输出结果与人类给出的参考结果之间的对应关系来自动评估程序生成质量的算法.它也可以用来计算原句对应程序和变异体对应程序之间的相似度.BLEU分数的详细信息、描述和动机可以在程序文献中找到[49].由于篇幅有限,这里只做一个概述.

BLEU先统计句子(在本文中是程序)和句子之间匹配子序列的数量计算精度$p_n$(称为修改后的n-gram精度[49],其中$n$表示子序列长度).例如,在句子"A A B C"($s_1$)和"A B B C"($s_2$)中,$s_2$中有三个2-gram子序列:A B、B B和B C.其中两个与$s_1$中的匹配:A B和B C.因此,$p_2$是2/3.

BLEU分数的计算还需要通过预定义的惩罚因子BP来惩罚过短的程序,如公式5所示.$c$表示$t(s_i)$的长度,$r$是$t(s)$的长度.

$$BP=\begin{cases}1 & if\ c>r\\ e^{(1-\frac{r}{c})} & if\ c\leqslant r\end{cases} \quad (5)$$

BLEU分数最后由公式6计算得到,在此公式中,权重$w_n=1/N$(本文设置$N=4$).

$$BLEU=BP*\exp\left(\sum_{n=1}^{N}w_n log p_n\right) \quad (6)$$

由于BLEU是单向的(即BLEU($s$, $s'$)≠BLEU($s'$,$s$)),因而本文使用其两种分数中较高的分数来衡量$s$和$s'$之间的相似性.这与本文在公式2中的意图是一致的:计算获得一致性度量的上限,从而避免关于因为程序问题而导致的误报.

4.1.3 程序生成系统

本文的实验考虑了常用的CodeGPT程序生成系统.本文使用CodeGPT,因为它是一个在程序生成中常见的开源模型,也是一个高质量的程序生成系统.

本文将CodeGPT迁移到本文所使用的数据集.本文使用默认设置来训练CodeGPT.CodeGPT基于一个程序生成数据集(Concode数据集)进行训练.Concode数据集[50]包含100,000条训练数据,其

中每条数据包括其自然语言及上下文代码．验证数据（帮助调整超参数）也来自Concode数据集，其中包含2，000条输入自然语言描述．我们使用Transformers与Pytorch深度学习库对模型进行训练[51]．

4.1.4　测试集

本文使用来自Concode数据集的验证集[50]作为测试集．该数据集用以测试CodeGPT模型的稳定性．验证集包含不同于训练集中的2，000条数据．本文从中随机选择了500条数据作为最终的测试集．

4.1.5　实验参数设置

COTE-N的实现基于公开可用的标准BERT[51]．BERT模型包含24层、1024大小的隐含层、16个头．该模型在16个TPU芯片上进行了100万步的训练，训练的批次大小为256．

参考TransRepair[10]，本文在测试过程中为每个输入句子生成最多5个有效变异体，在修复过程中为每个输入句子中生成最多16个有效变异体．

本文的实验是在256GB RAM和四个总共包含32个内核的Intel E5-2620 v4 CPU（2.10 GHz）的Ubuntu 16.04上进行的．本文使用的神经网络都是在八块Nvidia Titan RTX（24 GB内存）上训练和预测的．

### 4.2　实验结果

本节回答研究问题对应实验结果．

4.2.1　对输入生成的效果(RQ1)

要回答这个问题，对于本文数据集中的每条输入句子（总共500个），本文分别使用测试和修复算法COTE-N和COTE-L的方法生成变异体．每个变异体与原始句子配对以形成测试输入．然后记录数量（即测试输入的总数）及有效性（即测试输入中的变异和原始句子是否合适于用作约束不一致性测试）．

数量：对于501个输入句子，COTE-N通过同位替换总共生成了2，362个变异体，这些变异体可以与原始句子配对，作为程序生成模型的测试输入．对于COTE-L，它则产生163个测试输入．

有效性：有效性衡量生成的测试输入句子符合约束不一致性要求的比例．如果生成的变异体语法正确；语义上合理；产生与原始句子语义相同的程序（不包含所替换的单词所对应的程序部分），本文则认为测试输入是有效的．本文为COTE-N和COTE-L分别随机抽取100个输入用例．

本文的人工检查表明COTE-N生成的测试输入中有80％是有效的．由于数据集是从代码注释中收集的，因而其中存在很多不确定性描述，为测试输入生成带来了困难．如果输入描述越接近于常见自然语言，其对应的输入生成的有效率越高．对于COTE-L，其测试输入的75％是有效的．这些结果表明COTE-N相比于轻量级的COTE-L生成了更多有效的测试输入．

总的来说，对于RQ1，本文有以下结论：

结论1(RQ1)：COTE-L可以生成163个测试输入，其生成输入的有效率为75％．COTE-N可以生成2，362个测试输入，其生成输入的有效率为80％．

4.2.2　COTE在揭示约束不一致性问题方面的能力(RQ2)

在本实验中，与CAT[29]相同，本文设定四个相似度度量指标LCS、ED、tf-idf和BLEU的阈值为0.963、0.963、0.999、0.906，该阈值根据人工标注的数据通过搜索的方法得到．对于COTE-N和COTE-L生成的每个测试输入，本文将其输入CodeGPT当中以获取其对应程序，然后应用相似度指标及阈值来自动化确定其是否存在约束不一致性问题（低于我们设定的阈值）．

表2显示了COTE-N和COTE-L报告的每个相似性指标的对应测试到的问题数量．对于COTE-L而言，在CodeGPT上不一致的生成程序个数和相比于总输入用例数量的百分比分别是LCS：75（46％）；ED：75（46％）；tf-idf：77（47％）；BLEU：73（45％）．因此，总体而言，大约五分之二的程序低于本文选择的一致性阈值．本文观察到COTE-N在报告的问题数量上明显优于COTE-L．对于COTE-N而言，在CodeGPT模型上不一致的程序个数和相比于总输入用例数量的百分比分别是LCS：928（39％）；ED：928（39％）；tf-idf：926（39％）；BLEU：911（39％）．平均而言，COTE-N在CodeGPT模型上测试到的问题比COTE-L多十余倍，但是所占输入用例数量的百分比有所下降．

为了验证测试结果的有效性，本文统一从每种方法的测试结果中采样50个报出约束不一致性问

**表2　所测试出的约束不一致性问题数量(RQ2)**

| | 指标 | COTE-L | COTE-N |
|---|---|---|---|
| CodeGPT | LCS | 75(46％) | 928(39％) |
| | ED | 75(46％) | 928(39％) |
| | TFIDF | 77(47％) | 926(39％) |
| | BLEU | 73(45％) | 911(39％) |

题的输入用例. 对于每个采样的测试输入及其程序生成结果，本文手动检查测试输入及其输出程序是否存在不一致的问题.

人工检查结果表明，对于COTE-L，其可以以72%的准确率找出CodeGPT中所包含的约束不一致性问题. COTE-N同样以72%的准确率找出CodeGPT中所包含的约束不一致性问题. 这些结果表明COTE-N报告的问题的有效性与COTE-L相似. 本文进一步分析了其中假阳性的情况，发现假阳性的出现有多种原因，举例来说：(1)测试输入生成出错；(2)相似性度量指标不准确(由于程序生成模型生成的程序大多较短，因而即使一些小变化也会影响度量指标的准确率. 举例来说，LCS指标认为两个语义相同的程序"return createRefactoringWizardDialog ( ) ;"和"return (createRefactoringWizardDialog ( ) ) ;"的相似度不高). 总的来说，本节实验从自动化和人工验证两个角度说明了COTE的有效性；并且，本文的人工检查表明COTE-N报告的问题的有效性类似于COTE-L报告的问题.

结论2(RQ2)：在测试CodeGPT模型中的约束不一致性问题时，COTE-L可以测试到平均75个约束不一致性问题，而COTE-N则可以多测试到十倍多的约束不一致性问题. COTE-L的测试准确率为72%，COTE-N和COTE-L具有类似的效果，其准确率也为72%.

#### 4.2.3 COTE在修复约束不一致性问题方面的能力(RQ3)

为了便于比较，本文让COTE-N和COTE-L修复同一组测试到的问题，即由COTE-N或COTE-L测试到的问题. 对于每个问题，本文让COTE-N和COTE-L在原输入句和测试用例中的变异体上都生成最多16个变异体(第4.1.5节介绍的修复变异数上限)进行自动修复.

为了回答RQ3，本文先展示了通过相似度度量计算得到的已修复问题的数量. 最后，本文展示了通过人工检查所生成的程序的修复效果.

通过相似性度量访问的修复效果：修复由COTE-L和COTE-N测试到的问题的结果分别显示在表3和表4中. 每个单元格根据相似性指标显示已修复的问题数量，以及已修复问题的比例(相对于COTE所测试到的问题总数).

从该数据可以得出，COTE-L在CodeGPT上以黑盒/灰盒修复的形式可以修复平均51%/67%的由COTE-L测试到的约束不一致性问题. 而对于

**表3 修复COTE-L所测试出约束不一致性问题的数量和比例(RQ3)**

| 方法 | 指标 | 基于预测概率 | 基于交叉引用 |
|---|---|---|---|
| COTE-L | LCS | 41(55%) | 54(72%) |
| | ED | 41(55%) | 53(71%) |
| | TFIDF | 39(51%) | 55(71%) |
| | BLEU | 32(44%) | 39(53%) |
| COTE-N | LCS | 37(49%) | 41(55%) |
| | ED | 37(49%) | 41(55%) |
| | TFIDF | 38(49%) | 41(53%) |
| | BLEU | 39(53%) | 43(59%) |

**表4 修复COTE-N所测试出约束不一致性问题的数量和比例(RQ3)**

| 方法 | 指标 | 基于预测概率 | 基于交叉引用 |
|---|---|---|---|
| COTE-L | LCS | 37(4%) | 38(4%) |
| | ED | 37(4%) | 38(4%) |
| | TFIDF | 55(6%) | 58(6%) |
| | BLEU | 20(2%) | 20(2%) |
| COTE-N | LCS | 308(33%) | 378(41%) |
| | ED | 308(33%) | 377(41%) |
| | TFIDF | 308(33%) | 398(43%) |
| | BLEU | 302(33%) | 397(44%) |

由COTE-N测试到的问题，其修复效果则降低至4%/4%. 本文观察到COTE-N修复的由COTE-L测试到的问题比相比于COTE-L偏低(其修复了平均50%和56%的问题). 在由COTE-N测试到的问题上，其比COTE-L多，以黑盒修复的方式来说，COTE-N修复了42%的报告问题，而COTE-L仅修复了4%. 总的而言，在各自方法所测试到的问题上，其自身方法都比其他方法效果好. 在不同的方法测试到的问题上，COTE-N方法则效果较为稳定，而COTE-L方法在COTE-N所测试到的问题上效果较差.

修复程序的有效性：本文手动检查了LCS相似性指标上使用交叉引用的方式对CodeGPT模型上由COTE-N所测试到的问题的约束不一致性问题修复结果. 目的是检查基于相似度度量的问题修复是否确实提高了生成程序的一致性. 特别的，本文选择COTE-N所测试到的问题是因为其问题数量和种类较多.

在验证COTE修复生成程序约束不一致性问题的基础上，本文验证了感知扰动约束的最终目标(提高生成程序可接受性的能力). 可接受性表示人工检测对程序生成效果的评价. 在很多情况下，即

使程序生成没有生成正确的程序，其提供的参考程序也有效地辅助了开发人员的开发．因此，人工检查考虑两个方面：修复前后生成程序的一致性；修复前后的程序可接受性．对于每个维度，本文设置了三个标签“提升”、“不变”和“下降”．对于一致性来说，评价标准为：程序语义的相似性：如果修复后程序语义比修复前更接近于其相应不一致性问题中另一个程序的语义，则认为其修复的一致性提高；命名相似性：如果修复后程序命名（包括变量、函数名等）比修复前更接近于其相应不一致性问题中另一个程序命名，则认为其修复的一致性提高；对于以上两种相似性，如果判断结果都是一致性提高或下降，则最终结果为一致性提高或下降，否则则为不变．对于可接受性来说，其评价标准为：语义正确性：如果修复后程序语义比修复前更接近于自然语言描述中的语义，则认为其修复的可接受性提高；如果其修复后程序语义和原程序语义相近，则认为其可接受性不变；如果不是前两者，则认为其可接受性降低．本文随机抽样了50个所报告的修复（由于COTE-L的修复数量不够，因而对于COTE-L本文只抽样了38个修复）．对于可接受性而言，本文检查了原始输入生成的程序和变异体生成的程序的修复，因此需要进行100/76（COTE-N/COTE-L）次人工检查．

表5显示了实验结果．从该表中，为了提高一致性，COTE-N的修复成功地提高了45（90％）个生成程序的一致性，其余5（10％）个生成程序的一致性保持不变．而对于COTE-L，它的修复提高了36（95％）个生成程序的一致性，1个（3％）生成程序的一致性保持不变，1个（3％）生成程序的一致性有所下降．

**表5　关于问题修复的人工检查结果(RQ3)**

| 方法 | 角度 | 提升 | 不变 | 下降 |
|---|---|---|---|---|
| COTE-L | 一致性 | 36 | 1 | 1 |
| | 可接受性 | 9 | 61 | 6 |
| COTE-N | 一致性 | 45 | 5 | 0 |
| | 可接受性 | 30 | 43 | 27 |

为了验证生成程序的可接受性，COTE-N提高了30个（30％）生成程序可接受性，降低了27（27％）个生成程序的可接受性．COTE-L提高了9个（24％）生成程序的可接受性，但导致6（16％）个程序的可接受性下降．特别的，在标记数据的过程中，本文发现该程序可接受性的提升依赖于原程序生成技术的生成效果．如果程序生成技术对一条输入句子及其测试输入所对应句子都具有效果较好的生成效果，则其可接受性的提升比例会较大，否则，可接受性的提升比例会较小．

结论3（RQ3）：在各自方法所测试到的问题上，其自身方法都比其他方法效果好．在不同的方法测试到的问题上，COTE-N方法则效果较为稳定，而COTE-L方法在COTE-N所测试到的问题上效果较差．后续的人工检查表明，COTE-L/COTE-N修复后的程序在95％/90％的情况下提高了一致性（降低了3％/0％），在24％/30％的情况下具有更好的可接受性（16％/27％更差）．

## 5　有效性威胁

对外部有效性的威胁：主要在于评估数据集和本文所使用的模型．首先，虽然本文所提出的方法适用于不同的数据集和翻译模型，但到目前为止，本文仅在相同Concode数据集和CodeGPT[8]生成模型上进行实现并评估．因此，未来的工作需要了解在其他数据集和模型上的性能．其次，虽然本文只使用BERT模型[37]来提取上下文感知词嵌入，但本文所提出的方法仍然与其他上下文感知的词嵌入模型兼容（例如，Roberta[52]）．这是未来有待探索的工作．

对内部有效性的威胁：主要在于本文所使用的人工评估及相似度衡量指标．在人工评估方面，为了减少比较不同方法时的偏差，本文将句子/测试输入随机化，以保证每个句子所属的方法对标记者来说是未知的．同时，本文的数据标注也通过多人的讨论达成一致，以降低标注偏差．在相似度衡量指标方面，本文中约束不一致性测试所使用的衡量指标在很多情况下并非是最佳的．其根本原因在于本文中的衡量指标使用词语相似度来衡量程序语义的变化．然而，不同的程序结构有时也具有相同的程序语义，这可能导致其在衡量程序语义上的偏差．该问题存在一些可以优化的策略：(1)使用代码克隆检测的方法作为衡量指标，该方法可以考虑多种类型的代码克隆，提升程序语义衡量的精度，从而提升自动测试的效果，(2)使用代码向量表示的方法，将代码的语义转换为高维空间的向量，并根据该向量的相似度替代该衡量指标．这两种方法都可以使得自动测试的效果获得提升，但是由于Ⅳ型代码克隆检测及代码语义表示仍旧是当今研究中的两大挑战，并且本文所使用的衡量指标也有一定有效性（人

工标注实验说明了其有效性),本文将该探索留作未来工作的一部分.

## 6 总 结

本文研究了程序生成系统中输入自然语言和输出程序之间应该满足的扰动约束.本文提出了COTE,这是第一种基于自动化测试和修复程序生成模型的方法.COTE接受一个句子作为输入并应用上下文相似的变异对其进行微小改动,用以测试程序生成模型.约束不一致性测试是通过比较原始句子对应程序和变异句子对应程序来进行的.为了判断是否满足扰动约束(一致性),COTE计算生成程序所得子序列的相似度.当输入未更改部分对应的程序相似度低于阈值时,COTE认为这是一个约束不一致性问题.进而,COTE通过参考多个变异句子的程序对原始生成的程序以多样本组合的方式自动化修复约束不一致性问题.实验结果表明,COTE可以修复约束不一致性问题进而提升程序生成技术所生成程序的可接受性.

## 参 考 文 献

[1] Fan X, Cao J, and Mao H. A survey of mobile cloud computing. ZTE Communications, 2011, 9(1): 4-8

[2] Bohu L, Lin Z, and Xudong C. Introduction to cloud manufacturing. ZTE Communications, 2020, 8(4): 6-9

[3] Ling W, Blunsom P, Grefenstette E, Hermann K. M, Kočiský T, Wang F, and Senior A. Latent Predictor Networks for Code Generation//Proceedings of the Annual Meeting of the Association for Computational Linguistics, Berlin, Germany, 2016: 599-609

[4] Dong L. and Lapata M. Language to Logical Form with Neural Attention//Proceedings of the Annual Meeting of the Association for Computational Linguistics, Berlin, Germany, 2016: 33-43

[5] Yin P. and Neubig G. A Syntactic Neural Model for General-Purpose Code Generation//Proceedings of the Annual Meeting of the Association for Computational Linguistics, Vancouver, Canada, 2017: 440-450

[6] Sun Z, Zhu Q, Mou L, Xiong Y, Li G, and Zhang L. A grammar-based structural CNN decoder for code generation//Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, USA, 2019: 7055-7062

[7] Sun Z, Zhu Q, Xiong Y, Sun Y, Mou L, and Zhang L. Treegen: A tree-based Transformer architecture for code generation//Proceedings of the AAAI Conference on Artificial Intelligence, New York, USA, 2020: 8984-8991

[8] Lu S, Guo D, Ren S, Huang J, Svyatkovskiy A, Blanco A, Clement C, Drain D, Jiang D, Tang D, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation//Proceedings of the 35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), 2021

[9] Zhu Q, Sun Z, Zhang W, Xiong Y, and Zhang L. Grape: Grammar-preserving rule embedding//Proceedings of the International Joint Conference on Artificial Intelligence, Vienna, Austria, 2022

[10] Sun Z, Zhang J. M, Harman M, Papadakis M, and Zhang L. Automatic testing and improvement of machine translation//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Korea, 2020: 974-985

[11] He P, Meister C, and Su Z. Structure-invariant testing for machine translation//Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering, Seoul, Korea, 2020: 961-973

[12] Zettlemoyer L. and Collins M. Online learning of relaxed ccg grammars for parsing to logical form//Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Prague, Czech Republic, 2007: 678-687

[13] Zettlemoyer L. S. and Collins M. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars//Proceedings of the 21th Conference on Uncertainty in Artificial Intelligence, Edinburgh, UK, 2005: 658-666

[14] Kushman N. and Barzilay R. Using semantic unification to generate regular expressions from natural language//Proceedings of the North American Chapter of the Association for Computational Linguistics, Atlanta, USA, 2013: 826-836

[15] Wang A, Kwiatkowski T, and Zettlemoyer L. Morpho-syntactic lexical generalization for CCG semantic parsing//Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qatar, 2014: 1284-1295

[16] Rabinovich M, Stern M, and Klein D. Abstract Syntax Networks for Code Generation and Semantic Parsing//Proceedings of the Annual Meeting of the Association for Computational Linguistics, Vancouver, Canada, 2017: 1139-1149

[17] Dong L. and Lapata M. Coarse-to-Fine Decoding for Neural Semantic Parsing//Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia, 2018: 731-742

[18] Yin P. and Neubig G. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation//Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Brussels, Belgium, 2018: 7-12

[19] Yin P, Zhou C, He J, and Neubig G. StructVAE: Tree-structured Latent Variable Models for Semi-supervised Semantic Parsing//Proceedings of the 56th Annual Meeting of

the Association for Computational Linguistics（Volume 1：Long Papers），Melbourne，Australia，2018：754-765

[20] Heigold G，Varanasi S，Neumann G，and Genabith J. How robust are character-based word embeddings in tagging and MT against wrod scramlbing or randdm nouse? //Proceedings of the 13th Conference of the Association for Machine Translation in the Americas，Boston，USA，2018：68-80

[21] Belinkov Y. and Bisk Y. Synthetic and natural noise both break neural machine translation//Proceedings of the International Conference on Learning Representations，Canada，2018

[22] Zhao Z，Dua D，and Singh S. Generating natural adversarial examples. CoRR，abs/1710.11342，2017. arXiv：1710.11342

[23] Pesu D，Zhou Z. Q，Zhen J，and Towey D. A monte carlo method for metamorphic testing of machine translation services//Proceedings of the 3rd IEEE/ACM International Workshop on Metamorphic Testing，Gothenburg，Sweden，2018，2018：38-45

[24] Cao J，Li M，Li Y，Wen M，Cheung S.-C，and Chen H. SemMT：A semantic-based testing approach for machine translation systems. ACM Transactions on Software Engineering and Methodology，2022，31(2)：1-36

[25] Zhang J. M，Harman M，Ma L，and Liu Y. Machine learning testing：Survey，landscapes and horizons. IEEE Transactions on Software Engineering，2020，48(1)：1-36

[26] He P，Meister C，and Su Z. Testing machine translation via referential transparency//Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering，Madrid，Spain，2021：410-422

[27] Gupta S，He P，Meister C，and Su Z. Machine translation testing via pathological invariance//Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering，USA，2020：863-875

[28] Sun L. and Zhou Z. Q. Metamorphic testing for machine translations：Mt4mt//Proceedings of the 25th Australasian Software Engineering Conference，Adelaide，Australia，2018：96-100

[29] Sun Z，Zhang J. M，Xiong Y，Harman M，Papadakis M，and Zhang L. Improving machine translation systems via isotopic replacement//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering，Pittsburgh，USA，2022

[30] Pennington J，Socher R，and Manning C. Glove：Global vectors for word representation//Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing，Doha，Qatar，2014：1532-1543

[31] Wikipedia，Wikipedia. https://dumps.wikimedia.org/，2014

[32] ParkerRobert，GraffDavid，KongJunbo，Ke Chen，MaedaKazuaki，English Gigaword Fifth Edition. https://catalog.ldc.upenn.edu/LDC2011T07，2011

[33] SpaCy，SpaCy. https://spacy.io/，2019

[34] WeischedelRalph，PalmerMartha，MarcusMitchell，HovyEduard，PradhanSameer，RamshawLance，XueNianwen，TaylorAnn，KaufmanJeff，FranchiniMichelle，El-BachoutiMohammed，BelvinRobert，HoustonAnn，OntoNotes，https://catalog.ldc.upenn.edu/LDC2013T19，2013

[35] Manning C. D，Surdeanu M，Bauer J，Finkel J，Bethard S. J，and McClosky D. The Stanford CoreNLP natural language processing toolkit//Proceedings of the Association for Computational Linguistics（ACL）System Demonstrations，Baltimore，USA，2014：55-60

[36] Taylor A，Marcus M，and Santorini B. The penn treebank：An overview. Treebanks：Building and using parsed corpora，2003：5-22

[37] Devlin J，Chang M，Lee K，and Toutanova K. BERT：pre-training of deep bidirectional transformers for language understanding//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics：Human Language Technologies，Minneapolis，USA，2019：4171-4186

[38] Vaswani A，Shazeer N，Parmar N，Uszkoreit J，Jones L，Gomez A. N，Kaiser Ł，and Polosukhin I. Attention is all you need//Proceedings of the Advances in Neural Information Processing Systems，Long Beach，USA，2017：6000-6010

[39] Li X，Bing L，Zhang W，and Lam W. Exploiting bert for end-to-end aspect-based sentiment analysis//Proceedings of the 5th Workshop on Noisy User-generated Text，Hong Kong，China，2019：34-41

[40] Zhou W，Ge T，Xu K，Wei F，and Zhou M. Bert-based lexical substitution//Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics，Florence，Italy，2019：3368-3373

[41] Jia Y. and Harman M. An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering，2011，37(5)：649-678

[42] Papadakis M，Kintis M，Zhang J，Jia Y，Le Traon Y，and Harman M. Mutation testing advances：An analysis and survey. Advances in Computers，2019，112：275-378

[43] Foundation F. S，wdiffGnu，https://www.gnu.org/software/wdiff/，2019

[44] Hunt J. W. and Szymanski T. G. A fast algorithm for computing longest common subsequences. Communications of the ACM，1977，20(5)：350-353

[45] Ristad E. S. and Yianilos P. N. Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence，1998，20(5)：522-532

[46] Gu J，Wang Y，Cho K，and Li V. O. Search engine guided neural machine translation//Proceedings of the 32th AAAI Conference on Artificial Intelligence，New Orleans，Louisiana，USA，2018：5133-5140

[47] Zhang J，Utiyama M，Sumita E，Neubig G，and Nakamura S. Guiding neural machine translation with retrieved translation pieces//Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics：Human Language Technologies，New Orleans，USA，2018：1325-1335

[48] Zhang Y，Jin R，and Zhou Z.-H. Understanding bag-of-words model：A statistical framework. International Journal of Machine Learning and Cybernetics，2010，1(1)：43-52

[49] Papineni K，Roukos S，Ward T，and Zhu W.-J. BLEU：A

method for automatic evaluation of machine translation//Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, Philadelphia, USA, 2002: 311-318

[50] Iyer S, Konstas I, Cheung A, and Zettlemoyer L. Mapping language to code in programmatic context//Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 2018: 1643-1652

[51] Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, Cistac P, Rault T, Louf R, Funtowicz M, Davison J, Shleifer S, Platen P, Ma C, Jernite Y, Plu J, Xu C, Scao T. L, Gugger S, Drame M, Lhoest Q, and Rush A. M. Transformers: State-of-the-art natural language processing//Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Online: Association for Computational Linguistics, 2020: 38-45

[52] Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, and Stoyanov V. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019

**SUN Ze-Yu,** Ph. D., assistant researcher. His research interests include software testing and code generation.

**ZHANG Jie,** Ph. D., assistant professor. Her main research interests are software testing, machine learning trustworthiness, machine learning fairness, code generation, and program analysis.

**XIONG Ying-Fei,** Ph. D., associate professor with tenure . His research interests include software engineering and programming language.

**HAO Dan,** Ph. D., professor. Her current research interests include software testing and debugging.

**ZHANG Lu,** Ph. D., professor. His current research interests include software analysis, software testing, and software maintenance and evolution.

**Background**

Programming is important in software development. In order to improve the efficiency of programming, many researchers are devoted to program generation, code search, and code completion.

As a representative task in these areas, program generation has attracted extensive attention from both academia and industry. Given an input natural language description, a program generation system generates the target program automatically.

Existing approaches apply deep neural-based machine translation technology to convert an input natural language description into the target program. However, the program generated by program generation is different from the natural language generated by machine translation. The program contains a lot of constraints that are not included in natural language: (1) The grammar constraint. The generated program must satisfy the predefined grammar, otherwise, its compilation should fail; (2) The semantics constraint. The variable names and function names in the generated program must satisfy a property that when they are renamed, the semantics should not be changed, otherwise their semantics must be incorrect; (3) The task-specific constraint. In the program generation task, the generated program and the input natural language description should satisfy the perturbation constraint. If the natural language description is perturbed without hurting the semantics, the structure of the generated program should be unchanged, otherwise, the program may be incorrect.

Encoding the constraints into program generation may improve the performance. However, it is a very challenging problem. Although the existing work has focused on grammar encoding and semantics encoding, their performance is still not ideal.

To tackle this problem, this paper focuses on the task-specific constraint in program generation.

This paper represents a kind of task-specific constraint as a metamorphic constraint. Based on this metamorphic constraint, this thesis proposes a detection approach, COTE, to automatically detects the corresponding issues in a program generation system. Then, COTE uses an ensemble learning-based approach to fix the detected issues. The proposed COTE was evaluated on a widely used program generation tool. The experimental results on CodeGPT show the effectiveness of COTE.