软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006200]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

程序智能合成技术研究进展*

顾 斌[1], 于 波[1], 董晓刚[1], 李晓锋[1], 钟睿明[1], 杨孟飞[2]

[1](北京控制工程研究所,北京 100190)

[2](中国空间技术研究院,北京 100094)

通讯作者:李晓锋,E-mail: li_x_feng@126.com

摘 要: 近年,随着信息技术快速发展,软件重要性与日俱增,极大地推动了国民经济的发展.然而,由于软件业务形态越来越复杂和需求变化越来越快,软件的开发和维护成本急剧增加,迫切需要探索新的软件开发模式和技术.目前各行业在软件活动中积累了规模巨大的软件代码和数据,这些软件资产为软件智能化开发建立了数据基础.与此同时,深度学习等人工智能技术在多领域取得的成功应用,促使研究者考虑使用智能化技术与软件工程技术相结合解决程序自动生成问题.程序智能合成方法是程序自动生成的新途径,通过实现软件开发过程自动化,提高软件生产率.本文首先分析了软件工程发展历程及挑战.进而研究了智能化程序合成技术领域的研究布局,以及各方法的优势和劣势.最后,对程序智能合成技术加以总结并给出了未来研究建议.

关键词: 软件工程;程序合成;软件开发

中图法分类号: TP311

中文引用格式: 顾斌,于波,董晓刚,李晓锋,钟睿明,杨孟飞.程序智能合成技术研究进展.软件学报. http://www.jos.org.cn/1000-9825/6200.htm

英文引用格式: Gu B, Yu B, Dong XG, Li XF, Zhong RM, Yang MF. Intelligent program synthesis techniques: literature review. Ruan Jian Xue Bao/Journal of Software, 2020(in Chinese).http://www.jos.org.cn/1000-9825/6200.htm

Intelligent Program Synthesis Techniques: Literature Review

GU Bin[1], YU Bo[1], DONG Xiao-Gang[1], LI Xiao-Feng[1], ZHONG Rui-Ming, YANG Meng-Fei[2]

[1](Beijing Institute of Control Engineering, Beijing 100190, China)

[2](China Academy of Space Technology, Beijing 100094, China)

Abstract: In recent years, with the rapid development of the information technology, the importance of software is increasing day by day, which greatly promotes the development of economic society. However, in the face of more and more complex business forms and faster and faster demand changes, the cost of software development and maintenance has increased dramatically, so it is necessary to study new technologies and explore new software development models, large scale software codes and data are accumulated in specific fields in software activities throughout the whole life cycle. These software assets establish a data base for software intelligent development. At the same time, AI technologies such as deep learning have been successfully applied in many fields, which prompted researchers to consider using the combination of intelligent technology and software engineering technology to solve the problem of automatic program generation: intelligent program synthesis. This method not only realizes the automation of software development process and improves software productivity, but also enables software to have the function of intelligent change with the change of environment and demand, greatly reducing maintenance costs. In this paper, we start from exploring the development process and challenges of software engineering. Then we study the research layout in the field of intelligent software synthesis technology, as well as the advantages and disadvantages of each method. Finally, we summarize the intelligent program synthesis technology in a comparative perspective and give suggestions for future research.

* 基金项目: 国家自然科学基金(61632005)
Foundation item: National Natural Science Foundation of China (61632005)
收稿时间: 2020-07-10; 修改时间: 2020-08-20; 采用时间: 2020-10-14; jos 在线出版时间: 2020-12-02

Key words: software engineering; program synthesis; software development

随着信息技术的不断发展,软件在国民经济的各个领域发挥的作用越来越重要.然而,面对不断变化的软件需求,软件的功能日趋复杂,软件的规模日益增大,软件的开发和维护重要性日益明显.传统的软件开发模式已经无法满足当下及未来软件的开发要求,亟需探索新的、革命性的软件开发方法.

近年来,人工智能技术得到快速发展,并已成功应用于多个领域,如语音识别、数据挖掘、图像识别、自然语言处理等.受此启发,应用人工智能技术与软件工程技术相结合来探索新的软件开发方法开始逐步受到关注.机器学习等人工智能技术可以学习存在于样本数据间的内在规律[1],实现像人类一样识别、分析和学习各种数据.如何利用诸如机器学习等人工智能技术来学习并利用这些资产,从而提高软件开发的智能化程度,是目前软件工程领域关心的热点问题.

程序合成是指根据用户意图自动地构造计算机程序[2],是程序自动生成的一种解决途径,最终目标是实现计算机自主编程.与模型驱动通过模型转换构造程序不同,程序合成强调是从底层编程语言的可能程序中查找满足用户意图的程序.传统程序合成方法如演绎与归纳等是基于严格的形式规约,然而编写形式规约的难度并不低于直接编写程序,并不能将编程人员从枯燥重复的开发活动中解放出来.程序智能合成指的是在传统的程序合成技术基础上,采用机器学习等人工智能技术,利用已有的大量代码知识自动合成满足用户意图的程序.

本文综述了程序智能合成技术,结合人工智能技术和软件工程技术探索新的软件开发方法,并展望未来的技术挑战和趋势.首先介绍了软件研制的发展阶段和当前状态;进而通过文献分析展示长期以来软件工程领域对程序合成技术的研究布局;根据现有程序智能合成研究工作的分析和研究,阐述目前主流的技术方法;最后,探讨程序智能合成技术面临的挑战和发展趋势.

1 软件研制发展阶段

1946年第一台电子数字计算机诞生,软件开始正式走进人类生活.软件的发展已逾70年[3],在这期间人们对于软件系统的需求急剧上升,科研人员从未停止对于软件开发方法的探索.研究主要可分为四大阶段:个性化研制阶段、软件工程阶段、软件自动生成阶段、全智能生成阶段(如图1所示).

个性化研制阶段:在计算机发展的初期,软件功能相对简单,规模较小.直到20世纪70年代,软件基本上采用“软件作坊”[4]的模式进行开发,这种开发模式耗费大量的人力,无法应对不断增加的软件需求和软件数量,出现了“软件危机”出现.如何更快更好地开发软件是解决软件危机所必须解决的难题.

软件工程阶段:为了解决软件危机,“软件工程”这一概念[5]于1968年在德国举行的北约会议上首次被提出,人们考虑用工程的思想进行软件开发.软件开发开始分为需求分析、设计、编码和测试几个阶段.用户提出需求之后,软件开发人员先进行软件需求分析,编写需求规约,然后经过概要、详细设计之后进行编码和各种测试[7].至此软件开发走向“工程”和“协同”.这期间出现了软件复用、计算机辅助软件工程、软件再工程等方法.

软件自动生成阶段:随着软件规模与复杂度不断提高,软件的开发效率和质量仍然难以满足经济社会快速发展对软件开发的需求.实现从需求规约自动构造软件代码对于提高软件生产率至为重要,经过多年研究,出现了演绎综合、程序转换、归纳综合以及过程实现等诸多软件自动生成方法,但这些方法发展较为缓慢.机器学习等人工智能技术加速了软件自动化的进程[6],学习并利用各行业已储备的大量软件资产,实现智能化的软件开发是软件自动生成的重要目标.

软件全智能生成阶段:人工智能等技术高度发达,能够实现从自然语言描述的任务需求直接生成代码程序,计算机彻底取代程序员的工作,实现完全自主编程.

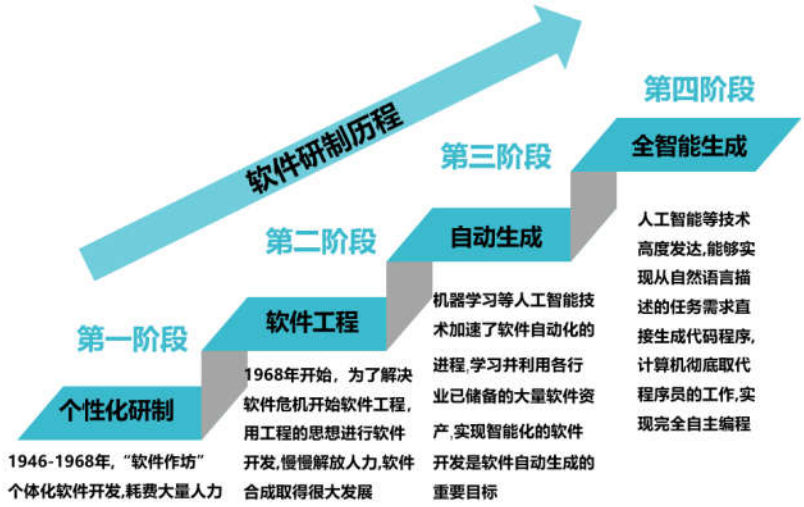

Fig.1 The four stages of sofeware development

图 1 软件研制的四个阶段

可以看出,当前软件研制活动还处于第二阶段,正向第三阶段迈进.随着我国软件规模的不断发展和增大,软件开发复杂度和质量要求的不断提高,传统软件方法很难适应当前及未来软件系统的开发要求.迫切需要探索新型软件开发方法和模式来取代传统的软件开发方法.

从软件工程方法学的角度看,其处理对象是千姿百态的客观世界,因而其行为必然是大量知识密集型的[8].目前各行各业积累了大量可复用的软件资产供人类学习及应用.人工智能技术善于解决知识的表示与使用问题,而这正是知识密集型的软件工程活动需要依靠的技术.人工智能技术能够对知识进行智能化表示和管理,提取软件系统中适应于变化的部分并将其封装在规则集中,实现整个规则库与推理机完全分离.这样,从最基本的非启发式规则入手,简化将要引入启发式规则和启发式推理机的系统的建立过程.因此,人工智能的理论与技术非常适用于软件智能开发,并有望取得突破.

程序合成方法与程序员根据需求规约和设计规约手工编写程序不同,它关注让计算机理解用户需要什么,进而自动编程,从而在一定程度上取代程序员的工作,使程序员的主要工作重点不再是重复性的编码,而是设计模型结构和编写需求规约.程序合成由Alonzo Church于1957年提出[9],长期以来,一直是计算机科学的核心问题和挑战之一,并被认为是计算机科学的圣杯,其挑战主要体现在“需求规约的描述”、“程序搜索空间的刻画”、“搜索技术的设计”[2][6]三个方面.由于描述完整需求规约难度过大,传统程序合成方法发展缓慢,而人工智能技术为程序合成技术的发展带来新的思路,各种需求规约描述方式不断涌现.近年来,程序智能合成技术得到了快速的发展.总体而言,现有的程序智能合成技术由三部分组成(如图2所示):需求规约的描述、合成模型、软件资产库.其中软件资产库是实现程序合成的支撑,对源代码进行挖掘处理后,构建软件资产库;合成模型为程序合成的核心,利用人工智能技术学习代码资源中隐含的特征和知识;需求规约的描述为程序合成的基础和前提,描述能用于程序合成的需求规约的挑战主要有两方面,一方面为如何权衡需求规约的完备性与复杂性,另一方面为如何处理自然语言具的二义性.需求规约包括输入输出示例[47-49]、代码框架[50]、部分代码[51-53]、自然语言[54-57]、逻辑规约[2]等.程序智能合成技术目前主要有3种需求规约描述方法,将在第三章详细阐述.

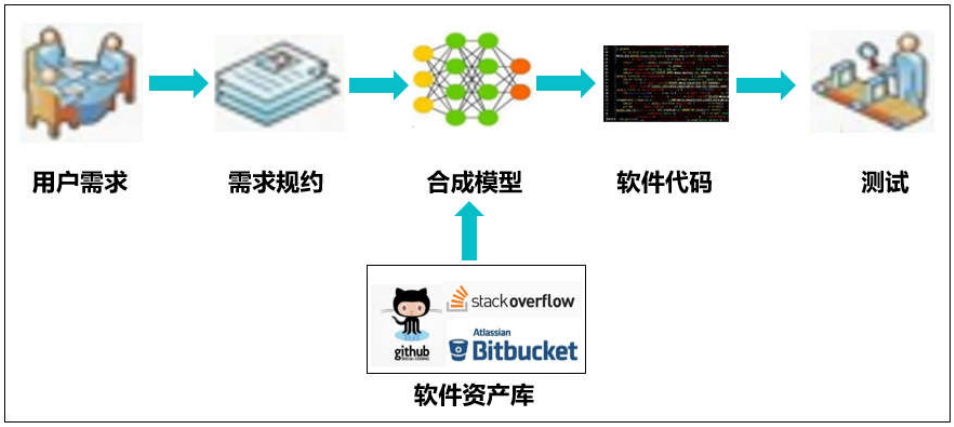

Fig.2 Framework of intelligent program synthesis model

图 2 程序智能合成软件开发模型

2 程序合成方法研究关注度分析

本文的主要目的是总结分析程序合成相关工作,提出未来的研究方向以及可探索的工程应用方案.为了解全世界范围上研究机构及研究人员对程序合成研究方向的关注情况,在 IEEE Xplore Digital Library 搜索“程序合成(Program Synthesis)”,检索到 15128 篇文章,其中 90%以上的论文发表均于 1970 年之后,因此对 1970-2020 年中的 14926 篇论文进行处理,从中筛选出研究主题为程序合成的论文,得到 150 篇与程序合成方法和应用相关的论文,从文献发表时间来看(如下图所示),全世界范围上研究机构及研究人员对程序合成研究方向的关注度逐渐上升.

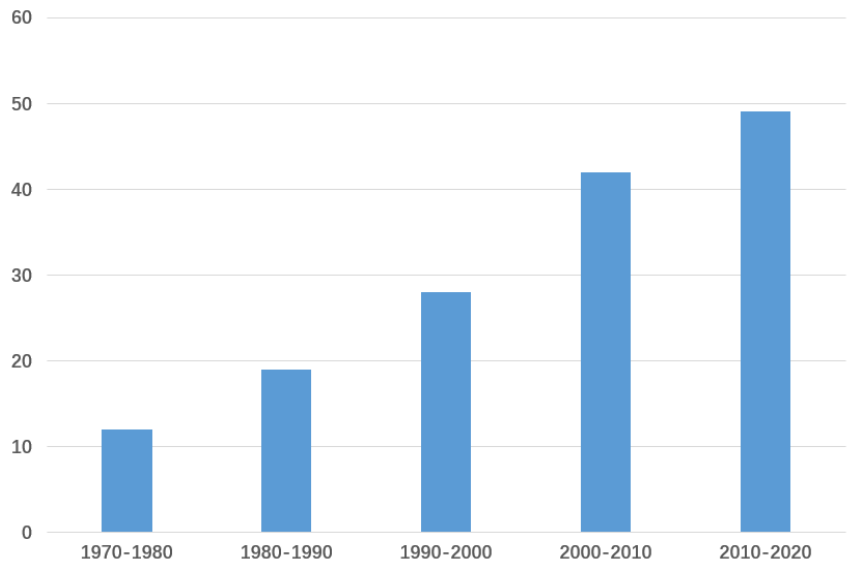

Fig.3 The submissions from 1970 to 2020

图 3 1970 年-2020 年文献发表数量统计

本文通过软件工程及人工智能领域国际重要会议和期刊中 2017 年至今录用的论文进行统计,包括 ICLR、

NIPS、ICSE、TOSE、IJSEKE、AAAI、ASE、SAMER、TOSEM、ESEC/FSE、CAV.从中筛选出与程序自动生成相关的文章数量,其中ICLR论文2篇,NIPS论文5篇,ICSE论文8篇,TOSE论文6篇,IJSEKE论文6篇,AAAI论文4篇,ASE论文3篇,SAMER论文10篇,TOSEM论文2篇,ESEC/FSE论文4篇,CAV论文2篇.经过对统计结果的分析可以得出结论,已有的研究集中在程序合成、程序修复、代码推荐、缺陷预测、代码克隆、代码搜索等方向(如下图所示).图中环线为文章数量参考线,蓝色区域为各个方向的论文数量,从图中可以看出程序合成所占比重最高,说明程序合成近年是程序自动生成方向的研究热点.

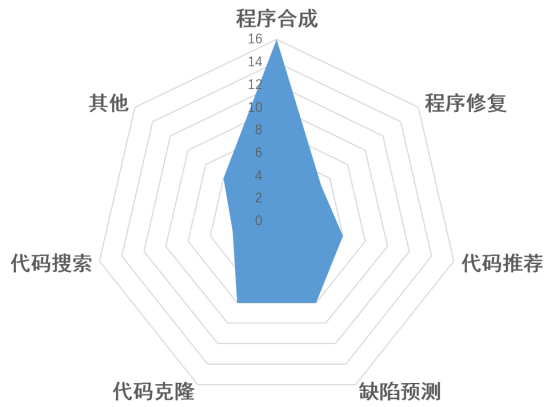

Fig.4 Analysis of the research content of articles from 2010 to 2020

图4 2010年-2020年顶级会议相关论文研究内容分析

程序合成技术受到了国际上多家研究机构的关注.美国国家科学基金会在2012年发布了四项远征计算奖(Expeditions in Computing awards),并为每个奖项提供1000万美元的资助,是当时美国国家科学基金会在计算机科学研究方面的最大投资.ExCAPE(Expeditions in Computer Augmented Program Engineering)就是被该奖金资助的一个项目,ExCAPE 旨在将编程从纯粹的手动任务转变为程序员和自动程序合成工具可以协作生成符合其规范的软件.在五年(2012-2017)时间里 ExCAPE 在多方面取得了丰硕的成果[10],其参研团队也成为程序合成领域极具影响力的单位.该团队提出Syntax-Guided Synthesis程序合成方法并建立多种原型解算器,这项工作有助于推进计算方法的最新技术,它促进了程序综合的新应用,例如量子计算机程序的自动优化;美国国防部高级研究计划局(DARPA)2015年发布了资源适应性软件系统(BRASS)项目招标书,该项目为建造使用寿命长、存活性高、扩展性强的自适应软件系统寻求新的研究方法[11].BRASS 需要集成新的资源感知程序分析和相关方法,以发现应用程序意图.这里需要用到程序转换和程序合成技术,保证系统的可信并提供验证,即面对不断演变的底层生态系统,意图与代码的一致性得以保证;2019年7月, DARPA又发起了IDAS(Intent Defined Adaptive System)项目[12],该项目于2020年2月正式启动,四年内支持经费6500万美元.IDAS目标是研究软件工程的新方法,快速实现代码生成,大大降低软件开发和维护成本.该项目包含4个技术领域:自动软件生成、问题集生成、集成测试和评估、实验与转换.其中自动软件生成为该项目的重点研究内容,通过研究用户意图及约束的捕获、理解和表达方法,利用自动化手段生成软件,研究软件自动化的代码生成和演化技术,以快速应对软件开发需求和运行环境的变化.

3 程序智能合成技术

程序合成问题自上世纪60年代初提出以来一直是软件工程领域的重大挑战之一.最初的程序合成方法是基于转换的程序合成(transformation-based synthesis),其核心是将一个高层的程序规约通过反复转换成较低层

的程序规约,最终生成目标程序代码[13-14].该方法自动生成的代码规模小,很长一段时间并没有进一步发展.20世纪60年代,人们提出了许多基于演绎推理的程序合成方法,该方法的主要思想为:首先,使用定理证明器构造程序规约的一个证明;然后,基于Curry-Howard同构关系,使用该证明来生成相应的程序代码.其中代表性工作是Z Manna[15]和R Walding[16]提出的程序合成系统,它将数学定理的构造性证明与程序开发相联系,将程序规约转换为待证明的命题, 将命题证明过程视为程序开发步骤, 证明成功即获得所需程序, 从而完成基于一阶谓词演算的程序规约到程序的自动生成.基于演绎的程序合成方法需要以完整的形式化规约为前提, 这一前提的达成通常具有较大的难度, 因而这种方法很少被应用.随着人工智能等技术的发展,程序合成技术不断探索出新的技术途径.与传统的程序合成方法不同,程序智能合成不再只是利用演绎和归纳进行合成,而是更多的利用机器学习等人工智能技术,学习已有的大量代码知识,智能化地进行程序开发[17-18].

现有的程序智能合成技术还有诸多挑战需要攻克,其中需求规约的描述是亟需解决的难题,一方面,完整的需求规约描述极其复杂,所花费的时间成本过高.不完整的需求规约又很难准确表达真实的用户意图.另一方面,自然语言的需求规约描述具有二义性,很难被计算机理解.根据需求规约的不同描述方式,可分为基于输入输出示例的程序合成方法、基于代码框架(语法)的程序合成方法、基于自然语言的程序合成方法.程序智能合成技术主要使用的合成模型包括遗传算法模型、循环神经网络模型、递归神经网络模型、卷积神经网络模型、Word2Vec模型等.表1为现有程序智能合成技术的总结,后文会进行详细介绍.

Table 1 Summary of existing studies of intelligent program synthesis

表 1 程序智能合成现有工作的技术总结

需求规约表示方式	文献	合成模型	技术特点
基于输入输出示例的程序合成	Reed 等人[19] Chengtao 等人[20] Cai 等人[21]	循环神经网络 递归神经网络 循环神经网络	利用深度学习方法生成代码片段
	Devlin 等人[23] Emilio 等人[28]	循环神经网络 递归神经网络	利用深度学习方法实现 Excel 字符串转换
	Balog 等人[22]	编码器-解码器模型	利用深度学习与搜索技术生成代码片段
	Becker 等人[25]	遗传算法模型	利用遗传算法模型生成代码片段
基于代码框架(语法)的程序合成	Armando 等人[29] Murali 等人[30] Navid 等人[34]	编码器-解码器模型 编码器-解码器模型 Word2Vec 模型	利用 Sketch 框架的方法生成代码片段
	Alur 等人[31] Alur 等人[32] Alur 等人[33]	深度神经网络 深度神经网络 深度神经网络	利用语法制导的方法生成代码片段
	Sun 等人[35] Luan 等人[36] Chen 等人[37] Hu 等人[38]	卷积神经网络 深度神经网络 长短期记忆网络 长短期记忆网络	利用代码结构化信息生成代码片段
基于自然语言的程序合成	Quirk 等人[39] Liu 等人[40]	深度神经网络 深度全连接网络+Attention	利用深度学习方法生成 IFTTT 程序
	Gu 等人[41] Gu 等人[42]	循环神经网络 循环神经网络	利用深度学习方法生成 API 序列
	Zhong 等人[43] Manshadi 等人[44]	编码器-解码器模型 深度神经网络	利用深度学习方法生成 SQL 查询语句
	Dong 等人[45]	编码器-解码器模型	利用深度学习方法生成逻辑形式
	Desai 等人[46]	深度神经网络	利用深度学习方法生成领域特定语言

3.1 基于输入输出示例的程序合成方法

基于输入输出示例的程序合成方法以输入输出示例作为需求规约描述.近年来,很多学者尝试将深度学习等方法与基于输入输出示例的程序合成相结合,并取得了一定的进展.该方法首先通过大量的输入输出示例对训练合成模型;合成模型训练好后可以进行合成任务,输入领域特定语言(Domain Specific Language, DSL)描述的需求规格,该模型可给出与之相对应的程序.

2015年, Reed等人开发了一个名为“神经程序解释器”(Neural Programmer Interpreters，NPI)[19]的框架，该框架利用神经网络及其组合来生成程序执行所需要的语句以及参数，可以根据输入输出示例生成加法、排序等简单程序.此后，许多研究工作基于NPI展开，通过对NPI进行改进以提高程序生成的效率.如Chengtao等人提出NPI的改进模型NPL(Neural Program Lattices)[20],其对传统神经程序解释器NPI进行监督学习方面的改进,使用弱监督代替大多数强监督的情况下,实现性能与NPI相当,使得NPI能够从只包含低级操作的示例中推断出高级的程序结构.Cai等人[21]提出NPI的改进模型RNPI,允许程序调用其自身,通过增加递归模块扩展了神经网络结构,改善了神经网络的泛化能力.对给定长度的数据进行排序操作,当待排序数据个数超过11时,NPI的预测正确性随着数据个数增长不断下降,而RNPI始终可以给出正确的数据排序结果.

2017年,Balog等人提出了智能化自动编程方法DeepCoder[22],该方法能够根据输入输出示例,使用深度神经网络和集束搜索技术实现特定领域的代码自动合成.该方法缩小了搜索空间,用大小为34的DSL指令集进行表示,通过将深度神经网络和集束搜索技术相结合生成简单的程序,与传统方法相比,该方法实现效率较高.虽然DeepCoder的指令集和操作数有限,无法描述复杂任务,通常只能生成10行左右的代码,但该方法作为一种代码自动合成方法,将深度学习技术和代码的搜索生成技术相结合,具有创新性.

2017年,Devlin等人提出了面向字符处理的神经网络模型RobustFill[23],该模型可以实现与FlashFill[24]相同的字符串处理功能,使用了一种改进的RNN神经网络,在字符串处理测试集上的正确率更高(RobustFill的正确率为92%,FlashFill的正确率只有不到50%).但是,由于RobustFill完成相同功能需要更多的数据样本(FlashFill仅需要1~3个示例,Robust需要至少5个示例)及计算资源,因此目前尚未在Excel中进行部署.

Becker等人在2017年提出了能够自动生成可运行程序的AI系统“AI Programmer”[25].该系统利用遗传算法进行复杂指令的模拟,并根据程序的输入输出生成程序.该系统的指令集由八个基础的计算机指令构成, 通过随机初始化获得一个初始的目标程序,并根据程序的输出测试其适合度.该方法生成的程序越接近目标程序,适应度就越高,因而越有可能继续进行下一代进化.尽管AI Programmer能够完成初级程序的自动生成任务,但其生成的程序可读性差,且算法模型仅能针对单一任务.此外,AI Programmer运行过程时间开销巨大,且容易陷入死循环和状态空间爆炸,因此,暂时还无法达到工业化的应用标准.

Jha等人提出了Oracle指导的归纳合成(OGIS)[26].当程序合成存在程序模板将大大降低程序合成的难度,然而,在大多数情况下,相关领域特定的模板是不具普适性的.基于Oracle指导的程序合成就试图解决这一难题,首先,使用简化的规约搜索候选程序,然后验证该候选程序Oracle有效性.为了降低验证的难度,该团队提出通过一个I/O Oracle可以在任意给定的输入上生成有效的程序输出的方法[27],该方法认为一个正确的候选程序如果与整个规约ϕ一致,那么它将与有限数量的有效输入/输出对一致.首先通过公式对所有可能的程序空间进行编码;再给出一组输入-输出对,然后进一步约束该公式,以便它能仅对正确运行的程序进行编码;最后通过此约束生成候选解决方案.

Guo等人提出了反例指导的归纳合成方法[20,30].该方法分为求解器和验证器两部分.首先将原始的二阶规约简化为一阶规约,将一阶规约输入到求解器中,产生满足该规约的候选程序.验证器判断候选程序是否满足原始二阶规约,如果不满足,验证器将生成一个反例,然后将其返回给求解器.求解器重复搜索以寻找满足此反例的新候选程序.此过程将反复进行,直到某候选程序被接受,或者求解器找不到与当前规约相符的候选程序(即无解).

Emilio等人提出两个新的神经模块进行神经程序合成[29].第一个模块称为互相关I/O网络(the cross correlation I/O network),它给出了一组输入输出示例,生成了一组I/O示例的连续表示.第二个模块,是递归神经网络(R3NN),给出了实例的连续表示,并通过增量展开部分程序来合成一个程序.通过将该方法应用于基于正则表

达式的字符串转换领域,展示了该方法的有效性.实验表明,R3NN模型能够从新的输入输出实例中构造程序.

3.2 基于代码框架(语法)的程序合成方法

基于代码框架(语法)的程序合成方法主要利用程序结构或者语法框架信息,将程序代码转换为抽象语法树、控制流图和数据流图、程序框架草图等某种结构化表示.该方法主要由三部分组成:需求规格说明、结构/语法限制、合成器.软件开发人员一般很难写出完整的需求规格说明,但在结构/语法限制的补充下可以不断丰富软件需求信息.结构/语法的限制条件同时可以限制合成器的输入,以提高合成效率.使用程序合成器来生成程序,成功的将程序合成问题转换为约束求解问题.目前常用的合成器有SAT(Boolean Satisfiability Problem)求解器和SMT(Satisfiability Modulo Theories)求解器.

Armando等人于2008年提出了基于Sketch的程序合成方法[30].该方法提供一个待填充的程序框架再由合成器根据需求规约填充程序空白.合成阶段,Sketch根据部分输入输出示例合成出一个可能的解;验证阶段,若发现当前解能够满足所有输入输出示例,则合成成功.若存在不满足的输入输出示例,则进入下一次的“合成验证循环”.该方法可实现简单Sketch框架下的程序合成,然后对于复杂Sketch框架下的程序合成依然无法实现.将用户意图转换为Sketch语言的过程比较繁琐,同时合成速度较为缓慢,因此基于Sketch的合成目前更多的是作为辅助程序合成的工具.在此基础上,2017年,Murali等人提出一种神经网络与代码框架Sketch相结合的进行程序合成方法[31],利用神经网络模型建立代码框架与标签(API调用、代码Token或数据类型)之间的联系.在软件开发过程中,模型会根据程序员输入的标签信息生成Java代码框架.该方法可以在编程过程中为开发人员提供辅助参考,提升开发效率.

Alur等人于2013年提出了一种基于语法制导(Syntax-Guided Synthesis)的程序合成方法[32],该方法的输入包括一组由逻辑表达式描述的需求语义规范, 和一组符合语法规则的候选实现集, 允许用户使用语法来补充候选集,并找到一个满足给定需求的解.该团队同时在基于语法制导的程序合成方法基础上加入了多种技术,如增加反例技术、激活学习技术、随机搜索技术等[36-37],这些方法利用语法进行程序搜索,大大缩小了代码空间,从而提升了搜索效率.目前基于语法制导的程序合成还处于理论研究阶段,对于复杂系统用户意图的表达还需进一步研究.

Navid等人于2017年提出了一种面向数据库应用的程序合成方法[33],该方法将用户给出的自然语言需求描述转换为Sketch语言, 然后基于Sketch语言合成SQL程序.该方法同时提供了错误修复功能,以保证自然语言描述与用户意图的一致性,使合成结果达到较高的准确率.因为数据库SQL语句相比于其他程序语言更加简单,便于机器分析与学习,因此该方法获得了良好的合成效果.

此外,还有很多研究利用基于程序上下文信息建立代码自动补全或推荐模型,Luan等人于2017年提出了代码智能推荐模型Aroma[36],该模型利用语法树的信息,找到在语法层面与输入代码类似的代码实例.自动将类似的搜索结果聚合在一起以生成代码建议.该方法考虑了代码结构化信息,支持JavaScript、Python等多种语言.Aroma根据代码上下文信息进行代码搜索和推荐,但是该方法仅能智能补全代码片段,无法生成完整可运行的程序代码,且补全过程还需要开发人员确认.Chen等人于2019年提出了一个名为DeepAPIRec[37]的API推荐模型,可以根据代码上下文自动推荐API及使用参数.Hu等人于2019年提出了一种代码补全模型Deep-AutoCoder[38],在方法调用补全和随机代码补全两种使用场景下优于其他方法.目前,基于代码上下文的程序合成方法多用于Token预测、程序短句补全、API调用推荐等方面.还无法直接生成可执行程序.

3.3 基于自然语言的程序合成方法

基于自然语言的程序合成方法是指针对自然语言描述的需求规约进行程序合成,由于自然语言具有二义性,与程序语言间有较大鸿沟,因此该技术是程序合成领域的一项巨大挑战,目前很多学者针对自然语言的理解以及领域特定代码的自动合成开展了研究.基于自然语言的程序合成方法还无法生成复杂程序,仅可以生成逻辑简单的程序(TFTTT程序和SQL程序)或者生成API调用序列、程序的逻辑形式等.

2015 年,Quirk 等人提出了一种利用神经网络模型从自然语言生成 If-This-Then-That(IFTTT)代码[39]的方

法,该方法将自然语言描述生成抽象语法树(Abstract Syntax Tree,AST),它首先将 AST 转换为代码 tokens 的序列,并训练了一个神经翻译机模型,将自然语言描述翻译为程序代码.虽然 If-This-Then-That 的结构是有限的,许多其他潜在的语法结构并没有被讨论,但该方法仍然具有很大的启发意义.2016 年,Liu 等人[40]针对 IFTTT 程序合成问题,提出了一种引入注意力(Attention)机制的神经网络结构,并对其进行端到端的训练,实现从自然语言文本描述生成 If-Then 程序,与其他方法相比可以显著降低程序预测的错误率.但训练数据的噪声仍会带来错误,因此该方法仍需改进.

2016年,Gu等人在Deep API项目[414]中提出了智能化API检索方法,能够根据自然语言形式的任务需求生成API调用序列.该方法的合成模型采用seq2seq,模型输入为特征向量,该特征向量由自然语言编码而成,模型输出为相应的API调用序列,利用神经网络模型学习自然语言与API序列间的关联联系,模型训练好后用于生成相应的API调用序列.2018年,该团队又提出了一种利用基于深度学习进行代码检索的方法[42],该方法首先要得到序列的编码信息,包括token序列、方法名序列以及API调用序列,然后将三部分信息组合为代码的向量表示,然后利用RNN得到自然语言语句向量表示,利用余弦相似度来衡量上述向量间的相似度,并以此建立自然语句与代码片段的联系.上述方法通常只能生成与自然语言描述相关的API调用序列或代码片段,给程序员的编程工作带来帮助.

2017年,Zhong等人提出了基于Seq2SQL的程序合成方法[43],该方法采用基于策略的强化学习使得自然语言描述合成为相应的SQL语句.在此基础上,还有学者在此基础上加入将深度学习技术和查询解析技术,该方法提高了合成准确率,并利用巴克斯范式约束SQL语句的合成.

Manshadi等人于2012年发表了通过群体智慧解决自然语言编程任务[44].该方法首先由非专业人员来提供大量的数据对,然后进行程序合成,生成相应的程序代码.通过群体智慧的方法能更好捕捉到自然语言所描述的真实需求,从而生成更精确的程序.在特定领域进行了试验,可以解决简单的自然语言编程任务,错误率仅为4%.由于该方法依赖于传统计算和人工计算的结合,但人工计算的正确性无法保证,因此还无法应用到工程实践中.

Dong等人于2016年发表了应用Encoder-Decoder模型结合注意力机制(Attention)进行程序合成方法[45].该方法合成模型的输入语句和输出逻辑形式都被视为序列,由不同的递归神经网络进行编码和解码.该方法采用两种神经网络模型:递归神经网络和树形循环神经网络.递归神经网络完成自然语言的编码;循环神经网络完成程序的生成.使用递归神经网络将输入语句编码成向量表示,并通过对编码向量的输出设置条件来生成它们的逻辑形式,例如将自然语言“jobs with a salary of num”转化为逻辑形式“job(ANS),salary_greater_than(ANS, num, year)”.并在四个数据集上验证了该方法的正确性.该方法实现了自然语言转到结构化的逻辑形式的转换.

Desai等人在2016年发表了以自然语言描述作为需求规约的程序合成方法[46].该方法利用神经网络模型学习自然语言与DSL之间的关系,训练好的神经网络模型可以将用户提供的自然语言描述转换为DSL,再通过DSL进行后续程序合成.实验证明该方法可得到较好的准确率.但该方法依然有许多问题需要继续研究,例如DSL的开发代价过大,且只能适用于相同的DSL,扩展性较差.

4 趋势与展望

总体而言,目前程序智能合成技术还处于早期阶段.引入人工智能技术支持软件开发是具有可行性的,也是近年新趋势,可提高软件开发效率,减少维护成本,未来有可能颠覆现有的软件开发模式.但该技术的后续发展仍存三点不足.一是当前的程序合成技术仅能实现生成规模小、功能单一的简单程序,且需要占用较大的系统资源;二是大部分程序合成技术只针对特定任务,对于新的任务需要重新进行计算或者训练;三是由于很多合成过程中使用了深度学习模型,对于合成得到的软件代码,只能通过黑盒测试的方法对软件功能、性能进行验证,难以利用程序分析等方法验证其实现逻辑的正确性.

程序智能合成技术已经取得了一定的进展,综合已有的技术来看,该技术在未来有以下两方面发展趋势.

1） 多种需求描述方法相结合.

目前常用的需求规约描述方法有多种,如输入输出示例对[47-49]、代码框架[50]、代码示例[51-54]、自然语言

[55-57]、逻辑规约[5]等.不同的任务特点决定了需求描述方法的选择,然而每种需求描述方法都有其局限性,直接影响着程序合成过程.最初的逻辑规约虽然能够完备的描述用户需求,但是这种逻辑规约书写难度过大;自然语言虽然简单易用,但由自然语言描述的需求规约存在二义性,很难合成高质量的软件代码.从上文的调研结果看,很多研究者已经不满足于使用一种需求描述方法来表达需求规约,而是采用多种需求描述方法相结合的方式完善用户意图信息.该方法能够更好的减轻用户负担,同时可以表达出更完善的需求规格,相信未来会有越来越多的程序合成方法采用多种需求描述方法相结合来表达需求规格.

2)深度学习技术应用越来越广泛.

本文归纳了多种程序智能合成技术方法,总体来看,这些方法在传统的程序合成方法基础上引入了人工智能技术,而深度学习技术应用的最为广泛.目前已有多种深度学习模型应用到程序合成技术中,如循环神经网络模型、递归神经网络模型、卷积神经网络模型等.这些应用中有些使用一种深度学习模型,有些使用一种深度学习模型的改进模型,也有些采用多种深度学习模型的组合模型.对深度学习模型的改良可以影响合成的速度和效率.因此未来会有更多的科研工作者研究如何加强深度学习技术,以取得更好的合成效果.

综上所述,程序合成目前还存在很多亟待突破的技术难点,亟需开展进一步的研究工作,可重点从以下三方面取得突破.

1） 建立需求规约描述语言,准确的识别用户意图.

虽然人工智能技术可以辅助开发人员识别、提取用户需求的关键特征,辅助开发人员提升编程效率,但目前开发人员需要花费大量精力描述需求规约;且不同项目对相同或相似任务的描述各不相同,描述方式不具有普适性.因此有必要探索一种结合多种表达形式且具有丰富描述能力的需求规约描述语言,在不同的抽象层次和维度上实现对软件系统的功能与性能、行为与结构、部分与系统的统一表示,攻克软件意图的表达与理解难题,为生成复杂程序奠定基础.

2） 充分利用已有领域资产中的软件知识.

利用人工智能方法可以通过数据驱动的方式挖掘程序代码的部分特征,并在代码智能搜索、补全等领域取得了一定的效果.但是已有的方法无法综合利用不同类型的软件资产进行程序合成,已有资产中可用于程序合成的关键特征有待进一步挖掘.

3） 突破合成软件的可信验证技术.

虽然目前可通过黑盒测试的方法对软件验证,但是仍缺少合成过程及合成产品的可信性分析与验证方法.因此,需要探索合成代码与需求规约的一致性评估方法,建立合成过程的质量追溯和评价体系,保证合成过程和合成代码的正确性、可靠性及安全性.

5 结束语

机器学习等人工智能技术的发展促进了软件工程学科的进步.同时随着各行各业的发展,专业领域已经积累了大量的软件资产,这些软件资产中存在着大量高质量可复用的软件代码.这些软件代码资产日渐庞大,促使人们探索软件开发的新方法,即如何利用大规模软件代码资产库中已有的可复用的代码实现软件开发方法的变革.基于软件资产和人工智能技术支持软件自动生成技术,已经得到了国内外学术界和工业界越来越多的关注.

本文通过对软件开发模式的发展进行分析,介绍了软件开发方法发展的四阶段,对近年来国内外程序智能合成技术的研究方法开展了调研和综述,梳理并总结了这些方法的原理和技术特点.在此基础总结了程序智能合成研究中面临的主要挑战和未来发展趋势.程序智能合成是软件自动生成的新途径,随着人工智能技术的蓬勃发展,相信在未来几年程序合成领域将会产生突破性进展,从而实现软件开发方法的变革.

References:

[1] Bordes A, Glorot X, Weston J, Bengio Y. Joint learning of words and meaning representations for open-text semantic parsing. In: Proc. of the Int'l Conf. on Artificial Intelligence and Statistics. 2012. 127-135.

[2] Gulwani S, Polozov O, Singh R. Program synthesis. Foundations and Trends® in Programming Languages, 2017,4(1-2):1-119.

[3] Yang FQ. Thinking on the development of software engineering technology. Journal of Software, 2005,16(1):1-7. http://www.jos.org.cn/1000-9825/16/1.htm

[4] DeMarco T. Structured analysis and system specification. Classics in software engineering. Yourdon Press, 1979: 409-424.

[5] Yang FQ, Mei H, Lü J, Jin Z. Some discussion on the development of software technology. Acta Electronica Sinica, 2002,30 (12A):1901-1906 (in Chinese with English abstract).

[6] Liu BB, Dong W, Wang J. Survey on intelligent search and construction methods of program. Ruan Jian Xue Bao/Journal of Software, 2018,29(8):2180-2197 (in Chinese). http://www.jos.org.cn/1000-9825/5529.htm

[7] Bai GY, Xu C, Fan ZH, Jiang DM. Software Engineering Technology, Method and Environment. Ruan Jian Xue Bao/Journal of Software, 1994,6():292-300.

[8] Hindle A, Barr ET, Su Z, Gabel M, Devanbu P. On the naturalness of software. In: Proc. of the 34th Int'l Conf. on Software Engineering. IEEE, 2012. 837-847. [doi: 10.1109/icse.2012.6227135]

[9] Church A. Logic, arithmetic and automata. In Proc. of the Int'1 Congess of Mathematicians.1962.23-35.

[10] Alur R, Martin M, Raghothaman M, *et al*. Synthesizing Finite-State Protocols from Scenarios and Requirements. In: Proc. of the 10th Int'l Haifa Verification Conference(HVC). 2014: 75-91.

[11] Hughes J, Sparks C, Stoughton A, *et al*. Building Resource Adaptive Software Systems (BRASS): Objectives and System Evaluation. ACM SIGSOFT Software Engineering Notes, 2016, 41(1): 1-2.

[12] Broad Agency Announcement. Intent-Defined Adaptive Software (IDAS), HR001119S0074, July 10, 2019. [Online]. Available: https://www.darpa.mil/program/intent-defined-adaptive-software.

[13] Shaw D, Wart W, Green C. Inferring LISP programs from examples. In: Proc. of the 4th Int'1 Joint Conf. On Artficial Intelligence (IJCAI). Vol.1. 1975. 260-267.

[14] Summers PD. A methodology for LISP program construction from examples. Journal of the ACM, 1977, 24(1): 161-175.

[15] Manna Z, Waldinger R. Synthesis: dreams→programs. IEEE Trans. on Software Engineering, 1979, 5(4): 294-328.

[16] Manna Z, Waldinger R. A deductive approach to program synthesis. ACM Trans. on Programming Languages and Systems (TOPLAS), 1980, 2(1): 90-121.

[17] E. Kitzelmann. Analytical inductive functional programming. 18th Int'1 Symp. on Logic-Based Program Synthesis and Transformation(LOPSTR). 2009: 87-102.

[18] J Bornholt, E Torlak. Scaling program synthesis by exploiting existing code. Machine Learning for Programming Languages Workshop (ML4PL), 2015.

[19] Reed S, De Freitas N. Neural programmer-interpreters. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2016.

[20] Chengtao Li, Daniel Tarlow, *et al*. Neural Program Lattices. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2017.

[21] Cai J, Shin R, Song D. Making neural programming architectures generalize via recursion. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2017.

[22] Balog M, Gaunt AL, Brockschmidt M, *et al*. Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989, 2016.

[23] Devlin J, Uesato J, Bhupatiraju S, *et al*. RobustFill: Neural program learning under noisy I/O. In: Proc. of the 34th Int'l Conf. on Machine Learning. 2017. 990-998.

[24] Gulwani S. Automating string processing in spreadsheets using input-output examples. Proc. of the ACM SIGPLAN Notices, 2011,46(1):317-330.

[25] Becker K, Gottschlich J. AI Programmer: Autonomously Creating Software Programs Using Genetic Algorithms[J]. arXiv preprint arXiv:1709.05703, 2017.

[26] Susmit Jha and Sanjit A. Seshia. A Theory of Formal Synthesis via Inductive Learning. ArXiv e-prints, May 2015.

[27] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. Oracle guided component-based program synthesis. In 2010 ACM/IEEE 32nd International Conference on Software Engineering (ICSE), volume 1, pages 215–224. IEEE, 2010.

[28] Zheng Guo, Michael James, David Justo, et al. Program Synthesis by Type-Guided Abstraction Refinement. POPL, 2020.

[29] Parisotto E, Mohamed A, Singh R, et al. Neuro-symbolic program synthesis. arXiv preprint arXiv:1611.01855, 2016.

[30] Solar-Lezama A. Program synthesis by sketching[D]. University of California, Berkeley, 2008.

[31] Murali V, Qi L, Chaudhuri S, *et al*. Neural Sketch Learning for Conditional Program Generation. arXiv preprint arXiv:1703.05698, 2017.

[32] Alur R, Bodik R, Juniwal G, *et al*. Syntax-guided synthesis. Formal Methods in Computer-aided Design. IEEE, 2013.

[33] Alur R, Singh R, Fisman D, *et al*. Search-based program synthesis. Communications of the ACM, 2018, 61(12): 84-93.

[34] Alur R, Černý P, Madhusudan P, *et al*. Synthesis of interface specifications for Java classes. In: Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages(POPL). ACM Press, 2005: 98-109.

[35] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, *et al*. Type-and content-driven synthesis of SQL queries from natural language. arXiv preprint arXiv:1702.01168, 2017.

[36] Luan S, Yang D, Barnaby C, *et al*. Aroma: Code Recommendation via Structural Code Search[J]. arXiv preprint arXiv:1812.01158, 2018.

[37] Chi C, Xin P, Jun S, *et al*. Generative API usage code recommendation with parameter concretization. Science China Information Sciences, 2019(9): 51-72.

[38] Hu X, Men R, Li G, *et al*. Deep-AutoCoder: Learning to Complete Code Precisely with Induced Code Tokens. In: Proc. of the 43rd Annual Computer Software and Applications Conf. (COMPSAC). IEEE, 2019. 159-168.

[39] Quirk C, Mooney R, Galley M. Language to code: Learning semantic parsers for if-this-then-that recipes. In: Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Int'l Joint Conf. on Natural Language Processing, Vol.1. 2015. 878− 888.

[40] Liu C, Chen X, Shin EC, *et al*. Latent attention for if-then program synthesis. In: Proc. of the Advances in Neural Information Processing Systems. 2016. 4574− 4582.

[41] Gu X, Zhang H, Zhang D, Kim S. Deep API learning. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2016. 631−642. [doi: 10.1145/2950290.2950334]

[42]Gu X, Zhang H, Kim S. Deep code search. In: Proc. of the 2018 40th Int'l Conf. on Software Engineering (ICSE). IEEE, 2018: 933-944.

[43]Zhong V, Xiong C, Socher R. Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103, 2017.

[44]Manshadi M, Keenan C, Allen J. Using the crowd to do natural language programming. In: Proc. of the AAAI. 2012.

[45]Dong L, Lapata M. Language to Logical Form with Neural Attention. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics. 2016: 33-43.

[46]Desai A, Gulwani S, Hingorani V, Jain N, Karkare A, Marron MRS, Roy S. Program synthesis using natural language. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016: 345-356.

[47]Gulwani S. Programming by examples. Dependable Software Systems Engineering, 2016, 45(137): 3-15.

[48]Singh R, Gulwani S. Synthesizing number transformations from input-output examples. Proc. of the 24th Int'l Conf. on Computer Aided Verification(CAV). 2012.

[49]Singh R, Gulwani S. Transforming spreadsheet data types using examples. In: Proc. of the 43rd annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages(POPL). ACM Press, 2016: 343-356.

[50]Perelman D, Gulwani S, Ball T, Grossman D. Type-directed completion of partial expressions. In: Proc. of the 33rd ACM SIGPLAN Conf. on Programming Language Design and Implementation(PLDI). ACM Press, 2012: 275-286.

[51]Bornholt J, Torlak E. Synthesizing memory models from framework sketches and litmus tests. Proc. of the ACM SIGPLAN Notices, 2017, 52(6): 467-481.

[52]Gvero T, Kuncak V, Kuraj I, R Piskac. Complete completion using types and weights. In: Proc. of the 34th ACM SIGPLAN Conf. on Programming language design and implementation. ACM Press, 2013: 27-38.

[53]Zhang H, Jain A, Khandelwal G, et al. Bing developer assistant: improving developer productivity by recommending sample code. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2016: 956-961.

[54]Nori AV, Ozair S, Rajamani SK, et al. Efficient synthesis of probabilistic programs. Proc. of the ACM SIGPLAN Notices, 2015, 50(6): 208-217.

[55]Yin P, Neubig G. A syntactic neural model for general-purpose code generation. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2017. 440-450. [doi: 10.18653/v1/P17-1041].

[56]Beltagy I, Quirk C. Improved semantic parsers for if-then statements. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2016. 726-736.

[57]Cai R, Xu B, Yang X, et al. An encoder-decoder framework translating natural language to database queries. arXiv preprint arXiv:1711.06061, 2017.

附中文参考文献:

[3] 杨芙清.软件工程技术发展思索.软件学报,2005,16(1):1−7.

[5] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展.电子学报,2002,30(12):1901−1906.

[6] 刘斌斌,董威,王戟.智能化的程序搜索与构造方法综述.软件学报,2018,29(8):2180-2197.

[7] 白光野,徐崇,范植华,蒋东溟.从软件工程的发展看软件自动化.软件学报,1994,6():292-300.