计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.20200586
59(1):236-250,2022

基于 Petri 网可达分析的代码搜索方法

丁雪儿[1] 钮 俊[1,2] 张开乐[1] 毛昕怡[1]
[1](宁波大学信息科学与工程学院 浙江宁波 315211)
[2](嵌入式系统与服务计算教育部重点实验室(同济大学) 上海 201804)
(dingxuer@yeah.net)

Code Search Method Based on the Reachability Analysis of Petri Nets

Ding Xue'er[1], Niu Jun[1,2], Zhang Kaile[1], and Mao Xinyi[1]
[1](*Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, Zhejiang* 315211)
[2](*Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education, Shanghai* 201804)

Abstract Reusing existing high-quality source codes can improve efficiency of software development and quality of software. At present, code search based on input/output queries provided by users is one of the main approaches in the field of code semantic search, but existing approaches are difficult to describe the complete behavior of codes and can only handle a single input type. This paper proposes a code semantic search approach based on the reachability analysis of Petri Nets for matching multiple forms of type. First, the semantic processes of code snippets consisting of the number of data objects and types of data objects in the code corpus are converted into improved Petri Net models. Second, the initial marking and target marking of Petri Net models are constructed according to the number of data objects and types of data objects contained in users' queries. Matching code snippets is obtained by the analysis of reachable paths in reachability graphs and induced networks of Petri Nets. Analysis and experimental results show that this approach contributes to seeking out desired code snippets by queries that possess multiple forms of input/output types provided by users, and compared with traditional approaches, it can significantly improve accuracy and efficiency of code search.

Key words code reuse; semantic search; matching type; Petri Net model; reachability analysis

摘 要 复用已有高质量源代码可提高软件开发效率及软件质量.当前,基于用户提供的输入/输出对的匹配判断是代码语义搜索的主要方法之一,但该方法难以刻画完整代码行为,且仅能处理单输入类型.提出一种针对多种形式类型匹配的代码语义搜索方法.首先将代码集内各个代码片段中数据对象个数及类型的加工过程转换为 Petri 网模型;其次根据用户查询中蕴含的数据类型及个数、输出数据类型等约束来构造 Petri 网初始标识和目标标识;然后在 Petri 网中通过可达图及诱发网分析判断是否存在相应的可达路径,从而获得代码匹配依据.分析及实验表明,该方法能有效实现多种形式的输入/输出类型匹配的代码搜索,且相对于传统类型匹配方法,能明显提高搜索准确度和效率.

关键词 代码复用;语义搜索;类型匹配;Petri 网模型;可达分析

中图法分类号 TP311

收稿日期:2020-07-31;**修回日期**:2021-02-22

基金项目:国家自然科学基金项目(61672384);嵌入式系统与服务计算教育部重点实验室开放课题(ESSCKF2019-07);宁波市自然科学基金项目(2019A610088)

This work was supported by the National Natural Science Foundation of China (61672384), the Open Project of Key Laboratory of Embedded System and Service Computing of the Ministry of Education of China (ESSCKF2019-07), and the Natural Science Foundation of Ningbo (2019A610088).

通信作者:钮俊(niujun@nbu.edu.cn)

随着计算机技术的发展和应用的逐渐深入,出现大量复杂大规模软件系统.软件系统功能的不断增强与完备等也正导致软件复杂度急剧增加,给设计、开发和部署、运维等带来新的挑战.通过代码复用来构造软件系统可提高软件开发效率,且对已被成功使用的各种代码的复用还可提高软件质量、降低软件风险[1].随着开源运动的兴起,互联网上已聚集大量开放代码可被学习和使用[2];同时,企业组织内部也存在大量已被成功实践的程序代码[3],为基于复用的高效率、低成本软件开发提供了基础和条件.

为了复用已有代码,首先需对已有代码进行有效组织管理,而快速、准确地搜索到期望代码则是代码复用的关键.当前已存在大量针对代码复用的代码搜索方法,从程序接口、语义、语法等角度去匹配候选代码[4].其中基于语义的代码搜索方法主要从用户期望程序的功能或行为的角度去搜索代码,能够更为快速、准确地定位用户所需代码,近年来获得学术界、工程界重点关注[5].代码、语句的功能或行为一般体现为对输入数据进行处理并获得输出结果,故基于输入/输出匹配的代码搜索是一种典型的基于语义的代码搜索方法.该方法[6-8]需要用户用多组输入输出对的形式给出自己期望代码的模糊功能描述,通过静态分析或符号执行等技术判断候选代码能否对给定输入获得期望输出.2014 年,美国爱荷华州立大学的 Stolee 等人[6]提出一种基于输入/输出值匹配的代码搜索方法,该方法通过符号分析技术将顺序结构程序代码的语义转换为逻辑表达式,同时将用户提供的输入/输出值对转换为语义约束,借助于可满足性模理论(satisfiability modulo theories, SMT)求解器对二者的组合进行求解,进而判断是否匹配.2016 年,文献[7]对该方法进行扩展,使其能够支持对分支结构代码的分析,并通过执行路径覆盖度排序等手段解决多路径代码的匹配.该方法能在一定程度上提高搜索准确度和效率,但也存在一些局限性:1)由于计算过程的复杂性,部分程序代码的输出值难以直接给出;2)用户只能提供有限的输入/输出值,且难以给出一些特殊的边界值;3)该方法难以应对如连接数据库等不具有显式输入/输出值的代码片段.

为了弥补上述不足,一种有效做法是用户在以输入/输出对形式进行功能描述时,不必给出具体值,仅给出输入输出数据的类型即可,从而提出基于输入/输出类型匹配的代码搜索方法[9-12].该类方法一般将代码功能行为表示为数据对象之间的类型转换图,基于用户提供的输入/输出类型,通过对图进行遍历以搜索从输入类型到输出类型的子图,每个子图所对应代码序列则构成一个解.这种方法忽略具体数值,从数据类型出发,在较高语义层次搜索代码,具有较好实用性,但已有工作也存在不足.首先,在查询形式上已有方法仅能处理单个输入类型,缺乏对多个输入类型匹配问题的研究.其次,在类型分析过程中,对具有多个同类型输入参数的情形缺乏分析.

针对上述问题,本文提出一种基于 Petri 网可达分析的代码语义搜索方法,整个搜索过程如图 1 所示.首先,借助程序分析技术解析候选代码,构建候选代码数据类型转换过程的 Petri 网模型.其次,对于用户提供输入/输出类型对,利用改进后的 Petri 网可达分析算法在 Petri 网上搜索从输入类型到输出类型的可达路径,并返回该路径所对应代码片段.最后采用路径长度和代码复用率 2 个评价指标对返回结果进行重排序.注意本文以“函数”粒度源代码作为分析对象.

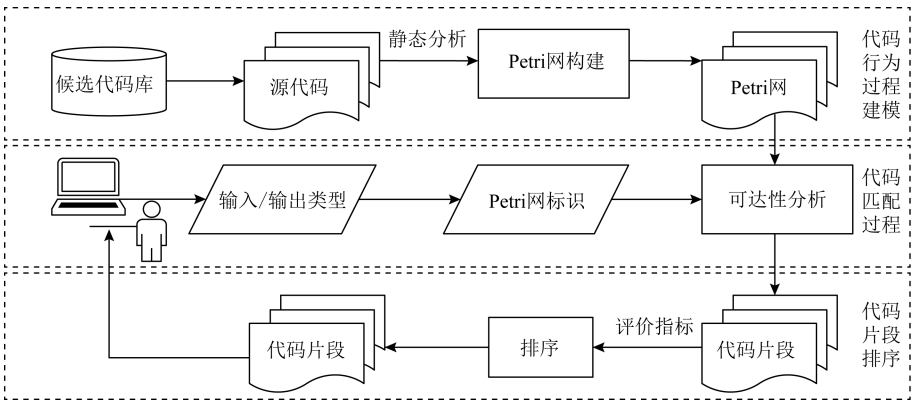


Fig. 1 Overview of code search method based on the reachability analysis of Petri Nets

图 1 基于 Petri 网可达分析的代码搜索方法概览图

本文的主要贡献有2个方面：

1) 提出一种基于Petri网的代码行为建模方法，以描述代码对数据类型的加工变换过程.与已有工作中所采用的图模型不同，本文基于Petri网对代码行为过程作进一步解析，采用Petri网有向弧表示类型间的转换，有向弧权重表示转换过程中参数类型个数，能够准确表达代码语句及数据对象依赖关系.

2) 提出一种基于Petri网可达分析的代码搜索方法.采用Petri网输入/输出标识表示用户提供查询中输入/输出数据对象的类型及个数，利用Petri网可达算法匹配由输入标识到输出标识的目标代码，能够有效地解决多种形式输入/输出类型的搜索问题；同时在基础Petri网可达算法上引入诱发网分析，以提高搜索效率.

1 相关工作

目前，基于输入/输出匹配的代码搜索方法大致分为2类：基于输入/输出测试值的代码匹配方法[6-8]和基于输入/输出类型的代码匹配方法[9-12].这2种方法的相同之处在于都是通过给定的输入/输出信息样例从语义角度描述用户搜索需求；区别则在于前者基于符号执行技术挖掘数据对象值的变化过程，后者则一般使用图模型描述数据对象类型的转换过程.实践表明，基于输入/输出测试值的代码语义搜索方法能够有效提高代码搜索的准确性，而基于输入/输出类型的代码匹配方法则适用更普遍的代码搜索问题.

根据图模型的不同，将基于输入/输出类型匹配的代码搜索方法分为基于全局图模型的方法和基于局部图模型的方法.其中，前者对所有源代码的行为过程进行统一建模，后者则对单个函数进行建模.为了更直观地描述现有方法的各种图模型，以图2(a)所示的数据库连接及表数据查询代码为例进行说明，图2(b)给出代码对应的类型变化过程.文献[9]根据类库和源代码中的函数依赖关系构建全局图模型，如图3(a)所示，节点表示数据对象类型，边表示实现类型间转换的函数调用，采用最短路径搜索算法求取输入类型到输出类型的函数序列.

```
String dburl, usename, password, query;
Connection cn = DriverManager.getConnection
                (dburl, usename, password);
Statement st = cn.CreatStatement( );
ResultSet rs = st.executeQuery(query);
```

(a) 代码示例

String a_1, a_2, a_3, a_4;
Connection $c = DrM.m_1(a_1, a_2, a_3)$;
Statement $d = c.m_2( )$;
ResultSet $e = d.m_3(a_4)$;

(b) 类型表示

Fig. 2 Code example and its type representation

图2 代码示例及其类型表示

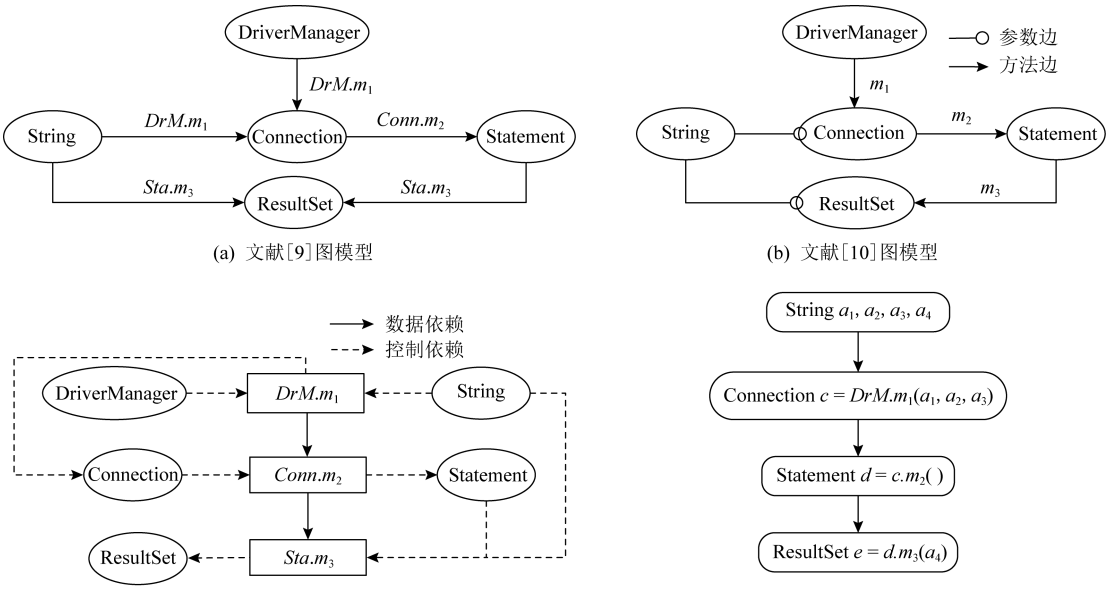


Fig. 3 Example of graph models in related work

图3 相关工作图模型示例

文献[10-12]根据特定单元源代码中的函数依赖关系构建局部图模型.这些局部图模型具有各自的特点,文献[10]的图模型区别于文献[9]的图模型,增加了专门指向函数参数的边,如图 3(b)所示;文献[11]的图模型将节点分为数据节点和动作节点,其中数据节点表示数据对象类型,动作节点表示函数调用,同时使用数据依赖边和控制依赖边描述数据节点以及动作节点间的关系,如图 3(c)所示;在文献[12]的图模型中,节点表示语句,边表示语句次序,如图 3(d)所示.同时上述工作的搜索策略也具有差异.文献[10,12]采用标准的图搜索算法查找输入类型到输出类型的最短函数序列.文献[11]则使用 Grouminer 工具挖掘常用的序列使用模式,然后匹配与查询相似的使用模式.总的来看,局部图模型能够缩小问题空间,相较于全局图模型有助于提升搜索的准确性,然而此类图模型缺乏考虑函数间的数据依赖关系(参数类型个数),导致其描述的代码行为不完整,准确率仍有提升空间.同时,已有研究无法满足多个输入类型查询,适用性较低.

此外,文献[13-17]基于数据对象的类型来进行代码搜索,是与本文较为相关的工作.文献[13]结合静态分析将源代码构建成有限状态自动机,并基于自动机,通过对象类型查找语义相关的代码片段.但是该方法的查询形式不支持输入/输出类型,因此不能匹配满足要求的程序代码或序列.文献[14]允许用户使用尽可能准确的规格描述,包括关键字、类型、方法签名、测试用例等,通过一组程序转换规则将用户要求的内容映射到候选方法或类.文献[15-16]将类型、函数、接口、成员等代码结构信息组织为有向图结构,然后通过图搜索算法查找与用户提供的自然语言查询相关的代码图.然而由于自然语言与代码存在较大差异,难以准确地定位正确解.文献[17]提出基于神经网络的深度学习方法,首先以向量形式语义关联自然语言查询与代码,然后通过计算二者的余弦距离返回与查询语义相近的代码段.但是其训练过程过于复杂和耗时,难以被推广使用.

2 程序代码的语义描述

2.1 基于类型转换的代码语义描述

程序代码为由若干程序语句所构成的语句序列,每个语句代表 1 个执行数据加工的操作,包括操作名以及被加工数据对象 2 个部分.这些操作的具体功能通常为读取或修改对应的数据对象,其具体表现形式为基本运算符的运用或函数(或方法)调用,数据对象可能为常量、变量及数组、结构体、对象等扩展数据类型所对应数据实例.本文暂不考虑与程序结构、流程控制等有关结构语句、预编译指令等不涉及显式数据加工的语句.注意,为了方便描述,本文将融合多个数据加工的复杂语句划分为原子操作后进行讨论,比如,将语句“int $a=b+c$;”看作“算术求和”操作及“赋值”操作的复合.

定义 1. 数据对象.程序语句的数据对象定义为二元组($type$,$value$),其中 $type$ 为数据类型,$value$ 为该数据对象的值.

定义 2. 原子操作.程序语句中的原子操作定义为三元组(c,I,o),其中 c 为操作符,I 为该操作输入数据对象集合,o 为输出数据对象.

定义 1,2 中暂未考虑能够携带多个输出值的函数调用语句,比如 C# 语言中声明为 out 类型的函数参数.一般地,一段程序代码的功能或行为由其数据加工序列体现.具体来说,代码中所有数据对象的值的变化过程对应着代码功能,比如程序分析技术就主要通过符号化手段从值变化过程来模拟代码行为[18].事实上,忽略数据对象的值的具体变化过程,单纯从被加工数据对象的个数、类型的动态变化过程也能从某种程度上去考察代码的功能行为[19-20].

本文用程序代码的操作序列所体现的数据对象的个数及其类型的动态变化过程来刻画代码行为,以应对诸如连接数据库、模拟 HTTP 响应等不具有显式数据对象值的程序代码.Petri 网作为一种经典的并发建模语言,能够很好地刻画并发系统的资源加工变化过程.本文将程序代码中的数据对象看作资源,用 Petri 网来刻画程序代码对数据对象的动态加工过程,并借助 Petri 网对应的分析方法来解决代码匹配问题.

2.2 Petri 网基本概念

Petri 网为由德国计算机科学家 Carl Adam Petri 于 20 世纪 60 年代提出的一种描述离散事件并发系统的形式化模型,具有严格数学定义和图形表达能力.Petri 网由库所、变迁、有向弧 3 种基本元素组成,库所和变迁由有向弧连接.采用 Petri 网对系统进行分析时,将库所看作系统资源,变迁看作引起资源变化的事件或者操作[21].库所能够容纳一定数量的令牌,表示所代表资源的数量.为了方便描述,先给出 Petri 网的形式定义及简单实例.

定义 3. Petri 网.Petri 网为五元组 $PN=(P,T,F,L,W)$,其中:

1) $P=\{p_1,p_2,\cdots,p_m\}$是库所的有限集合;

2) $T=\{t_1,t_2,\cdots,t_n\}$是变迁的有限集合;

3) $F\subseteq(P\times T)\cup(T\times P)$是连接库所和变迁的有向弧集合;

4) $L:P\to\mathbb{N}$是库所集合上的标记函数,用于指定库所 P 所对应的令牌数量;

5) $W:F\to\mathbb{N}_+$是 F 的权函数,表示令牌传递中的加权系数,$W(f)$表示 W 中弧 $f\in F$ 所对应的分量.

在变迁的触发过程中,各个库所的令牌数量会发生变化.各个库所对应的令牌数量所构成的向量称为标识 $\boldsymbol{M}$,记初始标识为 $\boldsymbol{M}_0$.在标识 $\boldsymbol{M}$ 中,库所 p 所对应的分量记作 $M[p]$.

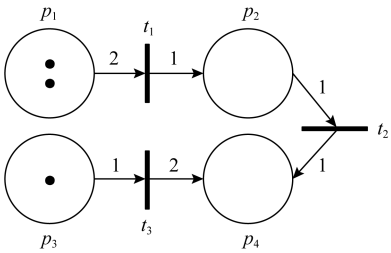
对 $\forall x\in P\cup T$,记 ${}^*x:=\{y\in P\cup T\mid(y,x)\in F\}$和 $x^*:=\{y\in P\cup T\mid(x,y)\in F\}$,称 ${}^*x$ 和 x^* 分别为 x 的前集或输入集和后集或输出集.如果 $\forall p\in{}^*t$:$M[p]\geqslant W({}^*t)$时,称变迁 t 在标识 $\boldsymbol{M}$ 下是使能的,记作 $\boldsymbol{M}[t\rangle$.如果状态标识 $\boldsymbol{M}$ 下 t 是使能的,表示资源的数量满足触发条件,称 t 可以触发.触发后得到的后继标识为 $\boldsymbol{M}'$,记作 $\boldsymbol{M}[t\rangle\boldsymbol{M}'$,且有:

$$M'[p]=\begin{cases}M[p]+W(t^*), & \text{当 } p\in t^*-{}^*t,\\ M[p]-W({}^*t), & \text{当 } p\in{}^*t-t^*,\\ M[p], & \text{其他.}\end{cases}$$

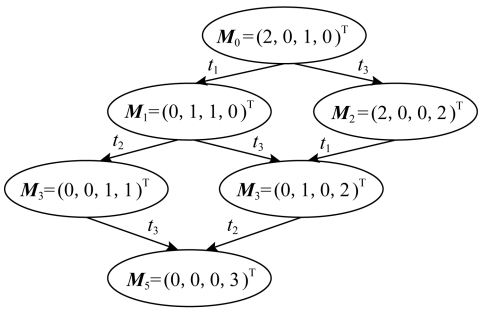
定义 4. 可达标识集.如果 Petri 网 $PN=(P,T,F,L,W)$中存在 $t\in T$,使 $\boldsymbol{M}[t\rangle\boldsymbol{M}'$,称 $\boldsymbol{M}'$是从 $\boldsymbol{M}$ 一步可达的.如果存在变迁序列 $t_1,t_2,\cdots,t_k$ 和标识序列 $\boldsymbol{M}_1,\boldsymbol{M}_2,\cdots,\boldsymbol{M}_k$,使得 $\boldsymbol{M}_0[t_1\rangle\boldsymbol{M}_1[t_2\rangle\cdots\boldsymbol{M}_{k-1}[t_k\rangle\boldsymbol{M}_k$,称 $\boldsymbol{M}_k$ 是从 $\boldsymbol{M}_0$ 多步可达的.从 $\boldsymbol{M}_0$ 可达的标识集合称为可达标识集,记作 $R(\boldsymbol{M}_0)$,且满足 $\boldsymbol{M}_0\in R(\boldsymbol{M}_0)$.

定义 5. Petri 网的可达图.Petri 网 PN 的可达图为二元组 $GR(PN)=(S,E)$,其中 $S=R(\boldsymbol{M}_0)$是节点集,代表 Petri 网 PN 中所有的可达标识或者状态;$E=\{(\boldsymbol{M},t,\boldsymbol{M}')\mid\boldsymbol{M},\boldsymbol{M}'\in R(\boldsymbol{M}_0),\boldsymbol{M}[t\rangle\boldsymbol{M}'\}$是有向弧集,代表不同可达状态之间的使能变迁.

例 1. Petri 网是一种直观的图形化建模工具,一般用实心矩形“▌”表示变迁,库所用圆圈“○”表示,令牌用实心圆点“•”表示.为便于理解,以图 4 为例给出 Petri 网的数学形式化定义的图形表示.图 4(a)为 1 个简单 Petri 网,其初始标识为 $\boldsymbol{M}_0=[p_1\mapsto2,p_2\mapsto0,p_3\mapsto1,p_4\mapsto0]$,记作向量$(2,0,1,0)^{\mathrm{T}}$,表示库所 p_1 的令牌数量为 2,库所 p_3 的令牌数量为 1,其余为 0;变迁 $t\in T$ 的前置集为:${}^*t_1=\{p_1\}$,${}^*t_2=\{p_2\}$等;$\forall p\in{}^*t_1$:$M_0[p]\geqslant W({}^*t_1)=2$,故 $\boldsymbol{M}_0[t_1\rangle$,记作 $\boldsymbol{M}_0[t_1\rangle\boldsymbol{M}_1$,其中$\boldsymbol{M}_1=(0,1,1,0)^{\mathrm{T}}$ 为可达标识.图 4(b)为该 Petri 网的状态可达图.



(a) Petri网示例



(b) Petri网状态可达图

Fig. 4 Formal definition and graphical representation of Petri Net

图 4 Petri 网形式化定义及图形表示

3 基于 Petri 网的代码行为建模

由 2.1 节可知,程序语句一般为执行数据加工的操作组成,而程序代码的功能行为则表现为数据对象类型及个数的转换过程.本文将操作的数据对象看作资源,程序操作则为导致资源流动的事件.根据定义 3 的 Petri 网定义,数据对象的类型和语句可以通过库所和变迁表示,数据对象的个数可以通过有向弧权重表示.基于 Petri 网的代码行为过程描述为:通过 Petri 网中有向弧的指向表示语句中数据对象间的类型转换关系,通过有向弧的权重表示语句中不同类型数据对象的个数.本节给出一种基于 Petri 网的代码行为过程建模方法.首先阐明基于 Petri 网的代码行为建模机制,制定 Petri 网元素与代码元素之间的对应关系;其次基于该对应关系,利用静态分析技术对程序代码进行分析处理,完成代码到 Petri 网的转换.

3.1 基于 Petri 网的代码行为模型

单个程序语句包括有关操作以及被加工的输入数据对象集和加工后所生成的输出数据对象.比如,

函数调用语句由函数名、调用该函数的主动调用数据对象(caller)、作为参数的被调用数据对象(callee)以及返回数据对象组成,其中函数名为操作符,调用和被调用数据对象均为输入数据对象,返回数据对象为输出数据对象.

定义 6. 程序语句为三元组(S_c,S_i,O),其中S_c代表操作符集合,S_i代表输入数据对象集合,O代表输出数据对象.对于$\forall c\in S_c$,语句输入数据对象$I\subseteq S_i$对应于操作c输入数据对象,语句输出数据对象o对应于优先级最低操作c的输出数据对象.

比如,语句"int $a=b+c$;"包括算术、赋值 2 个运算符,其中算术运算符优先级高于赋值运算符,显然整个语句功能应体现为赋值运算的最终结果,故将变量a作为最终输出对象.数据对象个数及类型变化过程可简单概括为:通过语句对数据对象进行加工处理,将一定数量的特定类型的输入数据对象转换为特定类型的输出数据对象.该过程在 Petri 网中可描述为:通过语句变迁的输入/输出弧的指向及权重,表示语句中输入数据对象到输出对象的转换过程,其中弧由输入数据对象类型指向输出数据对象类型,输入、输出弧的权重分别表示转换过程需要消耗的输入数据对象、生成的输出数据对象个数.

对于语句$s\in S$,用t_s表示该语句变迁,对于其输入数据对象类型$i\in S_i$,用p_i表示输入数据对象类型库所,对于其输出数据对象类型o,用p_o表示输出数据对象类型库所.为了方便描述,连接语句变迁与输入数据对象类型库所的输入弧表示为(p_i,t_s),输入弧权重表示为$W(p_i,t_s)$,即为语句s中类型为i的输入数据对象个数.连接语句变迁与输出数据对象类型的输出弧表示为(t_s,p_o),输出弧权重表示为$W(t_s,p_o)$,且$W(t_s,p_o)=1$.

在 Petri 网中用输入数据对象的消耗和输出数据对象的产生来建模程序语句,容易出现代码语义描述的不一致.比如,对于需要执行多次调用的某个主动对象x,当x执行一次方法调用后,在对应 Petri 网的语句描述上消耗了该数据对象x,从而导致不能准确描述将来其他涉及x调用的程序语句.因此,对于被多次重复使用的数据对象,用Y表示这类数据对象的类型.针对这类数据对象$y\in Y$,本文引入一个特殊的复制变迁C_y.连接复制变迁和数据对象类型库所的输入弧和输出弧分别表示为(p_y,C_y)和(C_y,p_y),其中p_y表示类型库所.输入弧权重表示为$W(p_y,C_y)$,且$W(p_y,C_y)=1$.输出弧权重表示为$W(C_y,p_y)$,即为该类型的数据对象在程序代码中被重复使用的次数.本文给出程序代码元素与 Petri 网元素之间的对应关系,如表 1 所示:

Table 1 Mapping of Program Code Elements to Petri Net Elements

表 1 程序代码元素到 Petri 网元素的映射

代码元素	Petri 网元素	转换语义描述
数据对象类型	库所集 P	库所集 P 表示数据对象类型集合
语句集合 S	变迁集 T	变迁集 T 表示程序语句集合
输入数据对象类型 i-语句 s	输入弧(p_i,t_s) 输入弧权重 $W(p_i,t_s)$	输入弧(p_i,t_s)表示类型 i 是语句 s 的输入数据对象 输入弧权重 $W(p_i,t_s)$表示语句 s 中类型为 i 的输入数据对象个数
语句 s-输出数据对象类型 o	输出弧(t_s,p_o) 输出弧权重 $W(t_s,p_o)$	输出弧(t_s,p_o)表示类型 o 是语句 s 输出数据对象 输出弧权重 $W(t_s,p_o)=1$
特殊情况(数据对象被重复使用)	复制变迁 C_y 输入弧权重 $W(p_y,C_y)$ 输出弧权重 $W(C_y,p_y)$	类型 $y\in Y$ 的数据对象被重复使用时,该类型存在特殊的复制变迁 C_y 输入弧权重 $W(p_y,C_y)=1$ 输出弧权重 $W(C_y,p_y)$表示类型为 y 的数据对象被重复使用次数

此外,对于没有明显输入或者输出类型的语句,为了充分保留语句中的语义信息,用 void 来表示这类语句的输入或者输出数据对象类型,比如 new 语句的输入类型、close 语句的输出类型等.

3.2 程序代码的 Petri 网构建过程

为了匹配程序代码,首先需要将所有候选代码转换成对应 Petri 网形式.代码片段到 Petri 网的转换过程如图 5 所示.本文以 Java 语言编写的程序代码为研究对象,首先借助已有代码分析工具解析候选代码片段,从而提取各个代码元素,如语句、操作、参数个数和类型等.其次基于 3.1 节所述代码元素到 Petri 网元素之间的映射关系,将候选代码构建成对应 Petri 网模型,存储于图数据库中,为后期代码匹配做准备.

为了便于与 Java 代码的无缝衔接及效率因素考虑,本文选择轻量级的 Eclipse JDT 代码解析组件 ASTParser.该组件可以快速将 Java 语言程序代码解析成基于文档对象模型(document object model,

DOM)结构的抽象语法树表示,代码中的每个元素对应于抽象语法树上某个节点.采用深度优先搜索策略并借助相关 API 便可获取代码元素如语句、操作、参数等以及彼此之间的上下文语义关系,即特定语句、操作、参数之间的对应,其中重复对象集的确定是通过判断抽象语法树上不同操作节点是否具有相同的输入变量.在此基础上,利用 3.1 节描述的代码元素与 Petri 网元素之间的映射关系,完成候选代码到 Petri 网模型的转换,详细步骤见算法 1.

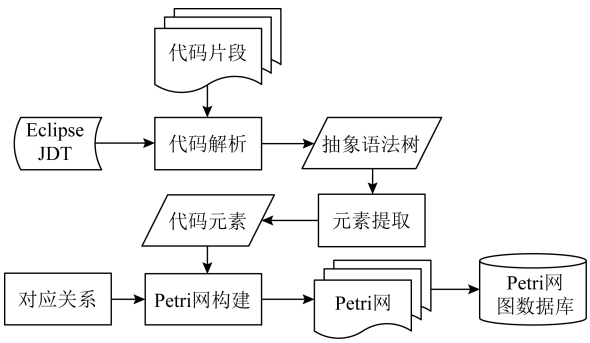


Fig. 5 Flow chart of Petri Nets construction process

图 5 Petri 网构建过程流程图

算法 1. Petri 网构建算法 $constructPN$.

输入:代码片段 $code\_Func$;

输出:Petri 网 PN.

① $PN \leftarrow \{P,T,F,W,L\}$;

② $P \leftarrow \varnothing, T \leftarrow \varnothing, F \leftarrow \varnothing, W \leftarrow \varnothing, L \leftarrow \varnothing$;
/* 初始化 Petri 网 */

③ $S \leftarrow parseStatement(code\_Func)$;
/* 将代码片段中的语句序列放入到语句集 S 中 */

④ while $S \neq \varnothing$ do

⑤ 选择 $s \in S$;

⑥ $S \leftarrow S - \{s\}$;

⑦ $T \leftarrow T \cup \{s\}$; /* 将语句添加到 Petri 网的变迁集中 */

⑧ $T\_in \leftarrow parseInputType(s)$;
/* 将语句的输入数据对象类型放入集合 T_in 中 */

⑨ while $T\_in \neq \varnothing$ do

⑩ 选择 $t \in T\_in$; /* 选择集合 T_in 中 1 个输入类型 */

⑪ $T\_in \leftarrow T\_in - \{t\}$;

⑫ if $t \notin P$ then /* 判断输入类型是否存在 Petri 网 */

⑬ $P \leftarrow P \cup \{t\}$; /* 若不存在则将类型添加到 Petri 网的库所集中 */

⑭ end if

⑮ $num \leftarrow parseInputNum(t,s)$;
/* 解析语句中该类型输入数据对象的个数 */

⑯ $F \leftarrow F \cup \{\langle t,s\rangle\}$; /* 添加输入弧 */

⑰ $W \leftarrow W \cup \{w\langle t,s\rangle = num\}$; /* 设置输入弧权重 */

⑱ end while

⑲ $T\_out \leftarrow parseOutputType(s)$; /* 获取语句中输出数据对象类型 */

⑳ if $T\_out \notin P$ then /* 判断输出类型是否存在 Petri 网 */

㉑ $P \leftarrow P \cup \{T\_out\}$; /* 若不存在则将类型添加到 Petri 网的库所集中 */

㉒ end if

㉓ $num \leftarrow parseOutputNum(T\_out,s)$;
/* 解析语句中该类型输出数据对象的个数 */

㉔ $F \leftarrow F \cup \{\langle s,T\_out\rangle\}$; /* 添加输出弧 */

㉕ $W \leftarrow W \cup \{w\langle s,T\_out\rangle = num\}$; /* 设置输出弧权重 */

㉖ end while

㉗ return PN.

需要指出,行③函数 $parseStatement()$用以分析并确定代码操作,包括程序语句以及本文针对程序代码中重复使用对象设置的复制变迁;行⑧函数 $parseInputType()$和行⑲函数 $parseOutputType()$用以分析某个具体操作的输入对象和输出对象,包括语句操作的输入类型和输出类型,以及复制变迁对应的重复使用对象类型;行⑮函数 $parseInputNum()$和行㉓函数 $parseOutputNum()$用以分析某个具体输入对象和输出对象在其操作语句中的数量,包括特定语句操作的某个输入类型个数和输出类型个数,以及特定复制变迁对应重复使用对象类型输入个数和输出个数(对象重复使用次数).

程序代码所对应的 Petri 网描述具有明显的网式结构化特征.为了提高后期代码匹配效率,本文采用高效的 Neo4j[①] 来存储程序代码所对应的 Petri 网

① https://neo4j.com/

信息.Neo4j 具有与 Petri 网类似的数据组织方式和结构,即通过节点和边来构成图,并提供类似 SQL 的语言 Cypher 以方便用户执行查询、编辑等操作.

例 2. 图 6 中为数据库连接的 Java 代码,该代码对应 Petri 网如图 7 所示.为了方便描述,将类型标记于对应库所,语句标记于对应变迁,其中 Class,String 类型库所为该代码片段的起始库所,void 类型库所为终止库所,其余类型库所为类型转换过程中的过渡或中间库所.由库所指向变迁的输入弧代表类型库所是语句变迁的输入数据对象,输入弧权重代表该类型的输入数据对象个数.由变迁指向库所的输出弧代表类型库所是语句变迁的输出数据对象,输出弧权重代表该类型的输出数据对象个数.此外,每个重复对象存在其相应的克隆变迁,由重复对象库所指向克隆变迁的输入弧权重为 1,由克隆变迁指向重复对象库所的输出弧权重为对象重复使用的次数.比如,在函数 *getConnection* 调用语句中,输入数据对象包括 3 个类型为 String 的参数以及 1 个类型为 DriverManager 的调用者,因此输入弧由 String 库所和 DriverManager 库所指向语句变迁,且输入弧权重分别为 3 和 1;输出数据对象是 1 个类型为 Connection 的返回者,因此输出弧由语句变迁指向 Connection 库所,且输出弧权重为 1.Connection 对象在程序片段中被重复使用 2 次,因此 Connection 库所处存在 C_c 克隆变迁,且 C_c 输入弧权重为 1,输出弧权重为 2.出于空间的考虑,在图 6 中采用函数名来表示函数调用语句.需要指出,示例中的异常处理及迭代仅作数据流分析.

```
public void testConnect(){
  String dbUrl="xxx";String username="xxx";
  String password="xxx";
  String dbClass="com.mysql.jdbc.Driver";
  String query="SELECT * FROM DVD Info Table";
  try{Class.forName(dbClass);
    Connection connection = DriverManager.getConnection(dbUrl,
      username, password);
    Statement statement=connection.createStatement();
    ResultSet resultSet=statement.executeQuery(query);
     while(resultSet.next()) String tableName = resultSet.
       getString(1);
    connection.close();}
    catch(ClassNotFoundException e){e.printStackTrace();}}
```

Fig. 6 "Java database connection" code example

图 6 "Java 数据库连接"代码示例

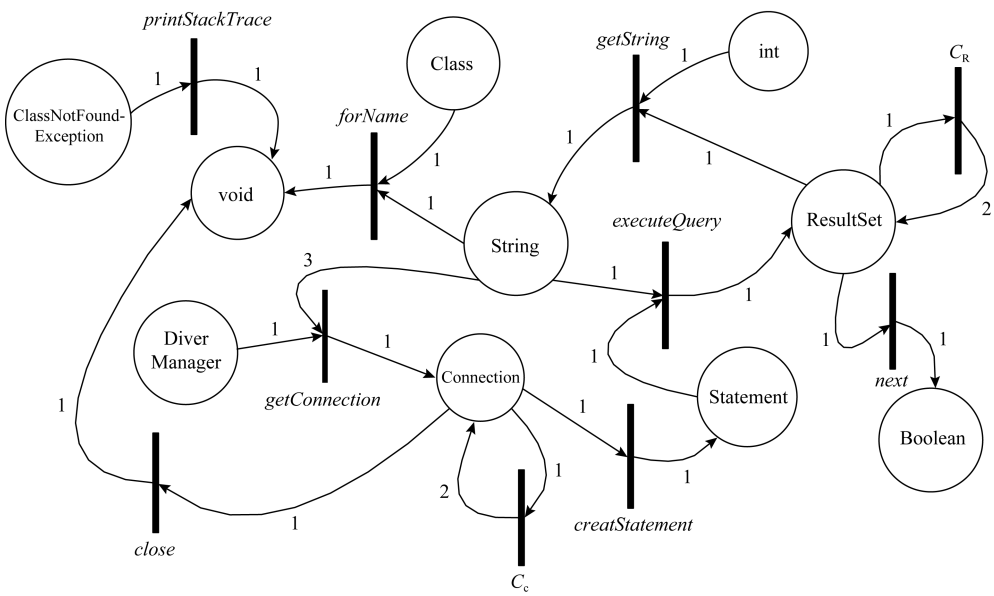


Fig. 7 Petri Net based on the example code of Fig.6

图 7 基于图 6 示例代码的 Petri 网

4 基于 Petri 网可达分析的代码匹配方法

本节引入一种基于 Petri 网可达分析的代码匹配方法.该方法根据用户提供的输入数据对象个数、类型信息及期望输出数据对象类型,从候选代码所对应 Petri 网模型中搜索存在功能行为满足将输入类型转换为输出类型的代码片段.整个过程分为 3 个阶段,如图 8 所示.首先,在各个候选代码 Petri 网中,根据输入、输出数据对象信息在候选 Petri 网中

确定初始标识 $\boldsymbol{M}_0$ 和目标标识 $\boldsymbol{M}^*$；然后基于初始标识 $\boldsymbol{M}_0$，对 Petri 网进行可达分析，生成反映标识转换的可达图；最后通过分析候选 Petri 网可达图中是否存在目标标识 $\boldsymbol{M}^*$ 来判断 Petri 网对应的程序代码是否为用户所期待代码片段.

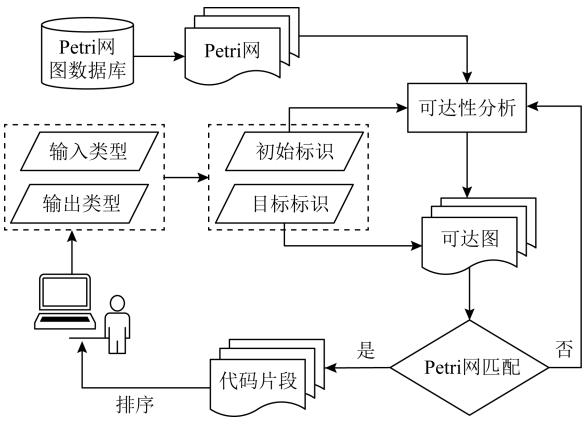


Fig. 8 Flow chart of method for matching code based on the reachability analysis of Petri Nets

图 8 基于 Petri 网可达分析的代码匹配方法流程图

4.1 确定初始标识与目标标识

为了分析 Petri 网的可达性，首先需借助用户提供的输入、输出数据对象信息确定 Petri 网 $PN=(P,T,F,L,W)$ 的初始标识 $\boldsymbol{M}_0$ 和目标标识 $\boldsymbol{M}^*$.

定义 7. 查询类型对.查询类型对为二元组 $T=(I,o)$，其中 I 代表输入数据对象的数据类型集，o 代表输出数据对象的数据类型.

初始标识是指 T 中输入类型在 Petri 网对应库所的等量令牌数，用来表示需要消耗的数据对象资源.

定义 8. 初始标识.设在查询类型对 T 中，类型为 $i\in I$ 的输入数据对象个数为 n，如果在 Petri 网中存在库所 $p\in P$ 的标记为 i，则 $M_0[p]=n$，$M_0[\text{void}]=1$，对于其他类型 $p'\in P$，$M_0[p']=0$.

考虑到 T 中通常不包含 void 输入类型，为了保证无明显输入类型程序语句的匹配过程不受影响，因此设置 void 类型库所的令牌数为 1.目标标识是指 T 中输出类型在 Petri 网对应库所中的等量令牌数，用来表示需要生成的数据对象资源.

定义 9. 目标标识.对于查询类型对 T 中输出数据对象 o，如果在 Petri 网中存在库所 $p\in P$ 的标记为 o，则 $M^*[p]=1$，$M^*[\text{void}]\geqslant 0$，对于其他类型 $p'\in P$，$M^*[p']=0$.

void 库所令牌数可能大于 0 的原因为：部分代码包含 1 个或多个无明显输出类型的语句，导致生成 1 个或多个 void 令牌；部分代码仅包含明显输入类型的语句，导致初始标识 void 库所中的令牌没有被消耗.

4.2 Petri 网可达图生成与剪枝

为了判断 Petri 网中是否存在由初始标识可达的目标标识，需对 Petri 网进行可达分析，生成所有标识间可达关系图.对于较为复杂的 Petri 网，为了提高分析效率，还需进行可达图修剪，即去除不影响分析结论的多余库所或变迁.

4.2.1 可达图的生成

可达性分析是研究 Petri 网动态行为的重要手段[22].对已确定初始、目标标识的 Petri 网 PN，其可达图生成过程如算法 2 所示.

算法 2. Petri 网可达图生成算法 *reachGraph*.

输入：Petri 网 PN、初始标识 $\boldsymbol{M}_0$、输出类型 p^*；

输出：PN 的可达图 GR^*.

① $GR^*\leftarrow(N,E)$；

② $N\leftarrow\{\boldsymbol{M}_0\}$；$E\leftarrow\varnothing$；/* 初始化可达图 */

③ $\Phi\leftarrow\{\boldsymbol{M}_0\}$；

④ while $\Phi\neq\varnothing$ do /* 当未处理标识集不为空时 */

⑤ 选择 $\boldsymbol{M}\in\Phi$；/* 选择未处理标识集中的 1 个标识 */

⑥ $\Phi\leftarrow\Phi-\{\boldsymbol{M}\}$；

⑦ for 所有 $T\in enabled(\boldsymbol{M})$ do /* 状态标识 $\boldsymbol{M}$ 下所有使能的变迁 T */

⑧ $(\boldsymbol{M}',p)\leftarrow fire(\boldsymbol{M},T)$；/* 计算 $\boldsymbol{M}$ 的后继标识 $\boldsymbol{M}'$ 以及变迁 T 的输出库所 p */

⑨ if $pathExists(p,p^*,\alpha(PN))$ then /* 判断 $\alpha(PN)$ 中是否存在 p 到 p^* 的路径 */

⑩ Continue；

⑪ end if

⑫ if $\boldsymbol{M}'\notin S$ then /* 判断 $\boldsymbol{M}'$ 是否为可达图 GR^* 的节点 */

⑬ $N\leftarrow N\cup\{\boldsymbol{M}'\}$；/* 若不是，则添加新节点 $\boldsymbol{M}'$ */

⑭ $E\leftarrow E\cup(\langle\boldsymbol{M},T,\boldsymbol{M}'\rangle)$；/* 添加新有向边 $\langle\boldsymbol{M},T,\boldsymbol{M}'\rangle$ */

⑮ $\Phi\leftarrow\Phi\cup\{\boldsymbol{M}'\}$；/* 将 $\boldsymbol{M}'$ 加入到未处理标识集中 */

⑯ end if

⑰ end for

⑱ end while

⑲ return GR^*.

算法 2 考虑不带行⑨～⑪的基本版本,该版本大致等同于构建 Petri 网可达图的标准算法.首先,初始化可达图 GR^* 和未处理标识集 Φ,其中 GR^* 的节点为 PN 的标识,GR^* 的起始节点为初始标识 $\boldsymbol{M}_0$,GR^* 的有向边$\langle \boldsymbol{M},T,\boldsymbol{M}'\rangle$连接相邻标识,表示标识 $\boldsymbol{M}$ 通过触发变迁 T 一步可达后继标识 $\boldsymbol{M}'$(行①～③);其次,对于每个未处理标识集中标识 $\boldsymbol{M}$,计算 $\boldsymbol{M}$ 的所有后继状态,后继状态包括后继标识 $\boldsymbol{M}'$以及变迁 T 的输出库所 p(行④～⑧);然后,判断 $\boldsymbol{M}'$是否已处理,如果 $\boldsymbol{M}'$尚未处理,则将 $\boldsymbol{M}'$插入 Φ,并将 $\boldsymbol{M}'$和$\langle \boldsymbol{M},T,\boldsymbol{M}'\rangle$添加到可达性图 GR^*(行⑫～⑰);最后,当 Φ 中所有标识都处理完毕,则循环结束,返回 GR^*(行⑱⑲).

例 3. 以图 4 为例,假设输出类型为 p_4,表 2 说明 Petri 网可达图生成算法的执行过程.

Table 2 Steps of Reachability Graph Generation Algorithm

表 2 可达图生成算法执行步骤

步骤	未处理标识集	可达图({节点},{有向边})
步骤 1	$\boldsymbol{M}_0$	$(\{\boldsymbol{M}_0,\boldsymbol{M}_1,\boldsymbol{M}_2\},\{\langle \boldsymbol{M}_0,t_1,\boldsymbol{M}_1\rangle,\langle \boldsymbol{M}_0,t_3,\boldsymbol{M}_2\rangle\})$
步骤 2	$\boldsymbol{M}_1,\boldsymbol{M}_2$	$(\{\boldsymbol{M}_0,\boldsymbol{M}_1,\boldsymbol{M}_2,\boldsymbol{M}_3,\boldsymbol{M}_4\},\{\langle \boldsymbol{M}_0,t_1,\boldsymbol{M}_1\rangle,\langle \boldsymbol{M}_0,t_3,\boldsymbol{M}_2\rangle,\langle \boldsymbol{M}_1,t_2,\boldsymbol{M}_3\rangle,\langle \boldsymbol{M}_1,t_3,\boldsymbol{M}_4\rangle,\langle \boldsymbol{M}_2,t_1,\boldsymbol{M}_4\rangle\})$
步骤 3	$\boldsymbol{M}_3,\boldsymbol{M}_4$	$(\{\boldsymbol{M}_0,\boldsymbol{M}_1,\boldsymbol{M}_2,\boldsymbol{M}_3,\boldsymbol{M}_4,\boldsymbol{M}_5\},\{\langle \boldsymbol{M}_0,t_1,\boldsymbol{M}_1\rangle,\langle \boldsymbol{M}_0,t_3,\boldsymbol{M}_2\rangle,\langle \boldsymbol{M}_1,t_2,\boldsymbol{M}_3\rangle,\langle \boldsymbol{M}_1,t_3,\boldsymbol{M}_4\rangle,\langle \boldsymbol{M}_2,t_1,\boldsymbol{M}_4\rangle,\langle \boldsymbol{M}_3,t_3,\boldsymbol{M}_5\rangle,\langle \boldsymbol{M}_4,t_2,\boldsymbol{M}_5\rangle\})$
步骤 4	$\boldsymbol{M}_5$	

4.2.2 可达图的修剪

随着代码规模的扩大,状态空间的组合爆炸将使可达性分析变得非常困难,基于 Petri 网的可达分析的复杂度呈指数增加[23].为了应对可能的状态爆炸问题或提高分析效率,本文通过分析 Petri 网对应有向网中的路径对可达图 $GR(PN)$进行剪枝,使可达图 $GR(PN)$得到化简,同时保证化简后的可达图 GR^* 也足以反映给定输入/输出类型转换过程.

如果在 PN 中没有从某类型库所到输出类型库所的路径,则不需分析从该类型库所到输出类型库所的路径是否可达,即没有必要对该类型库所分配非零数量的令牌.本文基于这个观察来修剪 $GR(PN)$冗余节点.为了分析 PN 类型库所间的路径存在的问题,本文定义诱发网 $\alpha(PN)$的概念,$\alpha(PN)$忽略变迁及有向弧权重,仅用无权有向弧表示类型库所间可达关系.

定义 10. 诱发网.诱发网 $\alpha(PN)$是由 Petri 网 $PN=(P,T,F,L,W)$诱发的表示为(V,E')的有向网,其中 $V=P$,$(P,P')\in E'$,并且存在变迁 $t\in T$ 使得$(P,t)\in E$,$(t,P')\in E$.

例 4. 图 7 中 Petri 网的诱发网 $\alpha(PN)$如图 9 所示.出于空间考虑,长单词用缩写表示.

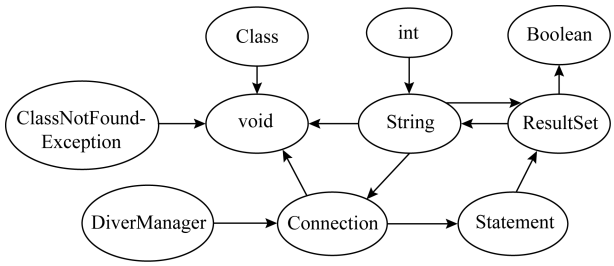


Fig. 9 Induced Net of Petri Net in Fig. 7

图 9 基于图 7 中 Petri 网的诱发网

命题 1. 令 $\alpha(PN)$为 Petri 网 PN 的诱发网,设 $\alpha(PN)$中没有从某库所 p 到目标库所 p^* 的路径,设 $\boldsymbol{M}$ 为 PN 中库所 p 内令牌数大于 0 的标识,$\boldsymbol{M}^*$ 为库所 p^* 内令牌数为 1 的目标标识,则 $GR(PN)$中没有从 $\boldsymbol{M}$ 到 $\boldsymbol{M}^*$ 的路径.

根据该命题,在不影响完整性的情况下,算法 2 的行⑨～⑪判断 $\alpha(PN)$是否存在由 p 到 p^* 的路径,若不存在,修剪此类标识($M[p]>0$).

定理 1. 约简模型中标识可达性是一致的.

证明. 设 PN 的化简前可达图 $GR(PN)=(SM,SL)$,化简后可达图 $GR^*(PN)=(SM^*,SL^*)$,其中 SM 和 SM^* 表示可达标识,SL 和 SL^* 表示连接相邻标识的有向边.$GR^*(PN)$满足:

1) $\forall \boldsymbol{M}\in SM$,其中 $M[P]>0$,$P=\{p_1,p_2,\cdots,p_n\}$,若 $\forall p\in P$,$\alpha(PN)$中都存在 p 到 p^* 的路径,则 $\boldsymbol{M}\in SM^*$.

2) $\forall \boldsymbol{M},\boldsymbol{M}'\in SM$,$\boldsymbol{M},\boldsymbol{M}'\in SM^*$,若 $\exists l(\boldsymbol{M},t,\boldsymbol{M}')\in SL$,则 $l(\boldsymbol{M},t,\boldsymbol{M}')\in SL^*$.

上述描述表明 $GR^*(PN)$保留 $GR(PN)$中所有可能到达目标标识 $\boldsymbol{M}^*$($M^*[p^*]=1$)的标识 $\boldsymbol{M}$ 及 $\boldsymbol{M}\rightarrow\boldsymbol{M}^*$ 的路径,即约简模型中标识可达性是一致的. 证毕.

4.3 基于可达图的代码匹配算法

Petri 网可达图反映了 Petri 网在初始标识 $\boldsymbol{M}_0$ 下可达的标识空间.因此对于生成的可达图 GR^*,若 GR^* 存在目标标识 $\boldsymbol{M}^*$,则 Petri 网上存在从 $\boldsymbol{M}_0$ 到 $\boldsymbol{M}^*$ 的可达路径,表明该 Petri 网匹配.算法 3 描述了 Petri 网匹配的整个过程:首先初始化匹配

Petri 网集合、搜索次数(行①②);其次,依次获取 Petri 网进行分析,生成其对应的可达图(行③～⑤);然后,判断可达图是否存在目标标识,若存在,则将对应 Petri 网添加到匹配 Petri 网集(行⑥～⑧);最后,当所有 Petri 网分析完毕,则循环结束,返回匹配 Petri 网集合(行⑨⑩).对于返回的匹配 Petri 网集,其对应的代码片段存在将输入类型转换为输出类型的语句序列,这些代码片段匹配,将其纳入候选代码片段集合.

算法 3. Petri 网匹配算法 $matchPN$.

输入:待搜索的 Petri 网库 $PN\_database$、输出类型 p^*、初始标识 $\boldsymbol{M}_0$、目标标识 $\boldsymbol{M}^*$、搜索最大次数(Petri 网个数)n_max;

输出:匹配 Petri 网集合 Ok_PNs.

① $Ok\_PNs \leftarrow \varnothing$;
② $searchCount \leftarrow 0$;
③ while($one\_PN \leftarrow nextFile(PN\_database)$ and $searchCount \leqslant n\_max$) do /* 获取 Petri 网 */
④ $searchCount \leftarrow searchCount + 1$;
⑤ $GR^* \leftarrow reachGraph(one\_PN, \boldsymbol{M}_0, p^*)$; /* 生成 Petri 网可达图 */
⑥ if $\neg NodeExist(\boldsymbol{M}^*, GR^*)$ then /* 判断可达图是否存在目标标识 $\boldsymbol{M}^*$ */
⑦ $Ok\_PNs \leftarrow Ok\_PNs \cup \{one\_PN\}$; /* 若存在,Petri 网匹配成功,添加到匹配 Petri 网集 */
⑧ end if
⑨ end while
⑩ return Ok_PNs.

4.4 候选代码排序

对于候选代码片段集合,本文使用 2 种代码质量评价指标对代码片段进行排序,帮助程序员快速识别所需的代码片段.

基于程序员经常倾向于使用较短序列而不是较长序列来实现其任务的现象[9-10,24],本文采用序列长度作为评价代码质量的指标,越短的序列优先级越高.本文将代码片段中功能序列的长度作为序列长度,功能序列指将输入类型转换为输出类型的方法序列.

本文还采用代码复用率作为评价代码质量的指标.在许多代码搜索工作如 PARSEWeb[10],DERECS[25]中都用到了复用率对结果进行排序,其基本考虑是代码的复用率(代码片段在软件开发过程中重复使用的频率)越高,参考价值越高,但计算方式各有差异.本文考虑相似代码片段在候选代码库中出现的频率.采用已有代码克隆检测方法 CCFinderSW[26]来识别候选代码库中的相似代码,并将相似代码聚成一类,计算每一类集合中代码段个数作为此类代码段的复用率.CCFinderSW 的源代码可以在 GitHub 上找到,本文用 Java 语言重新实现了此算法.

5 实验以及分析

为了验证本文所提出方法的有效性,本文进行了 2 个实验.在第 1 个实验中,我们验证了本方法的有效性,即能否解决多种形式的类型转换问题,并与其他检索方法的准确率进行对比;第 2 个实验则讨论了本方法中涉及的技术点对搜索效率的影响.

5.1 实验建立

5.1.1 实验准备

为了确保所收集源代码的真实性和可靠性,本文选择从软件项目托管平台 GitHub 上收集源代码.根据托管项目的 Most Stars 排序,本文选择了 200 个用户评价较高的 Java 语言开源项目,将项目以函数为粒度进行切分,得到函数级代码片段 189 442 个.

为了构建客观的用户真实查询测试集,我们从 Stack Overflow 中找到带有 Java 标签的问答对,并选取票数为正、答案中含有代码片段的热门的 20 个问题,同时提取答案代码示例中的输入/输出类型作为代码搜索工具的查询输入如表 3 所示.需要指出,我们将数组视为特殊的数据对象,其类型为元素类型.

5.1.2 评估指标

为了评价代码搜索的准确度或效率,本文采用 $P@n$[27-28], MAP (mean average precision)[7,29], MRR(mean reciprocal rank)[7]作为评价指标.$P@n$ 表示返回前 n 个结果的准确率,计算公式为

$$P@n = \frac{1}{n}\sum_{k=1}^{n} rel_k, \tag{1}$$

其中,n 表示返回的前 n 个结果,rel_k 表示第 k 个结果的相关性,如果相关,$rel_k=1$,否则 $rel_k=0$.MAP 表示平均准确率,该指标对出现在列表中较高位置的相关结果给予较高的权重,计算公式为

$$MAP = \frac{1}{Q}\sum_{i=1}^{Q} AveP(i), \tag{2}$$

Table 3 Experimental Query Test Set

表 3 实验的查询测试集

编号	查询	输入/输出类型
1	将字符串转换为日期	String/date
2	关闭最外层的字符串写入流	StringWriter/void
3	将输入流转换为字符串	InputStream/String
4	模拟 HTTP 响应	HttpTransport/HttpResponse
5	在特定范围内生成 1 个随机整数	Random/int
6	创建并写入文本文件	String,String/PrintWriter
7	连接 MySQL 数据库	String,String,String,String/Connection
8	如何使用 JSP/Servlet 将文件上传到服务器	HttpServletRequest,HttpServletResponse/InputStream
9	将输入流转换为字节数组	InputStream/byte[]
10	遍历 HashMap	Map/Iterator
11	使用 Java 解析 JSON	String/JSONArray
12	如何将文本附加到现有文件中	String/FileWriter
13	如何用 Java 确定 1 个数的小数位数	String/double
14	如何比较 2 个字符串	String,String/boolean
15	将字符串写入流转换为字符串	StringWriter/String
16	生成 MD5 散列	String/byte[]
17	将 java.util.Date 转换为 XMLGregorianCalendar	DatatypeFactory/XMLGregorianCalendar
18	如何使用 Java 从 Internet 下载和保存文件	String/FileOutputStream
19	计算 2 个日期实例之间的差异	date,date,TimeUnit/long
20	确定数组是否包含特定值	String[],String/boolean

$$AveP=\frac{1}{num}\sum_{k=1}^{n}(P(k)\times rel_k), \tag{3}$$

其中,$AveP$ 表示单个查询的准确率,num 表示相关结果的个数,$P(k)$表示对于查询 Q_i 相关结果列表中排名为 k 的权重,例如结果列表共有 2 个相关代码段位于排名 1 和 3,则 $P(1)=1$,$P(3)=0.67$. MRR 表示第 1 个相关结果排名的倒数,计算公式为

$$MRR=\frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i}, \tag{4}$$

其中,$rank_i$ 表示对于查询 Q_i 第 1 个相关结果的排名.

5.1.3 比较对象

为了检测搜索方法的有效性,我们与 2 种典型的代码检索方法进行对比.第 1 种为基于文本匹配的方法.该方法源于大多源代码搜索引擎,通过相似度计算模型[30]计算查询与代码片段的文本相似度.第 2 种为基于类型匹配的方法.该检索方法只考虑有向图中是否存在一条路径以输入/输出类型作为起点和终点节点,不考虑路径的可达问题.

5.2 实验结果及分析

5.2.1 搜索结果对比分析

针对表 3 中的 20 个查询问题,本文对比了 5.1.3 节中 2 种方法检索结果的 $P@5$,$P@10$ 评价指标,如图 10 和图 11 所示.

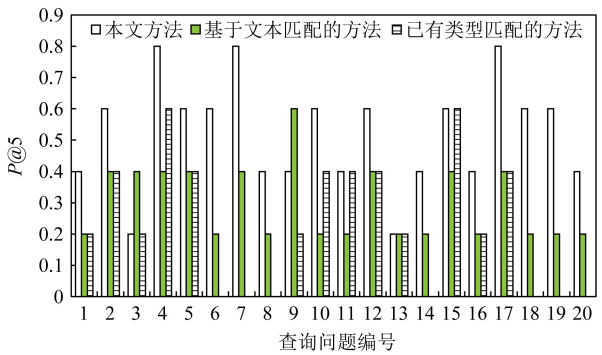


Fig. 10 $P@5$ of each question under different methods

图 10 不同方法下每个查询的 $P@5$

从图 10 和图 11 中可以看出,相比于基于输入/输出值的方法,本方法适用于更普遍的代码搜索问题,例如较好地解决了无法用具体值描述的查询 4,6,7 等编程任务.同时,本方法还能够有效地解决多种形式的类型转换问题,在已有类型匹配工作已解决的单个输入类型转换的基础上,解决了多个输入类型转换.为了能够更直观地验证本方法的有效性,

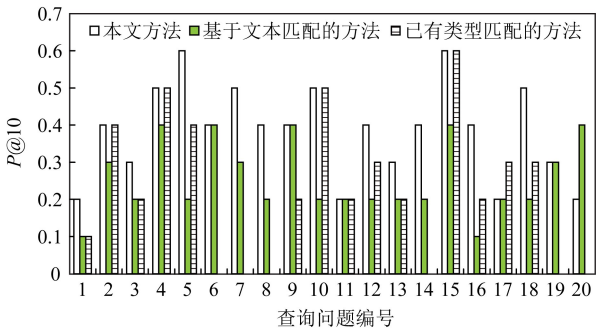


Fig. 11 $P@10$ of each question under different methods

图 11 不同方法下每个查询的 $P@10$

针对单个输入/输出类型的查询,对比了 2 种方法检索结果的 MAP 和 MRR 这 2 种评价指标,实验结果如表 4 所示:

Table 4 Experimental Results

表 4 实验结果

代码搜索方法	MAP	MRR
本文方法	0.56	0.60
基于文本匹配的方法	0.35	0.31
已有类型匹配的方法	0.41	0.47

从表 4 中可以看出,本方法的搜索准确率要优于基于文本匹配和已有类型匹配的方法.基于文本匹配的方法获得了最低的准确率,表明仅利用文本相似性解决代码匹配问题,而忽略代码的语义信息,难以获得理想的结果.本文结合静态分析技术挖掘代码的行为过程,并采用 Petri 网将其整合起来,添加到搜索中,使检索的准确性得到了显著提升.已有类型匹配的方法在搜索过程中只是遍历路径的存在问题,而本文在此基础上将参数类型个数作为新的约束条件,利用 Petri 网可达分析挖掘更全面的代码行为,从而匹配到更多相关的代码片段.

5.2.2 Petri 网构造过程分析

设单个代码片段中操作数为 m,每个操作的输入数据对象类型数为 n,则构建代码片段 Petri 网的时间复杂度为 $O(m\times n+m)$.本文设计实验分别从参数、语句、函数调用数目及重复使用对象集 4 个方面对 Petri 网的构造时长进行考察,如图 12 所示.实验结果表明,Petri 网的构造时间与语句数量、函数调用数量及重复使用对象数量均成正比关系,与参数数量大约为正比关系(除参数为 0 的情况).原因分析为:由时间复杂度可知,构造时长取决于代码操作与输入数据对象类型,且构造时长随二者数量的增加而增加,其中代码操作涉及代码语句及本文针对重复对象集设置的相应复制变迁;从语句内部分析,运算符操作通常对同类型数据对象加工,而函数调用往往需要处理多个不同类型的输入数据对象,因此代码中函数调用数量越多,涉及输入数据对象类型越复杂,导致构造时长增加;此外,在一般情况下参数数量与语句输入数据对象存在一定关联,参数数量越多,语句输入数据对象越多,但当参数为 0 时,输入数据对象与参数无直接关系,比如图 6 代码示例.

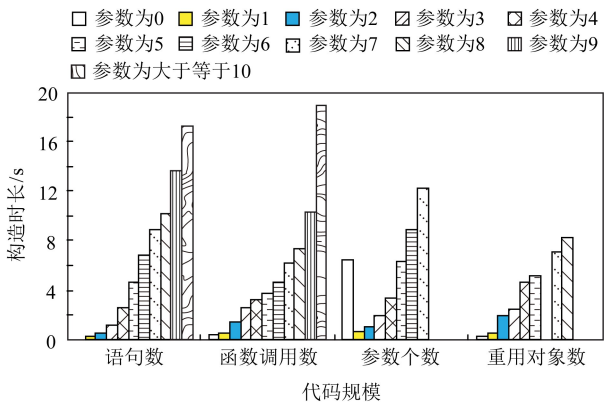


Fig. 12 Trend diagram of Petri Nets construction time and code scale

图 12 Petri 网构造时间与代码规模的趋势图

5.2.3 技术点分析

1) 复制变迁

为了表明在 Petri 网构建过程中增加复制变迁的重要性,本文针对表 3 的查询集进行检索实验,如图 13 所示.从图 13 中可知,复制变迁对提高搜索准确率起关键作用.原因分析为:传统 Petri 网模型难以蕴含程序片段中数据对象的重复使用过程,导致 Petri 网可达分析过程中重复使用对象的资源数不够,对应操作变迁无法触发,可达图生成不完整,进而影响搜索匹配结果.

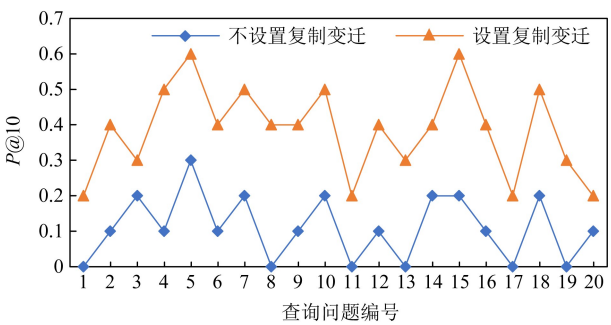


Fig. 13 Influence of cloned transition on search precision

图 13 复制变迁对搜索准确率的影响

2) 可达图修剪

图 14 说明了分析 Petri 网可达时设置可达图修剪的必要性.总的来看,在标准可达图生成算法的基础上,增加可达图修剪的操作能够明显减少搜索时间.

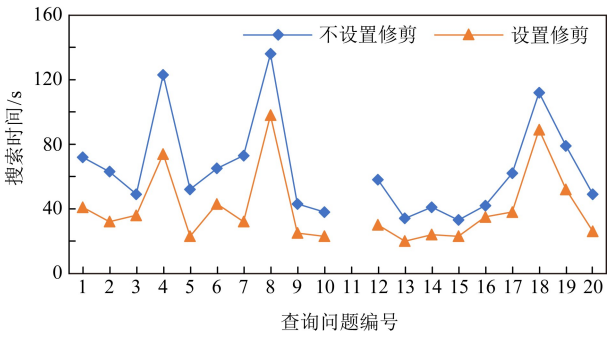


Fig. 14 Influence of reachability graph pruning on search time

图 14 可达图修剪对搜索时间的影响

6 总结及未来工作

为了帮助程序员更好地检索以及复用已有代码,本文提出了一种基于 Petri 网可达分析的代码搜索方法.该方法基于已有代码库,结合静态分析技术分析、提取代码元素,构建代码的 Petri 网模型;根据给定输入/输出类型,通过 Petri 网可达分析匹配将输入类型转换为输出类型的代码片段,并采用序列长度和复用率代码质量评价指标对结果进行排序.使用真实的查询测试集,验证了本文方法的有效性.

在未来的工作中,我们将针对搜索结果的多种情况,完善该方法.例如对于返回代码段过少的问题,采取对查询进行一定程度的放松与收缩(如父类、子类替换);对于返回代码段过多的问题,采用测试用例(具体的输入/输出值)进一步筛选代码段.此外,我们将进一步全方面探讨代码 Petri 网模型的可覆盖性、安全性、结构有界性等行为性质,增强代码 Petri 网模型的完备性.

作者贡献声明:丁雪儿负责论文思路的提出、论文写作、论文算法和论文实验;钮俊负责论文思路的把关、论文写作和实验监督;张开乐负责数据收集和论文写作完善;毛欣怡负责结果验证和论文写作完善.

参 考 文 献

[1] Gharehyazie M, Ray B, Filkov V. Some from here, some from there: Cross-project code reuse in GitHub [C] //Proc of IEEE MSR'14. Piscataway, NJ: IEEE, 2017: 291-301

[2] Qi Qing, Cao Jian, LiuYancen. The evolution of the software ecosystem in GitHub [J]. Journal of Computer Research and Development, 2020, 57(3): 513-524 (in Chinese)
(齐晴, 曹健, 刘妍岑. GitHub 中软件生态系统的演化[J]. 计算机研究与发展, 2020, 57(3): 513-524)

[3] Potvin R, Levenberg J. Why Google stores billions of lines of code in a single repository [J]. Communications of the ACM, 2016, 59(7): 78-87

[4] Liu Binbin, Dong Wei, Wang Ji. A review of intelligent program search and construction methods [J]. Journal of Software, 2018, 29(8): 2180-2197 (in Chinese)
(刘斌斌, 董威, 王戟. 智能化的程序搜索与构造方法综述[J]. 软件学报, 2018, 29(8): 2180-2197)

[5] Garcia-Contreras I, Morales J F, Hermenegildo M V. Semantic code browsing [J]. Theory & Practice of Logic Programming, 2016, 16(5): 721-737

[6] Stolee K T, Elbaum S, Dobos D. Solving the search for source code [J]. Transactions on Software Engineering and Methodology, 2014, 23(3): 26-67

[7] Stolee K T, Elbaum S, Dwyer M B. Code search with input/output queries: Generalizing, ranking, and assessment [J]. Journal of Systems and Software, 2016, 116(6): 35-48

[8] Jiang Renhe, Chen Zhengzhao, Zhang Zejun, et al. Semantics-based code search using input/output examples [C] //Proc of IEEE SCAM'18. Piscataway, NJ: IEEE, 2018: 92-102

[9] Mandelin D, Xu Lin, Bodik R, et al. Jungloid mining: Helping to navigate the API jungle [J]. ACM SIGPLAN Notices, 2005, 40(6): 48-61

[10] Thummalapenta S, Xie Tao. PARSEWeb: A programmer assistant for reusing open source code on the web [C] //Proc of the 22nd Int Conf on Automated Software Engineering. New York: ACM, 2007: 204-213

[11] Nguyen A T, Nguyen H A, Nguyen T T, et al. GraPacc: A graph-based pattern-oriented, context-sensitive code completion tool [C] //Proc of IEEE ICSE'34. Piscataway, NJ: IEEE, 2012: 1407-1410

[12] Sahavechaphan N, Claypool K T. XSnippet: Mining for sample code [J]. ACM SIGPLAN Notices, 2006, 41(10): 413-430

[13] Mishne A, Shoham S, Yahav E, et al. Typestate-based semantic code search over partial programs [J]. ACM SIGPLAN Notices, 2012, 47(10):997-1016

[14] Reiss S P. Semantics-based code search [C] //Proc of IEEE ICSE'31. Piscataway, NJ: IEEE, 2009: 243-253

[15] Lin Zeqi, Zhao Junfeng, Xie Bing. A code structure analysis and search method based on graph database [J]. Journal of Computer Research and Development, 2016, 53(3): 531-540 (in Chinese)
(林泽琦, 赵俊峰, 谢冰.一种基于图数据库的代码结构解析与搜索方法[J]. 计算机研究与发展, 2016, 53(3): 531-540)

[16] Ling Chunyang, Zou Yanzhen, Lin Zeqi, et al. Source code retrieval method for software projects based on graph embedding [J]. Journal of Software, 2019, 30(5): 1481-1497 (in Chinese)
(凌春阳, 邹艳珍, 林泽琦, 等. 基于图嵌入的软件项目源代码检索方法[J]. 软件学报, 2019, 30(5): 1481-1497)

[17] Gu Xiaodong, Zhang Hongyu, Kim S, et al. Deep code search [C] //Proc of IEEE ICSE'40. Piscataway, NJ: IEEE, 2018: 933-944

[18] Baldoni R, Coppa E, Delia D C, et al. A survey of symbolic execution techniques [J]. ACM Computing Surveys, 2018, 51(3): 50-89

[19] Ruizrube I, Person T, Dodero J M, et al. Applying static code analysis for domain-specific languages [J]. Software and Systems Modeling, 2020, 19(1): 95-110

[20] Amato G, Meo M C, Scozzari F. On collecting semantics for program analysis [J]. Theoretical Computer Science, 2020, 7(3): 1-25

[21] Kabir S, Papadopoulos Y. Applications of Bayesian networks and Petri Nets in safety, reliability, and risk assessments: A review [J]. Safety Science, 2019, 115(5): 154-175

[22] Han Zandong, Lee G B. Reduction method for reachability analysis of Petri Nets [J]. Tsinghua Science and Technology, 2012, 8(2): 231-235

[23] Czerwinski W, Lasota S, Lazic R, et al. The reachability problem for Petri Nets is not elementary [C] //Proc of the 51st Annual ACM SIGACT Symp on the Theory of Computing. New York: ACM, 2019: 24-33

[24] Raghothaman M, Wei Yi, Hamadi Y, et al. SWIM: Synthesizing what I mean [C] //Proc of the 38th Int Conf on Software Engineering. New York: ACM, 2016: 357-367

[25] Li Xuan, Wang Qianxiang, Jin Zhi. Code search method based on enhanced description [J]. Journal of Software, 2017, 28(6): 1405-1417 (in Chinese)
(黎宣, 王千祥, 金芝. 基于增强描述的代码搜索方法[J]. 软件学报, 2017, 28(6): 1405-1417)

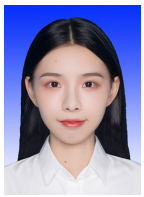
[26] Semura Y, Yoshida N, Choi E, et al. CCFinderSW: Clone detection tool with flexible multilingual tokenization [C] //Proc of IEEE APSEC'21. Piscataway, NJ: IEEE, 2017: 654-659

[27] Satter A, Sakib K. A search log mining-based query expansion technique to improve effectiveness in code search [C] //Proc of IEEE ICCIT'19. Piscataway, NJ: IEEE, 2016: 586-591

[28] Nie Linming, Jiang He, Ren Zhilei, et al. Query expansion based on crowd knowledge for code search [J]. IEEE Transactions on Services Computing, 2017, 9(5): 771-783

[29] Rahman M M, Roy C K, Lo D, et al. RACK: Code search in the IDE using crowdsourced knowledge [C] //Proc of IEEE ICSE'39. Piscataway, NJ: IEEE, 2017: 51-54

[30] Lü Fei, Zhang Hongyu, Lou Jianguang, et al. CodeHow: Effective code search based on API understanding and extended Boolean model [C] //Proc of IEEE ASE'30. Piscataway, NJ: IEEE, 2015: 260-270



Ding Xue'er, born in 1996. Master. Member of CCF. Her main research interests include code search and recommendation.
丁雪儿,1996 年生.硕士,CCF 会员.主要研究方向为代码搜索与推荐.

Niu Jun, born in 1976. PhD, associate professor. Member of CCF. His main research interests include code search, formal verification, and service computing.
钮 俊,1976 年生.博士,副教授,CCF 会员.主要研究方向为代码搜索、形式化验证、服务计算.

Zhang Kaile, born in 1994. Master. His main research interests include code search and machine learning.
张开乐,1994 年生.硕士.主要研究方向为代码搜索、机器学习.

Mao Xinyi, born in 1993. Master. Her main research interests include formal verification and service computing.
毛昕怡,1993 年生.硕士.主要研究方向为形式化验证、服务计算.