基于 k 近邻最弱前置条件的程序多路径验证方法

郭 曦[1),4)] 王 盼[2)] 王建勇[1)] 张焕国[3)]

[1)](华中农业大学信息学院 武汉 430070)

[2)](武汉电力职业技术学院电力工程系 武汉 430079)

[3)](软件工程国家重点实验室(武汉大学) 武汉 430072)

[4)](佐治亚理工学院计算机科学系 亚特兰大 30332 美国)

摘 要 程序多路径验证方法是对软件性质进行发掘的重要方法之一,现有的验证方法主要通过求解路径条件或者自动构造不同的输入来触发生成不同的路径,从而分析程序中潜在的安全问题,但存在对路径条件不加选择地进行多路径扩展而生成缺乏针对性的路径的问题,另外由于路径条件过长而难以求解也限制了它的使用范围.该文提出基于 k 近邻最弱前置条件的程序多路径验证方法,该方法通过后向符号分析对程序调用图的构建过程进行改进,同时对指定的程序检测点生成最弱前置条件,并以该最弱前置条件为引导信息使用符号执行的方法在保证检测点可达的前提下有针对性地生成对程序性质进行验证的精简路径集合.实验结果表明,该方法可以提高程序验证的精度和准确性,并减少误报.

关键词 程序验证;静态分析;最弱前置条件;符号执行;控制流图

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2015.02203

Program Multiple Execution Paths Verification Based on k Proximity Weakest Precondition

GUO Xi[1),4)] WANG Pan[2)] WANG Jian-Yong[1)] ZHANG Huan-Guo[3)]

[1)](*School of Informatics*, *Huazhong Agricultural University*, *Wuhan* 430070)

[2)](*Department of Electric Power Engineering*, *Wuhan Electric Power Technical College*, *Wuhan* 430079)

[3)](*State Key Laboratory of Software Engineering*, *Wuhan University*, *Wuhan* 430072)

[4)](*Department of Computer Science*, *Georgia Institute of Technology*, *Atlanta*, 30332, *USA*)

Abstract Program multiple paths verification is one of the key methods in the exploring of software properties. Current verifications usually trigger the generation of different execution paths via solving the path conditions or constructing inputs automatically to verify the underlying security problems in programs. However, this method extends the multiple paths without choosing the proper path conditions, which leads to the generation of redundant paths, meanwhile, the length of path conditions is usually too long, which restrains its application domain. A method of program multiple paths verification based on k proximity weakest precondition is proposed in this paper, which improves the generation of call graph of the program, and combines the backward symbolic analysis to generate the weakest preconditions in specific checking points. The results of weakest precondition are used as the guidance for symbolic execution to generate proper paths

收稿日期:2013-04-28;最终修改稿收到日期:2015-03-26.本课题得到国家自然科学基金(61332019,61173138,61272452,91118003)、国家"九七三"重点基础研究发展规划项目基金(2014CB340600)、国家"八六三"高技术研究发展计划项目基金(2015AA016002)、湖北省自然科学基金(2014CFB144)和中央高校基本科研业务费专项基金(2662015QC009)资助.**郭 曦**,男,1983 年生,博士,讲师,主要研究方向为可信软件安全性分析与测试、信息安全等.E-mail:seyeyesx@163.com.**王 盼**,女,1987 年生,硕士,讲师,主要研究方向为电力电子功率变换等.**王建勇**,男,1974 年生,硕士,副教授,主要研究方向为计算机网络、生物信息学.**张焕国**,男,1945 年生,教授,博士生导师,主要研究领域为信息安全、密码学.

that can verify the properties of the program on the premise of reaching the specific checking points. The experimental results demonstrate that this method can enhance the precision and accuracy of program verification, and reduce the false positive.

Keywords program verification; static analysis; weakest precondition; symbolic execution; control flow graph

1 引 言

程序分析的目的是为了验证系统代码是否满足给定的安全准则,以达到对程序所具有的性质进行验证的目的.传统的人工分析方法由于效率较低且容易出现误报和漏报等现象,故自动化的程序验证方法和错误检查方法成为当前主流的研究方向,如定理证明[1]、约束求解[2]、模型检验[3]等.

程序验证的重点是对程序属性的分析,例如程序中某条语句是否可达,多线程程序中是否存在竞争或互斥等逻辑属性.目前程序属性验证工具的主要目标是判断程序中是否存在违反待验证属性的执行路径,即判断该工具是否会生成对应的反例程序.

路径可行性判定问题是程序验证中的重要问题[4],它指的是对于一条指定的程序执行路径,通过符号执行[5](Symbolic Execution,SE)和约束求解等方法判断该程序的输入变量是否存在一组初始取值,使得程序能够沿着该路径执行.由于属性验证工具大多都采用对源代码进行抽象的方法来生成反例,故源代码对应的抽象模型会直接影响到反例的正确性.最弱前置条件[6](Weakest Precondition,WP)作为一种后向符号分析方法(Backward Symbolic Analysis),能够可靠地对程序的执行语义进行建模,并减小静态分析过程中对程序状态进行抽象操作所带来的验证精度上的损失.但最弱前置条件分析在实际应用过程中,往往由于程序对应的控制流图(Control Flow Graph,CFG)规模较大,其路径条件的解空间与程序分枝数量呈指数关系,即存在路径搜索空间爆炸的问题.此外最弱前置条件在计算过程中依赖循环变量,故可能存在死循环等问题影响验证的效果.

目前程序多路径分析与验证存在的主要问题是分析效率较低,在分析过程中容易产生大量不可达或不能对性质进行验证的无用路径,同时大多数分析方法仅用求解器对路径约束条件进行优化求解,缺乏对程序检测点和路径条件之间关联关系的深入研究,导致由于没有足够的路径引导信息而产生较多缺乏针对性的程序执行路径,从而降低对性质进行验证的精度与效率.本文针对以上问题,以程序路径的可满足性为研究对象,提出 k 近邻最弱前置条件(kWP)的多路径程序验证方法:在分析过程中不立即对路径条件进行求解,而是加入后向符号分析所生成的路径信息,当遇到程序的检测点的时候,根据路径信息有针对性的生成执行路径并进行筛选,从语义上对路径表达式进行简化与合并,减少待分析的路径数量,并减少求解器的误报与漏报.具体的过程是通过对程序的控制流图进行改进,消除循环结构,使得每条执行路径都可以表示成惟一的语句串;通过引入因子 k,可以得到一个与待分析的路径的编辑距离(Edit Distance)不超过 k 的路径集合,对该集合中的每条路径进行 kWP 运算并对运算结果进行析取操作,这样就可以得到对输入变量的一个近似的取值,通过不断优化调整 k 的大小,可以得到更为精确的变量输入值,从而引导程序执行到待验证的检测点,并减小路径空间爆炸在程序验证过程中所带来的影响,故 kWP 是对传统的最弱前置条件的一种近似.基于 kWP 的路径可行性问题是需求驱动(demand driven)的,符号分析的结果可以及时反馈于符号执行过程中的程序分支路径的选择,故与传统的符号执行相比,本文的方法在路径搜索过程中有更强的针对性.

本文的主要贡献是:

(1)在程序调用图的构建过程中加入后向符号分析,使得后向符号分析的结果能够引导程序调用图的构建,从而避免生成与待验证性质无关的程序调用图,提高验证的精度与效率.

(2)提出基于 k 近邻最弱前置条件的程序多路径分析方法,通过计算程序中断言的最弱前置条件并调整 k 的取值,从而生成有针对性的可行路径并对待验证的性质进行判定.

本文第 2 节介绍与论文相关的概念,给出最弱前置条件和过程间程序调用图的计算方法;第 3 节给出 kWP 的形式化定义和 k 近邻路径生成算法;

第 4 节从输入域的角度对本文方法的有效性进行证明，并给出本文方法的分析过程；第 5 节介绍实验环境和系统的框架，并对基准测试集的结果进行分析；第 6 节给出相关研究进展和对比；第 7 节总结全文并展望下一步研究工作.

2　最弱前置条件的生成方法

定义 1.　控制流图(Control Flow Graph，CFG). 程序的控制流图是由节点和边组成的有向图，可以形式化地表示为$\langle N,E\rangle$. 其中 N 是节点的集合，每个节点代表程序中指定位置的语句，边的集合 $E\subseteq N\times N$ 表示程序在执行过程中节点间可能的流向.

定义 2.　CFG 路径的可行性. 它表示为$N\times N\rightarrow\{T,F\}$的计算过程，对于一对节点$(n_i,n_j)$，若存在一个节点序列 $\pi:=\langle n_0,n_1,\cdots\rangle$，使得$(n_k,n_{k+1})\in E$，其中 $0\leq k\leq|\pi|-1$，$n_0=n_i$，$n_{|\pi|-1}=n_j$，那么这一对节点之间存在可达的路径；反之若不存在节点序列，则表示不存在可达路径. 用$(n_0,n_1,\cdots,n_k]$表示所有以 n_0的后继节点为起点，以 n_k为终点的路径；用$[n_0,n_1,\cdots,n_k)$表示所有以 n_0为起点，以 n_k的前驱节点为终点的路径，使用$|\pi|$表示当前路径的长度.

定义 3.　路径条件(Path Condition). 它是在控制流迭代过程中产生分支路径的数据流值. 在 CFG 中，新的路径只会在条件语句与循环语句的判断条件处产生，故在 CFG 中节点的路径条件为此处条件分支或循环分支的约束所组成的一阶逻辑公式(First-Order Logic，FOL).

在本文方法的分析过程中，可以将 CFG 的表示形式扩展为$\langle N,E,N_{\text{infea}},\varphi\rangle$. 其中 N_{infea}表示程序的可行路径不能到达的节点的集合；φ 表示待验证路径所表示的状态，即断言的集合. 程序从 CFG 的起始节点开始，依据路径的可行性定义运行到待验证的位置，若要验证程序在该处的断言 φ 成立，则需要验证程序以不同的执行路径达到该处时，该断言 φ 都要成立.

定义 4.　静态单赋值(Static Single Assignment，SSA). 静态单赋值是程序的一种中间表示(IR)形式，在程序中若出现了对变量 a 的多次赋值，即该变量存在多处定义，则通过对变量 a 采用下标的方式$(a_0,a_1,\cdots)$来区分 a 在每处的定义，同时在程序的控制流图中加入一个 φ 函数节点：$a=\varphi(a_0,a_1,\cdots)$，使得每个变量 a_i均有惟一的一处定义，同时变量 a_i也只有惟一的一条到达该定义的路径.

在计算过程内最弱前置条件之前，需要将程序转换为其对应的静态单赋值形式，使得程序的过程内控制流图中每一个基本块(Basic Block)最多代表一条语句，同时过程内控制流图拥有惟一的入节点(Entry node)和出节点(Exit node). 静态单赋值操作可以有效地区分程序中的变量与其存储位置，使得程序控制流图中每条边代表一次路径条件的运算过程，若程序中存在异常退出等情况，则有一条指向出节点的边，以提高后向符号分析的效率.

定义 5.　最弱前置条件(Weakest Precondition，WP). 它是保证程序中一条语句正常执行并满足结果断言(后置条件)的限制最小的前提条件. 它表示为一组谓词公式：WP(S,R)，其中 R 是语句 S 执行后所期望的结果断言.

在证明程序正确性过程中，以程序的执行结果作为最后一条语句的后置条件，通过该后置条件和最后一条语句可以计算出最后一条语句的最弱前置条件，以此类推到程序的开始，第一条语句的前置条件即为满足结果断言所需的条件. 若该条件包含在程序的输入中，则可证明程序是正确的. 表 1 列举了程序中常见类型语句的最弱前置条件的转换方法，当过程内控制流图中的节点有出边的时候就需使用该转换方法. 其中 $\varphi[s_1/s_2]$表示使用语句 s_2 去替换语句 s_1，谓词 get 和 set 分别表示获取和设置变量的值.

表 1　常见语句的 WP 转换方法

语句 s	最弱前置条件 WP(s,φ)
$v=u$	$\varphi[u/v]$
$v=u_1$ op u_2	$\varphi[(u_1\ \text{op}\ u_2)/v]$
$v=u.m$	$(u\neq\text{null})\wedge\varphi[\text{get}(u,m)/v]$
$v.m=u$	$(v\neq\text{null})\wedge\varphi[\text{set}(u,m)/v]$
$u[i]=v$	$(u\neq\text{null})\wedge(i\geq 0)\wedge(i<\text{get}(length))\wedge\varphi[v/\text{get}(u,i)]$
$v=u[i]$	$(u\neq\text{null})\wedge(i\geq 0)\wedge(i<\text{get}(length))\wedge\varphi[\text{get}(u,i)/v]$
$v=(T)v'$	$(v'\neq\text{null})\wedge\text{subType}(\text{typeOf}(v'),T)\wedge\varphi[v'/v]$
assume v	$v\wedge\varphi$

算法 1 是过程内最弱前置条件的生成算法. 对于程序中的每一条语句 s 和它对应的后置条件 φ_{post}，该算法计算过程内控制流图中每个节点的符号状态的集合 S，并对结果进行简化(simplify)，同时在其基础上进行合并(merge)操作. 简化操作是基于最弱前置条件所具有的析取性质：

$$\text{WP}(s,\varphi_1\vee\varphi_2)=\text{WP}(s,\varphi_1)\vee\text{WP}(s,\varphi_2).$$

它是状态集合中被选取的条件的析取范式(Disjunctive Normal Form，DNF).

下面给出该算法所采用的合并策略,即对当前的符号状态进行重写操作,例如 merge($\varphi \wedge a, \varphi \wedge \neg a$).

$$(x \leq y) \wedge (x \neq y) \rightarrow x < y;$$

$$(x < y) \wedge (x \neq y) \rightarrow x < y;$$

$$0 \leq \text{get}(arrayLength) \rightarrow \text{true};$$

$$(\text{typeOf}(x) = T) \wedge (x = \text{null}) \rightarrow \text{false};$$

$$\text{subType}(a, t_1) \wedge \text{subType}(a, t_2) \wedge \text{subType}(t_1, t_2) \rightarrow \text{subType}(a, t_1);$$

其中谓词 get 可同时用来判断数组是否出现越界现象,谓词 typeOf 与 subType 用来处理程序的类型判断与类型转换语句.此外在合并过程中还使用了常量折叠(constant folding)的分析方法,该方法作为程序优化中的一种技术可以将程序中的常量表达式存入常量表,以减少最弱前置条件计算过程中求解器对常量表达式的计算求值次数,同时该方法可以增加程序中需要多次被调用的过程的分析结果的重用率.

算法 1. 过程内最弱前置条件(WP_{intra})的生成算法.

输入: CFG, φ_{post}

输出: intraprocedural weakest precondition

1. var F: $\{Formula\} \leftarrow Statement$;
2. var $List$: pair of ($Statement$, $Formula$);
3. $\forall s \in Statement, F(s) \leftarrow \varnothing$;
4. $List \leftarrow \{(ExitNode, \varphi_{\text{post}})\}$;
5. WHILE $List \neq \varnothing$ DO
6. $(s', \varphi_{\text{post}}) \leftarrow$ select from $List$;
7. WHILE $(s, s') \in E$ of CFG DO
8. $\varphi_{\text{pre}} \leftarrow \text{merge}(F(s), \text{simplify}(\text{WP}(s, \varphi_{\text{post}})))$;
9. IF $\varphi_{\text{pre}} \neq$ false THEN
10. $F(s) \leftarrow F(s) \cup \varphi_{\text{pre}}$;
11. $List \leftarrow List \cup (s, \varphi_{\text{pre}})$;
12. ENDIF
13. ENDWHILE
14. ENDWHILE
15. RETURN $F(EntryNode)$

对于过程间的最弱前置条件 WP_{inter},本文采用上下文敏感的分析方法,其过程间控制流图(ICFG)由多个过程内控制流图连接而成.对于一个调用 $v = u.f(\cdots)$(此处省略其参数列表),WP_{inter} 首先将条件 φ_{post} 映射到被调用的函数 $f(\cdots)$ 的命名空间,得到函数 $f(\cdots)$ 在出节点 $ExitNode$ 处的分析结果,然后通过上述过程内最弱前置条件算法 WP_{inter} 获得函数 $f(\cdots)$ 在入节点 $EnterNode$ 处的最弱前置条件 φ_{pre},即产生从后置条件到前置条件的一个映射:$\varphi_{\text{post}} \xrightarrow{f(\cdots)} \varphi_{\text{pre}}$,该映射表明 φ_{pre} 作为前置条件可以引导程序获得出节点 $ExitNode$ 处的后置条件 φ_{post},最后不断通过过程内控制流图的传递计算出控制流图的前置条件 φ_{pre}.

在得到过程间最弱前置条件的基础上,由于程序中存在大量的过程调用,此时若直接依据过程间的关系生成调用图,则会产生较多无用的调用关系,影响程序验证的效率.本文在过程间最弱前置条件生成过程中,使用后向符号分析的结果来引导程序调用图的构建,从而生成与待验证的兴趣点相关联的过程调用图.

算法 2. 带过程间最弱前置条件(WP_{inter})的程序调用图生成算法.

输入: CFG, φ_{post}

输出: call graph under WP_{inter}

callGraphWithWP (CFG, φ_{post})

1. var $CG \leftarrow \text{WP}_{\text{inter}}(\text{CFG}, \varphi_{\text{post}})$;
2. var $CG' \leftarrow \{cg \in CG \mid cg \text{ can be solved}\}$;
3. IF $CG' = \varnothing$
4. FOREACH $m \in CG$ DO
5. select one $method$ from m;
6. select $s \notin$ subroutines in $method \wedge (m \wedge method = s$ can be solved);
7. IF s does not exist
8. CONTINUE;
9. ENDIF
10. $\text{CFG}' \leftarrow \text{CFG} + s$;
11. RETURN callGraphWithWP($\text{CFG}', \varphi_{\text{post}}$);
12. ENDFOR
13. ELSE
14. RETURN CG';

算法 2 中第 4~11 行通过判断过程内控制流图的函数调用关系来扩展程序中的调用关系,并使用递归运算得出与兴趣点相关的程序调用图.由于该精简的调用图删除了与待验证性质无关的函数调用集合,故可以在此基础上生成有针对性的路径验证集合.

图 1 表示一个程序片段(图 1(a))以最弱前置条件为引导信息的程序验证过程.程序第 10 行存在一条断言语句,若该语句被触发即表示验证过程失败.文献[7]指出:若程序中一条具体的执行路径不可行,那么其临近路径也是不可行的概率较高.本文据此理论采用后向符号分析方法,使用最弱前置条件和断言得到的具体执行路径来构建有限符号分析树,并生成与此路径临近的尽可能多的可行路径来

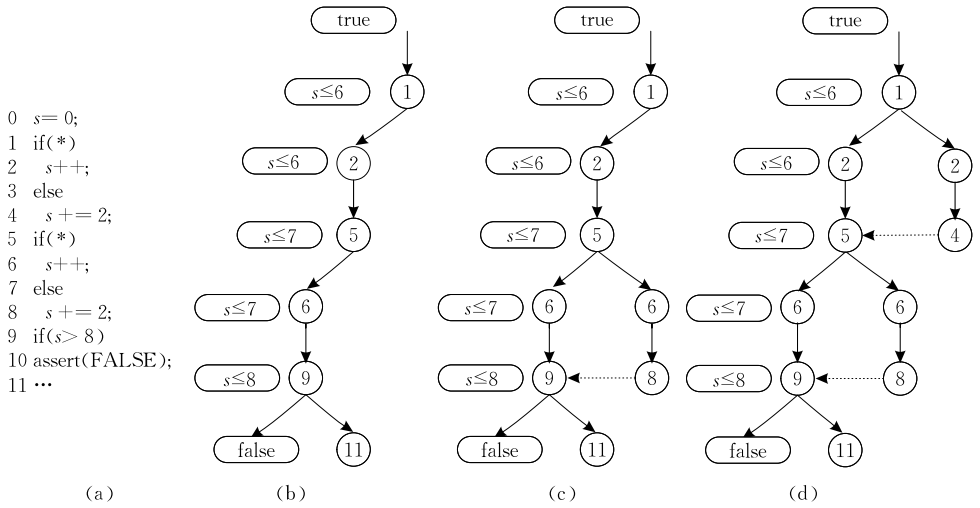


图 1　带最弱前置条件的符号分析树

对程序进行验证. 在最弱前置条件的计算过程中的两个主要的规则表示如下：

$$\mathrm{WP}(x:=e,Q)=Q[e/x];$$

$$\mathrm{WP}(\mathrm{if}(c)s_1\ \mathrm{else}\ s_2,Q)=$$

$$(c\Rightarrow \mathrm{WP}(s_1,Q))\wedge(\neg c\Rightarrow \mathrm{WP}(s_2,Q));$$

图 1(b)～(d)表示该程序片段所对应的符号分析树，程序中(*)表示该处的验证条件可能为 true 也可能为 false，在分析过程中主要的问题就是如何使用这些规则来产生路径条件对应的合取公式. 在符号分析过程中，求解器会在每一个分支位置对路径条件进行计算以删除那些冗余的或者不可解的路径条件. 图 1(b)表示后向符号分析的第 1 阶段，其中在每个节点处加入了最弱前置条件：$\tau_1: s_0\leq 6$；$\tau_2: s_0\leq 6$；$\tau_5: s_1\leq 7$；$\tau_6: s_1\leq 7$；$\tau_9: s_2\leq 8$，例如 $\tau_6: s_1\leq 7$ 可以通过 $\mathrm{WP}(s_2=s_1+1, s_2\leq 8)$ 计算得出，注意此处采用了 SSA 方法的表达形式. 图 1(c)表示后向符号分析的第 2 阶段，在第 9 行由于符号状态 $s_0=0\wedge s_1=s_0+1\wedge s_2=s_1+2\rightarrow s_2\leq 8$，故在此处可以使用上述 simplify 与 merge 操作来对路径分析的结果进行约简，其中虚线箭头表示被包含的路径. 类似地，在图 1(d)中由于 $(s_0=0\wedge s_1=s_0+2)\rightarrow s_1\leq 7$，故通过该分析方法之后，符号分析树的规模与分支的数目呈线性关系，有效地减少了分析空间的规模.

3　k 近邻路径生成方法

设程序 $Prog$ 在其对应的过程间控制流图中的执行路径为集合 $Paths$，为了验证 $Prog$ 的可达性，需要在程序中待分析的检测点设置断言 φ，通过计算 φ 与此处的路径条件(Path Condition，PC)的逻辑包含关系来对 $Prog$ 进行验证，PC 是程序能执行到此处的由约束条件所组成的一阶逻辑公式.

定义 6.　编辑距离(Edit Distance). 它是指两个串 (S,T) 之间，由一个串通过替换、插入、删除操作转换成另一个串的最少操作次数.

在实际应用过程中，常采用动态规划的方法来计算编辑距离，其动态规划公式为

(1) $i==0\&\&j==0$, $\mathrm{edit}(i,j)=0$；

(2) $i==0\&\&j>0$, $\mathrm{edit}(i,j)=j$；

(3) $i>0\&\&j==0$, $\mathrm{edit}(i,j)=i$；

(4) $i>0\&\&j>0$, $\mathrm{edit}(i,j)=\min(\mathrm{edit}(i-1,j)+1, \mathrm{edit}(i,j-1)+1, \mathrm{edit}(i-1,j-1)+f(i,j))$；

其中 $\mathrm{edit}(i,j)$ 表示 S 的子串 $[0\cdots i]$ 到 T 的子串 $[0\cdots j]$ 的编辑距离，$f(i,j)$ 表示 S 中第 i 个字符 $S[i]$ 转换到 T 中第 j 个字符 $T[j]$ 所需要的操作次数，若 $S[i]==T[j]$，则 $f(i,j)=0$；否则 $f(i,j)=1$.

本文使用编辑距离计算两条执行路径的相似度，因子 k 用来指导生成一个与当前分析路径 l 相似的路径集合 L，其中每条路径与 l 的编辑距离相差不超过 k，显然 $L\subset Paths$：

$$L=\{\varepsilon\mid\varepsilon\in Paths\wedge\triangle(l,\varepsilon)\leq k\},$$

其中算子△用来计算待分析的路径与当前生成的路径之间的编辑距离. kWP 的定义如下：

$$k\mathrm{WP}:\{(s,pc,path,k)\in Prog\times PC\times Paths\times k\rightarrow PC\}.$$

上式所得到的是经最弱前置条件计算后的更新的路径条件集合，通过 SMT 求解器[8]可得到路径 l 所对应的输入变量的近似值. 对于程序中常见的循环语句，其执行路径在过程间控制流图中表示为一个带回边的节点集合. 图 2 是一个带循环的控制流

图及其对应的消除循环节点的控制流图 CFG$^{\infty}$. 图中每条边都有一个符号与其对应,这样每条执行路径都有一个特定的符号串,即该路径的编码. 对于循环体执行 n 次的某条执行路径,其路径编码可以记录为 CFG^{n},显然 CFG^{n}是 CFG$^{\infty}$的一个子图. 算法 3 是计算与指定路径的编辑距离不超过 k 的执行路径集合的生成算法.

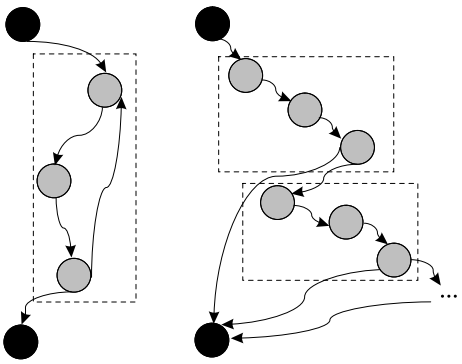


图 2 循环结构过程间控制流图及其 ICFG$^{\infty}$形式

算法 3. k 近邻路径生成算法.

输入: k,ICFG,l

输出: $Paths$ with k distance to l

1. $G\leftarrow$reverseArcs(ICFG);
2. $l'\leftarrow$reverse(l);
3. $Paths\leftarrow\varnothing$;
4. $q.append(l[0],\langle\ \rangle)$;
5. WHILE $q\neq\varnothing$ DO
6. $\langle v,p\rangle\leftarrow q.dequeue()$;
7. IF $(v=l'[|l'|-1])\wedge(\triangle(p,l')\leq k)$ THEN
8. $Paths\leftarrow Paths\cup\{p\}$;
9. ENDIF
10. $index\leftarrow\min(|p|-1,|l|)$;
11. IF$\triangle(p,l'[0,index])\leq 2k$ THEN
12. $p.append(v)$;
13. select n from V;
14. $q.append(\langle n,p\rangle)$;
15. ENDIF
16. ENDWHILE
17. RETURN $Paths$

其中算子△用来计算两个路径串之间的编辑距离,对于一条执行路径 l,变量$|l|$表示 l 中节点的个数,$l[i,j]$是路径 l 的子串$(0\leq i<j\leq|l|-1)$,$l[|l|-1]$表示 l 中最后一个节点. 该算法通过对参数 ICFG 所表示的控制流图和待分析的执行路径的指向进行翻转,若生成的执行路径与 l 的编辑距离不超过 $2k$,则不断扩展搜索的路径,在能够到达程序入节点的路径中,若 k 的值不能更小,则保留该执行路径. 故对于距离执行路径 l 的编辑距离不超过 k 的路径集合中,存在对应的 CFG^{n}. 值得注意的是该算法可能生成不可达的执行路径,例如程序的循环次数超过了设定的一个阈值,那么在随后的符号分析过程中,符号执行工具所采用的求解器会对当前的谓词集合进行求解以确定其可满足性. 例如对于 $\alpha<3\wedge\alpha=5$ 这样谓词集合不存在解,故对应的条件在传递到下一个节点前会被删除. 对于最后所生成的可执行路径集合 L,其 kWP 是各路径最弱前置条件的析取:

$$k\mathrm{WP}(Prog,PC,Paths,k)=\bigvee_{l\in L}\mathrm{WP}(l,\varphi).$$

当 $k=0$ 时,kWP 对当前待分析的路径进行分析;随着 k 的增大,算法 3 将会生成越来越多的符合条件的路径. 在 $k\to\infty$的情况下,kWP 的计算方法与传统最弱前置条件的计算方法存在如下关系:

$$\lim_{k\to\infty}k\mathrm{WP}(Prog,PC,Paths,k)=\mathrm{WP}(Prog,PC),$$

当 k 值发生变化的时候,算法会生成不同的路径集合,这样就需要在分析精度和分析的规模之间进行权衡,以获得最优的分析结果,本文的方法比传统的最弱前置条件分析方法能更加有效地获得反例程序.

4 kWP 的有效性

为了验证本文方法的有效性,从路径可满足性的角度提出验证方法. 对于程序 $Prog$,其输入域为 I,对应的值域为 O,则程序可以表示为输入集合到输出集合上的映射 $I\to O$. 对于输入 $i(i\in I)$,其对应的执行路径记为 l,待验证的断言为 φ,一条可满足的路径可以表示为 $Prog(i)\models\varphi$.

定义 7. 失效路径输入域 $I^{\mathrm{infea}}=\{i\mid i\in I,Prog(i)\models\neg\varphi\}$. 上节分析到 kWP 可能会生成不可达的路径,故使用 I^{infea}表示那些不能被满足或者不能到达指定位置的路径所对应的输入值. 本文的方法生成与设定的因子 k 相关的路径集合,在此阶段可以不断调整 k 的取值,以发掘类似的可达路径集合.

定义 8. kWP 可达路径输入域 $I^{\mathrm{fea}}_{k\mathrm{WP}}=\{i\mid i\in I\backslash I^{\mathrm{fea}},Prog(i)\models\varphi\}$. 本文可达路径输入域是在已知可满足的路径的基础上通过 kWP 计算得出的,其结果是对程序的可达路径输入域 I^{fea}的一种逼近,显然 $I^{\mathrm{fea}}_{k\mathrm{WP}}\subset I^{\mathrm{fea}}$,理想情况下有 $I^{\mathrm{fea}}_{k\mathrm{WP}}=I^{\mathrm{fea}}$. 图 3 表示这几种输入域之间的包含关系,$I^{\mathrm{fea}}_{k\mathrm{WP}}$中的阴影区域

表示 kWP 生成的路径集合中那些无效的路径集合或者不能对性质进行验证的路径集合，在最后生成的可行路径集合中需要将该无效路径集合删除.

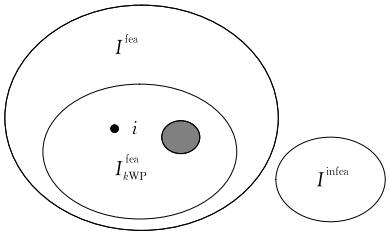


图 3　几种输入域之间的关系图

在定义 4 中，记那些在 kWP 计算过程中对应输入 i 所生成的不可达路径为 P^{i}_{infea}，则有如下定义.

定义 9.　kWP 程序验证问题. 设待分析的程序 $Prog$，它的一条可执行程序路径 l 对应的输入为 i，待验证的条件为 φ，经 kWP 计算得到与 l 的编辑距离不超过 k 的路径集合及其对应的前置条件 WP，对于其中的可达路径，以 WP 为引导条件对程序进行符号执行操作所得到的后置条件记为 P，通过计算逻辑蕴含 $P\rightarrow\varphi$ 来判断路径的可满足性.

对于 k 的不同取值所生成的路径集合，为了比较所生成的路径可行性的程度，可以对 k 的优化取值做出如下定义.

定义 10.　k 的优化取值问题. 对于不同的 k 值所生成的可行路径后置条件 P^{k_1}、P^{k_2}，以及所生成的不可行路径所对应的输入域 I^{infea}_1、I^{infea}_2，称 k_1 比 k_2 更加优化，当且仅当 P^{k_1} 逻辑蕴含 P^{k_2}，且 I^{infea}_1 是 I^{infea}_2 的子集，即 $k_1 \rhd k_2 \Leftrightarrow (P^{k_1}\rightarrow P^{k_2} \wedge I^{infea}_1 \subseteq I^{infea}_2)$.

下面给出本文验证程序可行性所采用的计算方法：

$$\mathrm{exe}_{SE}(Prog, k\mathrm{WP}(Prog, PC, \mathrm{exe}(Prog(i)), k) \wedge PC).$$

(1) 上式中的 exe($Prog(i)$)函数表示以数据 i 为输入来执行程序 $Prog$，获得指定的路径 l；

(2) kWP($Prog$, PC, exe($Prog(i)$), k)在计算过程中采用本文的方法生成与 l 的编辑距离不超过 k 的路径集合，并计算出最弱前置条件 R_{WP}；

(3) 最后使用 R_{WP} 作为引导条件进行符号执行的操作，获得程序的后置条件 P，通过计算路径可满足性验证条件 δ：$P \wedge PC$ 来验证生成的路径的可达性，即采用逻辑蕴含的方法来判断 $P\rightarrow PC$ 是否成立，如果成立，则表示生成的路径是可达的；如果不成立，则可以生成一组对应的路径条件，表明当前的路径不可达，并生成相应的程序输入数据.

定理 1.　程序过程内控制流图中一条可行路径 l 的起始节点和末尾节点分别记为 n_h 和 n_t，在 n_t 处待验证的断言为 φ，若使用本文方法生成的路径可满足验证条件 δ 是不可满足的，则在程序的过程间控制流图中流经 l 的执行路径在 n_t 处可使得 φ 成立.

证明.　本文计算路径可满足的验证条件 δ 是在 n_t 处程序的路径条件 pc 与 $\neg\varphi$ 的合取操作：$pc \wedge \neg\varphi$. 若验证条件 δ 是不可满足的，则表示 $\neg(pc \wedge \neg\varphi)$ 成立，即 $\neg\delta\rightarrow\neg(pc \wedge \neg\varphi)$. 依据命题逻辑可得 $\neg(pc \wedge \neg\varphi)\Leftrightarrow pc\rightarrow\varphi$，故 $pc\rightarrow\varphi$ 也成立，即在程序的过程间控制流图中流经 l 的执行路径在 n_t 处可使得 φ 成立.　证毕.

5　实验与分析

本文采用 k 近邻最弱前置条件和符号执行的方法来分析程序的路径可行性问题，通过比较 kWP 与传统的符号执行在发掘程序可行路径方面的数量与效率来验证本文方法的有效性. 本文以 WALA① 作为实验平台，使用平台提供的 T. J. Watson Libraries 作为程序过程内和过程间分析的工具，并构建程序的控制流图与调用图，在因子 k 的作用下生成待分析的路径集合，并使用 CVC3 工具[9] 对本文算法 1 过程中所得到的路径谓词进行计算，保留那些可以求解的路径谓词集合，最后符号执行工具 Java PathFinder[10] 以该谓词集合作为前置条件引导程序运行，并对路径的可行性进行验证. 在实验过程中涉及到的验证性质包括缓冲区溢出、悬空指针、内存泄漏、类型状态属性等. 实验的整体分析流程如图 4 所示.

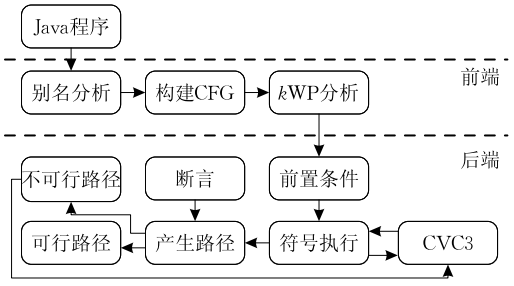


图 4　本文方法整体分析流程

本文的实验环境是Intel Core i5 2.5 GHz 的CPU，内存大小为 4 GB，操作系统采用 Ubuntu 12.04，内核版本为 3.2.0-24，Java 运行时的版本为 JRE 6.

① http://wala.sf.net

本文采用的测试基准程序集合来自 DaCapo[11] 和常见的开源 JAVA 程序，如表 2 所示. 其中 hedc 和 weblech 分别是网络爬虫和网站下载工具，lusearch 是文本索引工具，sunflow 是照片渲染工具，avrora 是微控制器仿真分析工具，hsqldb 是一款关系数据库引擎，antlr 是一款支持结构化文本和二进制文件的扫描器生成工具，batik 是一个基于 Java 的支持可缩放矢量图的浏览器. 这些程序的主要特点是规模较大，具有较多的分支语句和循环语句，可较为方便地设置待验证的断言并验证本文方法的有效性.

表 2 实验数据

测试程序	代码行数	圈复杂度	断言个数	因子 k 的取值		平均运行时间/s
				$k^{\max}$	$k^{\min}$	
hedc	1581	109	35	24	13	31.594
weblech	2793	167	72	31	11	85.190
lusearch	2934	121	41	42	19	67.891
sunflow	5210	371	85	39	24	138.561
avrora	4761	423	69	37	15	112.197
hsqldb	3593	287	47	21	21	81.693
antlr	8312	620	102	37	19	310.493
batik	3841	273	35	25	12	157.394

本文所提出的 kWP 方法由于是对最弱前置条件的一种近似，故 k 的值越大，其分析精度也越高，相应的时间开销也会越大，在验证程序路径的可行性方面，由于最弱前置条件是一种后向符号分析方法，故需要将 kWP 的结果反馈于符号执行工具，以引导其发掘可行性路径. 本文的主要贡献在于通过使用 kWP 方法更加高效地通过当前的失效路径来发掘尽可能多的失效路径，而不用像单纯的符号执行工具那样去分析整个程序状态空间. 在实验过程中，主要考察本文方法对符号执行工具在发掘可行路径方面的效率，主要考察指标为本文方法的运算时间以及因子 k 的取值对可行路径数量的影响.

测试基准程序的相关信息如表 2 所示，其中圈复杂度[12](Cyclomatic Complexity，CC)表示程序中判定节点的个数，它是一种代码复杂度的衡量标准，其值表示的是可行路径的条数，即为了验证程序的某个断言所需的最少路径条数. 圈复杂度越大，表明程序可能存在的错误数量越多，从而其验证的难度也相对较大. 圈复杂度的计算方法为 $CC(G)=e-n+2$，其中 e 表示控制流图中边的数量，n 表示节点的数量. 断言表示程序中待验证的路径所到达的位置需满足的条件，即待验证的兴趣点. 对于不同的断言，有一个对应的 k 值来表示对当前路径可行性的判定，表中 $k^{\max}$ 和 $k^{\min}$ 分别表示在待验证的断言集合中 k 的最大取值与最小取值，平均运行时间是在各自程序 k 的取值范围内对断言集合进行验证的运行时间的平均值，这样可以从整体上对比本文方法的时间开销.

在图 5 中我们定义了加速比(Speedup Ratio)的概念，它表示在不同 k 值的影响下，对于当前待分析的一条失效路径，本文方法在发掘与其近邻的失效路径集合所使用的时间($T_{k\mathrm{WP}}$)与直接采用符号执行的方法(T_{SE})所使用的时间的比值($T_{k\mathrm{WP}}/T_{\mathrm{SE}}$). 为了更加直观地体现本文方法的效果，对于每一个测试基准程序的不同兴趣点在同一个 k 值的约束下采用取平均值的方式来计算该处的加速比的值. 从图可看出加速比的值域为(0,1)，并没有出现加速比大于 1 的情况，表明本文的方法由于在路径搜索过程中加入了与失效路径相关的最弱前置条件，在发掘失效路径过程中消耗的时间较单纯的符号执行时间要少. 加速比的值越小，表明时间开销也越少. 从图 5 可以看出，加速比的值随 k 的取值的增大而增大，反映出发掘失效路径的时间开销会不断增加. 尤其在 k 值较小的时候，加速比上升的幅度较快，当 k 的值逐渐增加，其加速比的值趋向于 1，即表明本文方法的效率逐渐下降，趋向于单纯使用符号执行方法所得到的结果. 这是由于随着 k 值的增加，最弱前置条件在计算过程中所分析的路径谓词也越多，根据本文第 3 节 kWP 与传统最弱前置条件之间的关系，其分析精度也趋向于单纯使用符号执行工具的结果. 当 k 的取值较小的时候，加速比上升较快的现象反映出与失效路径的编辑距离较近的路径会较快地被符号执行工具检测出来，体现为加速比有较小的值，同时也验证了距离失效路径较近的路径其失效的概率也较高的结论. 对加速比有直接影响的是程序内部的控制流结构的复杂程度以及数据流的对象数量，从本组实验可以看出程序 lusearch 的加速比值最小为 0.73，即表明本文的方法相对于传统

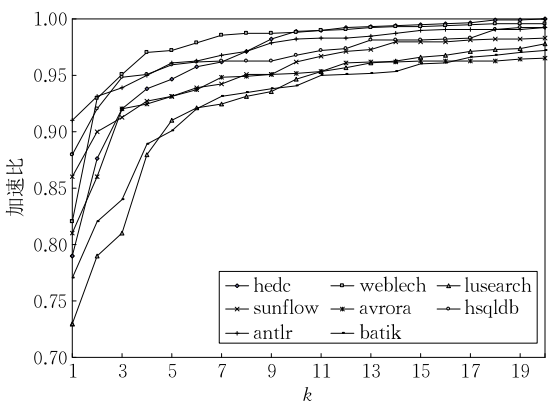


图 5 加速比随因子 k 的变化情况

的符号执行方法最多有 0.73 的加速比；而程序 weblech 的加速比值为 0.91，其原因是程序中有较多的全局变量和环境变量作为路径分支的判断条件，对程序的控制流结构的影响较为明显. 另外在 kWP 分析过程中，由于 SSA 对变量的重命名以及上下文敏感的分析过程导致产生了较多需要识别的新对象，从而对 kWP 的时间和空间复杂度带来了一定的影响，但是从图 5 中可以看出，由于 kWP 为符号执行工具提供了与待验证性质相关的路径引导信息，故能够对控制流结构以及数据流对象进行约减，使得加速比值能够小于 1（$T_{k\mathrm{WP}}/T_{\mathrm{SE}}<1$），即表明本文的方法相对于传统的符号执行分析方法有更高的执行效率.

图 6 表示在路径可行性验证过程中，在符号执行工具 Java PathFinder 运行前加入 kWP 分析的可行路径百分比对比图. 柱状图中的黑色部分表示只使用 Java PathFinder 的可行路径占所生成路径集合的百分比；灰色部分表示加入 kWP 分析后，可行路径所占的百分比相对于加入之前的增加幅度；斜纹部分表示加入 kWP 分析后，不可行的路径或者冗余的路径所占的比例. 从图 6 可以看出基于符号执行的路径搜索方法在加入 kWP 后，能较大幅度地提高路径可行性的验证效率，同时误报和漏报数量均相应减少. 在实际分析过程中，Java PathFinder 存在循环条件的计算较为复杂难以求解，导致容易在设定时间阈值内出现死循环的现象，故对于程序中的循环变量，本文中采用图 2 的方法将循环结构展开，并逐渐增加 k 的取值来调整循环体执行的次数，通过计算不可解的循环条件来生成对应的反例程序来引导生成可行的路径. 影响可行路径百分比的因素与影响加速比的因素类似，不同之处在于加速比突出的是在某个 k 值的作用下本文方法与传统

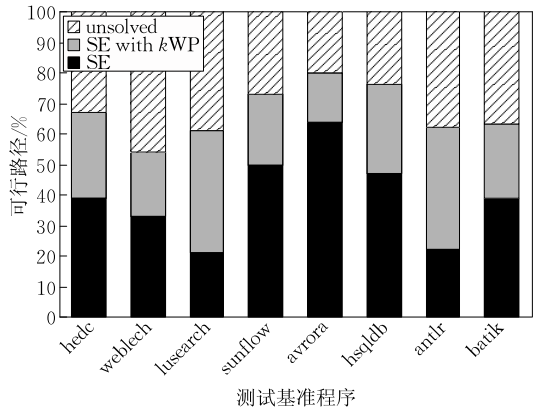


图 6　加入 kWP 前后可行路径对比

符号执行方法在时间开销方面的比较；而可行路径百分比强调的是能够发掘的可行路径的数量. 如图 6 所示，本文方法在发掘与兴趣点相关的路径集合方面相对于传统符号执行方法有大的性能提升，程序 lusearch 的提升幅度最高，幅度约为 41.2%；提升幅度最少的为程序 avrora，其提升幅度约为 17.1%. 测试基准程序集的平均提升幅度为 27.6%，表明本文的方法 kWP 在发掘程序可行路径方面有较高的精度提升.

图 7 表示因子 k 的取值与测试基准集的可行路径之间的关系：随着 k 取值的增加，所能生成的可行路径的数量也在不断增加. 在本文方法中 k 的取值对分析的精度与生成的可行路径数量有直接的影响，即因子 k 的取值可以平衡程序的分析规模以及分析的精度，故在实验过程中，我们主要从发掘不可行路径效率的角度出发，在设定的执行时间阈值内来分析 k 的取值. 另外本文方法在执行过程中，可能存在引入新的不可解路径条件的现象，故本文结合测试基准集的实际分析效果，设定 k 的取值一般不超过 20. 若 k 的取值继续增大，则可能导致分析时间过长而超时或者符号执行工具和求解器开销过大而内存资源耗尽. 从图 7 中可以看出对于有的测试基准程序，其可行路径数量在因子 k 的某个取值区间内无变化，其原因是循环结构的执行次数在不断增加，或者求解器在求解过程中出现了不可解的情况.

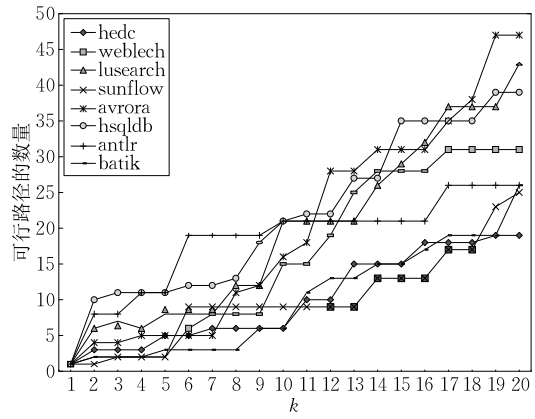


图 7　因子 k 的取值对可执行路径数量的影响

6　相关工作对比

可信计算是信息安全领域的重要研究方向，TCG（Trusted Computing Group）对可信的定义是：若一个实体是可信的，那么该实体在实现预定目标

过程中,其行为也是符合预期的.文献[13]认为:可信≈可靠+安全,故需要对软件的可信性进行验证.目前,程序验证过程中通常使用定理证明和模型检验的方法,ESC/JAVA[14]采用定理证明的方法验证函数是否满足给定的前置条件和后置条件,但该方法需要人工地在程序中增加注释来引导其分析过程.模型检验的方法可以对程序属性的正确性进行证明,SLAM[15]采用逐步精化的方法对程序待验证的性质进行近似,然后以目标制导的方式进行抽象精化;BLAST[16]采用惰性抽象和基于反例的抽象求精的方法对程序属性进行验证,但是这些方法常常遇到状态空间爆炸或者由于抽象所带来的分析精度较低等问题.在通过验证来发现软件缺陷方面,文献[17]从程序漏洞溢出的角度提出一种基于有限约束满足性的检测方法,可在无源代码的情况下直接分析可执行程序.Metal[18]是一个采用局部数据流分析的上下文不敏感错误检测工具,但它采用的分析方法是路径不敏感的,故其分析精度有限.PREFix[19]依据 CFG 图采用自底向上的方法来构建近似的过程摘要(procedure summary),但是它不能描述函数的输入域,故其采用保守的方式来预测程序的行为,相比之下本文的方法由于是一种基于需求驱动的路径敏感分析方法,其分析过程有较强的针对性.PSE[20]是一款使用过程间后向符号分析对程序的缺陷进行验证的工具,但是由于该工具不能获得程序所有的路径条件,故常需对程序的堆栈进行摘要操作,而该工具并没有提出相应的近似分析方法.ESC/Java 也是一种基于最弱前置条件的分析方法,但是它采用的只是过程内的分析方法,依赖用户的注释来推理程序中的调用关系.DSD-Crasher[21]通过使用 ESC/Java 生成测试用例作为程序反例,但是该工具通过符号分析所生成的最弱前置条件中的一个子集可能会被其它变量消除,从而导致整个最弱前置条件无法求解.文献[7]也采用最弱前置条件的方法引导符号执行操作来对程序缺陷进行修复和验证,但是没有对生成的最弱前置条件进行化简与合并.文献[22]采用基于路径的最弱前置条件来定位并纠正函数中的错误语句,其分析过程类似本文方法中的 $k=0$ 的情况,相比之下本文的方法可以调整 k 的取值来扩展分析范围,具有更好的灵活性.

7 结论及将来工作

路径可行性是验证程序安全性质中的主要内容,但是由于在验证过程中容易出现状态空间爆炸等问题而生成较多无用的路径,且大量的路径缺乏针对性.本文在文献[23]的工作基础上,提出一种改进的最弱前置条件与符号执行相结合的多路径程序验证方法,首先对程序的控制流图进行改进,使得路径的分析不再依赖于程序中的循环变量,从而每一条执行路径都有惟一的路径编码,这样就可以在其基础上使用最弱前置条件的方法生成引导信息供符号执行生成与验证性质相关的路径集合.本文方法通过分析与可行路径近邻的路径集合来生成程序近似的有效输入域,从而避免对所有路径的最弱前置条件进行计算,为高效地分析程序的路径可满足性提供了一种思路.实验结果表明本文的方法较传统的符号执行方法能够在较短时间内针对路径的可满足性问题生成更加精简而高效的路径集合,同时在分析效率上有较大提高.

本文的方法还存在需要改进的地方,主要体现在对浮点运算,按位操作等语句的语义还不能较为精确地进行分析,此外还不能处理并发程序,在以后的工作中可以采用诸如线程逃逸的方法进行研究;另外的一个研究方向是对本文的方法进行扩展,建立起程序的功能属性与非功能属性之间的关联关系,通过对功能属性的分析来对非功能属性进行间接地验证.

致 谢 感谢美国佐治亚理工学院计算机科学系的 Alessandro Orso 教授和审稿专家对本文提出的修改建议!

参 考 文 献

[1] Nipkow T, Paulson L. Isabelle/HOL: A proof assistant for higher-order logic. Lecture Notes in Computer Science 2283. Berlin: Springer, 2008

[2] Cruz J. Constraint Reasoning for Differential Models. Amsterdam: The IOS Press, 2005

[3] Clarke M, Grumberg O, Peled D. Model Checking. Massachusetts: The MIT Press, 1999

[4] Zhang Jian. Sharp static analysis of programs. Chinese Journal of Computers, 2008, 31(9): 1549-1553(in Chinese)
(张健. 精确的程序静态分析. 计算机学报, 2008, 31(9): 1549-1553)

[5] King J. Symbolic execution and program testing. Communications of the ACM, 1976, 19(7): 385-394

[6] Dijstra E. A Discipline of Programming. Englewood Cliffs: Prentice Hall, 1976

[7] Zhongxian G, Earl T, David J. Has the bug really been fixed//Proceedings of the 32nd Conference on International Conference on Software Engineering (ICSE 2010). Cape Town, South Africa, 2010: 55-64

[8] Lahiri S, Qadeer S. Back to the future: Revisiting precise program verification using SMT solvers//Proceedings of the 35th Conference on Principles of Programming Languages (POPL 2008). San Francisco, USA, 2008: 171-182

[9] Barrett C, Tinelli C. CVC3//Proceedings of the 19th International Conference on Computer Aided Verification (CAV 2007). Berlin, Germany, 2007: 298-302

[10] Willem V, Corina P, Sarfraz K. Test input generation with Java PathFinder//Proceedings of the 5th International Symposium on Software Testing and Analysis (ISSTA 2004). Boston, USA, 2004: 97-107

[11] Blackburn M, Garner R, Hoffman C, Khan M. The DaCapo benchmarks: Java benchmarking development and analysis//Proceedings of the 21st International Conferences on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006). Portland, USA, 2006: 169-190

[12] McCabe J. A complexity measure. IEEE Transactions on Software Engineering, 1976, 2(4): 308-320

[13] Shen Chang-Xiang, Zhang Huan-Guo, Wang Huai-Min. Research and development of trusted computing. Science in China, Series E, 2010, 40(2): 139-166(in Chinese)
(沈昌祥，张焕国，王怀民. 可信计算的研究与发展. 中国科学 E 辑，2010，40(2)：139-166)

[14] Flanagan C, Leino K, Lillibridge M, et al. Extended static checking for Java//Proceedings of the 23rd International Conference on Programming Language Design and Implementation (PLDI 2002). New York, USA, 2002: 234-245

[15] Ball T, Rajamani S. Automatically validating temporal safety properties of interfaces//Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN 2001). Toronto, Canada, 2001: 102-122

[16] Henzinger T, Jhala R, Majumdar R, Sutre G. Software verification with BLAST//Proceedings of the 10th International SPIN Workshop on Model Checking of Software (SPIN 2003). Portland, USA, 2003: 235-239

[17] Chen Kai, Feng Deng-Guo, Su Pu-Rui. Dynamic overflow vulnerability detection method based on finite CSP. Chinese Journal of Computers, 2012, 35(5): 898-909(in Chinese)
(陈恺，冯登国，苏璞睿. 基于有限约束满足问题的溢出漏洞动态检测方法. 计算机学报，2012，35(5)：898-909)

[18] Engler D, Chelf B, Chou A, Hallem S. Checking system rules using system-specific, programmer-written compiler extensions//Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI 2000). New York, USA, 2000: 2-11

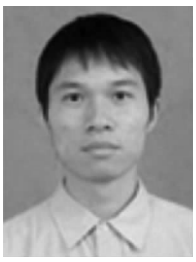
[19] Bush W, Pincus J, Sielaff D. A static analyzer for finding dynamic programming errors. Software: Practice and Experience, 2000, 30(7): 775-802

[20] Manevich R, Sridharan M, Adams S. PSE: Explaining program failures via postmortem static analysis//Proceedings of the 7th Fundamental Approaches to Software Engineering (FASE 2004). Barcelona, Spain, 2004: 63-72

[21] Csallner C, Smaragdakis Y, Xie T. DSD-Crasher: A hybrid analysis tool for bug finding. ACM Transactions on Software Engineering and Methodology, 2008, 17(2): 1-37

[22] He H, Gupta N. Automated debugging using path-based weakest preconditions//Proceedings of the 7th Fundamental Approaches to Software Engineering(FASE 2004). Barcelona, Spain, 2004: 267-280

[23] Guo Xi, Zhang Huan-Guo. Approach for reduced test suite generation based on predicate abstraction. Journal on Communications, 2012, 33(3): 35-43(in Chinese)
(郭曦，张焕国. 基于谓词抽象的测试用例约简生成方法. 通信学报，2012，33(3)：35-43)



GUO Xi, born in 1983, Ph. D., lecturer. His research interests include software analysis and testing, information security.

WANG Pan, born in 1987, M. S., lecturer. Her research interest is power electronics transformation.

WANG Jian-Yong, born in 1974, M. S., associate professor. His research interests include computer network and bioinformatics.

ZHANG Huan-Guo, born in 1945, professor, Ph. D. supervisor. His research interests include information security, trusted computing.

Background

The purpose of program analysis is verifying whether the source codes satisfy the given security criteria, and then verify the properties. Current analysis tends to false positive and false negative due to the manual analysis, so more and more research groups around the world have turned to the automatic verification and analysis. The path feasibility problem is the key problem in the domain of program verification, and weakest precondition is a backward symbolic

analysis, which can model the program execution semantics, but always confronts the phenomenon of state space explosion in the course of CFG construction, meanwhile, weakest precondition relies on the loop invariant during the process of computation.

Previous works in this domain mainly lack the combination of checking points and path conditions, and just merely use the constraint solver to analysis this problem, which is prone to generate redundant or infeasible paths, and can not verify the properties. There are a lot of methods and tools can analysis the verification problem, such as ESC-JAVA, SLAM, BLAST and so on, but these methods are either context insensitive or suffer from the state explosion, and thus remains many issues for others to explore.

The authors propose a new method to generate feasible paths to verify the properties of the target programs. During this method, the form of CFG is improved, so that it does not solve the path conditions during the computation, rather add the information generated by the backward symbolic analysis, which can generate proper paths and then simplify and merge them, and reduce false positives and false negatives. Thus, the main contributions of this paper are expanding the form of the CFG, which can express the loop and condition statements and explore more efficient paths. Another contribution is proposing the k proximity weakest precondition which can help to generate paths based on the given execution path, and according to adjust the k invariant, more feasible paths can be obtained, and assist in verifying the properties. Experimental results demonstrate that this method can get more feasible paths in contrast to the traditional symbolic execution, and reduce the false positives and false negatives.

This project is sponsored by the National Natural Science Foundation of China under Grant Nos. 61332019, 61173138, 61272452, 91118003, and the National Basic Research Program (973 Program) of China under Grant No. 2014CB340600, and the National High Technology Research and Development Program of China under Grant No. 2015AA016002 and Natural Science Foundation of Hubei Province under Grant No. 2014CFB144, and Fundamental Research Funds for the Central Universities under Grant No. 2662015QC009.