软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(2):386−399 [doi: 10.13328/j.cnki.jos.004538]

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

基于缺陷关联的静态分析优化*

张大林, 金大海, 宫云战, 王 前, 董玉坤, 张海龙

(网络与交换技术国家重点实验室(北京邮电大学),北京 100876)

通讯作者: 张大林, E-mail: dalin@bupt.edu.cn, http://www.dtstesting.com

摘 要: 缺陷检测一般包括静态分析与人工审查两个阶段.静态检测工具报告大量缺陷,但是主要的缺陷确认工作仍由人工完成,这是一件费时、费力的工作.巨大的审查开销可能会导致软件开发人员拒绝使用该静态缺陷检测工具.提出一种可靠的基于缺陷关联的静态分析优化方法,能够分组静态检测工具所报告的缺陷,在分组后的任意一组缺陷中,如果其主导缺陷被证明是误报(或者是真实的),就能确认其他缺陷也是误报(也是真实的).实验结果表明,基于缺陷关联的静态分析优化方法在较小的时间和空间开销下减少了 22%的缺陷审查工作,能够较好地适应于大型的关键嵌入式系统程序缺陷检测.

关键词: 静态分析;优化;缺陷关联;抽象解释;状态切片

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 张大林,金大海,宫云战,王前,董玉坤,张海龙.基于缺陷关联的静态分析优化.软件学报,2014,25(2):386−399. http://www.jos.org.cn/1000-9825/4538.htm

英文引用格式: Zhang DL, Jin DH, Gong YZ, Wang Q, Dong YK, Zhang HL. Optimizing static analysis based on defect correlations. Ruan Jian Xue Bao/Journal of Software, 2014,25(2):386−399 (in Chinese). http://www.jos.org.cn/1000-9825/4538.htm

Optimizing Static Analysis Based on Defect Correlations

ZHANG Da-Lin, JIN Da-Hai, GONG Yun-Zhan, WANG Qian, DONG Yu-Kun, ZHANG Hai-Long

(State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications), Beijing 100876, China)

Corresponding author: ZHANG Da-Lin, E-mail: dalin@bupt.edu.cn, http://www.dtstesting.com

Abstract: Defect detection generally includes two stages: static analysis and defect inspection. A large number of defects reported may lead developers and managers to reject the use of static analysis tools as part of the development process due to the overhead of defect inspection. To help with the inspection tasks, this paper formally introduces defect correlation, a sound dependency relationship between defects. If the occurrence of one defect causes another defect to occur, the two defects are correlated. This paper presents a sound optimized method to static analysis that can classify the defects reported by static defect detection tool into different groups, in which all defects are false positives (true positives) if the dominant defect is false positives (true positives). The experimental results show a decrease of 22% the time inspecting all defects and the capability and flexibility of this method to detect defects of large, critical or embedded systems.

Key words: static analysis; optimizing; defect correlation; abstract interpretation; state slicing

随着计算机硬件设备性能的迅速提高以及嵌入式系统应用领域的不断拓宽,嵌入式系统软件的规模和复杂性急剧增加,软件已经成为嵌入式系统的主要使能部件.软件测试是提高软件质量的重要手段,今天的软件工业越来越依赖自动的缺陷检测工具来检测软件缺陷[1,2].静态缺陷检测工具报告大量缺陷,但是主要的缺陷确认工作仍然由人工完成,这是一件费时、费力的工作.根据相关人员统计,人工审查这些缺陷的速度非常低,高耗时

* 基金项目: 国家自然科学基金(91318301, 61202080); 国家高技术研究发展计划(863)(2012AA011201)
收稿时间: 2013-05-08; 修改时间: 2013-09-29; 定稿时间: 20123-12-05

的缺陷审查工作可能导致软件开发和管理人员拒绝在软件开发流程中使用该静态缺陷检测工具.

静态分析求精技术试图通过抽象和近似更理想的程序语义进而减少缺陷检测的误报和漏报.根据 Rice 定理[3],静态分析针对程序的任何非平凡属性(例如是否存在运行时错误),不可能做到既是可靠的(sound)又是完备的(complete),导致其计算结果可能出现误报(false positive)和漏报(false negative).通过提升静态缺陷检测工具的分析能力来减少误报是一个难解的问题,因此现实中,静态缺陷检测工具往往会在分析精度和执行时间两者中做一个平衡.

静态分析优化技术试图利用静态分析过程中的信息进一步对静态分析结果进行优化,进而减轻缺陷确认负担.静态分析优化技术不是对现有可靠的和(或)完备的静态分析算法求精,相反,优化技术试图利用静态分析过程中的相关信息在工具运行过程中引导缺陷检测,或者在静态缺陷检测工具运行后进行缺陷报告优化.例如,将缺陷按某种依赖关系分组,同一组内如果其主导缺陷是误报(真实缺陷),那么组内其余缺陷也是误报(真实缺陷),进而实现缺陷快速审查.静态分析优化技术从另一个角度提升了静态分析工具的检测效率和精度.

缺陷关联是一种有效降低缺陷确认负担的静态分析优化技术[4].缺陷关联依据不同缺陷间的依赖信息将缺陷分组.我们可以这样认为:如果缺陷 *a* 可靠地依赖于缺陷 *b*,那么,如果缺陷 *b* 被确认为一个误报(真实缺陷),那么缺陷 *a* 逻辑上也是这个结论.当发现一个缺陷集合依赖于其中某一缺陷(我们称其为主导缺陷)时,我们就将这些缺陷分为一类.一旦发现这样一组缺陷,我们仅仅需要确认他们的主导缺陷是否为误报(或真实缺陷)即可.

图 1 程序片段展示了缺陷检测系统 DTS[5−7]在检测嵌入式开源工程 git-1.8.2 时所报告的 4 个空指针解引用缺陷,分别是:第 198 行函数 *binary_search*(·)的第 1 个参数 *sequence* 可能为空指针并在 198 行解引用;第 199 行 *sequence*[*i*]可能为空指针并在 199 行解引用;第 200 行 *sequence*[++*i*]可能为空指针并在此行解引用;第 212 行 *sequence*[*longest*−1]可能为空指针并在此行解引用.经过分析发现,这些缺陷全部都是因为 *xdl_malloc*(·)函数存在内存分配失败的可能,进而造成结构体类型变量 *sequence* 可能为空,其成员变量同样可能为空.基于这个实例可以得到以下结论:(1) 如果结构体变量 *sequence* 在 198 行为空,那么随后所有的引用都是空指针解引用;(2) 如果结构体变量 *sequence* 在 198 行非空,那么随后所有其他缺陷都是误报.基于对上述实例的分析,缺陷关联可以有效地减少缺陷确认负担,进而提升了软件测试整体效率.上述 4 个缺陷的关联关系是由本文的关联算法自动计算得到的.

```
git-1.8.2\xpatience.c
#define xdl_malloc(x) malloc(x)
189: static struct entry *find_longest_common_sequence(struct hashmap *map)
190:{
191:   struct entry **sequence=xdl_malloc(map→nr * sizeof(struct entry *));
            ...
198:       i=binary_search(sequence,longest,entry);
199:            entry→previous=i<0? NULL: sequence[i];
200:            sequence[++i]=entry;
           ...
212:       entry=sequence[longest−1];
           ...
220:}
```

Fig.1 An example of defect reports

图 1 缺陷报告实例

本文提出了一种可靠的缺陷关联方法.该方法能够在静态缺陷检测过程中自动发现可靠的缺陷关联关系.结合这些关联关系,静态缺陷检测工具将缺陷分组.如果主导缺陷被确认是误报(或者真实缺陷),则能够确认同组内的其他缺陷也为误报(或者真实缺陷).

本文的贡献主要有:(1) 基于抽象解释理论,我们给出了缺陷关联的相关定义.(2) 我们引入一个可靠的抽象缺陷关联方法.我们的缺陷关联算法可被认为是静态缺陷检测工具的副产品.我们的框架是通用的,可以被应用到任何一个基于抽象解释的静态缺陷检测工具上.它是缺陷求精研究和缺陷分级研究的正交.(3) 我们证明

了我们的缺陷关联方法的有效性.将该方法应用于缺陷检测系统 DTS,通过对 10 个 GCC 开源工程检测发现,该方法减少了 22%的缺陷确认数量.

本文第 1 节给出缺陷与缺陷关联的定义.第 2 节描述缺陷关联算法.第 3 节介绍静态缺陷检测工具 DTS 的框架以及实验环境,并对 10 个 GCC 程序的检测结果进行分析.第 4 节给出相关工作比较.第 5 节总结全文并展望未来的工作.

1 缺陷及缺陷关联

本节首先利用程序的踪迹语义给出程序缺陷以及缺陷关联的定义.我们引入了符号化的表达式抽象域,并且给出了抽象的缺陷关联表示,为下一步缺陷关联计算奠定基础.抽象解释[8,9]、区间集抽象域[10]以及基于缺陷模式的软件测试技术[11]为本研究的理论基础.

1.1 程序及其缺陷

程序 P 可被描述为一个迁移系统五元组$(\mathbb{L},\mathbb{X},\mathbb{V},\rightarrow,\mathbb{S}^i)$,其中,$\mathbb{L}$为程序点可粗略代表程序某一时刻的执行状态;$\mathbb{X}$代表变量即程序的内存定位;$\mathbb{V}$代表变量值的集合;符号$\rightarrow$代表迁移关系,$(\rightarrow)\subseteq\mathbb{S}\times\mathbb{S}$描述了程序执行时如何从一个状态进入另一个状态,$\mathbb{S}$为程序的状态集合,$\mathbb{S}=\mathbb{L}\times\mathbb{M}$,$\mathbb{M}=\mathbb{X}\rightarrow\mathbb{V}$描述某一时刻程序变量取值的存储状态;$\mathbb{S}^i$为程序的初始状态集合,$\mathbb{S}^i=\{l^i\}\times\mathbb{M}$,其中,$l^i\in\mathbb{L}$可理解为一个程序的入口程序点.

现实世界中的程序由于存储和运算错误,可能会导致系统故障.因此,$\mathbb{S}$中还需要包含一些指定的错误状态.关于错误状态,我们会在后续中给出定义.同时考虑到过程间程序,一个函数调用点可以用一个有序对(k,l)表示,其中,k是调用栈(函数名),l是程序点.因此,一个过程间程序状态就是一个$\mathbb{K}\times\mathbb{L}\times\mathbb{M}$上的三元组.

程序的一次执行可以用一个状态序列来表示,我们称这个序列为程序的踪迹.在抽象解释的程序语义体系中,踪迹语义最为精确,抽象解释中所涉及的程序语义都可以被描述为相应语义函数的最小不动点,较高抽象层次的语义是其下层语义的可靠近似,低层次语义函数的最小不动点可由高层次语义函数的最小不动点来可靠地近似.上述语义的可靠性由伽罗瓦连接保证.下面我们给出程序的踪迹语义描述,并根据程序的踪迹语义给出缺陷及缺陷关联的定义.

定义 1(踪迹语义). 程序可以看作是一个踪迹集合,$[\![P]\!]=\{\langle s_0,\ldots,s_n\rangle\in\mathbb{S}^*|s_0,\ldots,s_n\in\mathbb{S}\wedge\forall i,s_i\rightarrow s_{i+1}\}$.我们可以将程序 P 的语义$[\![P]\!]$描述为最小不动点形式:$[\![P]\!]=lfp^{\subseteq}_{\mathbb{S}^i}F_{\bar{P}}$, 其中, $F_{\bar{P}}$ 为语义函数,被定义为

- $F_{\bar{P}}:\mathbb{S}^*\rightarrow\mathbb{S}^*$;
- $\varepsilon\mapsto\varepsilon\cup\{\langle s_0,\ldots,s_n,s_{n+1}\rangle|\langle s_0,\ldots,s_n\rangle\in\varepsilon\wedge s_i\rightarrow s_{i+1}\}$.

定义 2(抽象踪迹). 假设存在一个抽象域 $(D^{\#}_{\mathbb{M}},\sqsubseteq)$ 代表程序变量抽象取值,相应的具体函数是:$\gamma_{\mathbb{M}}:D^{\#}_{\mathbb{M}}\rightarrow\mathcal{P}(\mathbb{M})$,该抽象域可以是区间抽象域(非关系抽象域)或关系抽象域.进而,可以引入如下的抽象语义 T 近似描述程序的踪迹语义$[\![P]\!]$:

- 抽象域:$D^{\#}=\mathbb{L}\rightarrow D^{\#}_{\mathbb{M}}$;
- 具体函数:$\gamma:I\in D^{\#}\mapsto\{\langle(l_0,\rho_0),\ldots,(l_n,\rho_n)\rangle|\forall i,\rho_i\in\gamma_{\mathbb{M}}(I(l_i))\}$.

上述抽象过程将程序踪迹语义抽象为集合语义.相应地,用 $lfp^{\#}$表示一个抽象的后置不动点(post fixpoint).给定任意 $x\in\mathcal{P}(\mathbb{M})$, $d\in D^{\#}_{\mathbb{M}}$, 并且有具体域上迁移函数 $F:\mathcal{P}(\mathbb{M})\rightarrow\mathcal{P}(\mathbb{M})$及抽象域上迁移函数 $F^{\#}:D^{\#}_{\mathbb{M}}\rightarrow D^{\#}_{\mathbb{M}}$, 如果 $x\subseteq\gamma_{\mathbb{M}}(d)$ 并且 $F\circ\gamma_{\mathbb{M}}\subseteq\gamma_{\mathbb{M}}\circ F^{\#}$, 那么 $lfp_xF\sqsubseteq lfp_dF^{\#}$.因此,由静态分析工具所计算的程序抽象语义 T 是可靠的.

程序的时序安全属性是描述“某些坏事情不会发生”的一类属性,通常,时序安全属性规定了一系列有序的事件,要求在程序中不能发生这些事件,若程序违反其时序安全属性,则进入错误状态.静态分析则是近似找出程序的可达错误状态集合,并报告一个相应的缺陷.

定义 3(缺陷). 在程序 P 的某一程序点 l 上,如果 $\gamma_{\mathbb{M}}(T(l)\cap\Omega(l))\neq\varnothing$, 则静态分析工具在程序点 l 上报告一个缺陷 φ_l,其中,T 为程序的抽象语义;$\Omega(l)$为错误状态函数,代表在程序点 l 上的所有错误程序状态集合.如果静态分析工具报告一个缺陷 φ_l 并且$[\![P]\!](l)\cap\Omega(l)=\varnothing$,那么 φ_l 为误报;反之,如果$[\![P]\!](l)\cap\Omega(l)\neq\varnothing$,则 φ_l 为一个真实缺陷.

定义 4(缺陷特征). 给定一个缺陷 φ_l,则表明在程序点 l,程序某一属性违反了该程序的语法或语义规则,该属性所对应的变量及相应取值即为这个缺陷的缺陷特征(defect feature),记为 $Df(\varphi_l)$,$Df(\varphi_l)\subseteq\rho_l$.

程序的一类时序安全属性违反可用缺陷模式来描述.静态缺陷检测方法关键是对缺陷模式进行定义和检测,能够处理的缺陷模式种类越多,分析检测能力则越强.缺陷模式是指程序中经常发生的缺陷所呈现出的语法或语义特征.DTS 中的缺陷模式是对程序属性的一种描述,若违反该属性,则造成一个缺陷.对于时序安全类型的缺陷模式,例如资源泄露缺陷(resource leak,简称 RL)、空指针解引用缺陷(null pointer dereference,简称 NPD)等,有限状态自动机是一种常用且易于理解的抽象表示,因此,缺陷模式也可以用缺陷模式状态机来表示.

定义 5(缺陷模式状态机). 用于描述缺陷模式的有限状态机(finite state machine,简称 FSM),包括状态集合 S、状态迁移集合 *Transfers* 及迁移条件集合 *Conditions*,其中,$S=\{\$start,\$error\}\cup S_{other}$,$Transfers: S\times Conditions\rightarrow S$. $\$start$ 和$\$error$ 分别表示起始状态和错误状态,S_{other} 表示其他中间状态的集合.

缺陷模式检测根据数据流分析的结果,从每个状态机实例(FSM instance)的$\$start$ 状态进行状态迭代.随着状态实例(FSM state instance)中数据流信息的更新,会触发设定的状态迁移条件(FSM condition),从而,缺陷状态会由$\$start$ 状态迁移到某些中间状态,进一步地会迁移到$\$end$ 或$\$error$ 状态.

DTS 采用 XML 文件对不同种类的缺陷模式状态机进行形式化描述.在缺陷检测时,根据待测程序的语法、语义特征决定是否创建某类型的状态机实例,并将产生的状态机实例集合置于待测函数控制流入口处;然后在遍历控制流图过程中,根据当前控制流节点的数据流信息更新状态机实例的缺陷状态,同时判定是否会发生状态迁移,以检测某种类型的缺陷.缺陷模式状态机可以用有向图直观地表示,例如,RL 和 NPD 缺陷模式分别可用状态机描述,如图 2 所示.

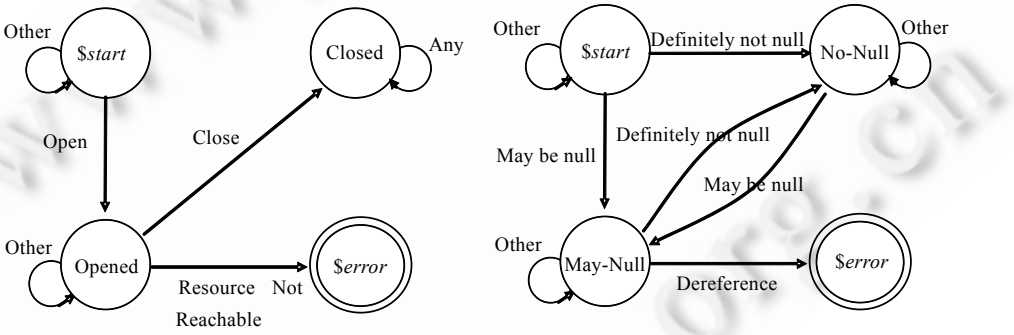


Fig.2 FSM of RL and NPD

图 2 资源泄露和空指针解引用缺陷模式有限状态机

在现实的基于抽象解释技术的静态缺陷检测过程中,静态分析在报告真实缺陷的同时也引入了大量误报,即,所报告的缺陷.其错误状态实际是不可达的.在大型的高可信的嵌入式软件测试过程中,一定规模的真实缺陷和大量的误报给缺陷确认工作带来挑战,因此,我们的目标是依据缺陷的错误状态对缺陷分组.分组后,同一组内只要主导缺陷的错误状态是可达的(真实缺陷),组内其余缺陷的错误状态就都是可达的;反之,同一组内只要主导缺陷其错误状态是不可达的(误报),组内其余缺陷的错误状态都是不可达.

1.2 缺陷关联及其抽象

本节正式引入缺陷关联来优化抽象解释技术.本节首先给出具体语义下的缺陷关联定义,然后正式地引入程序语义切片、错误状态切片技术,同时给出基于抽象解释框架的可靠性说明.基于上述定义,本节最后给出了抽象缺陷关联的定义并对其可靠性给出了形式化证明,为下一步在静态分析工具中实现奠定了基础.

定义 6(缺陷关联). 在程序 P 中,给定两个已知缺陷 φ_m 和 φ_n,$m,n\in\mathbb{L}$,当且仅当假设缺陷 φ_m 不发生总能得出缺陷 φ_n 也不发生,即称 φ_m 和 φ_n 具有关联关系,记为 $\varphi_m\mapsto\varphi_n$.

依据上述定义,缺陷关联的具体应用分两种情况:

(1) 给定两个已知缺陷 φ_m 和 φ_n,如果缺陷 φ_m 为误报,则总能得出缺陷 φ_n 也为误报,即称 φ_m 关联 φ_n,记作:$\varphi_m\mapsto\varphi_n$ (即,只判定 φ_m 即可).即,如果 $[\![P]\!](m)\cap\Omega(m)=\varnothing\Rightarrow[\![P]\!](n)\cap\Omega(n)=\varnothing$,那么 $\varphi_m\mapsto\varphi_n$;

(2) 给定两个已知缺陷 φ_m 和 φ_n,如果缺陷 φ_m 为真实缺陷,则总能得出 φ_n 缺陷也为真实缺陷,即称 φ_n 关联 φ_m,记作:$\varphi_n\mapsto\varphi_m$(即,只判定 φ_m 即可).即,如果 $[\![P]\!](m)\cap\Omega(m)\neq\varnothing\Rightarrow[\![P]\!](n)\cap\Omega(n)\neq\varnothing$,那么 $\varphi_n\mapsto\varphi_m$.

在实际的缺陷检测及缺陷关联计算中,我们仅关心那些实际的执行踪迹,这些程序踪迹都起始于一个程序的初始状态集合,并且导致一个缺陷产生.

定义 7(缺陷踪迹). 给定一个程序 P 的缺陷 φ_l,其初始状态集合为 $\mathcal{I}\subseteq\mathbb{S}$,那么该缺陷的实际执行踪迹为 $\overline{T}_{\varphi_l}=\{\langle s_0,...,s_n\rangle\in[\![P]\!]\mid s_0\in\mathcal{I}\}$,$\overline{T}$ 可描述为最小不动点形式:$\overline{T}_{\varphi_l}=lfp_{\varphi}\overline{F}$, 其 $\overline{F}$ 为

$$E\mapsto\{\langle s\rangle|s\in\mathcal{I}\}\cup\{\langle s_0,\ldots,s_n,s_{n+1}\rangle|\langle s_0,\ldots,s_n\rangle\in E\wedge s_n\to s_{n+1}\}.$$

为了抽象近似缺陷踪迹,令 $\mathcal{I}^{\#}\in D^{\#}$ 为缺陷踪迹的一个可靠近似,即 $\{\langle s\rangle|s\in\mathcal{I}\}\subseteq\gamma(\mathcal{I}^{\#})$;按相同方式,令 $\mathcal{F}^{\#}\in D^{\#}$,定义一个函数 $\sigma_{l,l'}:D^{\#}\to D^{\#}$ 计算程序的任意抽象迁移,$\sigma_{l,l'}$ 是一个可靠的基本抽象操作,即,$\sigma_{l,l'}$ 满足:

$$\forall\rho,\rho'\in\mathbb{M},\ \forall d\in D^{\#},\ \rho\in\gamma(d)\wedge(l,\rho)\to(l',\rho')\Rightarrow\rho'\in\gamma(\sigma_{l,l'}(d)).$$

下面继续给出缺陷踪迹的抽象语义描述.

定义 8(抽象缺陷踪迹). 基于定义 2,缺陷踪迹的抽象语义 $\overline{T}'$ 的抽象函数可描述为:

- $\overline{F}^{\#}:D^{\#}\to D^{\#}$;
- $I\mapsto\lambda(l\in\mathbb{L}).\sqcup\{\sigma_{l,l'}(I(l))|l'\in\mathbb{L}\}$;
- $\overline{T}'_{\varphi_l}=lfp^{\#}_{\mathcal{I}^{\#}}\overline{F}^{\#}$.

由上述定义可知,$\overline{T}_{\varphi_l}\subseteq\gamma(lfp^{\#}_{\mathcal{I}^{\#}}\overline{F}^{\#})$, 因此,缺陷踪迹的抽象语义是可靠的. $\overline{T}_{\varphi_l}$ 和 $\overline{T}'_{\varphi_l}$ 分别可以视为程序的一个具体语义切片和抽象语义切片.

由定义 4 可知,一个缺陷所对应的缺陷特征描述了导致程序进入缺陷状态的相关变量取值.在某一抽象域下,如果人为地将该程序点所包含的错误状态切除,那么静态分析工具将不会报告该缺陷.本节后续部分给出了状态切片、精化的缺陷踪迹语义这两个概念的定义,同时,分别给出了上述定义的可靠性说明.最后,基于上述概念给出了抽象缺陷关联的定义,并对其可靠性给出了形式化证明.

定义 9(状态切片). 在程序 P 的某一程序点 l 上,静态分析工具报告一个缺陷 φ_l,那么,在程序点 l 上切除缺陷 φ_l 所对应的错误状态后的状态切片可定义为 $\overline{\rho}_{\varphi_l}=\overline{T}(l)\ominus\Omega(\varphi_l)$, 其中,$\Omega(\varphi_l)$表示 φ_l 所对应的具体错误状态,$\ominus$为具体的错误状态切除操作.相应的状态切片的抽象语义可以表示为 $\overline{d}_{\varphi_l}=\overline{T}'(l)\,\hat{\ominus}\,\alpha(\Omega(\varphi_l))$, 其中, $\hat{\ominus}$ 为可靠的抽象的错误状态切除操作.具体实践中,静态分析工具需要针对某一抽象域实现这个抽象切除操作.

现实的应用程序中存在大量的外部输入,这些外部输入往往可能导致缺陷发生.静态分析工具往往通过某种程度的近似来处理这些输入的约束,进而达到对程序的一个可靠分析.在本研究中,我们借鉴程序外部输入约束的处理思想,将某一程序点上的状态切片作为一个输入约束,进而得到一个基于外部输入约束的程序语义.下面首先给出程序输入的一般性描述定义.

定义 10(输入). 在程序 P 中,给定一个程序的输入程序点集合 $\mathbb{L}_{in}=\{l\in\mathbb{L}|l:input(x_l)\}$,那么程序的输入可以用一个输入函数 δ 来表示:$\delta:\mathbb{L}\to\mathcal{P}(\mathbb{V})$,函数 δ 完成了程序点到该程序点相应变量输入值的映射.输入函数 δ 所代表的踪迹集合可表示为 $\gamma_{\mathbb{V}}(\delta)=\{\langle(l_0,\rho_0),...,(l_n,\rho_n)\rangle\mid\forall i,l_i\in\mathbb{L}_{in}\Rightarrow\rho_i(x)\in\delta(l_i)\}$, 这一踪迹集合满足任一外部输入对程序变量取值的修改.

本研究中不考虑真实程序的外部输入情况.我们将缺陷状态切片视作一种外部输入约束进而覆盖原有程序错误状态,那么 $\mathbb{L}_{in}=\{l\in\mathbb{L}_{\mathcal{F}}\mid l:input(x_l)\}$, 其中, $\mathbb{L}_{\mathcal{F}}$ 为所有缺陷发生程序点,相应的输入函数 $\delta_{\mathcal{F}}$ 所代表的踪

迹集合可进一步表示为 $\gamma_{\mathbb{V}}(\delta_{\mathcal{F}})=\{\langle(l_0,\rho_0),...,(l_n,\rho_n)\rangle \mid \forall i,l_i\in\mathbb{L}_{\mathcal{F}}\Rightarrow\overline{\rho}_{l_i}(x)\in\delta_{\mathcal{F}}(l_i)\}$.

定义 11(求精语义). 考虑将缺陷 φ_l 的状态切片 $\overline{\rho}_{\varphi_l}$ 所对应的变量取值作为外部输入约束,缺陷踪迹 $\overline{T}_{\varphi_l}$ 将被进一步求精,缺陷踪迹的求精语义 $\hat{T}_{\varphi_l}$ 可表示为 $\hat{T}_{\varphi_l}=\overline{T}_{\varphi_l}\cap\gamma_{\mathbb{V}}(\delta_{\mathcal{F}}(l))$.

假设 $\delta^{\#}_{\mathcal{F}}(l)$ 是对 $\delta_{\mathcal{F}}(l)$ 的可靠抽象,即满足 $\{\overline{\rho}_{\varphi_l}\mid\overline{\rho}_{\varphi_l}(x)\in\delta_{\mathcal{F}}(l)\}\subseteq\gamma(\delta^{\#}_{\mathcal{F}}(l))$. 下面我们给出缺陷踪迹求精语义的抽象语义.

定义 12(抽象求精语义). 求精语义的抽象语义 $\hat{T}'_{\varphi_l}$ 可表示为 $\hat{T}'_{\varphi_l}=lfp^{\#}_{\mathcal{I}^{\#}}\overline{F}^{\#}\sqcap\delta^{\#}_{\mathcal{F}}$, $\hat{T}'_{\varphi_l}$ 是对 $\hat{T}_{\varphi_l}$ 的可靠抽象,即

$$\hat{T}_{\varphi_l}\subseteq\gamma(lfp^{\#}_{\mathcal{I}^{\#}}\overline{F}^{\#}\sqcap\delta^{\#}_{\mathcal{F}}).$$

在静态分析过程中,因为具体的缺陷关联通常是不可计算的,故使用可靠的抽象缺陷关联来近似地表示具体缺陷关联关系.

定义 13(抽象缺陷关联). 在程序 P 中,静态分析工具输出两个缺陷:φ_m 和 φ_n,$m,n\in\mathbb{L}$,φ_m 抽象关联 φ_n,当且仅当 φ_m 在抽象求精语义 $\hat{T}'_{\varphi_m}$ 下不被该静态分析工具报出,即,当且仅当 $\gamma(\hat{T}'_{\varphi_m}(\varphi_n))\cap\Omega(\varphi_n)=\varnothing$. 当 φ_m 抽象关联 φ_n 时,记为 $\varphi_m\rightsquigarrow\varphi_n$.

下面对定义 13 所述抽象缺陷关联的可靠性进行证明.

证明:假设 φ_m 和 φ_n 为静态分析工具在某一个抽象域之下所报告的两个缺陷,且 $\varphi_m\rightsquigarrow\varphi_n$,那么由定义 13 可知,$\gamma(\hat{T}'_{\varphi_m}(\varphi_n))\cap\Omega(\varphi_n)=\varnothing$. 由 Galois 连接的定义[8]可知,$\hat{T}_{\varphi_m}(\varphi_n)\sqsubseteq\gamma(\hat{T}'_{\varphi_m}(\varphi_n))$, 所以有 $\hat{T}_{\varphi_m}(\varphi_n)\cap\Omega(\varphi_n)=\varnothing$. 由定义 6 和定义 11 可知, $\hat{T}_{\varphi_m}(\varphi_n)\cap\Omega(\varphi_n)=\varnothing\Leftrightarrow\varphi_m\mapsto\varphi_n$, 由此可证,$\varphi_m\rightsquigarrow\varphi_n$ 是对 $\varphi_m\mapsto\varphi_n$ 的可靠抽象. □

2 缺陷关联计算

如第 1.2 节所示,我们需要计算抽象的缺陷关联来揭示静态分析所报告的缺陷间的具体缺陷关联关系.首先,在第 2.1 节给出我们的一个基于数据流分析的缺陷检测与缺陷关联计算的算法概要,然后,第 2.2 节具体描述该缺陷关联算法,最后在第 2.3 节对算法的复杂性进行分析.

2.1 算法概述

缺陷关联计算包含两个相对独立又互相影响的过程:缺陷检测和缺陷关联.在缺陷检测过程,我们记录缺陷及缺陷发生的缺陷点.在缺陷关联过程,我们获得在缺陷检测过程中所报告的缺陷.我们切掉缺陷的错误状态,生成一个新的状态切片作为外部约束,进而得到缺陷的抽象求精语义,并进一步判断该抽象求精语义下是否对其他缺陷的发生产生影响.第 2 个过程的目标是,判断切除一个缺陷的错误状态是否影响第 1 个过程中另一个缺陷的报告.在整个缺陷检测和缺陷关联计算过程中,我们扩展了原有 DTS 的函数摘要模型[8,12],算法通过关联信息函数摘要实现过程间分析.

如图 3 所示,在程序控制流图的每一个节点上,算法通过更新相应路径上的程序状态信息进而实现在每个程序点上的程序抽象状态更新,同时,程序的被求精的状态信息也会得到同步更新计算,进而确保程序的求精语义的可靠性.程序某一路径上的原始状态和求精状态被更新后,首先进入缺陷检测阶段.如果静态分析在原始程序的抽象语义之下报告一个缺陷,则进入缺陷关联计算阶段,进一步判断该缺陷是否在求精语义下被报告.然后,对缺陷的错误状态进行切片操作,并将新生成的程序抽象求精状态加入到抽象求精状态集合中,供后续程序点上缺陷关联计算使用.需要说明一点,算法并不是对所有缺陷的错误状态都进行切片操作.对于相同变量或表达式引起的多个缺陷,如果变量或表达式的取值状态相同,则只对首个缺陷进行缺陷状态切片操作,进而避免大量重复关联判断.如果静态分析工具在求精抽象状态下没有报告该缺陷,则说明该求精抽象状态所对应的缺陷与该缺陷存在关联关系.详细的处理细节将在下一节具体算法中给出.

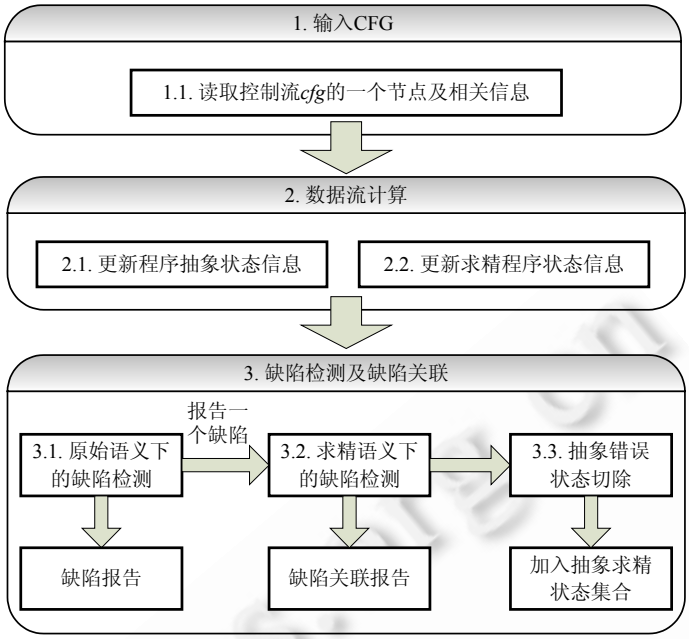


Fig.3 Flowchart of defect detection and defect correlation

图 3 缺陷检测及缺陷关联框架图

2.2 缺陷关联算法

根据第 2.1 节缺陷关联计算的思想框架,本节我们给出一个具体算法来静态地计算缺陷关联.程序的时序安全属性可以用有限状态机来描述,对属性状态机进行扩展,将路径信息作为状态约束条件,则实现了路径敏感的静态分析.在算法 1 中,输入为缺陷模式状态机集合 *dfsm* 和程序的控制流图 *cfg*,一个程序时序安全属性违反对应于一个程序的缺陷.缺陷检测过程计算每一个程序点上缺陷模式状态机的可能属性状态集合,如果可能属性状态集合中包含$*error* 状态,则报告一个可能的缺陷.算法 1 的输出为程序缺陷及缺陷关联信息,缺陷关联信息将可以直接用于减少软件开发人员的缺陷确认负担.

算法 1 的第 1 行~第 3 行是缺陷检测及缺陷关联计算的初始信息:算法的第 1 行将所有缺陷模式状态机置于程序控制流图 *cfg* 的入口节点,并且将缺陷模式状态机初始状态置为$*start*;第 2 行的 *begin* 用于标识是否进行缺陷关联计算的标签,*firstdefect* 用于标识是否进行错误状态切片的标签;第 3 行中的 *RefSemSet* 为在所有求精语义下程序抽象状态集合,而 $\hat{T}'_{\varphi_i}(n)$ 表示为缺陷 φ_i 下的程序求精语义 $\hat{T}'_{\varphi_i}$ 在节点 *n* 的抽象状态.算法的第 4 行~第 11 行为算法主体部分,主要包括缺陷检测、缺陷关联计算及错误状态切片这 3 个阶段.

静态分析可以被视为一个数据流运算过程.在缺陷检测和缺陷关联阶段,为了得到精确的程序状态信息,首先要进行相应路径上的数据流运算.在算法的第 12 行~第 15 行,*CallDomainSet*(*n*)函数依据当前节点 *n* 及前驱状态信息来更新当前节点程序状态信息.算法的第 16 行~第 23 行是缺陷检测过程的实现细节:(1) 首先更新当前节点程序状态信息(第 17 行);(2) 然后利用变量抽象取值信息,更新当前缺陷模式状态机的状态及状态条件,同时删除当前节点缺陷模式状态机的矛盾状态(由不可达路径造成),进而降低误报(第 18 行);(3) 如果相应缺陷模式状态机当前状态为$*error*,则报告一个缺陷(第 19 行、第 20 行);(4) 如果在求精语义下程序抽象状态集合 *RefSemSet* 不为空,则将缺陷关联计算标签置为 true(第 21 行、第 22 行).

在算法 1 中,第 24 行~第 31 行的缺陷关联计算是一个直接过程.对于 *RefSemSet* 集合中任一抽象状态 $\hat{T}'_{\varphi_i}(n)$,首先进行求精语义下的程序抽象状态信息更新(第 26 行),然后将缺陷关联问题转化为求精语义下的缺陷检测

问题(第 27 行).如果在缺陷检测阶段报告的缺陷 *def* 在抽象求精语义 $\hat{T}'_{\varphi_i}$ 下没有报出,就说明 $\varphi_i \rightsquigarrow def$(第 28 行、第 29 行).缺陷的抽象错误状态切除操作在缺陷关联计算完成之后进行(第 8 行~第 10 行).算法 1 并不是对所有的缺陷都进行错误状态切片操作,如果缺陷间变量或表达式相同并且其抽象取值信息也相同,那么只对首个报告的缺陷进行状态切片操作.在缺陷检测过程阶段,静态分析工具通过一个符号化的函数摘要模型实现了上下文敏感的函数间分析.我们的算法不是试图通过求精抽象解释进而达到精确的缺陷检测,而是通过利用静态分析过程中的信息进而优化抽象解释输出的结果.我们通过扩展现有函数摘要模型,增加缺陷关联摘要信息,以此在缺陷关联计算过程中替代其展开.

算法 1. 基于缺陷检测的缺陷关联算法.

输入:缺陷模式状态机集合 *dfsm* 及控制流图 *cfg*⟨*N*,*E*⟩,其中,*N* 为节点集合,*E* 为边的集合.

输出:缺陷 *def* 及其关联信息.

1. *OUT*[*entry*]:=*dfsm*{$*start*:⊥}
2. set *begin*:=false, *firstdefect*:=true;
3. set *RefSemSet*:=*null*; $\hat{T}'_{\varphi_i}(n)$:=*null*;
4. for each *n* in *N* except *entry* do
5. *call DefectDetection*(*dfsm*,*n*);
6. if (*begin*) then
7. *call DefectCorrelation*(*n*,*RefSemSet*);
8. if (*firstdefect*) then
9. *Stateslice*(*def*), add $\hat{T}'_{def}(n)$ into *RefSemSet*;
10. set *firstdefect*:=false;
11. end do
12. procedure *CallDomainSet*(*Node n*){
13. *calulateIn*(*n*);
14. *calulateOut*(*n*);
15. }
16. procedure *DefectDetection*(*dfsm*,*n*){
17. call *CalDomainSet*(*n*);
18. *update state conditions and remove contrary states*;
19. if ($*error*) then
20. report defect *def*;
21. if *RefSemSet*!=*null* then
22. set *begin*:=true;
23. }
24. procedure *DefectCorrelation*(*Node n*,*RefSemSet*){
25. for each $\hat{T}'_{\varphi_i}(n)$ in *RefSemSet* do
26. call *CalDomainSet*(*n*);
27. *DefectDetection*(*dfsm*,*n*);
28. if not report defect *def* then
29. report *defect correlation information*;
30. end do
31. }

2.3 算法时间复杂性分析

通过对上述缺陷关联算法的分析可知,不考虑原有缺陷检测环节,缺陷关联计算的复杂度主要集中在程序求精语义 $\hat{T}'_{\varphi_l}$ 在每个节点 n 的程序状态更新以及当新的缺陷报告时在 $\hat{T}'_{\varphi_l}(n)$ 下属性状态机状态迁移判断.假设控制流图中节点个数为 N,边数为 E,属性状态机状态数为 D,程序中相关变量个数为 V,用于表示变量取值的偏序抽象语义域上基本操作(交、并、补、判相等、切片、判是否矛盾等操作)的最大复杂度为 Q.内层 for 循环的计算复杂度由下述因素共同决定:节点数和边数、每个节点可能出现的最多状态数 D、状态条件中包含的变量数 V、表示变量取值的偏序抽象语义域上基本操作最大复杂度 Q.因此,内层 for 循环的最大复杂度为 $O((N+E)DVQ)$.外层 while 循环的终止条件为所有节点的 $out[n]$集合不发生变化,而 while 循环的每次迭代只能使 $in[n]$和 $out[n]$集合变大或者其状态条件发生变化,假设用于表示状态条件的各抽象语义域的最多增大次数为 H,则 while 循环的迭代次数上限为 $NDVH$.假设一个方法内报告的缺陷个数为 M,因此,$\hat{T}'_{\varphi_l}$ 求精抽象语义计算的最坏情况下的复杂度为 $O((N+E)NMVQH)$.在实际的软件中,每个方法内的时序安全属性违反所导致的缺陷个数一般在数十个以内,缺陷关联计算并不是在每个节点上都需要进行 $\hat{T}'_{\varphi_l}(n)$ 下属性状态机状态迁移判断,而是只有当新缺陷产生时才进行.同时,求精抽象语义计算并不是从程序方法的入口节点开始,而是由缺陷的发生点决定.因此在实际中,缺陷关联计算的复杂度可被认为介于 $O((N+E)VQH)$~$O((N+E)NVQH)$之间.文献[5]中分析了 DTS 中缺陷检测算法的复杂度在 $O((N+E)VQH)$~$O((N+E)NVQH)$之间,由此可见,我们的缺陷关联计算过程的时间复杂性大约是原缺陷检测复杂性的 1~2 倍.

3 实验结果及分析

本节首先介绍基于抽象解释技术的静态缺陷检测系统 DTS 及缺陷检测的整体分析流程,给出我们 DTS 的实验环境设置;然后,为了验证本文方法的有效性,在 DTS 中实现了本文方法,并针对本文方法的时间开销和对原有检测结果的优化情况进行了对比实验分析.

3.1 实验环境配置

DTS 采用状态机来统一描述缺陷模式。目前能检测的缺陷模式包括 179 种,而且可以根据用户需求进行快速缺陷模式定制.DTS 将缺陷模式按其造成的影响分为 4 类:故障模式、安全漏洞模式、疑问代码模式、规则模式.本文讨论的 NPD 模式属于故障模式.

DTS 总体框架及整体的缺陷检测流程如图 4 所示.

(1) 输入与输出.输入部分负责读取待测程序源文件,并从缺陷模式配置文件中读取待检测的缺陷模式;测试结果被写入缺陷库并提供 UI 界面以方便人工缺陷确认.

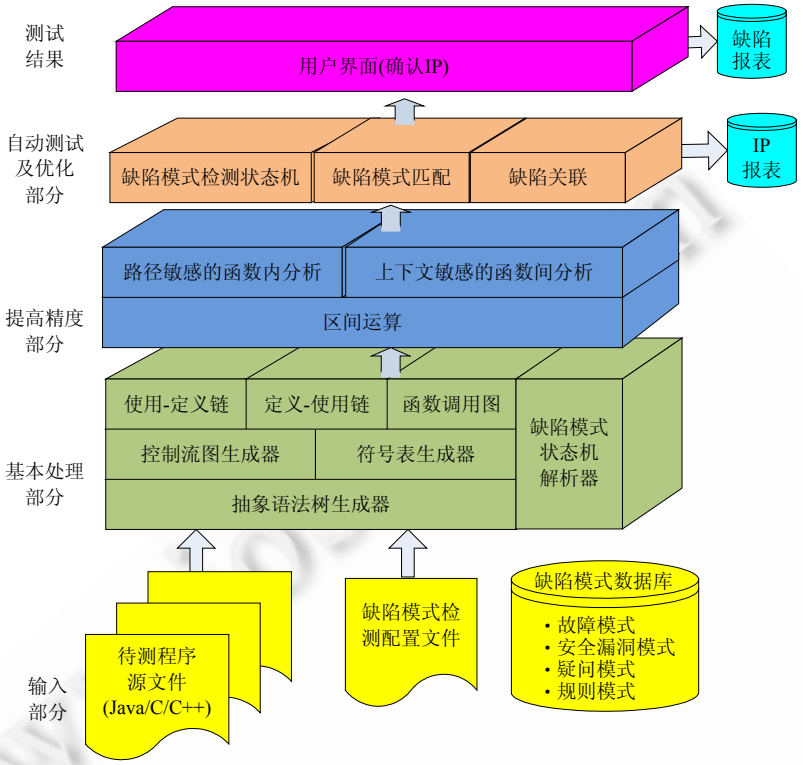
(2) 基本处理部分.该部分生成待测代码的抽象语法树(abstract syntax tree,简称 AST),然后由 AST 生成符号表,进一步生成各函数的控制流图,并基于控制流图进行定义使用链分析.

(3) 提高精度部分.区间分析的结果直接影响整个缺陷检测的精度,因此,DTS 分别实现了基于函数摘要的上下文敏感的函数间分析及基于相同缺陷状态合并的路径敏感的函数内分析.

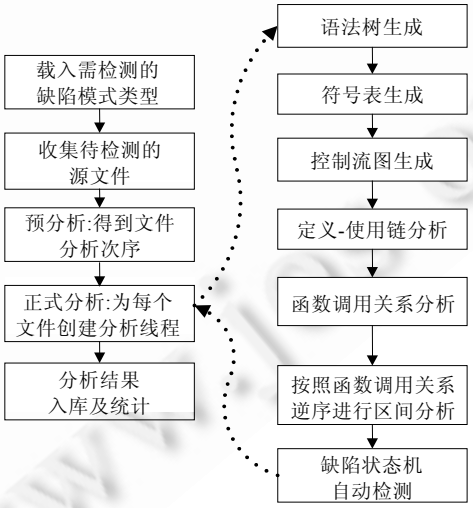
(4) 自动测试及优化部分.该模块按照函数调用关系的拓扑逆序依次对每个函数进行缺陷自动机状态迁代.缺陷状态迁移为$*error* 表示发现某类型的缺陷,或者迁移到$*end* 状态表示未发现缺陷并自动销毁状态机实例;自动测试优化则利用静态分析过程中的信息进一步对静态分析结果进行优化,使得最终缺陷确认更加高效.本文的研究内容即是 DTS 的自动优化部分.

GCC 作为 Linux 系统中开发 C 语言程序的主流编译器,在通过对 ANSIC 标准进行语法扩展获得了更高灵活性的同时,其程序语义也更加复杂,导致其出现软件缺陷的机率也更高,且难以检测.DTS 不仅可以检测 Linux 中 GCC 标准的源程序,而且可以针对 Keil C 标准和 HP C 标准的工程进行检测:首先,修改工程配置文件中的预处理选项,并调用修改后的配置文件编译原工程,使其保留编译过程中产生的中间文件(后缀名为.i);然后,以此预处理后的中间文件作为 DTS 的输入,即可以按照图 4 中的分析流程进行缺陷检测.本文使用 DTS 采用不同的

分析方法对 Linux 中 10 个开源工程进行了缺陷检测对比实验,扫描的缺陷模式为 NPD.在这 10 项开源工程中,源代码量最小的 acpid-1.0.8 为 1 680 行,最大的 httpd-2.4.4 为 20 万行.实验所使用电脑基本配置为 Intel Pentium U5600 1.33G CPU,2.92G 内存,Windows 7 操作系统.



(a) DTS 总体框架



(b) DTS 整体分析流程

Fig.4 DTS overall architecture and analysis process

图 4 DTS 总体框架及整体分析流程

3.2 基于DTS的实验对比

实验过程中使用两种不同的分析方法:

方法 1. 当前 DTS 采用的相同状态合并的路径敏感方法,该方法的实现细节详见文献[5,6].

方法 2. 本文提出的基于缺陷关联的静态分析优化方法.

基于以上实验设置,对静态缺陷检测结果进行了人工统计和确认,结果见表 1,其中,文件数只统计后缀为.c或.h的源文件,源代码行数为去除了空行后的统计结果.DTS并不对C源文件直接进行检测,而是通过DTS编译器对其进行预处理(包括头文件展开、条件编译执行、宏定义替换),对得到的中间文件进行处理,因此,实际测试代码量为中间文件的代码量.由于应用本文基于缺陷关联的静态分析优化方法后,通过缺陷间的关联关系减少了缺陷确认数量,从而可以粗略地估计通过该优化方法,缺陷确认效率提升的情况.

Table 1 Results of comparison experiment

表 1 对比实验结果

Program	Files	Lines	Lines of intermediate file	DTS		DTS with defect correlations		
				Time (s)	Defects	Time (s)	Investigation reduction	Reduction (%)
antiword-0.37	78	20 213	126 925	304	47	558	9	19.1
barcode-0.98	18	3 409	19 591	72	34	92	1	2.9
spell-1.0	6	1 991	4 847	22	67	33	7	10.4
sphinxbase-0.3	120	22 517	157 332	1 356	407	2 006	88	21.6
uucp-1.07	252	52 595	518 067	1 428	513	2 420	73	14.2
sudo-1.8.6	191	56 866	207 123	395	307	406	42	13.7
acpid-1.0.8	6	1 680	13 491	24	0	26	0	0.0
ffmpeg-0.4.8	247	124 911	11 342	1 310	113	1 918	33	29.2
git-1.8.2	468	164 500	2 998 401	6 651	1 368	9 381	293	21.4
httpd-2.4.4	374	204 229	1 369 624	379	434	524	179	41.2
Total	1 760	652 911	5 426 743	11 941	3 290	17 364	725	22.0

统计结果表明:对表 1 中 10 个工程检测其空指针解引用缺陷,共检测源代码量 65 万行,中间文件代码量 542 万行,方法 1 用时 3.32 小时,共上报 3 290 个缺陷,而本文方法用时 4.82 小时.两种方法在时间上的统计结果进一步验证了第 2.2 节的算法时间复杂性分析,即,方法 2 在时间上保持在方法 1 所消耗时间的 2 倍以内.通过人工确认统计发现:利用本文的缺陷关联方法共计减少缺陷确认 725 个,占总缺陷个数的 22%.下面分别对本文方法所计算的缺陷关联情况及方法有效性进行分析:

(1) 缺陷关联结果分析.表 1 统计结果表明,缺陷关联程度最高的工程是 httpd-2.4.4,通过缺陷关联可以有效减少 41.2%的缺陷识别工作.对于 barcode-0.98,由于代码量相对较小并且所报告缺陷数量较小,缺陷关联程度不高.DTS 实际检测文件为中间文件,中间文件数量往往远远大于实际源代码行数,但是对于 ffmpeg-0.4.8,由于其缺少一个头文件,造成部分源文件编译不能通过,进而无法生成全部的中间文件,这就造成了实际检测的中间文件行数小于源代码行数的现象.另外一个极端的例子是 acpid-1.0.8,DTS 没有检测出空指针解引用缺陷,故实际被关联的缺陷为 0.

(2) 本文方法的有效性.从表 1 可以观察到,DTS 真正分析的中间文件代码量一般是源代码量的几倍甚至十几倍,其中,git-1.8.2 和 httpd-2.4.4 工程更是达到了几百万行代码.在我们的实验环境中,采用本文方法,DTS 内存占用最多 1.4GB 左右,CPU 占用 70%左右,百万行原代码量 DTS 采用原方法平均可在 5 小时内完成检测,采用本文方法平均可以在 7.4 小时内完成检测.上述测试结论验证了 DTS 采用本文方法在检测代码量较大工程时,与同类静态检测工具相比不但具有较好的分析效率,同时还对检测结果进行了较好的优化,进而增强了静态分析工具的可用性.

4 相关工作对比

抽象解释技术在静态分析领域被广泛应用.在静态分析过程中,程序的状态通过抽象域中的域元素来近似,程序语义动作通过抽象域中的域操作来可靠建模[6,8].抽象域的设计和实现往往是根据静态分析目的的需要,在

计算效率和分析精度间取得合理平衡,因此实际的缺陷检测结果往往会伴随一定数量的误报.然而验证这些缺陷的正确性仍然靠人工完成,这是一项非常耗时的工作[13].对于一个新开发的大型系统软件,静态测试工具有可能生成数十万甚至上百万的警报,但是处理这些报告的速度是非常低的.过多的警报生成以及很大比例的错误警报可能会导致开发者拒绝使用该款测试工具.

在本研究中,基于缺陷关联的抽象解释优化试图通过缺陷间的关联性对静态分析报告的缺陷分组,对于同一组内的被关联的缺陷,只要确认其主导缺陷即可完成改组缺陷的确认,进而提高确认效率,从另一角度来说,也间接地提高了静态分析的精度.Le 等人首次基于符号执行和约束求解技术提出了路径敏感的故障关联,生成故障关联图以进行故障诊断[4].Le 等人的故障关联方法通过传播一个故障的错误状态,观察这个错误状态能否对新故障产生影响,进而确认故障间的关联关系.我们的缺陷关联技术基于抽象解释.我们通过切掉一个缺陷的错误状态,并将状态切片视为一个外部约束,得到该缺陷的求精语义,并判断在该求精语义下,其他缺陷是否发生,进而优化抽象解释.我们的缺陷关联方法是可靠的,而 Le 等人的方法由于涉及不可达路径判断及指针别名分析,可能会存在关联误报.

我们的工作与 Rival[14]的工作都是通过探索缺陷的错误状态信息,进而求精抽象语义.Rival 通过前向分析及后向分析得到一个缺陷相关的求精语义切片,然后观察是否求精后的初始状态与错误状态仍然是可达的,如果经过语义切片求精,初始状态为空,则说明缺陷被证明为一个误报.另一方面,我们的工作通过引入状态切片,首先切掉缺陷的错误状态,并将该状态切片视为一个程序的外部约束,进而探索在程序的求精语义下,程序的其他缺陷状态是否可达.

Rival 的误报消除原理在某种程度上更类似于 PSE[15].PSE 通过一种路径敏感的事后分析对程序所报告的故障进行回放,如果在程序的某一条路径上,从故障发生点至程序的初始点都没有发现导致程序故障的条件,则排除该路径发生故障的可能.我们的工作不去试图验证静态分析所输出缺陷的正确性,而是对输出的缺陷进行分组.PSE 通过对每个状态元组(state tuple)中出现的函数进行重复处理,实现上下文敏感的分析,而本文的静态分析工具 DTS 通过函数摘要避免了这种效率与精度的损失.

我们的工作与 Lee 等人[16]的工作都是在抽象解释技术框架下,试图对缺陷进行分组.不同之处是,Lee 等人借鉴了反例求精的思想方法.他们的方法首先需要对被测程序生成一个超级控制流图,进而进行完整的程序分析.这种方式无疑将需要大量的时间和内存开销,在大型程序的分析过程中无法实现.Lee 等人的工作探索的是缺陷依赖子集.即,他们只考虑相邻两个缺陷之间的依赖关系,当两个相邻缺陷之间不存在依赖关系时,将这两个缺陷的错误状态合并,进而判断两个连续的缺陷是否与后续缺陷存在关联关系,这样就放弃了计算一个缺陷与后续所有缺陷的关联关系.我们的方法借鉴了程序的外部输入约束思想.我们将缺陷的错误状态切片作为一个外部约束,进而考察在基于外部约束的求精语义下,其他缺陷是否发生.我们的工作考虑了所有缺陷间的关联关系.我们的静态分析工具 DTS 是基于函数摘要的.我们的缺陷关联计算通过添加关联摘要属性,扩展了 DTS 原有函数摘要信息.我们的方法并不是对所有的缺陷生成关联摘要属性信息,而是只对那些影响函数返回值及全局变量取值的缺陷生成关联摘要属性信息.

我们的静态分析工具 DTS 在前期已经得到了大量求精与优化.肖庆等人通过使用变量取值信息来表达程序的路径状态,进而实现了 DTS 的路径敏感的分析[5].赵云山在原有表达式区间抽象域的基础上引入了符号化三值逻辑区间抽象域[17],不但可以表示变量间的线性关联关系,而且还可以表达变量间的逻辑关联关系.周虹伯等人扩展了 DTS 原有的符号化区间域,实现了基于域敏感指向分析的区间抽象域,进一步提高了 DTS 指针相关缺陷的检测精度[18].赵云山等人通过将前向数据流分析与逆向约束搜索技术相结合,进一步对 DTS 所产生的误报进行消除[7].我们的工作可以看作是上述静态分析求精工作与缺陷误报消除工作的一个正交.

5 结论与未来的研究

本文提出了一种基于缺陷关联的可靠的静态分析优化方法.我们首先给出了缺陷关联的定义,然后正式提出了缺陷错误状态切片.我们将缺陷的错误状态切片作为一种程序的外部输入约束,进而得到了基于外部约束

的程序求精语义.我们探索在该求精语义下,程序的其他缺陷是否发生.实验结果表明,我们的优化方法可以有效减少缺陷的确认工作,为静态分析工具适用于大规模的、高可信的嵌入式软件测试提供有力的支持.

本文工作的局限性主要在于,在缺陷错误状态切片过程中,由于符号化区间抽象域的表示及计算能力不足.对于涉及表达式相关的缺陷,如何在现有静态分析工具所提供的抽象域下精确地切除其错误状态,仍然是下一步需要改进的方向.

References:

[1] Bush WR, Pincus JD, Sielaff DJ. A static analyzer for finding dynamic programming errors. Software-Practice and Experience, 2000,30(7):775−802. [doi: 10.1002/(SICI)1097-024X(200006)30:7<775::AID-SPE309>3.0.CO;2-H]

[2] Das M, Lerner S, Seigle M. ESP: Path-Sensitive program verification in polynomial time. ACM SIGPLAN Notices, 2002,37(5): 57−68. [doi: 10.1145/543552.512538]

[3] Rice HG. Classes of recursively enumerable sets and their decision problems. Trans. of the American Mathematical Society, 1953, 74(2):358−366. [doi: 10.1090/S0002-9947-1953-0053041-6]

[4] Le W, Soffa ML. Path-Based fault correlations. In: Proc. of the 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2010. 307−316. [doi: 10.1145/1882291.1882336]

[5] Xiao Q, Gong YZ, Yang ZH, Jin DH, Wang YW. Path sensitive static defect detecting method. Ruan Jian Xue Bao/Journal of Software, 2010,21(2):209−217 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3872.htm [doi: 10.3724/SP.J.1001.2010.03872]

[6] Xiao Q, Chen JL. Research on key technologies of improving the accuracy of static defect detecting [Ph.D. Thesis]. Beijing: Beijing University of Posts and Telecommunications, 2011 (in Chinese with English abstract).

[7] Zhao YS, Gong YZ. Research on symbolic analysis based static defect detection technique [Ph.D. Thesis]. Beijing: Beijing University of Posts and Telecommunications, 2012 (in Chinese with English abstract).

[8] Cousot P, Cousot R. Abstract interpretation frameworks. Journal of Logic and Computation, 1992,2(4):511−547. [doi: 10.1093/logcom/2.4.511]

[9] Cousot P, Cousot R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages. ACM Press, 1977. 238−252. [doi: 10.1145/512950.512973]

[10] Cousot P, Cousot R. Static determination of dynamic properties of generalized type unions. ACM SIGOPS Operating Systems Review, 1977,11(2):77−94. [doi: 10.1145/390018.808314]

[11] Wang YW, Chen JL. Research on software testing technology based on defect pattern [Ph.D. Thesis]. Beijing: Beijing University of Posts and Telecommunications, 2009 (in Chinese with English abstract).

[12] Zhao Y, Gong Y, Liu L, Xiao Q, Yang Z. Context-Sensitive interprocedural defect detection based on a unified symbolic procedure summary model. In: Proc. of the 2011 11th Int'l Conf. on Quality Software (QSIC). IEEE, 2011. 51−60. [doi: 10.1109/QSIC.2011.15]

[13] Dillig I, Dillig T, Aiken A. Automated error diagnosis using abductive inference. ACM SIGPLAN Notices, 2012,47(6):181−192. [doi: 10.1145/2254064.2254087]

[14] Rival X. Understanding the origin of alarms in Astrée. In: Proc of the 12th Int'l Conf. on Static Analysis. London: Springer-Verlag, 2005. 303−319. [doi: 10.1007/11547662_21]

[15] Manevich R, Sridharan M, Adams S, Das S, Yang Z. PSE: explaining program failures via postmortem static analysis. ACM SIGSOFT Software Engineering Notes, 2004,29(6):63−72. [doi: 10.1145/1029894.1029907]

[16] Lee W, Lee W, Yi K. Sound non-statistical clustering of static analysis alarms. In: Proc. of the Verification, Model Checking, and Abstract Interpretation. Berlin, Heidelberg: Springer-Verlag, 2012. 299−314. [doi: 10.1007/978-3-642-27940-9_20]

[17] Zhao Y, Wang Y, Gong Y, Chen H, Xiao Q, Yang Z. STVL: Improve the precision of static defect detection with symbolic three-valued logic. In: Proc. of the 2011 18th Asia Pacific Software Engineering Conf. (APSEC). IEEE, 2011. 179−186. [doi: 10.1109/APSEC.2011.23]

[18] Zhou H, Wang Q, Jin D, Gong Y. A Static detecting model for invalid arithmetic operation based on alias analysis. In: Proc. of the 2012 IEEE 23rd Int'l Symp. on Software Reliability Engineering Workshops (ISSREW). IEEE, 2012. 183−188. [doi: 10.1109/ISSREW.2012.14]

附中文参考文献:

[5] 肖庆,宫云战,杨朝红,金大海,王雅文.一种路径敏感的静态缺陷检测方法.软件学报,2010,21(2):209−217. http://www.jos.org.cn/1000-9825/3872.htm [doi: 10.3724/SP.J.1001.2010.03872]

[6] 肖庆,陈俊亮.提高静态缺陷检测精度的关键技术研究[博士学位论文].北京:北京邮电大学,2011.

[7] 赵云山,宫云战.基于符号分析的静态缺陷检测技术研究[博士学位论文].北京:北京邮电大学,2012.

[11] 王雅文,陈俊亮.基于缺陷模式的软件测试技术研究[博士学位论文].北京:北京邮电大学,2009.

张大林(1983－),男,内蒙古宁城人,博士生,主要研究领域为软件测试,程序静态分析.
E-mail: dalin@bupt.edu.cn

王前(1983－),男,博士生,主要研究领域为软件测试,程序静态分析.
E-mail: wavericq@163.com

金大海(1974－),男,博士,副教授,CCF 会员,主要研究领域为软件工程,软件测试.
E-mail: jindh@bupt.edu.cn

董玉坤(1981－),男,博士生,CCF 学生会员,主要研究领域为软件测试,程序静态分析.
E-mail: dongyk@upc.edu.cn

宫云战(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件测试.
E-mail: gongyz@bupt.edu.cn

张海龙(1981－),男,硕士生,主要研究领域为软件工程.
E-mail: zhhailon@gmail.com