软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(12):2905−2918 [doi: 10.3724/SP.J.1001.2011.03956]

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

程序代码中隐含数据与控制的 Petri 网建模技术*

周国富[1,2], 杜卓敏[2+]

[1](武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

[2](武汉大学 计算机学院,湖北 武汉 430072)

Petri Nets Model of Implicit Data and Control in Program Code

ZHOU Guo-Fu[1,2], DU Zhuo-Min[2+]

[1](State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

[2](School of Computer, Wuhan University, Wuhan 430072, China)

+ Corresponding author: E-mail: zmdu@whu.edu.cn

Zhou GF, Du ZM. Petri nets model of implicit data and control in program code. *Journal of Software*, 2011, 22(12):2905−2918. http://www.jos.org.cn/1000-9825/3956.htm

Abstract: To describe the implicit data and the implicit control in program code, CNets, one extension of Petri nets, is applied. By data view nets and control view nets, data and control of the program code are modeled. Based on the CNets specification, interactions between data flow and control flow, relations among data, operations, and resources are also captured respectively. Meanwhile, mapping rules from CNets specification to Petri nets are presented. According to the rules, from CNets specification, properties of the program are analysized through Petri nets techniques without a running a program.

Key words: formalization; Petri net; program semantics; static analysis

摘 要: 应用一种 Petri 网扩展 CNets,描述程序代码中所隐含着的数据与控制,分别给出了程序代码的数据视图和控制视图.在 CNets 规范的基础上,研究程序中的数据与控制流之间的交互以及程序中数据、操作和资源之间的关系,同时给出 CNets 规范向经典 Petri 网规范的转换规则.在不运行程序的前提下,根据这一映射机制,通过 CNets 规范,利用经典 Petri 网理论对程序的性质进行分析.

关键词: 形式化;Petri 网;程序语义;静态分析

中图法分类号: TP311 **文献标识码**: A

随着软件系统越来越复杂和庞大,软件中的安全漏洞数目急剧增加[1],单纯靠人工来保证系统的安全也越来越困难,人们急需一种自动化分析方法来帮助程序员发现软件中的缺陷[2−6].自 2006 年以来,Coverity 公司的 SCAN 项目已经完成扫描并分析 6 000 多万行不重复的程序代码,涉及各类项目 26 181 个,其中有 280 多种著名开源项目(如 Firefox,Linux,Perl,Python 以及 PHP 等).在 2009 年度报告[7]中指出,最常见的缺陷仍然是 NULL pointer,resource leaks,unintentional ignored expressions 等.事实上,当前国内外研究机构关于系统漏洞研究的主

* 基金项目: 国家自然科学基金(61040036); 国家教育部留学回国人员科研启动基金; 湖北省自然科学基金(2009CDB218); 中央高校基本科研业务费专项资金(6082015); 高等学校学科创新引智计划(B07037)

收稿时间: 2010-04-20; 修改时间: 2010-07-06; 定稿时间: 2010-10-11

要切入点也是指针和资源泄漏[8−11].

程序形式化验证是保证程序正确性的重要手段.经过多年的积累,已经获得了大量的程序建模工具[12−17].然而,应用程序形式化证明技术发现程序缺陷(如漏洞)尚有较大困难,原因在于:首先,实际的软件开发过程很难达到形式化验证所需要的理想状态;其二,经过理论验证后的形式规范在向程序代码的转换过程中,仍然可能会发生某种程度的语义变化甚至丢失.因此,Clarke 等人提出了介于动态测试[18]和程序验证之间的静态分析方法[3].静态分析并不能够证明程序的完全正确性,而是利用形式化技术尽可能多地发现程序中的错误或缺陷.20 世纪 90 年代后期,Lee 和 Necula 提出一种编译阶段的程序验证框架 PCC(proof-carrying code)[19],即在生成低级目标代码的同时对其进行验证.2003 年,Hoare 的题为“验证式编译器:计算研究的伟大挑战”的论文[20],更加激发了人们对编译阶段增加安全性分析和验证技术的关注.静态分析正是构建源代码的中间模型,分析并验证[18,21−24]程序的性质,从而发现缺陷;另一个客观事实是,现实中存在着大量待验证的、不具备理想形式规范的应用程序.因此,面向程序代码的静态分析技术显然更具有实际意义[25,26].

任何特定的程序是可计算问题(算法)的某一种实现,比如,解决同一问题可以有顺序性和并行性两类程序,其执行性与平台密切相关[27].程序代码的静态分析首先面对的是这种平台依赖性问题,也就是说,程序包含平台依赖的数据和控制.传统程序语义研究的是从控制流和数据流两个方面分别展开[28]的.

控制流图采用图形符号描述程序所有的执行路径[28],可达性、不可达性、循环与死代码等都能够在可达图中分析得到[29].程序中“条件转移”、“循环”等结构信息有助于程序性质的分析[30−32].比如,文献[33,34]应用 Petri 网描述临界区类物理资源(控制流是资源的流动轨迹),并讨论死锁的发现、诊断和避免.一般来说,动态性质难以通过静态方法来分析,比如从死锁、饥饿等问题[35],原因在于程序并不直接描述程序所有的控制流(获得动态性质).因此,当程序运行时,可能会出现许多“额外”的(未在代码显式描述的)控制流.这些“额外”的控制流受程序变量值和环境影响,有上下文特性.

数据流图与控制流图类似,也是采用图形方式来描述系统功能和数据的逻辑流向和逻辑变换[36−38],从而分析程序的可达、活性等属性.一般来说,数据流分析从控制流图开始,确定程序控制流图中某一点(语句)中变量的使用、取值以及变量值的作用范围.数据流分析技术最早被用于编译优化[39],检查程序中的数据异常问题[40].

数据流图通过函数机制抽象了使得变量发生变化的过程,把注意力集中在变化的结果上,而对产生这个结果的过程关注不足.控制流图则相反.程序动态性质是数据流和控制流相互作用的结果.因此,控制流和数据流结合的分析技术是非常富有吸引力的[41,42].

程序代码模型是变量(保存数据的机制)、操作、控制机制、资源约束等元素的混合体[43].变量(数据)是可观察的,是记录程序状态的重要构件,是认识程序性质的窗口.而程序的这种可观察性是通过计算平台资源实现的.

控制分成两大类:一是算法本身普遍具有的一些控制成份,比如因果关系、依赖关系等,是不依赖计算平台的;二是计算平台的控制,是程序中与平台相关的成分.平台控制又分为绝对控制和相对控制.绝对控制就是计算平台无条件的对程序运行的干涉,比如 CPU 时间分配;相对控制就是计算平台通过物理资源的分配对程序的运行产生间接影响,比如内存分配与回收.

程序中的一个非常重要的成分——变量,分为物理变量和算法变量两类.物理变量是计算机平台中存在的变量,是存放值的空间,具有物理地址和物理空间.物理变量所承担的操作是读出和写入两个基本的操作.物理变量的改变是具有过程性的(该过程性也是缺陷的潜在地之一).算法变量则是抽象的纯数学概念,承担数学运算.数学变量的值只能是初始值或当前值,变量值的变化过程/时间是瞬时不计的.算法变量通过物理变量而表现并被观察.

本文基于袁崇义教授所提出的 Petri 网扩展——CNets[43,44],结合面向程序代码特征,解决程序中变量、操作、资源以及读写的描述.对程序中隐含的控制、数据以及控制与数据之间的关系建模,并给出 CNets 与有色网之间的映射规则,实现经典 Petri 网理论技术的重用,从而达到对程序代码进行静态分析的目标.

本文第 1 节给出扩展的 CNets 的形式定义,该定义是在文献[43,44]基础上进行的,增加了有色网、数据视图、

控制视图等组成的讨论.第 2 节讨论从 CNets 向有色网映射的规则,以及介绍 CNets 保持的有色网相关属性.为进一步说明映射和 CNets 保持的属性,第 3 节通过实例详细说明程序代码的 CNets 建模特点.

1 形式定义

在文献[43,44]的工作基础上,本节给出 CNet 的定义,并增加与有色网之间关系的讨论,同时给出面向代码分析的数据视图和控制视图定义.有关 Petri 网的概念与术语采用文献[13,44]的定义.本文涉及到的集合均为有穷集合.

定义 1.1. 六元组 $N=(S_c,S_v,T;R,W,F)$,其中:

- $S_c\cap S_v=\varnothing$;
- $(S_c\cup S_v)\cap T=\varnothing$;
- $S_c\cup S_v\cup T\neq\varnothing$;
- $S=S_v\cup S_c$.

其中,S 是库所集合,S_v 是变量库所集合,S_c 是控制库所集合,T 是变迁集合.

- $R\subseteq S_v\times T$;
- $W\subseteq T\times S_v$;
- $F\subseteq S_c\times T\cup T\times S_c$.

其中,R,W,F 是关系集.

- $S_v=dom(R)\cup cod(W)$;
- $S_c\subseteq dom(F)\cup cod(F)$;
- $T\subseteq dom(F)\cup dom(W)\cup cod(F)\cup cod(R)$.

F 从 S_c 到 T,也可以从 T 到 S_c.F 表示库所中的托肯流动,图形描述与经典 Petri 网一致,用带箭头的线段“→”表示.

R 从 S_v 到 T,表示读出数据的关系,托肯无流动性.R 的图形表示是带有小圈的线条“—○”,小圈端指向变迁.

W 连接从 T 到 S_v,表示写入关系,托肯无流动性.W 的图形表示为“—○”,小圈端指向变量库所.

读写弧是读弧和写弧的合成,图形表示为两头都有小圈的线段“○—○”.

定义 1.2. 给定一个 CNet $N=(S_c,S_v,T;R,W,F)$,N_{cv} 称为 N 的控制视图 *cvNet*

$$N_{cv}=(S_c,T|_{S_c};F),$$

其中,$T|_{S_c}$ 是连接控制库所的变迁集合.

定义 1.3. 给定一个 CNet $N=(S_c,S_v,T;R,W,F)$,N_{dv} 称为 N 的数据视图 *dvNet*

$$N_{dv}=(S_v,T|_{S_v};R,W),$$

其中,$T|_{S_v}$ 是连接变量库所的变迁集合.

定理 1.1. 若 N_{cv},N_{dv} 分别是 CNet N 的 *cvNet* 和 *dvNet*,则

$$N=N_{cv}\cup N_{dv}.$$

这里,∪是关系的并操作符.

证明:根据图论和集合原理,结论易证. □

由以上结论可知,如果获得了程序的 CNet 规范描述,则程序性质可以从数据和控制两个角度进行研究.控制视图描述了程序中存在的控制流关系;数据视图描述了程序中的变量关系(或者实体关系).

定义 1.4. 四元组 $\Sigma=(N,\mathcal{S},\mathcal{Z},M_0)$,其中,$N=(S_c,S_v,T;R,W,F)$是 CNet;

$S:S\rightarrow\mathcal{C}$,

$S=S_c\cup S_v$;

$\mathcal{C}=Integer\cup DATATYPE$ 是颜色集合;

$\forall s_c \in S_c, \mathbb{S}(s_c)=Integer$;

$\forall s_v \in S_v, \mathbb{S}(s_v)=DATATYPE$;

$\mathbb{Z}: T \to G_c \cup G_v \cup L$;哨

$G_c: T \to BOOL$;控制哨

$G_v: T \to BooleanExpression$;变量哨

$L: T \to statement$;语句

M 表示 N 的标识,$M=M_c \cup M_v$;

$M_c: S_c \to \mathbb{N}$;控制库所中的标识,库所中托肯数.

$M_v: S_v \to DATA$;变量库所中的标识为变量值.

M_0 表示初始标识;M_{c0} 表示 S_c 的初始标识;M_{v0} 表示 S_v 的初始标识.

变迁的触发导致标识的变化在本节后半部分讨论.

$\mathbb{S}$是从库所集 S 到颜色集$\mathbb{C}$的函数

$$\mathbb{S}: S \to \mathbb{C}.$$

特别地,若颜色集存在关系$\mathbb{C}=\mathbb{C} \cup Integer \cup DATATYPE \cup BOOL \cup BooleanExpression \cup statement$ 以及 $cd=\mathbb{S} \cup \mathbb{Z}$, CNets 可以映射到有色网[55] $\mathfrak{M}=\langle P, T, Pre, Post, \mathbb{C}, cd\rangle$.映射规则将在下一节中继续讨论.

与有色网类似,CNets 变迁的触发由哨控制,哨是布尔型变量[13].G_c 与变量无关,而与资源控制相关,称为控制哨,定义为

$$G_c: T \to BOOL.$$

T 是变迁的集合.G_c 总是从 true 到 false 轮流变化.当变迁的前集库所中包含有足够托肯时,该变迁的哨为 true;否则,该变迁的哨为 false.当哨为 true,T 可触发;否则,T 不可触发.

扩展另一个哨来监控变量库所,这样的哨称为变量哨.令 G_v 是一个变量哨

$$G_v: T \to Boolean\ Expression,$$

其中的布尔表达式 *Boolean Expression* 描述了变迁触发所需的变量条件.布尔表达式中的操作数可以是常量或者变量库所.变迁由两个哨来控制其实际的触发.

变迁的第 3 个组成部分是语句集合.语句描述了变迁执行所期望的动作.该语句可以用编程语言编写,比如 C,Java 等.令函数 L 是变迁的动作脚本,那么,

$$L: T \to statement.$$

综上所述,变迁包含 3 部分组成:G_c,G_v 和 L.令函数$\mathbb{Z}$为

$$\mathbb{Z}: T \to G_c \cup G_v \cup L.$$

但是,并非每个变迁都必须同时具有上述 3 项.根据实际情况,变迁可以包含其中的一项或者两项,主要取决于变迁是否关联变量.对于$\forall t \in T$,

- 当 $R(t)=\varnothing \wedge W(t)=\varnothing$时,有 $L(t)=\varnothing \wedge G_v(t)=\varnothing$.无读写弧,变迁不关联变量库所;
- 当 $G_v(t)=\varnothing \wedge L(t)=\varnothing$时,变迁与有色网中的变迁一致,托肯从变迁的前集流向后集;
- 当 $G_c(t)=\varnothing$时,变迁不关联包含流动托肯的库所.

作为缺省值,定义:

- 如果 $G_v(t)=\varnothing$,令 $G_v(t)$=true;
- 如果 $G_c(t)=\varnothing$,令 $G_c(t)$=true.

定义 1.5. 给定 Cnets $\Sigma=(N, \mathbb{S}, \mathbb{Z}, M_0)$,称 N^{cs} 为 cvNets.

$$N^{cs}=(N_{cv}, \mathbb{S}|_{S_c}, \mathbb{Z}|_{S_c}, M_0|_{S_c}),$$

其中, $M_0|_{S_c}=M_{c_0}$;有色网的初始标识; $\mathcal{S}|_{S_c}:S_c\to integer$;是有色网的颜色集合; $\mathcal{Z}|_{S_c}=G_c$;变迁中的控制哨.

定义 1.6. 给定 CNets $\Sigma=(N,\mathcal{S},\mathcal{Z},M_0)$,称 N^{ds} 为 dvNets.

$$N^{ds}=(N_{dv},\mathcal{S}|_{S_v},\mathcal{Z}|_{S_v},M_0|_{S_v}),$$

其中,

- $M_0|_{S_v}=M_{v_0}$;变量库所的初始标识;
- $\mathcal{S}|_{S_v}:S_v\to DATATYPE$;变量库所的颜色(类型)集合;
- $\mathcal{Z}|_{S_v}=G_v\cup L$;变迁中的变量哨和动作脚本.

定理 1.2. 设 N^{cs} 是 N 的 cvNets,N^{ds} 是 N 的 dvNets,则

$$N=N^{cs}\cup N^{ds}.$$

这里,∪是关系的并操作符.

证明:根据图论和集合原理,结论易证. □

研究程序的动态特征时,基于 CNets 的规范可以从 cvNets 和 dvNets 两个视图来展开.特别地,由于$\mathcal{Z}$中哨 G_c 和 G_v 一起决定变迁的触发,因此可以根据哨来研究程序中与具体数据相关的属性,如代码覆盖率、关键路径、路径的选择等.

定义 1.7. $x\in S\cup T$,

$r_s(x)=\{a|(a,x)\in R\wedge x\in T\}$,表示变迁 x 所读的变量库所集合;

$w_s(x)=\{a|(x,a)\in W\wedge x\in T\}$,表示变迁 x 所写的变量库所集合;

$r_t(x)=\{a|(x,a)\in R\wedge x\in S\}$,表示读变量库所 x 的变迁集合;

$w_t(x)=\{a|(a,x)\in W\wedge x\in S\}$,表示写变量库所 x 的变迁集合;

$r(x)=r_s(x)\cup r_t(x)$,表示读 x;

$w(x)=w_s(x)\cup w_t(x)$,表示写 x;

${}^\bullet t=\{p|(p,t)\in F\}$,表示变迁 t 的前集;

$t^\bullet=\{p|(t,p)\in F\}$,表示变迁 t 的后集.

定义 1.8. 在标识 $M=M_c\cup M_v$ 下,t 是 *C_enabled*,当且仅当

$$C\_enabled(M,t)\equiv enabled(M_c,t),$$

其中,$enabled(M_c,t)$表示${}^\bullet t$ 具有足够的托肯数并引起控制哨 G_c=true.如果${}^\bullet t=\varnothing$,则令 G_c=true.

t 是 *V_enabled*,当且仅当

$$V\_enabled(M,t)\equiv enabled(M_v,t),$$

其中,$enabled(M_v,t)$表示变量哨 G_v 成真.如果 $r(t)\cup w(t)=\varnothing$,则令 G_v=true.

定义 1.9. t 在标识 M 下是可触发的,记为 $M[t\rangle$,当且仅当在标识 M 下,t 既是 *C_enabled* 又是 *V_enabled*,

$$M[t\rangle\equiv C\_enabled(M,t)\wedge V\_enabled(M,t).$$

变迁 t 由 G_v 和 G_c 一起控制触发.即变迁触发的条件为:所有需要的资源都能满足;同时,所有相关的变量都满足条件.

定义 1.10. 如果 $M[t\rangle$,令 M'是 M 的后续标识,则 $M[t\rangle M'$.这里,M'定义为

如果 $s\in S_c$,

$$M'(s)=\begin{cases}M(s)-1, & \text{如果}s\in {}^\bullet t-t^\bullet\\ M(s)+1, & \text{如果}s\in t^\bullet-{}^\bullet t\\ M(s), & \text{如果}s\notin {}^\bullet t\text{或者}s\in {}^\bullet t\cap t^\bullet\end{cases}.$$

如果 $s\in S_v$,

$$M'(s)=\begin{cases}\sum_i Val(\mathbb{Z}(t_i)), & \text{如果} s\in w(t_i)\\ M(s), & \text{如果} s\in r(t)\end{cases}.$$

这里,$Val(\mathbb{Z}(t_i))$表示可被 t_i 修改的 s 当前值.

2 与有色网的映射

Petri 网不仅具有丰富的理论基础,还具有图形特征,比纯文本形式的代数描述更加直观易懂.但是由于 Petri 网的绝对动态性,并不能完全描述程序的静态属性,原因是:

(1) 变量是程序的基本构件,而 Petri 网的基本构件之一是状态.变量是相对静止的,是一个存放值的位置,而且其值也并不是流动的.变量和状态并不总是一一对应的,更多情况是一个变量对应着多个状态;

(2) 由于 Petri 网描述资源的抽象性,无法区分资源的不同.然而,程序中总是存在多种资源,比如多个变量和操作等.为适应描述程序代码的需要,必须克服 Petri 网资源描述过于抽象的问题,以区别多种不同的资源;

(3) Petri 网适合描述控制流,也因此导致对数据流的描述不足.程序的动态性质中不仅包含控制流,还包含着数据流,还存在着数据与控制互相影响的关系;

(4) 经典 Petri 网把操作均抽象为变迁,因此无法描述具体的程序语句.程序代码的形式化需要解决不同操作的描述,以及如何实现程序指令在 Petri 网规范中描述.

如果仅仅采用经典 Petri 网形式化,那么经典 Petri 网存在的问题仍然不能解决,如状态爆炸、描述的复杂繁琐等.因此,有必要研究 CNets 与 Petri 网的转化.该过程不仅要保证映射之间的语义传递,还要保证语义的不受损.CNets 是 Petri 网的扩展,需要解决的是数据描述如何向经典 Petri 网描述转换.分析方法则从两个方向展开:一是重用经典 Petri 网的基本分析技术;另一个就是基于 CNets 规范来提出新的分析技术.正如上节所讨论,CNets 与有色网相比有 3 个不同构成元素:

(1) 每个变量库所可保持一个托肯,该托肯不能从变量库所中流进流出;

(2) 读弧表示变迁的触发并不消耗所关联变量库所中的托肯.读写弧没有权;

(3) 写弧表示相关的变迁更新该弧所关联的变量库所的当前值.写弧没有权.

因此,要实现 CNets 到有色网之间的转换,需要定义托肯、变量库所、读弧、写弧和变迁的语义.CNets 有控制库所和变量库所两种类型.控制库所的特点与有色网中的库所类似,因而不需要特别的规定,控制库所可以直接映射到有色网的库所.变量库所的托肯表示库所所描述变量的当前值.变量库所要映射到有色网中有颜色的库所,有两个问题需要解决:

- 如何处理变量库所中的托肯?
- 如何处理变量库所所关联的读弧和写弧?

为解决这两大问题,首先研究托肯的语义.CNets 包含两种托肯:一是控制库所中的托肯,二是变量库所中的托肯.控制库所中的托肯与有色网中库所中的托肯语义一样,可以沿着弧的方向流动.因此,这类托肯不需要更多的映射规则.变量库所中的托肯表示库所所描述的变量当前值.变量库所中的托肯是不流动的.显然,由 CNets 中非流动性的托肯映射到有色网中流动性的托肯,需要一些规则约束,特别是对于弧的语义约束.

在经典 Petri 网[13,44]中,弧的权是常量.然而,当库所描述变量时,弧的权将是不可确定的.有色网中,弧(权)由变量或者一个表达式来标识.变量或表达式描述了变迁的触发将会消耗掉或者产生的托肯数.在 CNets 中,弧是没有标识的.弧分为控制弧、读弧和写弧这三大类.

控制弧与经典 Petri 网的弧语义一致,其语义是:当变迁触发后,变迁将要消耗掉一定的托肯,同时会产生一定数目的托肯.控制弧仅连接控制库所和变迁.当控制弧映射到有色网弧时,语义保持不变.

读弧的语义是:当变迁触发时,变迁将会从相关联的变量库所中读取当前值,并不修改该值.为保证语义的一致性,读弧可以由两个控制弧模拟:一个控制弧表示消耗掉库所中所有的托肯,另一个控制弧表示将产生相同数目的托肯.消耗掉的托肯数和产生的托肯数是一致的,因此库所中的托肯总量保持不变.这两个弧由相同的变量或表达式来标识(如图 1 所示).

写弧的语义是:当变迁触发时,变迁将修改所关联的变量库保持的当前值.当写弧映射到有色网弧时,语义将变成:托肯将被消耗掉,同时产生新的托肯并流入下一个库所中.即,写弧将产生新的托肯,老的托肯将被丢弃.在映射时,与读弧类似,写弧由两个控制弧来模拟.与读弧不同的是,两个控制弧消耗和产生的托肯数并不完全相等.这些弧也可以由变量或者函数标识(如图 2 所示).

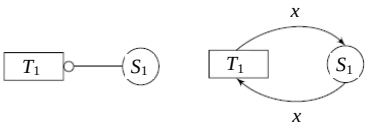


Fig.1 Readarc mapping rule

图 1 读弧映射

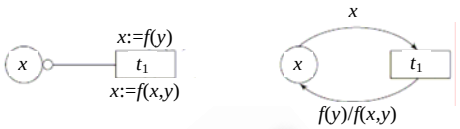


Fig.2 Writearc mapping rule

图 2 写弧映射

变迁的语义和库所密切相关.变迁的触发取决于与该变迁连接的是控制库所还是变量库所.当变迁连接控制库所时,其语义与经典 Petri 网一致;否则,变迁的具体行为由变迁的语句来描述.当变迁触发时,变量库所的托肯总是被完全移除,此时产生的新托肯数目依赖于语句的规定.当映射时,CNets 中变迁的语句将从变迁中移出,与变量哨一起放置到弧标识上.有色网弧的标识将由合成后的语句和哨来描述[13].

为有助于直观理解这些映射规则,我们从文献[13]中选取一个赛车的例子加以说明.P_1 中有两辆赛车 a,b,变迁 t 表示信号枪,枪响后,赛车处于就绪状态 P_4.为节省篇幅,忽略其中的细节,选取其中代表性的库所与 CNets 库所之间映射进行讨论.P_1 包含托肯 a 和 b,P_4 中的托肯取决于 P_1 中的托肯.P_4 除了依赖 P_1 之外,还依赖其他库所(如赛手);P_1 中的托肯也不绝对确定,可能依赖于其他库所(如允许的赛车)中的托肯.这种情况下,P_1 类似于 CNets 的被读变量库所,P_4 类似于 CNets 的被写变量库所(如图 3 所示).

变迁中的托肯颜色取值和变化由颜色函数决定.令函数 $Pre[p,t]:cd(t)\rightarrow Bag(cd(p))$定义了变迁 t 触发所接受的托肯颜色(前集).所有可能出现的颜色集合$\beta\in cd(t)$,当 t 以颜色β触发,$Pre[p,t](\beta)\in Bag(cd(p))$表示将托肯从 p 中移除.

同样地,定义 $Post[p,t](\beta)$表示当 t 以颜色β触发时,托肯将加入到 p 中.映射时,Pre,$Post$ 实际上就是函数 $sign(id)$通过参数 id 决定了 P_1 和 P_4 的托肯(如图 3 所示).弧的图形表示“→”变为“—○”.

规则 1(读弧). 当弧是读弧$(x,t)\in R$,这里,x 是库所,t 是变迁,则

- $Pre(x,t)=\{x\}$;
- $Post(x,t)=\{x\}$.

映射规则如图 1 所示.

规则 2(写弧). 当弧是写弧$(x,t)\in W$,这里,x 是库所,t 是变迁,则

- $Pre(x,t)=\{x\}$;
- $Post(x,t)=L$.

这里,L 是产生托肯的函数.映射规则如图 2 所示.

规则 3(读写弧). 当弧是读写弧$(x,t)\in W\wedge(x,t)\in R$,这里,$x$ 是库所,t 是变迁,则

- $Pre(x,t)=\{x\}$;
- $Post(x,t)=L$.

这里,L 是产生托肯的函数.映射规则如图 2 所示.

规则 4(哨). 控制哨直接映射到有色网中的哨,变量哨则和变迁中的语句一起被置于弧标识中.弧表达式 $POST$ 是

arc expression := if 变量哨 then statement
else ε

这里,ε表示特殊空.映射规则如图 4 所示.

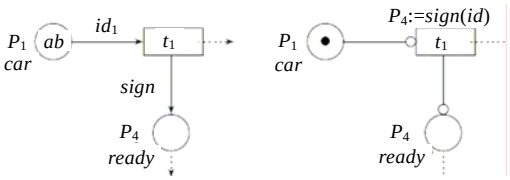


Fig.3 CPN arc function

图 3 有色网弧函数

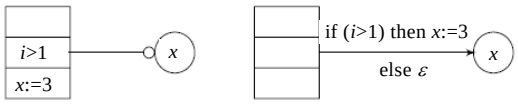


Fig.4 Arc expression

图 4 弧表达式

根据以上讨论,映射函数定义如下:

定义 2.1. 给定一个 CNets $\Sigma=(N,\mathbb{S},\mathbb{Z},M_0)$和 $N=(S_c,S_v,T;R,W,F)$,以及有色网 $\mathcal{M}=\langle P,T,Pre,Post,\mathcal{C},cd\rangle$.$\mathcal{A}$ 称为映射,如果:

- $P=S_c\cup S_v$;
- $T=T$;
- $\mathcal{C}=\mathbb{S}$;
- $cd:P\cup T\to\mathcal{C}$.

如果$\forall t\in T$,

$$Pre(p,t)=\begin{cases}M(s), & \text{如果}(s,t)\in R\\ M(s), & \text{如果}(t,s)\in W,\\ n, & \text{如果}(t,s)\in F\end{cases}\quad Post(p,t)=\begin{cases}M(s), & \text{如果}(s,t)\in R\\ L(s), & \text{如果}(t,s)\in W.\\ n, & \text{如果}(t,s)\in F\end{cases}$$

基于以上映射规则,一些有色网中的属性在 CNets 中仍然成立.为方便起见,令$\mathcal{Q}$是有色网,$\mathcal{R}$是 CNets.$\mathcal{Q}$是由$\mathcal{R}$经过映射得到的有色网.对于任意$\forall x\in\mathcal{R},x'$是 x 在$\mathcal{Q}$中的映射元素.反之亦然.

定理 2.1. 如果 $t'\in\mathcal{Q}$是可触发的,那么 $t\in\mathcal{R}$也是可触发的.

证明:如果 $t'\in\mathcal{Q}$是可触发的,那么有 3 个条件必须满足:

(1) t'的前集有足够的托肯;

(2) t'的后集有足够的容量;

(3) 弧表达式${}^\bullet t'\times\{t'\}$计算后的值必须成真.

在$\mathcal{R}$中,无论 t 是否触发都是依赖于 t 的前集中的托肯.令 p 是 t 的前集.p 可能包含控制库所 p_p、变量库所 p_v,或者两者都有.不失一般性,假设 p_p 和 p_v 分别是 t 的控制库所和变量库所.对于控制库所 p_p,其语义和有色网$\mathcal{Q}$中的 t'的前集语义是一致的.对于变量库所,如果是读弧$(p_v,t)\in R$,那么根据映射规则,读弧可以转化成两种弧:

- $(p_{v'},t')\in F$;
- $(t',p_{v'})\in F$.

因为 $t'\in\mathcal{Q}$是可触发的,因此 p_v 有足够的托肯和容量.

如果是写弧$(t,p_v)\in W$,那么根据映射规则,写弧可以转化成两种弧:

- $(p_{v'},t')\in F$;
- $(t',p_{v'})\in F$.

因为 $t'\in\mathcal{Q}$是可触发的,因此 p_v 有足够的托肯和容量.

在有色网$\mathcal{Q}$中,弧表达式${}^\bullet t'\times\{t'\}$可以正确地计算.在$\mathcal{R}$中,读写变量库所的托肯数表示的是表达式的值.根据映射规则,$\mathcal{Q}$中表达式${}^\bullet t'\times\{t'\}$中包含的逻辑条件被移除并置于 CNets 中 t 的变量哨位置.因此,t 触发的 3 个条件:

- 控制哨成真;

- 变量哨成真;
- 变量库所可以正确地计算,

都可以满足.因此,CNets 中的 t 也是可触发的. □

定理 2.2. 如果 $b'\in\mathcal{Q}$是可达的,那么 $b\in\mathcal{R}$也是可达的.

证明: b'是可达的,因 b'前集中的变迁是可触发的.令 b'的前集是 T',T'的前集是 P'.为证明 b 是可达的,b 的前集必须是可触发的.令 b 的前集是 T,T 的前集是 P.

对于$\forall p\in P$,如果 p 是控制库所,显然,t 与 $t'\in T'$语义相同,是可触发的.如果 p 是变量库所,有色网$\mathcal{Q}$中弧表达式(p',t')包含 t 的变量哨作为弧表达式的逻辑条件.由前提可知,弧表达式(p',t')可以正确地计算,因此,p 的变量哨可以成真,也就是 t 可以触发.因为 b 是 t 的后集,因此 b 是可达的. □

推论 2.1. 令 M_1' 是$\mathcal{Q}$系统的一个标识,M_1 是$\mathcal{R}$系统的一个标识.如果 M_1' 是可达的,那么 M_1 也是可达的.其中的出现序列是 $M_0[t_1\rangle\ldots t_i\rangle M_i[t_{i+1}\ldots t_n\rangle M_1$.

证明:根据定理 2.2,结论显然成立. □

定理 2.3. 如果$\mathcal{Q}$是活的,那么$\mathcal{R}$也是活的.

证明:因为$\mathcal{Q}$是活的,因此$\mathcal{Q}$中的任意变迁都是可以触发的.因此,$\mathcal{Q}$中任意标识也是可达的.根据定理 2.2 和推论 2.1,结论也是成立的. □

定理 2.4. 令 M'是$\mathcal{Q}$的一个标识.$\mathcal{R}$是有界的,当且仅当存在一个自然数边界 n,对于任意一个可达标识,都有 $M'\leqslant n$.

证明:如果 $M'\leqslant n,\forall p'\in M'$包含托肯,如果 $p\in\mathcal{R}$是一个控制库所,显然,p 中的托肯数与 p'中的托肯数是相同的.如果 p 是一个变量库所,那么根据变量库所的定义,变量库所中的标识为其描述变量的值.根据题设,有 $M'(p)\leqslant n_i$ 成立.因此对$\forall p\in\mathcal{R}$,总有 $M(p)\leqslant n_i$,从而 $M\leqslant\sum n_i$ 成立.可得$\mathcal{R}$是有界的.

如果$\mathcal{R}$是有界的,$\forall p\in\mathcal{R}$包含托肯,如果 p 是一个变量库所,那么 p 中的标识是 p 的值.因为$\mathcal{R}$是有界的,因此存在 n_i,$M(p)\leqslant n_i$,从而 $M'(p)\leqslant n_i$.如果 p 是一个控制库所,因为$\mathcal{R}$是有界的,因此存在一个自然数 n_i,同时 $M(p)\leqslant n_i$ 成立.因为扩展规则并不改变库所中托肯的数目,可知 $M'(p)\leqslant n_i$ 成立.因此$\forall p'\in\mathcal{Q}$,p'中的托肯数是和 p 中的托肯数或者相等或者小于 n_i.从而存在一个自然数 $n=\sum n_i$,$M\leqslant n$ 成立.所以,$M'\leqslant n$ 成立. □

3 CNets 规范

读出和写入是主流程序语言中最基本的特征,其根源来自于图灵模型的读写头机制,程序代码模型是变量(保存数据的机制)、操作、控制机制、资源约束等元素的混合体.因此,只要刻画了读写操作,其他复杂的操作可以根据一定的规则分解成读写两个最基本操作的组合.程序语言的其他扩展功能,如模块结构、循环、子函数(库函数)等,都是程序语言的一种封装机制,均可以依一定的技术分解成控制、操作、变量和资源相关语句的集合.在采用 CNets 描述程序时,把这些封装模块作为子图予以描述,其中,变迁中的哨(变量哨和控制哨)控制中止(出口)条件.

另外,本文采用 CNets 描述的目的之一就是尽量减少程序形式化过程中 CNets 规范中的状态数量,原因在于 CNets 中的变量描述并不是采用状态机制.因此,CNets 可以有效减少状态的引入数量.但是在 CNets 规范向 Petri 网规范转换过程中,可能会导致大量状态的引入,因而状态爆炸的可能性仍然存在.这也是本项目研究目标之一.为避免海量状态的 Petri 网规范的验证,应用较少状态的 CNets 规范的验证.

特别地,程序中不可避免地使用指针,而指针具有动态性和别名特点,很难进行静态跟踪.指针隐含着新代码起始点,并且是在动态运行中才能确定的.在传统的静态分析技术中,无法针对这种动态语义的特征进行描述,因此难以分析程序的动态行为.基于 Petri 网中资源和进程等观点认为,指针实际上就是平台资源的一种分

配.Petri 网的补库所可以方便刻画与正常流程互补的异常现象.由于篇幅所限,本例子只是描述了程序中最典型读写特征.我们仅仅讨论如何应用 CNets 来形式化程序代码,并略掉一些性质的证明细节.给定一个函数 *foo* 接收参数 *x* 和 *y* 并计算表达式 5*min(*x*,*y*)的值.该函数代码是:

```
int foo(int x,int y)
{
    int z;
    if x<y
        z:=x;
    else
        z:=y;
    z:=5*z;
    return z;
}
```

显然,函数 *foo* 包含有 3 个变量 *x*,*y* 和 *z*.令

$$S_v=\{x,y,z\}.$$

为简洁起见,我们忽略函数头、变量声明以及函数返回语句(return).代码有 3 条核心语句:

(1) if $x<y$ $z:=x$

(2) else $z:=y$

(3) $z:=5*z$

每条语句由一个变迁来描述.令变迁 t_{if},t_{else} 和 t_{min} 分别描述这 3 条语句,那么每个变迁中的 L 分别是:

- $L(t_{if})$:$z:=x$
- $L(t_{else})$:$z:=y$
- $L(t_{min})$:$z:=5*z$

在这 3 条语句中,语句 if $x<y$ $z:=x$ 和语句 else $z:=y$ 由 if-else 语句所包围.语句 if-else 中的布尔表达式是哨.

在变迁 t_{if} 的 L 中可以看出,存在两个变量:z 和 x.t_{if} 有两个哨:一个是控制哨,用来测试前集${}^\bullet t_{if}$ 是否有足够的托肯.当${}^\bullet t_{if}$ 有一个托肯时,控制哨成真,变迁是 *C_enable*,否则成假.控制哨决定了语句 if $x<y$ $z:=x$ 是否可以继续执行;另外一个是变量哨,用来测试判断语句 $x<y$ 是否成真.变量哨决定了语句 $z:=x$ 是否能够继续执行.因此,这两个哨分别是:

- $G_c(t_{if})=\{\text{true,false}\}$;
- $G_v(t_{if})=\{x<y\}$.

在 $G_v(t_{if})$中,变量 x 和 y 都是被读的.在 $L(t_{if})$中,变量 z 是被写的,而 y 是被读的.因此,变迁 t_{if} 涉及到 3 个变量:x(读),y(读)和 z(写).语句规范如图 5 所示.同理,变迁 t_{else} 可以形式化,如图 6 所示.

由于语句 if-else 具有互斥特性,也就是说语句 if $x<y$ $z:=x$ 和语句 else $z:=y$ 不能同时被执行.为描述这种互斥性,引入控制库 C_{ifelse},如图 7 所示.C_{ifelse} 包含一个托肯,表示同一时刻 $G_c(t_{if})$和 $G_c(t_{else})$中只有一个才能成真,同时只能执行一条语句.假设 $G_c(t_{if})$成真,在 t_{if} 触发,相应的托肯被消耗后,C_{ifelse} 失去其中的托肯,随之 $G_c(t_{if})$只能成假;反之亦然.一般来说,$G_c(t_{if})$根据 C_{ifelse} 是否可以获得足够的托肯,在取真值和假值之间变换.

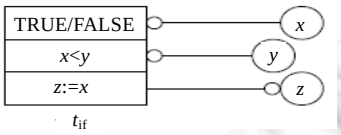


Fig.5 Transition t_{if}

图 5 变迁 t_{if}

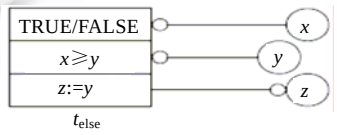


Fig.6 Transition t_{else}

图 6 变迁 t_{else}

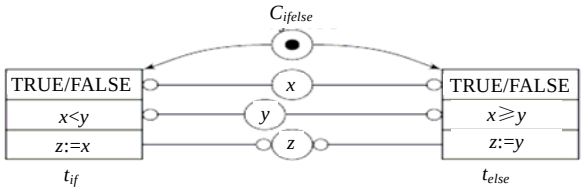


Fig.7 Exclusive transition

图 7 互斥变迁

在语句 z:=5*z 中,z 出现在赋值语句的左边和右边,意味着 z 既被读也会被写入.再者,语句 z:=5*z 只能在语句 if-else 完成之后执行.也就是说,t_{min} 是在 t_{if} 和 t_{else} 完成之后执行.令控制库所 C 决定这种执行顺序性.程序规范如图 8 所示.当 C 包含一个托肯时,t_{min} 将会触发运行,t_{min} 的变量哨不是必须的.

根据前面讨论的有色网映射规则,可获得相应的有色网规范(如图 9 所示).

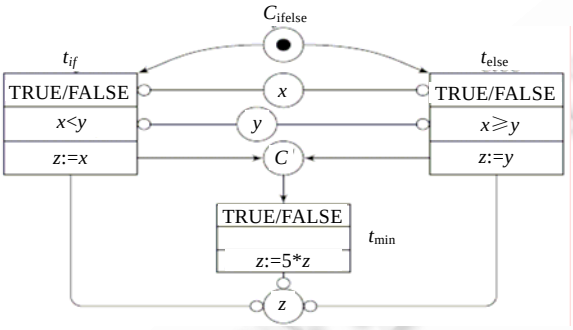


Fig.8 5*min(*x*,*y*) CNet specification

图 8 5*min(*x*,*y*) CNet 规范

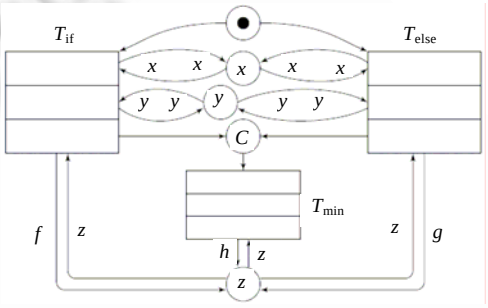


Fig.9 5*min(*x*,*y*)CPN specification

图 9 5*min(*x*,*y*)有色网规范

与图 9 对应的颜色定义为:

Colour sets:

place colours:

x=*integer*, *y*=*integer*, *z*=*integer*, *C*=*integer*, *C*_{*ifelse*}=*integer*;

transition colours:

$cd(t_{if})$=*integer*∪*BOOL*, $cd(t_{else})$=*integer*∪*BOOL*, $cd(t_{min})$=*integer*∪*BOOL*;

functions:

x *y* *z*

f: if *x*<*y* then *z*:=*x* else *z*:=ε

g: if *x*≥*y* then *z*:=*y* else *z*:=ε

h: *z*:=5*z

系统的前集和后集分别见表 1 和表 2.

Table 1 *Pre*

表 1 *Pre*

Preset	T_{if}	T_{else}	T_{min}
x	*x*	*x*	—
y	*y*	*y*	—
z	*z*	*z*	*z*
C_{ifelse}	1	1	—
C	—	—	1

Table 2 *Post*

表 2 *Post*

Postset	T_{if}	T_{else}	T_{min}
x	*x*	*x*	—
y	*y*	*y*	—
z	*f*	*g*	*h*
C_{ifelse}	—	—	—
C	1	1	1

CNets 中的变迁触发条件与有色网中的弧标识是一致的,并不发生改变.与数据相关的触发条件置于变迁中的变量哨中,而与传统有色网一致的触发条件则置于控制哨中.因此,CNets 并不改变有色网的语义.从而,CNets 所描述系统的活性、进展性、可达性等都与有色网一致.与有色网相比,CNets 有更灵活的机制获得控制流规范 cvNets 和数据流规范 dvNets.在 CNets 中,如果所有的变量库所、读弧、写弧都被略去不显式绘制,可以得出相应的程序 cvNets(如图 10 所示).显然,cvNets 描述了程序的控制流框架.事实上,图 10 是一个经典 P/T 网,可以重用 P/T 网分析技术.

同样地,如果所有的控制库所、控制弧和不含读写的变迁略掉,可以得到程序的 dvNets(如图 11 所示).因为 dvNets 不含有控制流,因此所有的变迁可以是并发的.dvNets 描述的是程序中的实体和数据流关系的框架.

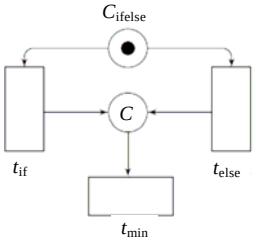


Fig.10 cvNets of 5*min(*x*,*y*)

图 10 cvNets of 5*min(*x*,*y*)

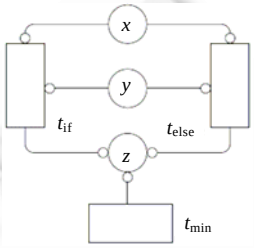


Fig.11 dvNets of 5*min(*x*,*y*)

图 11 dvNets of 5*min(*x*,*y*)

CNet 可以在程序设计的不同阶段提供 cvNets 模型和 dvNets 模型.当工程师开始设计一个程序时,可以先把控制流关系放在一边,把主要精力集中在数据流关系中去,即数据模型 dvNets.当确定了计算机平台时,可以基于该计算体系结构的控制限制来设计程序的控制流模型,即 cvNets.

4 相关工作

我们的研究目标是开发实用的工具,自动生成程序代码的 CNets,从而获取程序中隐含的控制和数据视图,进行分析、验证、模拟等静态分析.本项目应用 CNets 实现程序代码建模,描述程序中的变量、操作、资源以及读写操作,从而实现对程序中隐含的控制、数据以及控制与数据之间的关系分析.

本项目研究主要是采用 CNet 对程序代码进行建模,特别是研究实现 C 程序代码的自动化建模,并在程序模型的基础上分析模型的性质.分析方法从两个方向展开:一是重用经典 Petri 网的分析技术,因而提出 CNets 向经典 Petri 网映射的规则;另一方面是提出新的 CNets 程序缺陷分析技术.本文提出的 CNets 建模与映射规则,解决了程序代码的图形化描述、隐含的控制和数据关系视图,实现程序代码向经典 Petri 网的转换,从而重用经典 Petri 网理论,达到验证和分析的目标.

另外,正在研发的程序代码分析工具 CCNeter 采用变量库所描述变量,操作分解成读/写元操作,通过变迁来描述操作,其他系统中的状态或控制资源机制等通过控制库所来描述.CCNeter 目前暂时选择基于 C99 标准语言的程序.CCNeter 首先对 C99 标准的 C 程序代码进行词法分析,形成变量库所表、控制库所表、变迁表以及它们之间关系,这部分工作将来可以应用到编译阶段进行静态分析.在图形元素的自动布局方面,采用开源工具 graphviz[45]实现.在获得可视化的程序模型后,通过控制流和数据流结合的分析技术对程序的缺陷进行分析.

References:

[1] Mei H, Wang QX, Zhang L, Wang J. Software analysis: A road map. Chinese Journal of Computers, 2009,32(9):1697−1710 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2009.01697]

[2] Weber S, Karger PA, Paradkar A. A software flaw taxonomy: Aiming tools at security. SIGSOFT Software Engineering Notes, 2005,30(4):1−7. [doi: 10.1145/1083200.1083209]

[3] Clarke L. A system to generate test data and symbolically execute programs. IEEE Trans. on Software Engineering, 1976,2(3): 215−222. [doi: 10.1109/TSE.1976.233817]

[4] Hangal S, Lam MS. Tracking down software bugs using automatic anomaly detection. In: Proc. of the 24th Int'l Conf. on Software Engineering. Orlando: ACM, 2002. 291−301. [doi: 10.1145/581339.581377]

[5] Sekar R, Bendre M, Dhurjati D, Bollineni P. A fast automaton-based method for detecting anomalous program behaviors. In: Proc. of the 2001 IEEE Symp. on Security and Privacy. IEEE Computer Society, 2001. 144−155. [doi: 10.1109/SECPRI.2001.924295]

[6] David B. Source code analysis:A road map. In: Proc. of the Future of Software Engineering. 2007. 104−119. [doi: 10.1109/FOSE.2007.27]

[7] SCAN. Available from: http://scan.coverity.com/

[8] Xu G, Rountev A. Precise memory leak detection for Java software using container profiling. In: Proc. of the 30th Int'l Conf. on Software Engineering. 2008. 151−160. [doi: 10.1145/1368088.1368110]

[9] Ganapathy V, Jha S, Chandler D, Melski D, Vitek D. Buffer overrun detection using linear programming and static analysis. In: Proc. of the 10th ACM Conf. on Computer and Communications Security. Washington: ACM, 2003. 345−354. [doi: 10.1145/948109.948155]

[10] Xie Y, Aiken A. Context- and Path-Sensitive memory leak detection. SIGSOFT Software Engineering Notes, 2005,30(5):115−125. [doi: 10.1145/1081706.1081728]

[11] Li J, Wang L. Method for precisely detecting buffer overflow vulnerabilities in C programs. Journal of Beijing University of Aeronautics and Astronautics, 2008,34(3):319−322 (in Chinese with English abstract).

[12] Bjorner D, Jones CB, Airchinnigh MMA, Neuhold EJ. Vdm '87 Vdm—A formal method at work. Vol.252. Springer- Verlag, 1987.

[13] Girault C, Valk R. Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer-Verlag, 2002.

[14] Harel D. Statecharts: A Visual Formalism for Complex Systems. 1987. 231−274. [doi: 10.1016/0167-6423(87)90035-9]

[15] Hoare CAR, He J. Unifying Theories of Programming. Prentice Hall, 1998.

[16] Taguchi K, Araki K. The state-based CCS semantics for concurrent Z specification. In: Proc. of the Int'l Conf. on Formal Engineering Methods. 1997. 283−292. [doi: 10.1109/ICFEM.1997.630435]

[17] Woodcock JCP, Davies J. Using Z-Specification, Refinement, and Proof. Prentice-Hall, 1996.

[18] Cornelissen B, Graaf B, Moonen L. Identification of variation points using dynamic analysis. In: Proc. of the 1st Int'l Workshop on Reengineering Towards Product Lines (R2PL). 2005.

[19] Code. http://www.cs.cmu.edu/~fox/pcc.html

[20] Hoare T. The verifying compiler: A grand challenge for computing research. Journal of the ACM, 2003,50(1):63−69. [doi: 10.1145/602382.602403]

[21] Tevis JEJ, Hamilton JJA. Static analysis of anomalies and security vulnerabilities in executable files. In: Proc. of the 44th Annual Southeast Regional Conf. Melbourne, ACM, 2006. 560−565. [doi: 10.1145/1185448.1185570]

[22] Kruegel C, Robertson W, Vigna G. Detecting kernel-level rootkits through binary analysis. In: Proc. of the 20th Annual Computer Security Applications Conf. IEEE Computer Society, 2004. 91−100. [doi: 10.1109/CSAC.2004.19]

[23] Blanchet B, Cousot P, Cousot R, Feret JO. A static analyzer for large safety-critical software. In: Proc. of the SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). 2003. 196−207. [doi: 10.1145/781131.781153]

[24] Chess B, McGraw G. Static analysis for security. IEEE Security and Privacy, 2004,2(6):76−79. [doi: 10.1109/MSP.2004.111]

[25] Livshits B. Improving Software Security with Precise Static and Runtime Analysis [Ph.D Thesis]. Stanford University, 2006.

[26] Voung JW, Jhala R, Lerner S. Relay: Static race detection on millions of lines of code. In: Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. Foundations of Software Engineering 2007. ACM Press, 2007. 205−214. [doi: 10.1145/1287624.1287654]

[27] Zhou G, He G. One program model for cloud computing. In: Proc. of the 1st Int'l Conf. on Cloud Computing. Lecture Notes in Computer Science, 2009. 589−594. [doi: 10.1007/978-3-642-10665-1_57]

[28] Myers AC. Jflow: Practical mostly-static information flow control. In: Proc. of the 26th ACM Symp. on Principles of Programming Languages (POPL). 1999. 228−241. [doi: 10.1145/292540.292561]

[29] Nir-Buchbinder Y, Tzoref R, Ur S. Deadlocks: From exhibiting to healing. In: Proc. of the Runtime Verification. Lecture Notes in Computer Science, Heidelberg, 2008. 104−118. [doi: 10.1007/978-3-540-89247-2_7]

[30] Li BX, Zheng GL, Wang YF, Li XD. An approach to analyzing and understanding program—Program slicing. Journal of Computer Reseach and Development, 2000,37(3):284−291 (in Chinese with English abstract).

[31] Ngo MN, Tan HBK. Detecting large number of infeasible paths through recognizing their patterns. In: Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. Dubrovnik: ACM, 2007. 215−224. [doi: 10.1145/1287624.1287655]

[32] Gao D, Reiter MK, Song D. Gray-Box extraction of execution graphs for anomaly detection. In: Proc. of the 11th ACM Conf. on Computer and Communications Security. Washington: ACM, 2004. 318−329. [doi: 10.1145/1030083.1030126]

[33] Wang Y, Mahlke S, Lafortune S, Kelly T, Kudlur M. The theory of deadlock avoidance via discrete control. In: Proc. of the POPL 2009. 2009. 252−263. [doi: 10.1145/1480881.1480913]

[34] Kharitonov DI, Tarasov GV. Towards Petri nets application in parallel program debugging. In: Proc. of the 6th Asian Computaitional Fluid Dynamics Conf. 2005. 1–3.

[35] Naik M, Park CS, Sen K, Gay D. Effective static deadlock detection. In: Proc. of the the 31st Int'l Conf. on Software Engineering. Vancouver, 2009. 386–396. [doi: 10.1109/ICSE.2009.5070538]

[36] Livshits VB, Lam MS. Tracking pointers with path and context sensitivity for bug detection in C programs. SIGSOFT Software Engineering Notes, 2003,28(5):317–326. [doi: 10.1145/940071.940114]

[37] Yin H, Song D, Egele M, Kruegel C, Kirda E. Panorama: Capturing system-wide information flow for malware detection and analysis. In: Proc. of the 14th ACM Conf. on Computer and Communications Security (2007). 2007. 116–127. [doi: 10.1145/1315245.1315261]

[38] Khedker U, Sanyal A, Karkare B. Data Flow Analysis: Theory and Practice. CRC Press (Taylor and Francis Group), 2009.

[39] Kildall G. A unified approach to global program optimization. In: Proc. of the 1st Annual ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages. Boston: Massachusetts ACM, 1973. 194–206. [doi: 10.1145/512927.512945]

[40] Ayewah N, Hovemeyer D, Morgenthaler JD, Penix J, Pugh W. Using static analysis to find bugs. IEEE Software, 2008,25(5):22–29. [doi: 10.1109/MS.2008.130]

[41] Castro M, Costa M, Harris T. Securing software by enforcing data-flow integrity. In: Proc. of the 7th Symp. on Operating Systems Design and Implementation. Washington: USENIX Association, 2006. 147–160.

[42] Mohnen M. A graph-free approach to data-flow analysis. Lecture Notes in Computer Science, 2002,2304:185–213. [doi: 10.1007/3-540-45937-5_6]

[43] Zhou G, Yuan C. Mapping punity to uninet. Journal of Computer Science and Technology, 2003,18(3):378–387. [doi: 10.1007/BF02948908]

[44] Yuan CY. Petrinets Theory. Beijing: Publishing House of Electronics Industry, 2003 (in Chinese).

[45] Graphviz. http://www.graphviz.org

附中文参考文献:

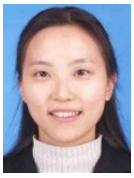
[1] 梅宏,王千祥,张路,王戟.软件分析技术进展.计算机学报,2009,32(9):1697–1710. [doi: 10.3724/SP.J.1016.2009.01697]

[11] 李吉,王雷.C 程序缓冲区溢出漏洞精确检测方法.北京航空航天大学学报,2008,34(3):319–322.

[30] 李必信,郑国梁,王云峰,李宣东.一种分析和理解程序的方法——程序切片.计算机研究与发展,2000,37(3):284–291.

[44] 袁崇义.Petri 网理论.北京:电子工业出版社,2003.

周国富(1970－),男,湖北大冶人,博士,副教授,CCF 会员,主要研究领域为 Petri 网,程序语义,形式化方法,软件工程.



杜卓敏(1975－),女,博士,副教授,CCF 会员,主要研究领域为并行计算,分布式 Agent,可信编译,软件工程及程序语义.