计算机科学
COMPUTER SCIENCE

实例编程研究进展与挑战

严倩羽, 李弋, 彭鑫

引用本文

严倩羽, 李弋, 彭鑫. 实例编程研究进展与挑战[J]. 计算机科学, 2022, 49(11): 1-7.

YAN Qian-yu, LI Yi, PENG Xin. Research Progress and Challenge of Programming by Examples[J]. Computer Science, 2022, 49(11): 1-7.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.211000225

实例编程研究进展与挑战

严倩羽　李　弋　彭　鑫
复旦大学计算机科学技术学院　上海 200438
上海市数据科学重点实验室(复旦大学)　上海 200438
(19212010002@fudan.edu.cn)

摘　要　程序合成指计算机自动地构造符合指定语法和用户给定规约的代码。实例编程是程序合成中一类以输入输出实例为规约形式的范式,它易用性高、学习成本低。近年来,该技术已经在数据处理、字符串变换等领域得到成功应用,具有很大的发展潜力。实例编程主要待解决的问题有两点:一是庞大程序空间中高效搜索的问题,二是程序合成解的歧义性问题。为解决第一个问题,实例编程方法在指定搜索策略时,需选取适当的领域特定语言,制定搜索算法,所应用的算法可分类为基于规则的算法和基于统计模型的算法。为解决第二个问题,实例编程方法需制定排序策略,所应用的排序策略可分类为基于给定实例的排序方法和基于用户交互的排序方法。文中对近年来的实例编程相关文献进行了整理,针对解决以上两个问题的方法、关键技术点进行了总结归纳,最后对实例编程领域未来的研究方向给出了建议。

关键词:程序合成;实例编程;搜索策略;歧义性
中图法分类号　TP311

Research Progress and Challenge of Programming by Examples

YAN Qian-yu,LI Yi and PENG Xin
School of Computer Science,Fudan University,Shanghai 200438,China
Shanghai Key Laboratory of Data Science(Fudan University),Shanghai 200438,China

Abstract　Program synthesis means that the computer automatically constructs code that conforms to the specified grammar and user's given specifications. Programming by examples is a kind of paradigm in program synthesis that takes input and output examples as the specifications. It has high usability and low learning cost. In recent years,it has been successfully implemented in applications such as data wrangling and string transformation,and has great potential for development. There are two main problems to be solved in programming by examples. One is the problem of efficient search in huge program space,and the other is the problem of ambiguity in solutions. To solve the first problem,a method for programming by examples needs to select the appropriate domain-specific language and formulate the search algorithm when specifying the searching strategy. The applied searching algorithm can be classified into a rule-based algorithm and an algorithm based on statistical model. To solve the second problem,a method for programming by examples needs to formulate a sorting strategy. The applied sorting strategy can be classified into a sorting method based on given examples and a sorting method based on user interaction. This paper sorts out related literature on programming by examples in recent years,summarizes the methods and key technical points to solve the above two problems,and finally gives suggestions for future research directions in the field of programming by examples.

Keywords　Program synthesis,Programming by examples,Search strategy,Ambiguity

1　引言

软件工程和程序语言等领域一直致力于提高软件开发的自动化程度,减少程序员的开发负担。给定程序语言的语法,程序合成自动构造满足用户规约的程序,是实现软件自动化开发的重要手段之一,可用于辅助甚至代替程序员编写代码。为了保证生成程序的正确性,用户提供的规约要完备可靠。在大部分情形下,编写逻辑规约和编写程序同样复杂[1-2]。为了使规约的表示更容易,一些工作提出了输入输出实例、自然语言描述、部分程序和断言等多种形式[2]。1977 年,Summers[3]在合成 Lisp 程序时,提出了实例编程(Programming By examples,PBE),用输入输出实例来表示规约。这里的输入输出实例是期望程序的合法输入的子集及其对应的输出集合。与逻辑规约相比,实例规约更容易获取和验证,对用户

到稿日期:2021-10-29　返修日期:2022-02-25
基金项目:上海市科委项目(19511132000)
This work was supported by the Science and Technology Commission of Shanghai Municipality(19511132000).
通信作者:李弋(liy@fudan.edu.cn)

更友好[2]。同时,实例编程被认为与人工智能、模仿学习息息相关[4]。人类的认知概念可以被视为认知程序(Cognitive Programs)[5],实例编程从实例中学习程序的过程与人类从教学中模仿、归纳、学习认知程序的过程类似。

实例编程求解的主要思想是在特定语法生成的程序空间中搜索满足输入输出实例规约的程序。这需要处理两个问题:1)程序搜索空间规模庞大,其他程序合成范式也面临这个问题,同时,实例编程被广泛应用于面向终端用户(End User)的任务中,如 Excel 的 FlashFill[6],这类场景需要更有效的搜索方法以满足实时性要求;2)歧义性消除,实例是一种不完备规约[7],这意味着程序空间中可能会有多个符合规约的程序。实例编程需要进行歧义消除以返回最佳的程序[8]。

实例编程的早期工作主要基于规则来限制搜索空间的规模。典型的方法可分为两类:1)枚举,结合设计好的剪枝策略对程序空间进行穷举,程序复杂度提高时,如程序变长,可枚举空间规模呈指数增长[2];2)约束求解,将给定语法中的语义转为表示程序和实例的变量的约束,利用 SAT/SMT 求解器求解。程序语言的语法复杂性决定了约束条件的规模,语法越复杂,约束条件越多,构造成本就越高。近年来,借助人工智能方法,很多研究都是基于已有程序学习语法规则的使用特征,通过统计模型对程序空间进行概率采样,减少基于规则的方法中对人工设计的要求,提高搜索效率。目前,实例编程的搜索策略有两方面的趋势:1)重视基于规则的方法和基于统计模型的方法相结合;2)强调合成过程中语义信息的利用。在歧义性消除方面,大部分工作通过排序的方式,使越可能符合用户意图的候选程序排序越靠前[7],如早期工作倾向于根据实例和程序特征设计排序函数。近年来,一些工作通过人机交互获取更多信息,挖掘用户意图,消除歧义性问题。

本文检索了 2012－2021 年间与实例编程相关的论文,选取了 38 篇高水平论文进行分析。我们利用谷歌学术、ACM Digital Library 和 IEEE Xplore Digital Library 等检索系统,以 programming by examples 为关键词进行检索。论文分别来自 ICSE, FSE, ASE, CAV, POPL, PLDI, OOPSLA, NIPS, ICML, AAAI 和 ICLR 等软件工程、程序语言与设计和人工智能 3 个领域的会议。

本文综述了实例编程研究的进展与挑战。首先介绍实例编程的研究背景和当前状态,进而阐述实例编程的一般过程与两个方面的挑战。本文针对近年来实例编程领域解决两方面挑战的方法,分为搜索策略和歧义性消除两部分论述。其中,针对搜索策略,本文进行了特定领域语言(Domain Specific Language,DSL)的选择和搜索算法设计的分类与总结。最后总结并展望了实例编程未来的技术挑战与趋势。

2 实例编程简介

通常,一个实例编程问题包含两方面的规约:给定的语法 G 和一组输入-输出实例对 $\varphi=\{(i_1,o_1),\cdots,(i_n,o_n)\}$。实例编程旨在找到一段满足语法 G 的程序 P,使得 $\forall(i,o)\in\varphi$, $P[i]=o$,即对于属于 φ 中的任意一组输入输出 (i,o),输出 o 均可以由输入 i 经过程序 P 的操作得到。一般来说,为了降低研究的复杂性,实例编程会使用 DSL 从语法上限制程序空间的大小。DSL 的选取和设计对于实例编程来说非常重要,它既需要对待解决的问题有足够的表现力,又需要尽可能地降低搜索的难度[8]。

实例编程的一般求解过程如图 1 所示。首先需要有预先设计的 DSL 来表示程序空间,其次需要用户提供的表示需求的输入输出实例集合。求解器(Solver)在程序空间中进行搜索,得到的候选程序要经过所有给定的 (i,o) 数据验证,验证通过的程序进入候选程序集合,若不通过,则求解器重新搜索,直到找到验证通过的程序。候选程序集合进行排序筛选,得到候选程序 P 即可返回作为合成结果。当有额外约束,如时间、程序语法树深度等限制求解器时,不满足额外约束时求解过程会直接停止。

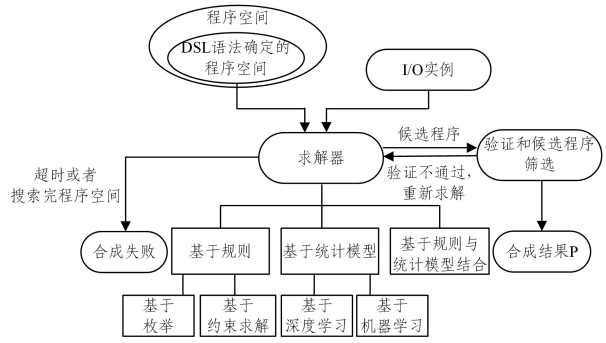


图 1 实例编程求解过程示意图

Fig. 1 Process of solving PBE

实例编程存在程序合成任务面临的普遍问题——庞大的程序搜索空间。即便使用了 DSL,实例编程面对的搜索空间依然是庞大的。例如,RobustFill[9]提供了足以表达常用字符串变换操作程序空间的 DSL 语法。随着程序的大小增长,其对应的程序搜索空间也会呈指数型增长。除了需要在庞大的程序搜索空间中找到符合规约的程序外,实例编程被应用在与用户有交互的场景中,这类场景对算法的实时性提出了更高的要求。

表 1 一组输入输出实例

Table 1 A set of (i,o)

i	o
Apple	apple
Banana	banana
Orange	orange
Grape	grape
Lemon	lemon

另一方面,求解器求解时,面临着潜在的歧义问题,因为用户能提供的实例往往是有限的,如 SyGuS 数据集中,每个字符串相关的问题提供实例的数量在 2～400 之间。有限的实例规约是一种不完备的规约[7],无法准确地描述用户的意图。程序空间往往存在多个行为一致、均能满足实例规约的代码,因此求解器合成时会面临潜在的歧义性问题。例如表 1 中的实例,程序 $p=$ ToCase(Lower) 和程序 $p=$ Concat(ToCase(Lower, GetAll(AllCaps), GetAll(lower))) 均符合实例(前者表示字符串中字符全部转小写,后者表示对大写部分转小写之后再与小写部分连接),即存在若干个程序 P,均

满足$\forall(i,o)\in\varphi,P[i]=o$,但这若干程序并不一定都满足用户的真正意图。对于实例编程来说,如何通过排序筛选手段选择出符合用户真正意图的程序是一个挑战。

3 搜索策略研究现状分析

面对庞大的程序搜索空间,通常一个合适的搜索策略首先会选取DSL来取代通用编程语言作为语法约束,其次需要设计合适的搜索算法。本节分别从DSL和搜索算法的设计两个方面来论述。

3.1 DSL的选择与设计

DSL适用于特定领域,通常用上下文无关语法表示,比通用编程语言(如Java和python等)有更多的限制,例如没有循环结构。与通用编程语言相比,DSL使用领域的特性,对领域内的问题有更简洁的表达,但是表达的范围有限,如针对字符串变换领域的DSL只处理字符数据,限定字符串替换、连接等操作。实例编程通常选择DSL进行程序合成,因为通用编程语言丰富的关键字和函数等会进一步扩大程序搜索空间。

DSL的选择与设计对搜索算法与后续的歧义消除过程均有影响[8]。表2列举了几个应用领域中使用的DSL及其特点。

表2 不同应用领域中的DSL总结

Table 2 Summary of DSL in different applications

应用领域	文献	DSL特点
字符串变换	[5,9-11]	包括字符串的连接、获取子串、字符替换、正则匹配等常见字符串相关操作
数据表处理	[8,12-13]	包括数据行列获取、数据项过滤、谓词连接、行列等价判断等操作
机器人	[14-16]	包括二维坐标系内机器人的4个方向上的移动、捡起物体和放下物体的操作,有循环和条件结构
软件工程	[17-19]	包括抽象语法树上节点的增、删、改、移,变量对应语法中节点的类型

通常实例编程选择的DSL需要能清晰刻画待解决的问题,关注领域问题内在逻辑的完备,设计中需要考虑包含的操作与运算、程序结构等因素。常见的实例编程研究以字符串变换、数据处理等作为任务。Gulwani[6]使用了一种针对字符串变换的DSL,包括子串、正则表达式、连接等常见处理字符串的操作,而Devlin等[9]在其基础上去掉了DSL中的循环结构,增加了字符类型的限制和获取正则匹配结果的操作。Harris等[12]针对数据表操作设计了一种名为TableProg的DSL,包括表格的过滤、取同行同列等操作。还有一些针对数据表的实例编程任务使用SQL或者类SQL的DSL[8,13]。

这些DSL缺乏现代语言具备的循环和条件语句,可表示的问题有限。跟上述DSL相比,Karel包含控制结构,其实例编程任务更加困难。为了降低难度,Karel简化了求解的问题,模拟一个虚拟机器人在二维坐标系中的走动,具备放捡物体的功能。在近年来的工作中,Karel成为了主要研究的目标语言[14-16]。

一些工作针对代码重构等软件工程领域任务设计DSL。Rolim等[17]针对代码重构设计DSL,其中一次代码重构被定义为一系列应用于抽象语法树的重写规则。每条重写规则对应着重构前后语法树上对应子树间的修改,包括插入、删除、更新和前置节点的操作。与Rolim等的工作类似,Le等[18]针对代码修复问题,设计了一种涉及整型和布尔型数据的DSL,其中包括了关系运算符、逻辑运算符和算术运算符。该DSL合成的程序通过GumTree[20]应用到抽象语法树上实现修复。

3.2 搜索算法的设计

最朴素的搜索策略就是遍历整个程序空间,直到找到符合规约的程序。然而,这样的做法难以应对庞大的搜索空间。普遍的做法是利用程序的特性来减小搜索空间。一些方法通过设计规则,挖掘规则在程序中的表现,来缩小程序空间。还有一些方法利用统计模型学习程序概率分布来加速搜索。本文根据搜索算法的指导思想将现有的算法分为基于规则、基于统计模型和基于规则与统计模型相结合三大类。其中基于规则的方法可以分为基于枚举和基于约束求解两类,基于统计模型的方法可以根据学习分布的方式分为基于机器学习和基于深度学习两类。表3列出了实例编程现有工作中的搜索算法的技术特点。

表3 实例编程现有工作搜索算法总结

Table 3 Summary of searching algorithms in PBE

方法类别		文献	技术特点
基于规则	基于枚举	[21-26]	按照特定顺序对程序空间中的程序枚举,根据枚举方式选择基于语法或执行结果的剪枝策略
	基于约束求解	[27-29]	利用提供的语法信息,如函数等,先合成不完整程序,再根据实例利用约束求解器等求解缺失部分
基于统计模型	基于机器学习	[30]	设计实例、DSL中产生式的相关特征,根据实例中出现的特征改变不同产生式的概率,按照概率组合形成最后的程序
	基于深度学习	[14,31-32]	神经程序归纳,实例编码成向量,直接使用网络作为黑盒的程序表示,通过对网络进行特殊结构的设计达到对程序结构的模拟
		[8-10,15]	神经程序合成,一般采用编码器-解码器的架构,编码器学习实例中的特征,解码器将特征与DSL中的原语或产生式相关,解码器按概率输出原语或产生式合成程序
基于规则与统计模型结合		[33-37]	枚举与统计模型相结合,通过实例学到不同子程序的概率或者不同原语或产生式的概率,改变枚举的优先级和范围
		[11,38-39]	规则约束与统计模型相结合,通过统计模型得到不完整程序,利用约束求解或者演绎推理验证不完整程序的可行性,或对缺失部分进行求解

3.2.1 基于规则的搜索算法

基于规则的搜索算法包括枚举和约束求解两种基本思想。枚举方法通常按照一定的语法规则直接对程序空间进行遍历,同时会结合特定的剪枝规则。在程序较短的场景中,这类方法比其他需要复杂计算的方法更有效[2]。但当程序变长时,待枚举的空间呈指数级增长,枚举方法的效率严重下降。约束求解类方法尝试将搜索问题简化为约束求解问题[7]。这类方法通常会将语法中包含的语义关系转为表示程序和实例的变量间的约束,再利用已有的约束求解器如 SMT 等求解。当程序复杂度上升时,语义关系的描述也变得复杂。基于规则的方法可以很好地利用程序语法结构中的知识,但由于复杂度随着程序规模的增加而提升,因此需要利用其他手段预先减小程序的搜索空间。

常见的枚举方式可以根据枚举方向分为自顶向下和自底向上两种。自顶向下的方式从程序的语法树根节点向叶节点扩展,将不完整程序扩展至完整程序。这类枚举常与基于语法的程序等价的剪枝策略组合。Feser 等[21]使用类型感知(Type-aware)的归纳,并将用户提供的实例泛化为关于程序的一组假设,使用演绎推理为假设寻找解,最后依赖枚举探索假设约束下的程序空间。Osera 等[22]提出了一种 Refinement Trees 来表示待合成代码的约束,利用类型信息将实例往叶节点方向靠近来修剪程序空间,实现在自顶向下枚举过程中评估程序的效果。自底向上的方式从语法树叶节点出发,先枚举叶节点,再到子表达式,最后将子表达式组合形成最终的语法树。实例编程中,自底向上的枚举常与基于观测等价(Observational Equivalence)的剪枝策略组合[23-25],即在输入实例上有相同输出的子表达式被视为等价,之后的枚举过程只需要其中一个子表达式参与。尽管使用基于观测等价的优化,自底向上枚举依然无法应对程序空间的指数式增长。为了解决此问题,Lee[26]尝试结合自底向上枚举和自顶向下扩展(Top-down Propagation),利用后者先将合成问题递归分解为规模较小的问题,再利用前者求解小问题,最后组合得到合成结果。

约束求解类方法尝试利用额外的信息限制搜索空间,常见的做法有两类。一类是基于组件(Component-based)的方式,使用组件作为合成程序的语法偏差(Syntactic Bias)。组件指合成程序中可能会出现的函数、API 等,如 Shi 等[27]根据给定的 Java 库函数、目标程序的方法签名和输入输出实例合成带有控制结构的 Java 程序。另一类是基于草图(Sketch-based)的方式,把程序的合成分为草图的生成与补全两部分[12]。草图表示程序的高层次意图,可以由用户提供,草图中缺失的部分可以通过实例求解出,如 Wang 等[28]针对表格数据,利用有限树自动机,根据实例对所提供的草图中的缺失部分进行合成。有的工作将基于组件和基于草图相结合,如 Feng 等[29]先利用给定的组件合成草图,再利用 SMT 推断草图的可行性,结合类型指导的搜索填补草图。

3.2.2 基于统计模型的搜索算法

基于规则的搜索方法需要人工设计剪枝规则和语法对应的语义。这样的人工设计系统难以扩展,且易被噪声数据破坏[9]。为了解决这类问题,近年来,很多工作将重心转移至基于统计模型的方法。基于统计模型的方法尝试学习整个程序空间上的概率分布,并将其作为合成的先验知识,将搜索导向符合实例规约可能性高的方向。本节将收集的文献中基于统计模型的搜索算法划分为基于机器学习和基于深度学习两大类。

基于机器学习的搜索算法针对实例设计特征,通过输入输出实例为语法中的不同产生式附加概率,组合概率高的产生式作为合成的程序。Menon 等[30]对比实例中的输入输出,将其间的特征联系称为规则(Rule),再与语法中的具体产生式关联,这样的关联被称为线索(Clue)。通过在输入输出实例中查找线索,为线索对应的产生式加权,利用这样的概率指导程序空间中的搜索。在实例编程中单独使用机器学习的方法较少,大多数工作选择将其与枚举搜索等其他方法结合。

机器学习方法面临着人工设计特征耗时、易出错的问题,深度学习有自动学习特征的能力,可用于解决这个问题,因此越来越多的工作尝试使用深度学习。利用深度学习神经网络合成实例编程的程序有两种思路。一种是神经程序归纳(Neural Program Induction)。这类方法使用网络学习程序的潜在表示,直接生成给定输入的输出,即将网络本身作为一种黑盒的程序,网络的设计往往受计算机内部计算结构的启发,如堆栈结构。Reed 等[31]将程序表示为向量,使用循环和组合神经网络来学习、表示和执行程序,网络的训练数据包含了子程序调用结构,通过网络学习如何组合这样的子程序来完成复杂任务。类似地,Chen 等[32]尝试学习操作不可微分机器的神经网络,将学习到的程序限制在特定空间内,加入递归的概念,使得学习到的神经网络程序泛化性更高。这类神经程序归纳方法的弊端主要体现在无法得到程序的可解释模型,且泛化性难以保证。

另一类基于深度学习的搜索算法被称为神经程序合成(Neural Program Synthesis)。这类方法利用网络学习输入输出实例中的特征,取代以往的人工设计特征过程,再由网络预测输出程序语句。为了充分学习特征,这类方法需要人为构造大量实例作为训练数据。这类方式常采用编码器-解码器(Encoder-decoder)的架构,其中编码器学习数量不定的实例中的特征,解码器输出程序或限制程序范围。Balog 等[8]针对一种类似 SQL 的查询 DSL 进行合成,设计了 DeepCoder,采取了编码器-解码器的网络架构,编码器从输入输出实例中提取特征,解码器预测程序中可能出现的关键字或不可能出现的关键字,限制程序的搜索范围。DeepCoder 利用网络增强了外部的搜索算法,而不是采用更为普遍的端到端模型。之后,Devlin 等[9]针对 FlashFill 的字符串变换任务设计了端到端的预测网络,使用注意力机制和循环神经网络编码输入输出实例,再解码输出组成程序的令牌(Token),并证明了模型可以处理基于规则的方法无法处理的噪声数据,如某些词的错误拼写。

神经程序合成工作采用的编码器-解码器架构在很多简单任务上取得了不错的效果,但在复杂任务上性能会有所下降[15]。大部分工作利用网络学习输入输出实例与语法之间的映射关系,以此得到不同语法在最终程序中出现的概率,这种方式往往忽视了输入输出实例和语义之间的联系。近年来,神经程序合成工作中出现了一类强调执行指导(Execu-

tion Guided)的方法。这类方法尝试利用语义信息引导网络训练,普遍认为程序的执行轨迹,即某个输入实例在程序操作下的变化过程,或者程序的中间执行结果,可以作为程序的语义信息,利用这样的语义信息一次生成一条语句,而不是整个程序。Chen 等[15]改写 Karel 的条件语句和循环语句,以便在编码器-解码器的架构下利用程序的中间执行结果,同时采用集成学习进一步提高合成的准确率。Ellis 等[10]对该想法进行了扩展,他们训练网络使其能评估中间程序的执行状态,在字符串变换任务和二维三维图形推断任务中设计了在 actor-critic 网络中使用程序中间执行结果优先搜索,通过合成-验证的循环利用程序的语义。这类利用程序语义信息来指导搜索的方式往往需要未合成完毕的程序的执行结果是网络可感知的。

3.2.3 基于规则与统计模型结合的搜索算法

与自然语言类似,对于程序语言来说,每个离散的原子结构是有含义的,难以映射到状态空间,但与自然语言不同的是,程序语言中的词汇量小[40],这使得统计模型要从简单的实例中学习复杂程序功能变得困难。近年来,越来越多的方法尝试结合基于规则的方法和统计模型的优势,即统计模型对程序模式的挖掘和基于规则的方法在程序结构与符号逻辑方面的处理,利用统计模型限制待搜索的程序空间,利用基于规则的方法在限制后的空间内构建有意义的程序。

Alur 等[33]利用分治思想,将枚举算法与决策树模型相结合,将程序语法拆分为项(Terms)和谓词(Predicates),两部分分别枚举后再训练决策树进行组合。Lee 等[34]、Ji 等[35]和 Barke 等[36]尝试使用基于结构概率的概率模型来加速程序合成。其中 Lee 等和 Barke 等在枚举过程中根据结构概率,优先枚举结构概率高的子表达式,后者在枚举过程中使用输入实例验证不同的子程序,从而更新不同产生式的概率,比前者减少了对训练过程的依赖。而 Ji 等认为动态规划是解决实例编程的一种重要思路,他们用结构概率指导动态规划过程来解决实例编程问题。与 Barke 等[36]类似,Odena 等[37]在自下而上枚举过程中训练了一个简单的二分类器,同时采取了与执行指导类似的思想,根据枚举过程的中间程序在给定输入实例上的输出,利用二分类器判断其是否可能出现在最终合成结果中,从而加快枚举搜索的速度。

还有一些工作尝试在统计模型合成中使用规则约束或逻辑推理,利用统计模型在模式识别上的优势先学习实例的特征,以获得一个不完整的解,再利用约束求解等手段获得完整的程序。Nye 等[11]利用网络生成程序草图,作为待合成程序的模板约束,再根据具体的输入输出实例对草图中缺失的表达式进行枚举。Bavishi 等[38]在利用图神经网络为 pandas 库中的 API 合成使用代码时,加入 API 文档中的规则约束,如参数类型,在合成中验证约束以加快合成速度。Chen 等[39]将强化学习和演绎推理相结合,在合成过程中,强化学习的奖励机制,使用求解器来检查不完整程序的不可行性,使得可能扩展出不可行程序的产生式的概率在下一次探索中变小。

4 歧义性消除研究现状分析

实例编程中歧义性消除的普遍思路是设计排序规则。排序筛选使得越有可能符合用户意图的候选程序拥有越高的排名[7]。表 4 列出了实例编程歧义性消除的现有工作。排序规则的设计可以按照是否需要用户提供额外信息划分为两类,一类基于给定实例,一类基于交互获取额外信息。

表 4 实例编程歧义性消除现有工作总结

Table 4 Summary of ambiguity resolution in PBE

方法类别	文献	技术特点
基于给定实例	[41-42]	设计与实例或程序结构特征相关的排序函数,利用实例训练排序函数
基于用户交互	[43-46]	利用用户提供的额外实例或选择,对候选程序进行排序或者消除可能有歧义的程序

第一类不需要用户提供额外信息的方法,基于给定实例,根据实例特征和程序结构特征进行排序,例如,朴素地认为越短的程序排名越高。这类方法的排序过程是静态的,排序结果获得后无法改变。一些工作利用机器学习根据设计好的特征训练出排序函数。Singh 等[41]针对字符串变换的程序选择了字符位置、原子(Atom)表达式和拼接(Concat)表达式作为特征,通过适当的损失函数,保证预期程序排名高于非预期程序,学习这些特征的权重。除了设计排序函数,也有工作尝试对用户提供的实例进行增强。An 等[42]尝试挖掘关系置换属性,即输出的变化和输入的变化之间的关系,他们认为这样的置换属性可以在程序搜索空间中加入语义偏差,增强用户提供的原有实例。这类方法的弊端在于相关的特征需要人工设计,设计的特征和实例编程求解问题的所属领域关联较大,鲁棒性不够,难以迁移。

第二类需要用户提供额外信息的方法会通过交互等其他手段,如利用用户提供的额外实例对候选程序进行筛选,这类方法的排序过程是动态的,排序结果可以随交互而改变。Mayer 等[45]将候选程序翻译为自然语言向用户展示,允许用户浏览合成出来的程序并进行选择,同时会主动向用户展示有歧义的数据并允许用户主动修改。此方法依赖用户直接进行选择,翻译的成本略高。其他的工作尝试让用户提供额外的实例。Ellis 等[46]尝试不依赖与程序结构相关的特征,设计了与程序行为相关的特征,利用用户提供的额外输入在候选程序上的输出与用户提供的相应额外输出之间的相似度,对候选程序进行重新排序。Narita 等[43]针对如何让用户提供额外实例的问题定义了两种交互模型,一种需要经过多轮的用户选择指定输入的候选输出来消除可能有歧义的程序,另一种需要用户判定指定的输入输出实例是否匹配,从而对候选程序进行二分后排序。这类工作都需要在程序执行前就确认好最终的候选程序,而 Raza 等[44]则认为不需要过早地排序出最有可能的候选程序,他们将排序步骤延迟到程序执行时,保存多种候选程序,直至用户提供额外实例时再选择最佳程序,这种方式使得他们的排序策略更具动态性。

5 前景与展望

总体来看,实例编程是一个备受软件工程、人工智能等领域关注的研究方向,并且已经有成功落地的应用。未来实例编程研究可以在以下几个方面进行进一步的发展。

(1)很多工作尝试设计基于统计模型与规则相结合的

搜索算法。其中很多基于深度学习的方法在训练过程中需要预先人为地构造实例，如何进行更合理的表示学习、减少对人为构造实例的依赖值得研究。

(2)实例编程算法目前主要依赖于语法导向，如通过概率等判断某个语法产生式出现的可能性，目前对语义信息的利用集中在利用语义来验证某个候选程序是否可行。现实中，程序员在写代码的过程中会更多地判断某个语义是否出现，如何将语法导向转变为语义导向是一个值得探索的方向。

(3)通过人机交互的方式做到歧义性消除是一个发展趋势。交互可以使用户提供关于意图的额外信息，从而对候选程序进行排序。现有的工作往往需要较多的交互次数，未来可以思考如何用较少的交互次数达到较好的排序效果。

结束语 实例编程是一类重要的程序合成范式，它使得非程序员的终端用户可以通过简单实例利用计算机完成日常任务。这种编程范式可以给多个领域带来编程交互方式的革新。本文总结了实例编程领域面临的主要挑战，对目前领域内解决挑战的主要方法进行了分类介绍，并在此基础上对实例编程研究的发展进行了展望。

参 考 文 献

[1] HU X, LI G, LIU F, et al. Program Generation and Code Completion Techniques Based on Deep Learning: Literature Review [J]. Journal of Software, 2019, 30(5): 1206-1223.

[2] GULWANI S, POLOZOV O, SINGH R. Program synthesis[J]. Foundations and Trends in Programming Languages, 2017, 4(1/2): 1-119.

[3] SUMMERS P D. A methodology for LISP program construction from examples[J]. Journal of the ACM (JACM), 1977, 24(1): 161-175.

[4] GKIOKAS A. Imitation learning in artificial intelligence[D]. Coventry: University of Warwick, 2016.

[5] LÁZARO-GREDILLA M, LIN D, GUNTUPALLI J S, et al. Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs[J]. arXiv: 1812. 02788, 2018.

[6] GULWANI S. Automating string processing in spreadsheets using input-output examples[C]// Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 2011: 317-330.

[7] GULWANI S, ESPARZA J, GRUMBERG O. Programming by Examples (and its applications in Data Wrangling)[J]. Dependable Software Systems Engineering, 2016, 45(137): 3-15.

[8] BALOG M, GAUNT A L, BROCKSCHMIDT M, et al. DeepCoder: Learning to Write Programs[C]// International Conference on Learning Representations (ICLR 2017). OpenReview. net, 2017.

[9] DEVLIN J, UESATO J, BHUPATIRAJU S, et al. Robustfill: Neural program learning under noisy i/o[C]// International Conference on Machine Learning. PMLR, 2017: 990-998.

[10] ELLIS K, NYE M, PU Y, et al. Write, execute, assess: program synthesis with a REPL[C]// Proceedings of the 33rd International Conference on Neural Information Processing Systems. 2019: 9169-9178.

[11] NYE M, HEWITT L, TENENBAUM J, et al. Learning to infer program sketches[C]// International Conference on Machine Learning. PMLR, 2019: 4861-4870.

[12] HARRIS W R, GULWANI S. Spreadsheet table transformations from examples[C]// Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. 2011: 317-328.

[13] WANG C, CHEUNG A, BODIK R. Synthesizing highly expressive SQL queries from input-output examples[C]// Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2017: 452-466.

[14] DEVLIN J, BUNEL R, SINGH R, et al. Neural program meta-induction[C]// Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017: 2077-2085.

[15] CHEN X, LIU C, SONG D. Execution-guided neural program synthesis[C]// International Conference on Learning Representations (ICLR 2018). OpenReview. Net, 2018.

[16] BUNEL R, HAUSKNECHT M, DEVLIN J, et al. Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis[C]// International Conference on Learning Representations (ICLR 2018). OpenReview. Net. 2018.

[17] ROLIM R, SOARES G, D'ANTONI L, et al. Learning syntactic program transformations from examples[C]// 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, 2017: 404-415.

[18] LE X B D, CHU D H, LO D, et al. S3: syntax-and semantic-guided repair synthesis via programming by examples[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 593-604.

[19] BAVISHI R, YOSHIDA H, PRASAD M R. Phoenix: Automated data-driven synthesis of repairs for static analysis violations [C]// Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019: 613-624.

[20] FALLERI J R, MORANDAT F, BLANC X, et al. Fine-grained and accurate source code differencing[C]// Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. 2014: 313-324.

[21] FESER J K, CHAUDHURI S, DILLIG I. Synthesizing data structure transformations from input-output examples[J]. ACM SIGPLAN Notices, 2015, 50(6): 229-239.

[22] OSERA P M, ZDANCEWIC S. Type-and-example-directed program synthesis[C]// Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2015: 619-630.

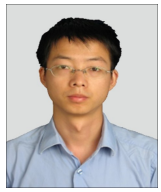
[23] ALBARGHOUTHI A, GULWANI S, KINCAID Z. Recursive program synthesis[C]// International Conference on Computer Aided Verification. Berlin: Springer, 2013: 934-950.

[24] ALUR R, BODIK R, JUNIWAL G, et al. Syntax-Guided Synthesis[C]// 2013 Formal Methods in Computer-Aided Design (FMCAD). IEEE, 2013: 1-8.

[25] PELEG H, POLIKARPOVA N. Perfect is the Enemy of Good: Best-Effort Program Synthesis[C]// 34th European Conference on Object-Oriented Programming (ECOOP 2020). Schloss Dags-

tuhl-Leibniz-Zentrum für Informatik,2020:1-30.
[26] LEE W. Combining the top-down propagation and bottom-up enumeration for inductive program synthesis[J]. Proceedings of the ACM on Programming Languages,2021,5(POPL):1-28.
[27] SHI K,STEINHARDT J,LIANG P. Frangel:component-based synthesis with control structures[J]. Proceedings of the ACM on Programming Languages,2019,3(POPL):1-29.
[28] WANG X,DILLIG I,SINGH R. Synthesis of data completion scripts using finite tree automata[J]. Proceedings of the ACM on Programming Languages,2017,1(OOPSLA):1-26.
[29] FENG Y,MARTINS R,VAN GEFFEN J,et al. Component-based synthesis of table consolidation and transformation tasks from examples[J]. ACM SIGPLAN Notices,2017,52(6):422-436.
[30] MENON A,TAMUZ O,GULWANI S,et al. A machine learning framework for programming by example[C]// International Conference on Machine Learning. PMLR,2013:187-195.
[31] REED S,DE FREITAS N. Neural programmer-interpreters[J]. arXiv:1511.06279,2015.
[32] CHEN X,LIU C,SONG D. Towards Synthesizing Complex Programs From Input-Output Examples[C]// International Conference on Learning Representations(ICLR 2018). OpenReview. Net. 2018.
[33] ALUR R,RADHAKRISHNA A,UDUPA A. Scaling enumerative program synthesis via divide and conquer[C]// International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Berlin:Springer,2017:319-336.
[34] LEE W,HEO K,ALUR R,et al. Accelerating search-based program synthesis using learned probabilistic models[C]// 39th ACM SIGPLAN Conference on Programming Language Design and Implementation,PLDI 2018. Association for Computing Machinery,2018:436-449.
[35] JI R,SUN Y,XIONG Y,et al. Guiding dynamic programing via structural probability for accelerating programming by example [J]. Proceedings of the ACM on Programming Languages,2020,4(OOPSLA):1-29.
[36] BARKE S,PELEG H,POLIKARPOVA N. Just-in-time learning for bottom-up enumerative synthesis[J]. Proceedings of the ACM on Programming Languages,2020,4(OOPSLA):1-29.
[37] ODENA A,SHI K,BIEBER D,et al. BUSTLE:Bottom-Up Program Synthesis Through Learning-Guided Exploration[C]// International Conference on Learning Representations. 2020.
[38] BAVISHI R,LEMIEUX C,FOX R,et al. AutoPandas:neural-backed generators for program synthesis[J]. Proceedings of the ACM on Programming Languages,2019,3(OOPSLA):1-27.
[39] CHEN Y,WANG C,BASTANI O,et al. Program Synthesis Using Deduction-Guided Reinforcement Learning[C]// International Conference on Computer Aided Verification. Cham:Springer,2020:587-610.
[40] KANT N. Recent advances in neural program synthesis [J]. arXiv:1802.02353,2018.
[41] SINGH R,GULWANI S. Predicting a correct program in programming by example[C]// International Conference on Computer Aided Verification. Cham:Springer,2015:398-414.
[42] AN S,SINGH R,MISAILOVIC S,et al. Augmented example-based synthesis using relational perturbation properties[J]. Proceedings of the ACM on Programming Languages, 2019, 4(POPL):1-24.
[43] NARITA M,MAUDET N,LU Y,et al. FlashAttention:Data-centric Interaction for Data Transformation Using Programming-by-Example[C]// Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology. 2020:65-67.
[44] RAZA M,GULWANI S. Disjunctive program synthesis:A robust approach to programming by example[C]// Proceedings of the AAAI Conference on Artificial Intelligence. 2018: 1403-1412.
[44] MAYER M,SOARES G,GRECHKIN M,et al. User interaction models for disambiguation in programming by example[C]// Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. 2015:291-301.
[46] ELLIS K,GULWANI S. Learning to learn programs from examples:going beyond program structure[C]// Proceedings of the 26th International Joint Conference on Artificial Intelligence. 2017:1638-1645.

YAN Qian-yu, born in 1997, postgraduate. Her main research interests include program synthesis and software engineering.



LI Yi, born in 1975, Ph.D, lecturer, is a member of China Computer Federation. His main research interests include robot software, program analysis and computer system architecture.

(责任编辑:何杨)