计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(10):114−119 [doi: 10.15888/j.cnki.csa.007659]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于动态分析的程序设计课程教学系统①

陈　哲[1,2], 王　冲[1,2], 黄志球[1,2]

[1](南京航空航天大学 计算机科学与技术学院, 南京 211106)
[2](高安全系统的软件开发与验证技术工业和信息化部重点实验室, 南京 211106)
通讯作者: 陈　哲, E-mail: zhechen@nuaa.edu.cn

摘　要: 程序设计是计算机专业的第一门核心必修专业课程, 但是在教学实践中, 学生很难掌握程序设计语言中一些复杂的或抽象的理论知识. 为了提高教学效果, 针对程序设计课程教学的难点, 本文设计和实现了面向程序设计课程的教学系统: 程序动态分析系统. 该系统通过综合运用程序设计等专业课程的知识, 实现了程序错误检测和源代码自动插桩, 同时可以展现这些知识之间的深度融合. 我们将该系统应用于程序设计教学实践, 有助于学生理解和掌握程序设计课程的难点, 以及这些知识在实际软件开发过程中的应用, 从而有效提高教学效果.

关键词: 程序设计; 教学系统; 动态分析; 教学实践; 教学效果提升

引用格式: 陈哲,王冲,黄志球.基于动态分析的程序设计课程教学系统.计算机系统应用,2020,29(10):114–119. http://www.c-s-a.org.cn/1003-3254/7659.html

Teaching System for Course of Programming Languages Based on Dynamic Analysis

CHEN Zhe[1,2], WANG Chong[1,2], HUANG Zhi-Qiu[1,2]

[1](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)
[2](Key Laboratory of Safety-Critical Software, Ministry of Industry and Information Technology, Nanjing 211106, China)

Abstract: The course of programming languages is usually the first core course for the majority of computer science. However, in the teaching practice, it is hard for students to master some complex and abstract knowledge in programming languages. To overcome the difficulties in teaching the course of programming languages, in this study, we design and implement a teaching system for the course of programming languages—a program dynamic analysis system. This system, by applying the knowledge of programming languages and some other core professional courses, implements program error detection and automated source instrumentation. We apply this system to the teaching practice, which helps students to understand and master the complex and abstract concepts in the course of programming languages, and their applications in real-world software development, thus to improve the teaching effectiveness.

Key words: programming languages; teaching systems; dynamic analysis; teaching practice; teaching effectiveness

程序设计是计算机、软件工程等工科专业的第一门核心必修专业课程, 其教学内容通常包括 C 程序设计语言[1]、面向对象程序设计语言 (例如 C++)[2]. 由于后续专业课程 (例如数据结构、编译原理、软件工程、软件测试等) 都需要使用程序设计来完成实验和课程设计, 而且程序设计也是计算机和软件行业招聘面试的核心考查内容, 所以熟练地掌握程序设计技能不仅是学习后续专业课程的基础, 也是获得理想工作机会的重要条件. 因此, 如何改进程序设计课程教学、让学生更好地掌握课程知识点是重要的研究课题[3–7].

① 基金项目: 国家自然科学基金 (U1533130); 高安全系统的软件开发与验证技术工业和信息化部重点实验室开放基金 (XCA18164-02)
Foundation item: National Natural Science Foundation of China (U1533130); Open Fund of Key Laboratory of Safety-Critical Software, Ministry of Industry and Information Technology (XCA18164-02)
收稿时间: 2020-03-20; 修改时间: 2020-04-21; 采用时间: 2020-05-10; csa 在线出版时间: 2020-09-30

笔者在教学实践中发现, 虽然采用了网络教学组织[3,4] 和在线判题[5−7] 教学系统, 学生还是很难掌握程序设计语言中一些复杂的或抽象的理论知识, 也很难将这些知识正确地运用到编程实践中. 以 C 程序设计教学为例, 指针和内存操作较为复杂[1]. 程序内存被划分为代码段、数据段、BSS 段、栈、堆等不同的段, 而每个段都有不同的访问规则[8], 例如: 代码段中的数据只能读不能写, 栈中的数据不能被显式释放, 堆中分配的内存块被释放后不能再次访问, 对内存块的访问不能超过该内存块的上界和下界等. 所以照本宣科式地讲授相关的理论知识, 学生难以消化吸收, 更无法正确运用, 使得编写的程序经常带有内存错误, 导致错误的运行结果和耗时的调试过程. 事实上, 在工业界的软件开发实践中, 内存错误也一直是最常见、危害最大的程序错误之一, 它可以导致数据腐败、安全漏洞和程序崩溃等致命后果, 例如阿丽亚娜 5 号火箭控制系统软件的内存溢出错误导致火箭在升空后爆炸[9]. 因此, 教师有必要使用教学系统帮助学生牢固掌握指针和内存操作的相关知识、更快地找到程序中的内存错误, 从而正确运用指针和内存操作.

以面向对象程序设计教学为例, 封装、继承、多态是面向对象的三大特性[2], 但是这些概念较为抽象. 注意, C 程序设计教学关注的是程序具体功能相关的知识点, 如循环语句、函数等. 与之不同, 面向对象程序设计教学关注的是非功能的知识点, 例如封装、继承、多态的主要目标是改善大规模代码的模块化、可维护性、可复用性、健壮性等. 所以咬文嚼字式地讲授相关的概念定义, 或者通过小例子辅助教学, 学生都难以理解这些特性的优点, 更无法主动地在编程实践中合理地运用, 使得编写的程序经常不具备面向对象的特征, 导致程序模块性差、难于维护. 因此, 教师有必要使用教学系统帮助学生深入理解面向对象的特性, 从而正确运用面向对象程序设计思想.

基于上述教学需求, 本文设计并使用 C++程序设计语言实现了面向程序设计课程的教学系统: C 程序内存安全性动态分析系统 (以下称为 Movec). 该系统包含了 C 程序编译器前端, 通过递归式遍历和重写程序源代码的抽象语法树 (AST), 实现对程序源代码的自动插桩; 通过编译运行插桩后的源代码, 实现对内存错误的动态检测. 该系统综合运用了 C 程序设计、面向对象程序设计 (C++)、数据结构、编译原理、软件工程、软件测试等计算机和软件工程专业核心课程的知识, 同时可以展现这些知识之间的深度融合. 学生通过学习和使用该系统有助于理解和掌握相关的理论知识, 以及这些理论知识在实际软件开发过程中的应用, 可以激发学生学习的主动性和钻研的兴趣, 从而有效地提高教学效果.

1 总体规划

1.1 基本技术

程序分析技术是一种对程序行为进行自动分析的过程, 通常用于发现程序中的缺陷和错误, 例如内存安全性缺陷、死循环等逻辑错误.

程序分析技术分为静态和动态两种. 静态分析不需要运行程序, 而是通过对程序代码进行逻辑推理来检测错误. 由于静态分析需要确保性能 (在短时间内报告结果), 而这是以检测精度为代价的, 因此通常有误报和漏报. 动态分析通过代码插桩技术在代码中插入用于错误检测的代码片段, 从而在运行时实现对程序执行过程的分析和错误检测[10]. 动态分析的优点是考虑了程序实际运行环境等因素, 理论上不存在误报. 因此本系统采用基于动态分析的方法.

动态分析所采用的代码插桩技术通常对程序的中间表示[11] 或二进制代码插桩[12], 导致无法报告错误在源代码中的准确位置, 给错误调试带来障碍. 因此本系统采用源代码插桩技术[13], 即直接对源代码进行重写, 在检测到错误时, 可以报告错误所在的源文件名和行号等.

1.2 系统结构

图 1 展示了 Movec 系统的总体结构, 主要由 C 程序解析器和递归式 AST 访问者组成. 对于用户给定的 C 程序, Movec 首先使用 C 程序解析器将程序源代码解析为 AST, 然后通过递归式遍历和访问 AST 上的每个节点, 分析程序语义, 并插入相应的用于错误检测的代码片段, 生成插桩后的源代码和接口文件 (包括实现错误检测算法的数据结构和接口函数). 用户通过编译插桩后的源代码, 得到插桩后的可执行程序. 最后用户运行该程序, 实现对内存错误的动态检测.

2 错误检测与自动插桩算法

内存错误动态检测的基本思想是为每个指针变量分配一个指针元数据 (Pointer Meta Data, PMD), 用于

记录指针变量所指向的内存块的下界、上界和状态信息, 并在程序通过指针变量访问内存时检测指针元数据所记录的信息是否允许这次访问. 下面我们通过实例介绍错误检测算法的基本思想.

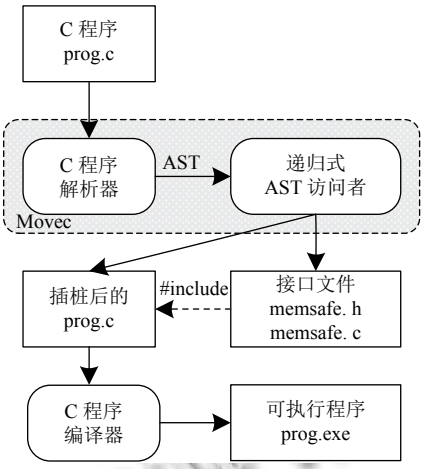

图 1 系统结构图

实例 1. 下列 C 程序首先将指针 p 初始化为指向变量 i, 然后对指针 p+5 进行解引用和赋值. 显然, 由于 p+5 超出了变量 i 所在内存块的范围, 因此会发生内存溢出错误.

```
int i=1, *p=&i;
*(p+5)=1;
```

我们的检测算法可以发现该错误. 在程序初始化指针 p 之后, 检测算法首先为指针 p 分配一个指针元数据 (如图 2 所示), 用于记录 p 所指向的内存块的下界&i、上界&i+1 和状态 stack, 其中 stack 表示该内存块 (变量 i) 是被存储在栈中的有效变量 (即没有被释放), 然后将该指针元数据插入指针元数据表, 并使用 p 的地址&p 作为该元数据的索引值, 用于加快查找指针元数据的速度.

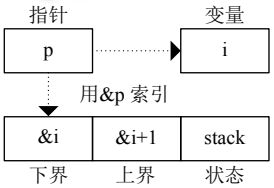

图 2 实例 1 中指针 p 的指针元数据

在程序对指针 p+5 进行解引用访问之前, 检测算法首先使用 p 的地址&p 作为索引值在指针元数据表中查找 p 的指针元数据, 然后检测程序即将访问的内存块范围是否在指针元数据所记录的下界和上界之间. 由于程序即将访问的内存块范围是从&i+5 到&i+6, 超出了元数据所记录的从&i 到&i+1 的范围, 因此存在内存溢出错误. 此时检测算法将报告内存错误及其所在的源文件名、行号、列号和导致错误的内存访问表达式“*(p+5)”.

实例 2. 下列 C 程序首先在函数 foo 中将指针 n 初始化为指向函数 foo 中的局部变量 x, 然后在退出函数 foo 后对指针 n 进行解引用和赋值. 显然, 由于 n 所指向的变量 x 此时是无效变量 (即已经被释放), 因此会发生内存释放后访问的错误.

```
void foo(int **pa)        void main() {
{                             int *n;
  int x;                     foo(&n);
  *pa=&x;                    *n=1;
}                           }
```

我们的检测算法可以发现该错误. 在程序初始化指针 n 之后, 检测算法首先为指针 n 分配一个指针元数据 (如图 3(a) 所示), 用于记录 n 所指向的内存块的下界&x、上界&x+1 和状态 stack, 然后将该指针元数据插入指针元数据表, 并使用 n 的地址&n 作为该元数据的索引值.

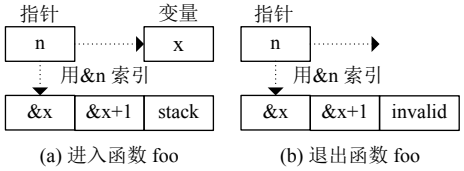

图 3 实例 2 中指针 n 的指针元数据及其变化

在程序退出函数 foo 之前, 程序释放函数 foo 中所有的局部变量, 检测算法将 n 的指针元数据中的状态改为 invalid, 表示 n 所指向的内存块 (变量 x) 已经被释放 (如图 3(b) 所示).

在程序对指针 n 进行解引用访问之前, 检测算法首先使用 n 的地址&n 作为索引值在指针元数据表中查找 n 的指针元数据, 然后检测指针元数据所记录的状态是否不是 invalid. 由于此时元数据所记录的状态是 invalid, 因此存在内存释放后访问的错误. 此时检测算法将报告内存错误及其所在的源文件名、行号、列号和导致错误的内存访问表达式“*n”.

我们通过以下 3 个接口函数来实现以上错误检测算法的主要功能:

1) 接口函数 pmd_tbl_update_as 用于更新给定指针变量 p 的指针元数据, 将&p 所索引的指针元数据的内容设置为给定的下界、上界和状态信息.

2) 接口函数 pmd_tbl_lookup 用于获取给定指针变量 p 的指针元数据, 返回&p 所索引的指针元数据的地址.

3) 接口函数 check_dpv 用于检测要访问的内存块范围是否在指针元数据所记录的下界和上界之间、状态是否有效.

接下来我们需要实现对程序源代码的自动插桩, 使得程序能在运行时调用以上接口实现对内存错误的动态检测. 算法 1 给出了自动插桩的基本思想: 通过递归式遍历和访问程序源代码 AST 上的每个节点, 在合适的位置插入对以上接口的调用. 例如, 在指针变量定义之后插入对其 pmd 的初始化操作, 在指针变量赋值语句之后插入对其 pmd 的更新操作, 在指针解引用表达式之前插入对比其 pmd 和实际访问的内存块范围的检测操作.

算法 1. 内存错误动态检测的自动插桩算法

遍历程序源代码 AST, 对于每个访问到的节点:

1) 如果该节点是指针变量定义, 例如“T *p;”或带有空初始值“T *p=NULL;”, 则在该语句后插入接口调用:

pmd_tbl_update_as(&p, NULL, NULL, NULL);

2) 如果该节点是从指针常量到指针变量的赋值语句, 例如“p=&i;”, 则在该语句后插入接口调用:

pmd_tbl_update_as(&p, &i, &i+1, i_status);

3) 如果该节点是从指针变量到指针变量的赋值语句, 例如“p1=p;”, “p1=p+I; ”或“p1=&p[i];”, 则在该语句后插入接口调用:

pmd=pmd_tbl_lookup(&p);

pmd_tbl_update_as(&p1, pmd->base, pmd->bound, pmd->status);

4) 如果该节点包含指针解引用表达式, 例如“i=*p1”或“*p1=i”, 则在该语句前插入接口调用:

check_dpv(pmd_tbl_lookup(&p1), p1, sizeof(*p1));

然后递归访问该节点的子节点.

3 系统开发

Movec 系统的开发包括两部分: 使用 C 程序设计语言实现错误检测算法, 使用 C++程序设计语言实现源代码自动插桩算法. 为了用户可以在不同的平台上使用 Movec, 我们充分利用 C/C++的跨平台特性, 采用基于 CMake 编译系统的跨平台架构, 使得 Movec 可以在 Linux 和 Windows 上运行.

3.1 错误检测函数库的实现

由于 C 程序设计语言并不自带容器类数据结构, 因此我们从零开始实现了一个高效的指针元数据表. 如图 4 所示, 指针变量 ptr 的指针元数据包括该指针变量的地址&ptr、所指向内存块的下界 base、上界 bound 和状态信息节点的地址&st. 由于多个指针 (例如 ptr1 和 ptr2) 可能指向同一个内存块, 因此内存块的状态信息节点必须是一个独立的节点, 这样才可能被多个指针元数据引用. 整个指针元数据表是使用指针变量地址为索引值的哈希表. 图 5 展示了指针元数据结构定义的 UML 图, 其中指针变量的地址是指向指针的指针类型, 下界和上界是指针类型, 状态信息节点的地址是指向状态信息节点的指针类型. 在状态信息节点中, 我们使用无符号整数表示内存块的当前状态, 例如 invalid、stack、heap 等.

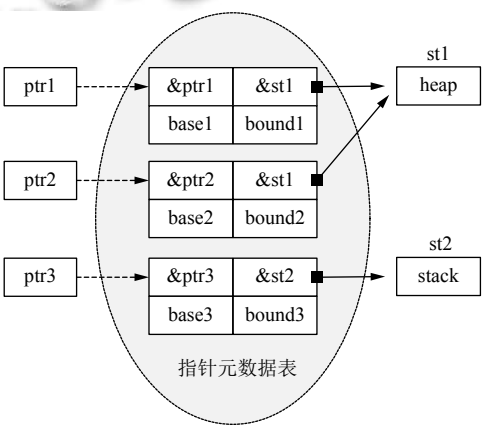

图 4 指针元数据表

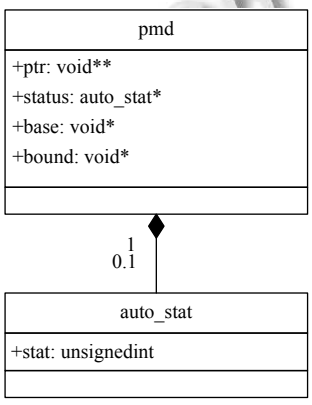

图 5 指针元数据

基于指针元数据表, 实现错误检测算法的 3 个接口函数是较为直接的. 例如, 接口函数 pmd_tbl_update_as 先在指针元数据表中查找给定指针变量 p 的指针元数据, 如果未找到则插入一个新的指针元数据, 然后对该元数据中的成员变量进行赋值.

3.2 源代码自动插桩的实现

自动插桩实现的关键在于递归式 AST 访问者的

实现. 为了展示面向对象程序设计的特性, 在实现中融合了封装、继承和多态. 首先, 设计了一个基本的 AST 访问者类 ASTVisitorBase, 包含对程序源代码 AST 上各种节点进行访问的成员虚函数 VisitX, 然后实现了 Traverse 成员函数, 在对 AST 进行递归式遍历的同时调用 VisitX 函数来对 AST 上的每个节点进行访问, 其中 X 可以是任意类型的节点, 例如访问声明 Decl 的函数 VisitDecl 和访问语句 Stmt 的函数 VisitStmt 等. 然后, 设计了一个实现插桩的 AST 访问者类 ASTVisitor, 继承于 ASTVisitorBase, 并对其中的成员虚函数 VisitX 进行虚函数重载, 实现源代码插桩功能.

4 实施方案

Movec 的使用非常简便. 用户只需要执行 3 个命令即可得到检测结果: 运行 Movec 对源代码进行自动插桩; 运行编译器将插桩后的源代码编译为可执行程序; 运行插桩后的可执行程序得到检测结果. 例如, 图 6 和图 7 分别展示了对第 2 节中的实例 1 和实例 2 进行检测的过程. 结果显示, 实例 1 中对指针 p+5 的解引用导致了空间错误 (即内存溢出错误), 实例 2 中对指针 n 的解引用导致了时间错误 (即内存释放后访问). 值得注意的是, 检测结果也报告了错误在源代码中的精确位置, 包括源文件名和行号等.

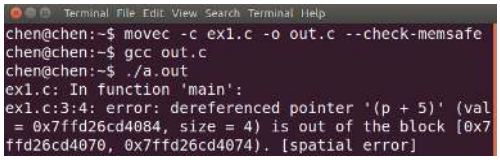

图 6 实例 1 的检测结果

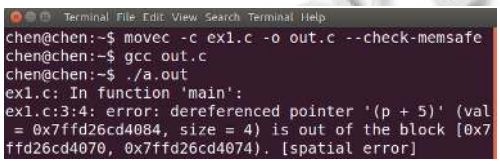

图 7 实例 2 的检测结果

Movec 可以被用于 C 程序设计语言的教学, 有效提高教学效果. 首先, 学生可以对自己编写的程序运行 Movec, 更快地找到程序中的内存错误, 极大地缩短调试时间, 提高学习效率. 通过长期使用 Movec, 学生能养成正确运用指针和内存操作的习惯, 提高编程能力. 第二, Movec 的错误检测函数库可以作为 C 语言的演示系统, 帮助学生掌握 C 语言中一些复杂的理论知识. 函数库的实现不仅大量使用了基本的条件语句、循环语句、数组、函数等, 还覆盖了较为复杂的指针和内存操作、结构体和链表等知识. 例如, 指针元数据采用结构体实现, 并包含了指向状态信息节点的指针; 指针元数据表采用基于动态内存分配的哈希表实现, 并采用链表来处理哈希冲突. 这可以帮助学生深入理解这些理论知识, 并掌握如何将这些知识正确地运用到编程实践中.

Movec 可以被用于面向对象程序设计语言 (C++) 的教学, 有效提高教学效果. Movec 的源代码自动插桩模块可以作为 C++语言的演示系统, 帮助学生掌握 C++语言中一些抽象的理论知识. 插桩模块的实现覆盖了重要但抽象的封装、继承和多态等知识. 例如, AST 中所有的语法节点类型都采用了封装的类来实现, 并且采用继承来描述相关节点类型之间的联系; 递归式 AST 访问者的实现也采用了封装、继承和多态, 即继承于一个基本的 AST 访问者类, 并对其中的 AST 节点访问函数进行虚函数重载; 通过这些设计改善了大规模代码的模块化、可维护性、可复用性和健壮性. 这可以帮助学生深入理解这些面向对象特性的优点, 并主动地将这些知识合理地运用到编程实践中.

Movec 也可以作为数据结构、编译原理、软件工程、软件测试等课程的案例分析材料, 全面贯穿计算机、软件工程等专业的培养计划.

5 结论

程序设计是计算机、软件工程等工科专业的第一门核心必修专业课程. 熟练地掌握程序设计技能不仅是学习后续专业课程的基础, 也是获得理想工作机会的重要条件. 笔者在教学实践中发现, 已有的教学系统只支持网络教学组织[3,4] 和在线判题[5–7], 并不能帮助学生掌握课程难点. 为此, 我们设计和实现了面向程序设计课程的教学系统: C 程序内存安全性动态分析系统. 该系统综合运用了 C 程序设计、面向对象程序设计 (C++)、数据结构、编译原理、软件工程、软件测试等计算机和软件工程专业核心课程的知识, 同时可以展现这些知识之间的深度融合. 从教学效果来看, 该系统不仅可以帮助学生理解和掌握相关的理论知识, 以及这些理论知识在实际软件开发过程中的应用, 还可以激发学生学习的主动性和钻研的兴趣, 从而有效地提高教学效果.

参考文献

1 Kernighan BW, Ritchie DM. The C Programming Language. 2nd ed. Englewood Cliffs: Prentice Hall, 1988.

2 Lippman SB, Lajoie J, Moo BE. C++ Primer. 5th ed. Upper Saddle River: Addison-Wesley Professional, 2013.

3 叶冬芬, 杨明霞, 方智敏. 基于 B/S 的《C 程序设计》网络教学系统. 计算机系统应用, 2016, 25(4): 56–62.

4 罗荣良, 吴明晖. 基于“MOOC+实验辅助平台”的面向对象程序设计教学实践. 计算机教育, 2020, (2): 170–174. [doi: 10.3969/j.issn.1672-5913.2020.02.041]

5 王金鹏, 曹旗磊, 王涵. 基于 Online Judge 的程序设计基础教学改革与实践. 计算机教育, 2020, (2): 101–104. [doi: 10.3969/j.issn.1672-5913.2020.02.025]

6 李旻朔, 林巧. 多核平台上程序在线评测辅助教学系统. 计算机系统应用, 2011, 20(6): 129–132. [doi: 10.3969/j.issn.1003-3254.2011.06.030]

7 韩志科, 王贵, 韩俊杰. 基于 API 自动测试的程序设计在线判题系统的研究与实现. 计算机系统应用, 2008, 17(7): 9–13. [doi: 10.3969/j.issn.1003-3254.2008.07.003]

8 Chen Z, Tao CQ, Zhang ZY, *et al*. Beyond spatial and temporal memory safety. Proceedings of ACM/IEEE 40th International Conference on Software Engineering. Gothenburg, Sweden. 2018. 189–190.

9 Chen Z, Gu Y, Huang ZQ, *et al*. Model checking aircraft controller software: A case study. Software: Practice and Experience, 2015, 45(7): 989–1017. [doi: 10.1002/spe.2242]

10 Chen Z, Wang ZM, Zhu YL, *et al*. Parametric runtime verification of C programs. Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Eindhoven, the Netherlands. 2016. 299–315.

11 Serebryany K, Bruening D, Potapenko A, *et al*. AddressSanitizer: A fast address sanity checker. Proceedings of the 2012 USENIX Annual Technical Conference. Berkeley, CA, USA. 2012. 28.

12 Nethercote N, Seward J. Valgrind: A framework for heavyweight dynamic binary instrumentation. Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA. 2007. 89–100.

13 Chen Z, Yan JQ, Li WM, *et al*. Runtime verification of memory safety via source transformation. Proceedings of ACM/IEEE 40th International Conference on Software Engineering: Companion. Gothenburg, Sweden. 2018. 264–265.