

任务安全关键软件构造时在线监控方法研究

王犇, 丁成钧, 林伟, 马春燕

(西北工业大学 软件学院, 陕西 西安 710072)

摘要: C语言因其灵活性和高效率在航空航天等多个任务关键领域得到广泛应用。然而, C语言程序存在安全风险, 如指针操作不严格限定、数组和字符串缺乏边界检查等, 容易引发潜在的运行时故障, 造成不可挽回的损失。针对此类问题, 提出一种任务安全关键软件 C 程序构造时的在线监控方法, 在构造 C 程序时对代码进行实时监控和静态分析, 高效检测潜在故障。针对在线编辑的 C 程序片段的实时编译及测试问题, 提出了一种混合式监控方法的片段程序可编译版本自动生成技术。针对任务安全关键软件 5 类运行时故障的产生条件归纳出 43 种故障类型, 基于抽象语法树建立在线编辑的 C 程序片段故障的规则库。提出了基于语法结构匹配算法, 实现在线编辑的 C 程序片段故障监控。实验选择 50 个安全关键软件常用的 C 程序代码进行验证, 共计匹配到 41 种、146 个潜在运行时故障, 结果表明, 文中监控方法能够有效识别潜在故障, 提高软件安全性和可信性。

关键词: 在线监控; 故障检测; 抽象语法树; 自动化测试

中图分类号: TP31

文献标志码: A

文章编号: 1000-2758(2025)03-0600-10

C 语言因其使用灵活、执行效率高, 既具有高级语言的优势, 又具备低级语言的特点, 因此广泛应用于航空、航天、电气、交通等高度依赖安全关键软件的领域^[1]。但是对于高安全、高可信的关键软件而言 C 语言程序缺少必要的安全措施, 例如对指针操作未进行严格限定、对数组和字符串未提供边界检查等, 易导致软件运行时出现难以发现的低概率异常, 造成灾难性的损失^[2]。因此, 如何发现潜在运行时故障, 提高软件质量和可信性^[3]已成为研究热点。近年来, 有关该课题的研究已经取得了巨大进步, 在诸多方法中, 软件监控技术 (software monitoring technology) 因其效果显著、应用广泛, 取得了许多重要成果。

软件监控技术可以节省开发和维护成本、提升开发效率并确保软件的可信性^[4], 其主要分为构造时监控和运行时监控 2 类方法。构造时监控基于代码静态分析方法, 对正在构造程序的代码进行在线分析并及时给出故障反馈。随着软件规模和复杂度不断提高, 对尽早发现故障的需求日益增加, 因此构

造软件时检测故障的监控技术被广泛研究。但现有研究所提出的相关技术可监控的故障范围较小或只面向某些特定故障, 软件构造过程中编写的代码片段, 因无法编译导致不能及时有效地检测故障。

本文所提出的任务安全关键软件构造时在线监控方法, 归纳了 5 类 43 种故障类型, 针对 C 代码以函数为单位代码量逐步增长的特点, 在构造 C 程序时对代码片段进行实时监控和静态分析, 能够全面、高效地检测潜在故障并反馈给开发人员, 从而提高软件安全性和可信性, 节省后续测试成本。本文贡献如下:

1) 针对在线编辑的 C 程序片段实时编译和测试问题, 提出了一种混合监控方法, 结合智能时间阈值和 deepfix 代码修复技术, 高效实现 C 程序构造时的可编译版本自动生成。

2) 为了全面有效地检测各类故障, 针对任务安全关键软件 5 类运行时故障产生条件归纳出 43 种故障类型, 基于抽象语法树建立在线编辑的 C 程序片段故障的规则库, 涵盖了断言违反、数组越界、多线程中的死锁和饥饿等多种故障类型。

3) 提出了一种基于规则库的语法结构匹配算法, 实现在线编辑的 C 程序片段故障监控。利用构

建的规则库来匹配 C 程序的抽象语法树结构,以识别潜在故障。算法通过递归遍历 AST 节点,并根据规则库进行匹配,能够识别包括违反断言、数组越界、空指针引用等在内的多种故障。

实验验证表明,本文所提出的 C 程序构造时在线监控方法能够有效地识别出实验对象中的潜在故障,具有较高的自动化程度,适用于人机结对编程的进一步研究。

1 相关工作

对软件实施合理的监控,可以及时获得软件系统的信息,准确分析并判断软件是否处于可信状态。软件监控技术在软件开发、软件调优、软件实时容错以及软件维护方面得到了广泛应用。现有软件监控技术大致分为构造时监控和运行时监控两大类。前者在编码过程中通过代码片段静态分析识别潜在故障,提高软件质量和开发效率;后者在运行过程中实时分析、处理软件的状态信息,可有效降低软件维护成本。随着计算机软件规模和复杂度的不断提高,尽早发现故障的需求日益迫切,因此在构造软件时检测故障的监控技术被广泛研究。

构造时监控是基于代码静态分析方法,对正在构造程序的代码实时进行分析并及时给出相应反馈的技术。现有研究主要采用形式化方法、软件质量模型方法和机器学习方法。

基于形式化的方法主要有:Li 等^[4]实现了一种支持 SOFL 形式化工程方法的软件工具,可以自动检查正在构建的形式化规范,确保内部一致性。Wang 等^[5]提出了基于人机结对编程特定编程范式的框架,该框架提供了故障建模的通用方法,检测程序构建时潜在安全漏洞。针对不完整程序的静态分析方法,Schubert 和 Wang 等^[6-7]提出可以在编码阶段进行分析,从而可以实时执行漏洞发现。Dai 等^[8]提出了结合风险编号和代码检查图的技术,旨在提高程序员在编程过程中自我检查代码的效率和准确性。

基于软件质量的度量值及模型定义监控方法,文献^[9]提出一个基于软件开发过程数据的可视化质量监控工具,该工具能够计算软件当前版本的度量值以反映产品质量的实时变化趋势。Ronchieri 等^[10]定义软件质量模型并设计用于监控软件质量的度量,形成可在任何开发阶段预测软件质量的数

学模型。Bielikova 等^[11]提出了一个独立于平台的代码监控环境,该环境使用信息标签作为文档模型和用户模型之上的描述性元数据,并支持多个软件开发监控工具。

基于机器学习的方法主要采用机器学习和人工智能算法提高代码检查的效率和精度。例如 Allamanis 等^[12]基于神经网络建立了软件构造的监控系统,其具有很强的识别软件故障的能力,可将软件开发效率提高约 20%,但仅适用于大型软件系统的开发,对于小型软件开发具有一定的局限性。Lal 等^[13]引入 bug 数据库,提出一种基于机器学习的代码评审方法,并在 Eclipse 上进行了可行性评估,快速、清晰地检查构建中的代码。还有一些研究者利用生成式预训练 Transformer (GPT) 进行代码检查^[14]。但这些研究主要侧重于检查完整的代码,而不是帮助程序员在程序构造时检测代码。

目前,国内外关于软件监控技术的研究,其可监控的故障范围较小,只面向某种故障,例如 Wang 的基于人机结对编程特定编程范式框架无法处理程序中的嵌套结构;Dai 的基于检查表的人机结对检查方法主要检测语义错误;Li 的支持 SOFL 形式化工程方法的软件工具主要检测形式化规范内部一致性;Lal 的基于机器学习的代码评审方法主要检测内存泄漏和逻辑错误。Georgsen 的基于生成式预训练 Transformer 的代码检查主要检查完整代码,而不是检查构造时的代码。

因此针对这些研究工作的不足之处,本文提出的 C 程序构造时在线监控方法,包括代码片段的可编译版本生成技术和基于规则库的 C 程序潜在故障的语法结构匹配方法,可在构造过程中实时识别潜在的 5 类运行时故障。其中,预识别 C 程序故障语法规则库为检测其他高级语言程序构造时的潜在故障扩展了思路,具有指导意义。

2 C 程序构造时故障检测流程

为实现安全关键软件的构造时在线监控和故障检测,首先要分析不同故障产生的条件及其表现形式。本文归纳了违反断言、数组越界、空指针引用、线程死锁和饥饿等 5 类运行时故障的特征,定义了各自产生的条件,划分了 43 种故障类型。

其次,定义每一种故障类型的关键语句到抽象语法树结构的映射规则,建立一个能够预识别潜在

故障语法结构的规则库,并设计相应的匹配算法以快速检测不同类型的故障。

然而,抽象语法树的构建依赖于完整的、可编译的 C 代码,这就需要在构造时能够将代码片段生成可编译的代码版本。本文采用混合监控机制,基于事件触发和时间阈值相结合的方法实现代码修复,生成可编译版本。

基于此,C 程序构造时的在线监控流程如图 1 所示,分为以下 4 步。

1) 基于混合监控方法触发检测流程,对构造中 C 程序代码片段构建副本并进行修复,得到可编译通过的 C 程序。

2) 对可编译通过的 C 程序进行词法语法分析生成其抽象语法树。

3) 遍历抽象语法树结构,与预识别 C 程序故障语法结构的规则库进行匹配。

4) 若存在与之对应的抽象语法树结构,则调用对应的故障检测方法;若不存在,则结束流程,等待下一次触发。

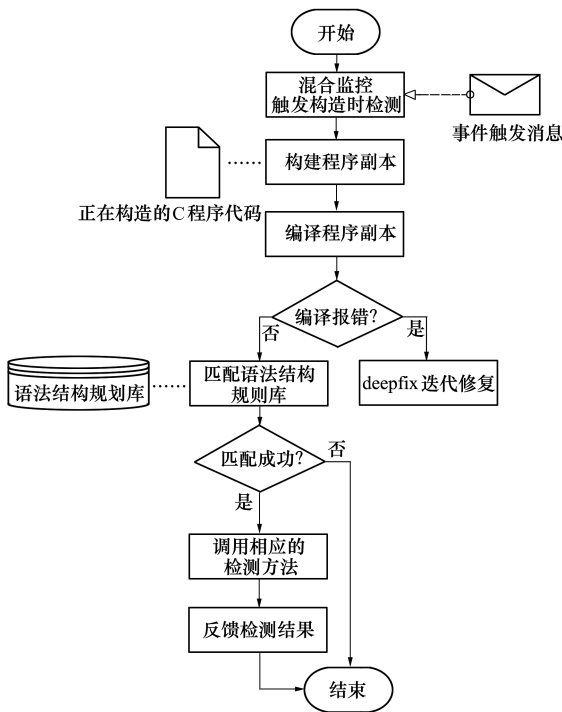


图 1 C 程序构造时故障检测流程

3 C 程序构造时可编译版本生成算法

由于构造中的 C 程序是不完整的,不能直接生成用于后续故障匹配以及检测的抽象语法树,因此

首先需要修复编译错误,目前已有 deepfix^[15]、SYN-FIX^[16]等技术。其中 deepfix 可通过多层序列到序列的神经网络实现全局分析,并对多个编译告警进行自动修复。deepfix 通过对当前版本 C 程序的编译错误给出修复建议,并检查该建议是否导致输入程序出现更多错误,若建议合理则更新程序,并进行下一个编译错误的修复。停止修复的情况有:①神经网络认为输入程序正确;②更新后的程序没有任何需要修复的编译错误;③超过迭代修复次数的预定义上限。

本文采用 deepfix 代码修复技术,并结合混合式监控机制,设计 C 程序构造时代码片段的可编译版本生成算法。由于 deepfix 修复的程序在达到迭代修复次数的预定义上限时可能依然存在编译错误,因此本文对该部分进行了修订,当达到迭代次数上限但依然存在编译错误时,则显示初始版本的编译告警,给予开发人员提示,并等待下一次事件触发,然后再重新执行可编译版本生成流程。

所谓混合式监控机制,是指当增加、删除、修改代码等事件触发较为频繁时则以时间触发模式来监控,否则以事件触发模式来监控。如算法 1 所示。

算法 1 C 程序可编译版本生成算法

input:构造时的 C 程序

output:可编译版本 C 程序

initialize: $T_{thre} \leftarrow threshold_value$ // 设置时间阈值

1. $T_{start} \leftarrow cur_time$ // 获取当前时间
2. L1: if event_trigger then // 事件触发
3. $T_{event} \leftarrow cur_time$ // 获取当前时间
4. $T_{dev} \leftarrow T_{event} - T_{start}$ // 计算时间间隔
5. if $T_{dev} < T_{thre}$ then // 间隔时间小于阈值
6. goto L1 // 继续等待事件触发
7. else
8. $ectype \leftarrow copy_source_code$ // 复制已构造的代码
9. $report \leftarrow compile\ ectype$ // 获取代码的编译结果
10. if error in report then // 编译结果存在错误
11. $ectype \leftarrow auto_repair$ // 调用 deepfix 进行自动修复
12. end if
13. $compilable_version \leftarrow ectype$ // 得到可编译版本
14. end if
15. end if

16. return compilable_version

设置一个时间阈值作为事件触发是否频繁的界定,当事件触发时,计算上一次记录的事件触发时间与当前事件的时间差,若时间差小于时间阈值,则采用时间触发方法,等待下一次事件的触发,直到时间差不小于时间阈值;否则时间差不小于时间阈值,触发不频繁,采用事件触发方法,进入后续的C程序代码片段可编译版本生成阶段。首先复制当前版本的C程序代码片段,构建一个程序副本,然后对该副本进行编译。当代码的不完整性或其他语法等问题导致程序不能编译通过时,则调用 deepfix 对编译告警的问题进行修复,形成可编译版本的C程序。

算法1的时间复杂度取决于代码复制和 deepfix 修复的时间复杂度。若 n 为输入代码长度,则代码复制的复杂度为 $O(n)$; deepfix 使用了多层 Seq2Seq 神经网络,并带有注意力机制,若最大迭代次数为 k ,则其时间复杂度为 $O(k \cdot n \cdot m \cdot d)$,其中 m 为输出代码的长度, d 为隐藏层的维度。因此算法1总时间复杂度为 $O(n+k \cdot n \cdot m \cdot d)$,即 $O(k \cdot n \cdot m \cdot d)$ 。算法1的空间复杂度取决于输入代码和输出代码所需的辅助空间,即 $O(n+m)$ 。

4 故障语法结构规则库与匹配算法

为了全面有效地检测各类故障,针对任务安全关键软件涉及的违反断言、数组越界、空指针引用、线程死锁和饥饿5类运行时故障分析了代码具体表现形式,归纳了43种故障类型。基于抽象语法树建立了在线编辑的C程序片段故障的规则库,提出了一种基于规则库的语法结构匹配算法,实现了在线编辑的C程序片段故障监控。

本节分别介绍5类运行时故障的产生条件、故障类型、抽象语法树映射规则,以及相应的匹配算法。由于篇幅限制,每一类故障类型,仅列举一个故障语法结构映射规则,以及相应的匹配算法。

4.1 违反断言

1) 故障类型

违反断言只与"assert(expression)"语句有关,因此只有一种产生条件和故障类型,如表1所示。

表1 违反断言故障类型划分信息表

产生条件	故障类型	编号
违反断言中的表达式	assert(expression)	F1

2) 映射规则 Rule1

Rule1 到抽象语法树结构的映射规则以及对于C程序的 assert(expression) 故障表现形式,其生成对应的抽象语法树结构如图2所示。

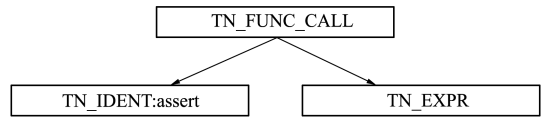


图2 Rule1 的抽象语法树

其中,根节点的左子节点调用的是 assert 函数,右子节点表示后续的表达 expression 表达式。

3) Rule1 语法结构匹配算法

如算法2所示,若根节点类型为 TN_FUNC_CALL,则递归遍历该节点,使当前节点为该节点的左子节点。若当前节点类型为 TN_IDENT,包含的字符串为"assert",且子节点的类型为 TN_EXPR,则根据 Rule1,将违反断言放入匹配结果中。

算法2 Rule1 语法结构匹配算法

input: 抽象语法树节点 node

output: 故障匹配结果 error_type

1. if (node is TN_FUNC_CALL) then
2. recur_traverse (node) // 递归遍历节点
3. if (node is TN_IDENT) then
4. if (node.value is "assert") then
5. recur_traverse (node)
6. if (node is TN_EXPR) then
7. error_type ← assert_violation
8. end if
9. end if
10. end if
11. return error_type

4.2 数组越界

1) 故障类型

数组越界有下标越界、API调用越界、指针访问越界3种,其中API调用越界主要与字符数组有关。以 strcpy 为例,当目标字符与源字符长度不一或源字符是一个不以'\0'结束的 char 型数组时,目标数组可能会出现越界写入,如表2所示。

表 2 数组越界故障类型划分信息表

产生条件	故障类型	编号
数组下标越界	常量下标越界(一维数组)	F2
	常量下标越界(二维数组)	F3
	变量下标越界(一维数组)	F4
	变量下标越界(二维数组)	F5
	数组长度不一致定义(一维数组)	F6
	数组长度不一致定义(二维数组)	F7
	API 调用	char * strcpy(char * dest, const char * src)
数组越界	int vsprintf(char * str, const char * format, va_list arg)	F9
	int sprintf(char * str, const char * format, ...)	F10
	char * strcat(char * dest, const char * src)	F11
	指针访问	一维数组
数组越界	二维数组	F13

2) 映射规则 Rule6

针对上述数组越界相关的故障类型建立到抽象语法树的映射规则:Rule2~13。本节以数组长度不一致定义(一维数组)的类型为例,建立映射规则 Rule6:数组定义(一维数组)到抽象语法树结构的映射规则,其生成对应的抽象语法树结构如图 3 所示。其中,根节点表示定义语句,左子节点和左子节点的子节点表示定义的类型,右子节点表示数组声明, TN_IDENT 的字符串值为数组名,右子节点的右子节点表示定义的数组长度。

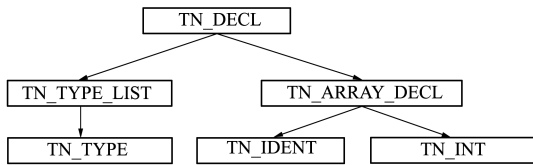


图 3 Rule6 的抽象语法树

3) Rule6 语法结构匹配算法

如算法 3 所示,如果节点类型为 TN_DECL,则递归遍历该节点;若左子树根节点类型为 TN_TYPE_LIST 且其子节点类型为 TN_TYPE,则继续递归遍历根节点的右子树;若右子树根节点类型为 TN_ARRAY_DECL,且其左子节点类型为 TN_IDENT,右子节点类型为 TN_INT,则符合 Rule6 的匹配规则,将数组越界放入匹配结果中。

算法 3 Rule6 语法结构匹配算法

input: 抽象语法树节点 node

output: 故障匹配结果 error_type

1. if(node is TN_DECL) then
2. recur_traverse(node) // 递归遍历节点
3. if(node is TN_TYPE_LIST) then
4. recur_traverse(node) // 递归遍历节点
5. if(node is TN_TYPE) then
6. recur_traverse(node)
7. if(node is TN_ARRAY_DECL) then
8. recur_traverse(node)
9. if (node is TN_IDENT) then
10. error_type←array_bound
11. end if
12. end if
13. end if
14. end if
15. end if
16. return error_type

4.3 空指针引用

1) 故障表现形式

空指针引用的产生条件是指针变量赋空,其中函数赋值指针是由于当字符串操作失败或内存未分配成功等特殊情况下函数会返回 NULL 值,此时可能会形成空指针,造成空指针引用,如表 3 所示。

表 3 空指针引用故障类型划分信息表

产生条件	故障类型	编号
指针变量赋空	一级指针定义时赋空	F14
	二级指针定义时赋空	F15
	一级指针调用时赋空	F16
	二级指针调用时赋空	F17
	复杂类型指针赋空	F18
	指针数组赋空	F19
	自定义函数的返回值为空	F20
	函数参数赋空	F21
	库函数 char * strchr(const char * str, int c) 赋值指针	F22
	库函数 char * strtok(char * str, const char * delim) 赋值指针	F23
	内存分配函数 malloc 赋值指针	F24
	内存分配函数 realloc 赋值指针	F25

2) 映射规则 Rule16

针对上述空指针引用相关的故障表现形式建立到抽象语法树的映射规则:Rule14~Rule25。本节以一级指针调用时赋空的表现形式为例,建立映射规则 Rule16:一级指针调用时赋空到抽象语法树结构

的映射规则,其生成对应的抽象语法树结构如图4所示。

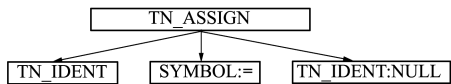


图4 Rule16的抽象语法树

其中,根节点表示赋值操作,第一个子节点中TN_IDENT的字符串值为指针变量名,第二个子节点和第三个子节点表示赋空。

3) Rule16 语法结构匹配算法

如算法4所示。

算法4 Rule16 语法结构匹配算法

input: 抽象语法树节点 node

output: 故障匹配结果 error_type

1. if (node is TN_ASSIGN) then
2. recur_traverse(node) //递归遍历节点
3. if (node is TN_IDENT) then
4. recur_traverse(node) //递归遍历节点
5. if (node is SYMBOL and node.value is "=") then
6. recur_traverse(node)
7. if (node is TN_IDENT and node.value is "NULL") then
8. error_type←null_pointer_dereference
9. end if
10. end if
11. end if
12. end if
13. return error_type

如果根节点类型为 TN_ASSIGN,则递归遍历该节点,若第一个子节点类型为 TN_IDENT,第二个子节点类型为 SYMBOL 且值为“=”,第三个子节点类型为 TN_IDENT 其值为 NULL,则符合 Rule16 的匹配规则,将空指针引用放入匹配结果中。

4.4 死锁与饥饿

1) 故障表现形式

由于死锁和饥饿都存在于并发多线程中,因此两者的产生条件都与 POSIX 标准头文件以及 Solaris 标准头文件有关,此处只罗列了其较为常见的表现形式,如表4所示。

表4 死锁、饥饿故障类型划分信息表

产生条件	故障类型	编号
	pthread_mutex_t	F26
	PTHREAD_MUTEX_INITIALIZER	F27
	int pthread_mutex_lock	F28
	(pthread_mutex_t * mutex)	
	int pthread_mutex_unlock	F29
存在	(pthread_mutex_t * mutex)	
头文件	pthread_cond_t	F30
pthread.h	int pthread_cond_wait(pthread_cond_t * restrict cond, pthread_mutex_t * restrict mutex)	F31
	int pthread_cond_signal	F32
	(pthread_cond_t * cond)	
	int pthread_setschedparam(pthread_t tid, int policy, const struct sched_param * param)	F33
	int thr_suspend(thread_t tid)	F34
存在	int thr_continue(thread_t tid)	F35
头文件	int mutex_lock(mutex_t * mp)	F36
thread.h	int mutex_unlock(mutex_t * mp)	F37
	int thr_setprio(thread_t tid, int newprio)	F38
	sem_t	F39
存在	int sem_wait(sem_t * sem)	F40
头文件	int sem_post(sem_t * sem)	F41
semaphore.h	int sem_init(sem_t * sem, int pshared, unsigned int value)	F42
	int sem_getvalue(sem_t * sem, int * sval)	F43

2) 映射规则 Rule30

针对上述死锁与饥饿相关的故障表现形式建立到抽象语法树的映射规则:Rule26~ Rule43。本节以 pthread_cond_t 为例,建立映射规则 Rule30:条件变量定义 pthread_cond_t 到抽象语法树结构的映射规则,其生成对应的抽象语法树结构如图5所示。

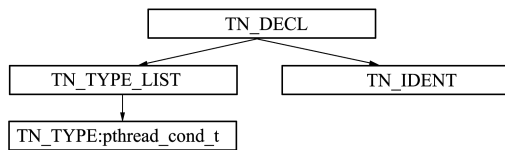


图5 Rule30的抽象语法树

其中,根节点表示定义语句,左子节点及其子节点表示定义的类型为 pthread_mutex_t,右子节点 TN_IDENT 的字符串值为互斥量名。

3) Rule30 语法结构匹配算法

如算法5所示,如果根节点类型为 TN_DECL_CALL,则递归遍历该节点,使当前节点为该节点的左子节点。若当前节点类型为 TN_TYPE,且包含的

字符串为"pthread_cond_t",则继续遍历根节点的右子树;若右子树根节点类型为 TN_IDENT,则符合 Rule30 的匹配规则,将死锁和饥饿放入匹配结果中。

算法 5 Rule30 语法结构匹配算法

input: 抽象语法树节点 node

output: 故障匹配结果 error_type

1. if (node is TN_DECL) then
2. recur_traverse(node)
3. if (node is TN_TYPE_LIST) then
4. recur_traverse(node)
5. if (node is TN_TYPE and node.value is "pthread_cond_t") then
6. recur_traverse(node)
7. if(node is TN_IDENT) then
8. error_type←deadlock or starvation
9. end if
10. end if
11. end if
12. end if
13. return error_type

4.5 算法性能分析

根据算法 2~5 可知,故障匹配过程包括源代码建立抽象语法树和抽象语法树的遍历 2 个阶段,其中抽象语法树的建立分为语法分析和词法分析,其

时间复杂度和空间复杂度均为 $O(n)$, n 为源代码的长度。抽象语法树的遍历采用 DFS 算法,其时间复杂度为 $O(t)$, t 为树中节点的数量;空间复杂度为 $O(h)$, h 为树的高度。因此故障匹配算法总的复杂度为 $O(n+t)$, 空间复杂度为 $O(n+h)$ 。可见,算法随问题规模,即源代码长度的增长,时间开销和空间开销呈线性增长,能够满足处理大规模代码时的性能需求。

5 实验验证

5.1 实验对象

本文选取了表 5 所示的 50 个程序作为实验对象,这些程序源码来自“软件定义卫星站控系统”和“基于国产操作系统的飞控核心软件形式化验证”等项目的工程代码,涵盖了航空、航天等安全关键领域常用的线性表、树、图等数据结构的查找、遍历、排序等操作,以及多线程处理、网络通信、总线通信等不同功能;程序规模从几十行到数千行不等;程序复杂度包括 $O(\log_2 n)$, $O(n)$, $O(n\log_2 n)$, $O(n^2)$, $O(n^3)$ 等。所选案例的程序源码覆盖了本文介绍的 43 种故障表现形式,具有较好的代表性。本文将依据上述方法,在构建这些程序案例时,检查是否按规则库匹配到相应的故障类型。

表 5 实验结果表(节选)

序号	程序名称	当前行数	程序功能	迭代次数	停止修复原因	匹配规则
1	qupdate.c	34	更新队列中的数值	2	②	Rule4
2	bankers.c	104	银行家算法	4	②	Rule4,5
3	management_ads.c	58	检测误差是否超限	2	②	Rule1,4
13	roundrobin.c	48	计算平均周转时间	5,2	③,②	Rule4
20	Driver_receive1553.c	97	处理从 1553B 总线接收的数据	2	②	Rule4,19
34	vector.c	125	实现 vector 容器	2	②	Rule4,17,21,24
39	floydwarshall.c	39	弗洛伊德算法实现	5,2	③,②	Rule5
40	huffman_encoding.c	210	霍夫曼编码实现	1	②	Rule4,18,20,25
50	shell_callback.c	236	每行查询结果使用 shell 排序	3	②	Rule1,4

5.2 实验过程

本节以表 5 中的 Driver_receiv1553.c 程序为例,介绍具体的检测流程和实验过程。该程序主要负责

从 1553b 总线接收数据,从接收到的数据读取相关信息,如命令字、数据长度、数据块等,对接收数据块的有效性进行判断,该程序定义了 10 个数组对象,

其中包括 2 个指针数组。

首先对于 Driver_receiv1553.c 程序随机选取了前 123 行作为当前正在构造的代码片段,基于混合式监控方法生成可编译版本,设置时间阈值为 5 s。当事件时间差超过阈值时,对当前程序的副本进行编译。由于程序不完整,存在编译错误,因此调用

```

118 | for (ii = 0; ii < ADS_REV_CNT; ii++)
119 | {
120 |     if (!ads_err[ii])
121 |     {
122 |         *ADS_var_safe[ii] = *ADS_cn_o
123 |     }
    
```

图 6 构造时的 C 程序

```

118 | for (ii = 0; ii < ADS_REV_CNT; ii++)
119 | {
120 |     if (!ads_err[ii])
121 |     {
122 |         *ADS_var_safe[ii] = *ADS_cn_o[ii];
123 |     }
    
```

图 7 第一次迭代修复后的 C 程序

```

118 | for (ii = 0; ii < ADS_REV_CNT; ii++)
119 | {
120 |     if (!ads_err[ii])
121 |     {
122 |         *ADS_var_safe[ii] = *ADS_cn_o[ii];
123 |     }
    
```

图 8 可编译通过的 C 程序
(第二次迭代修复)

接着,生成可编译版本代码的抽象语法树,由于完整的抽象语法树比较繁杂,本文只展示部分数据对象的抽象语法树。图 9 展示了指针数组 ADS_var_safe 与 ADS_var_o 的抽象语法树。抽象语法树的示意图中,节点均标明了类型,叶子节点中展示了具体的字符串值信息。

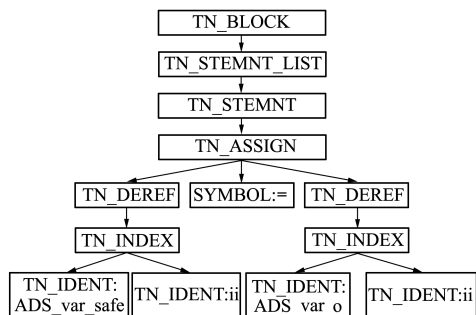


图 9 指针数组 ADS_var_safe 与 ADS_var_o 的抽象语法树示意图

最后,按照本文提出的 C 程序运行时故障语法规则的规则库及 C 程序构造时监控算法,生成匹配结果。在图 9 中,当遍历到 TN_ASSIGN 节点时,与规则库中的 Rule19 进行匹配,最终匹配成功,识别到运行时故障——“指针数组赋空”。

5.3 实验结果及分析

实验结果共计匹配到 41 种故障类型,146 个潜在运行时故障均正确反馈。实验数据信息如表 5 所示,其中停止修复的原因如第 3 节所述共有 3 种,编号分别为①,②,③。

在案例 13 和 39 中,初始构建的程序未能在达到迭代修复次数的上限前形成可编译版本,因此继

续构建代码,当事件时间差超过阈值时,重新执行可编译版本生成流程,此次在迭代修复 2 次后得到可编译版本。此外,经过 C 程序抽象语法树与规则库的对照分析,案例 34 和 40 由于已构建的程序未出现相应结构,因此结果分别缺少 F20、F21 故障类型。整体来说,实验结果均符合预期,表明本文所提出的监控方法能够有效识别潜在故障,提高软件安全性和可信性。

6 结 论

任务安全关键领域软件使用 C 语言开发的程序存在潜在的安全风险,易导致运行时故障,造成巨大损失,本文针对此类问题,总结、分析了现有软件监控方法的特点和不足,提出了一种任务安全关键软件 C 程序构造时的在线监控方法,在构造 C 程序时对代码进行监控和静态分析,高效监测潜在故障。首先,针对在线编辑的 C 程序片段的实时编译及测试问题,提出了一种混合式监控方法的片段程序可编译版本自动生成技术,可有效减少监控开销,并通过现有的代码修复技术处理当前不完整的 C 程序,生成可编译通过的版本。其次,针对任务安全关键软件常见的指针操作不严格限定、数组和字符串缺乏边界检查等 5 类运行时故障的产生条件归纳出 43 种故障类型,基于抽象语法树建立在线编辑的 C 程序片段故障的规则库。在此基础上,提出基于语法结构匹配算法识别潜在运行时故障,实现了在线编辑的 C 程序片段故障的构造时在线监控功能。最后,对 50 个 C 程序进行了实验验证,共匹配 41 种故障类型,发现 146 个潜在故障,表明本文所提出的

监控方法能够有效识别潜在故障,提高软件安全性和可信性,且执行效率高,开销较低,具有很好的实用性。后续将针对 C++ 和 Java 等其他主流编程语

言,优化可编译版本生成方法,并研究其故障类型,形成适用于不同语言的规则库。

参考文献:

- [1] ZHOU J, ZHANG M X. Survey on trustworthy software evaluation[J]. *Application Research of Computers*, 2012, 29(10): 3609-3613
- [2] 刘克, 单志广, 王戟, 等. “可信软件基础研究”重大研究计划综述[J]. *中国科学基金*, 2008(3): 145-151
LIU Ke, SHAN Zhiguang, WANG Ji, et al. Overview on major research plan of trustworthy software[J]. *Bulletin of National Natural Science Foundation of China*, 2008(3): 145-151 (in Chinese)
- [3] 郎波, 刘旭东, 王怀民, 等. 一种软件可信分级模型[J]. *计算机科学与探索*, 2010, 4(3): 231-239
LANG Bo, LIU Xudong, WANG Huaimin, et al. A classification model for software trustworthiness[J]. *Journal of Frontiers of Computer Science and Technology*, 2010, 4(3): 231-239 (in Chinese)
- [4] LI S, LIU S. A software tool to support scenario-based formal specification for error prevention[J]. *Lecture Notes in Computer Science*, 2018, 10795: 187-199
- [5] WANG P, LIU S, LIU A Z F. A framework for modeling and detecting security vulnerabilities in human-machine pair programming[J]. *Journal of Internet Technology*, 2022, 23(5): 1129-1138
- [6] SCHUBERT P D, NIXDORF H, HERMANN B, et al. Lossless, persisted summarization of static callgraph, points-to and data-flow analysis[C]//*Leibniz International Proceedings in Informatics*, 2021
- [7] WANG P, LIU S, JIANG L W. Detecting security vulnerabilities with vulnerability nets[J]. *The Journal of Systems and Software*, 2024, 208: 111902
- [8] DAI Yujun, LIU Shaoying, XU Guangquan. Enhancing human-machine pair inspection with risk number and code inspection diagram[J]. *Software Quality Journal*, 2024, 32(3): 939-959
- [9] 潘森, 林云, 彭鑫, 等. 基于软件开发过程数据的可视化产品质量监控工具[J]. *计算机应用与软件*, 2015, 32(9): 8-12
PAN Sen, LIN Yun, PENG Xin, et al. Visualised product quality monitoring tool based on software development process data [J]. *Computer Applications and Software*, 2015, 32(9): 8-12 (in Chinese)
- [10] RONCHIERI E, CANAPARO M. A software quality predictive model[C]//*International Conference on Software Engineering and Applications*, 2013
- [11] BIELIKOVÁ M, POLÁŠEK I, BARLA M, et al. Platform independent software development monitoring: design of an architecture[J]. *Lecture Notes in Computer Science*, 2014, 8327: 126-137
- [12] ALLAMANIS M, BROCKSCHMIDT M, KHADEMI M. Learning to represent programs with graphs[C]//*6th International Conference on Learning Representations*, 2018
- [13] LAL H, PAHWA G. Code review analysis of software system using machine learning techniques[C]//*2017 11th International Conference on Intelligent Systems and Control*, 2017
- [14] SZABÓ Z, BILICKI V. A new approach to web application security: utilizing GPT language models for source code inspection [J]. *Future Internet*, 2023, 15(10): 326
- [15] GUPTA R, PAL S, KANADE A, et al. Deepfix: fixing common C language errors by deep learning[C]//*National Conference on Artificial Intelligence*, 2017
- [16] AHMED T, LEDESMA N R, DEVANBU P. Synshine: improved fixing of syntax errors[J]. *IEEE Trans on Software Engineering*, 2021, 49: 2169-2181

Research on monitoring method during the construction of safety-critical software

WANG Ben, DING Chengjun, LIN Wei, MA Chunyan

(School of Software, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: The C language is widely used in aerospace and other critical areas due to its flexibility and high efficiency. However, C programs have safety risks, such as unrestricted pointer operations and lack of boundary checks for arrays and strings, which can easily lead to potential runtime faults. To address these issues, an online monitoring method for building safety-critical C programs that efficiently detects potential errors by monitoring and analysing the code in the program generation is proposed. To solve the problems of real-time compilation and verification of the online edited C program segments, a hybrid monitoring method and a technique for generating compliable versions of the segment programs are proposed. Then 43 types of error conditions are induced for 5 types of runtime errors in safety-critical software, and a rule library for error detection of online edited C program segments is constructed based on the abstract syntax trees. Finally, a syntax structure matching algorithm is proposed to implement the error monitoring of online edited C program segments. 50 commonly used C program segments from safety-critical software were selected for verification, resulting in a total of 41 matches and 146 potential runtime errors. The results show that the present monitoring method can effectively identify the potential errors and thus improve the safety and reliability of the software.

Keywords: construction-time monitoring; fault detection; abstract syntax tree; automated testing

引用格式:王犇,丁成钧,林伟,等.任务安全关键软件构造时在线监控方法研究[J].西北工业大学学报,2025,43(3):600-609

WANG Ben, DING Chengjun, LIN Wei, et al. Research on monitoring method during the construction of safety-critical software[J]. Journal of Northwestern Polytechnical University, 2025, 43(3): 600-609 (in Chinese)