软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(3):547−561 [doi: 10.13328/j.cnki.jos.004980]

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

基于变量访问序模式的中断数据竞争检测方法*

陈 睿[1,3], 杨孟飞[2], 郭向英[1,3]

[1](北京控制工程研究所,北京 100190)

[2](中国空间技术研究院,北京 100094)

[3](北京轩宇信息技术有限公司,北京 100190)

通讯作者: 陈睿, E-mail: chenrui@sunwiseinfo.com, http://www.sunwiseinfo.com

摘 要: 在航天嵌入式软件等中断驱动型软件中,中断数据竞争问题十分突出.然而,中断在并发语义、同步机制、调度机制等方面与线程(任务)有诸多不同,具有 Ad-hoc 特征,难以统一刻画,因此,主流的数据竞争检测方法并不适用.以航天嵌入式软件数据竞争案例库为基础进行了系统分析,提出刻画有害中断数据竞争的 7 种缺陷模式.针对其中最常见且最难解决的单变量访问序模式,基于抽象解释,提出一种支持过程间分析、中断并发分析的高效检测方法.设计并实现了相应的检测工具 SpaceDRC.实验结果表明,SpaceDRC 能够在 145ms 内检测出约 21 400 行程序中的真实数据竞争.SpaceDRC 已经在多个航天重点型号中进行了应用,使得中断数据竞争专项分析的效率提高了至少 5 倍,并且降低了问题遗漏率.

关键词: 中断驱动型程序;数据竞争;抽象解释

中图法分类号: TP311

中文引用格式: 陈睿,杨孟飞,郭向英.基于变量访问序模式的中断数据竞争检测方法.软件学报,2016,27(3):547−561. http://www.jos.org.cn/1000-9825/4980.htm

英文引用格式: Chen R, Yang MF, Guo XY. Interrupt data race detection based on shared variable access order pattern. Ruan Jian Xue Bao/Journal of Software, 2016,27(3):547−561 (in Chinese). http://www.jos.org.cn/1000-9825/4980.htm

Interrupt Data Race Detection Based on Shared Variable Access Order Pattern

CHEN Rui[1,3], YANG Meng-Fei[2], GUO Xiang-Ying[1,3]

[1](Beijing Institute of Control Engineering, Beijing 100190, China)

[2](China Academy of Space Technology, Beijing 100094, China)

[3](Beijing Sunwise Information Technology Ltd., Beijing 100190, China)

Abstract: Interrupt data race is a category of critical bugs in interrupt-driven software such as aerospace embedded software. However, interrupt is much different from thread or task on concurrency semantics, synchronization and schedule, and its ad-hoc characteristics is hard to describe. Thus the state-of-art data race detection techniques are not suitable to interrupt-driven software. In this paper, the data race bug repository of aerospace embedded software is reviews systematically, and seven bug patterns for harmful interrupt data race are proposed. For the pattern single variable access order, an efficient abstract interpretation based detection method is developed to support inter-procedural and interrupt concurrency analysis. A tool named SpaceDRC is designed to verify our method. The evaluation results show that SpaceDRC only takes 145ms to detect 21400 lines of code to find the true bugs. Up to now, SpaceDRC has been applied in several aerospace missions, increasing the efficiency of interrupt data race inspection by 5 times and making a significant reduction in bug omission rate.

Key words: interrupt-driven program; data race; abstract interpretation

* 基金项目: 国家自然科学基金(91118007)
Foundation item: National Natural Science Foundation of China (91118007)
收稿时间: 2015-07-14; 修改时间: 2015-10-20; 采用时间: 2015-11-27; jos 在线出版时间: 2016-01-05
CNKI 网络优先出版: 2015-10-16 11:53:29, http://www.cnki.net/kcms/detail/11.2560.TP.20151016.1153.001.html

数据竞争是各类并发软件中最突出的缺陷类型之一,这类问题发生在多个并发执行流(如线程、任务、中断)对同一数据单元进行同时读写,且至少其中一个操作是写操作时.由于数据竞争中涉及的两次访问之间的次序不可确定,程序可能会因此产生异常行为,严重时甚至导致软件或系统失效.

中断驱动型嵌入式软件是一类典型的并发软件,主要依赖嵌入式处理器的中断机制实现强实时的并发响应,在航天、汽车电子、医疗设备、无线传感器网络等安全关键系统中广泛应用.在各个应用领域,由于中断数据竞争导致的事故屡见不鲜.如:某卫星帆板驱动线路盒转角跳变导致控制偏差;Therac-25 放射治疗仪致 5 人丧生[1];2003 年,北美地区大停电事故[2]等等.据中国空间技术研究院软件产品保证中心统计:在航天器总装测试(A.I.T.)阶段发现的软件质量问题中,约 80%都与数据竞争密切相关.

然而在中断驱动型程序中,中断是表达并发的主要手段,在调度机制、同步机制、抢占关系等方面与线程有诸多不同,并没有特定的并发原语机制,因此,目前主流的数据竞争检测方法都无法适用.同时,由于在中断驱动型程序中绝大多数数据竞争是良性的,若以数据竞争作为检测目标,检测结果中会出现大量误报.

本文以航天嵌入式软件真实数据竞争案例为基础,对中断驱动型嵌入式软件的并发特征以及中断数据竞争缺陷的特征进行了系统分析,提出了中断数据竞争的 7 种缺陷模式,用于精确刻画有害的中断数据竞争;进而,针对其中最重要的单变量访问序模式提出了基于抽象解释的静态检测方法;最后,设计并实现了一个数据竞争检测工具 SpaceDRC 来验证方法的有效性.

本文第 1 节通过对航天嵌入式软件数据竞争案例进行系统分析,提出中断驱动型程序的特征以及缺陷模式.第 2 节给出检测方法和系统.第 3 节介绍实验评估结果.第 4 节介绍相关研究工作.最后对全文进行总结.

1 中断数据竞争缺陷特征概述

中断驱动型程序中的数据竞争检测具有非常重要的意义,但是缺乏有效的检测方法和技术.试图将面向线程、任务、进程的并发缺陷检测方法照搬到中断驱动型程序上,面临误报多、效率低等问题.这主要因为:

1) 中断的并发语义、并发机制与任务、线程不同;
2) 中断驱动型程序的体系结构、共享数据使用、内存模型等与多线程、多任务程序不同;
3) 中断驱动型程序中存在大量良性数据竞争,并不一定构成缺陷.

因此,如何根据中断驱动型程序的特征设计精确高效的并发程序分析方法以及如何准确刻画中断数据竞争的缺陷模式,是研究中断数据竞争检测方法的基础.

本文以航天嵌入式软件为例进行了实例研究,选取了 53 个真实型号软件以及 97 个典型中断数据竞争案例进行系统分析,给出了中断驱动型程序的特征以及中断数据竞争的缺陷模式.

1.1 中断驱动型程序的特征

针对本文研究内容,中断驱动型程序的特征体现在以下 4 个方面.

- 体系结构

中断驱动型程序一般采用主程序加多级中断的并发体系结构,其中:主程序是无限循环结构,周期性地等待中断触发,并进行相应的处理;中断负责处理外部硬件输入或其他定时任务.在这种体系结构下,主程序的几乎所有数据流都由中断驱动.

- 并发调度

与多线或任务通过软件调度器进行并发调度不同,中断调度是通过处理器硬件机制实现的,不同中断分配不同的优先级,高优先级中断可以抢占低优先级中断,中断可以抢占主程序,而反过来则不行,这是一种非对称的抢占关系.另一方面,中断的触发具有多种不确定性,如定时触发、程序请求触发、外部触发等,在时序上存在周期性、随机性可能,在频率上存在频发和偶发之分.

- 并发控制

由于硬件资源受限,中断驱动型程序往往不采用底层操作系统,因此缺乏规范的并发控制机制,如锁、信号量等.天然的并发控制机制是硬件中断开关,即,通过设置或清除中断使能寄存器来禁止中断触发.这种方式在

一定程度上会影响实时性,在实际工程中仅在必要时使用;除此之外,多采用 Ad-hoc 的同步方式进行并发控制.最常见的是通过自定义变量进行同步或互斥,其他常用的还包括设置双倍缓冲区、立标志进行二次通信等.

- 共享数据使用

中断驱动型程序中定义较多全局变量,主程序和中断以及不同中断之间通过这些共享变量实现互相通信和数据交互的是普遍的设计策略.因此,存在大量数据竞争是有意设计导致的,并不是真正的缺陷.

上述特征对精确检测中断数据竞争缺陷有关键意义.例如:对于中断驱动型程序,若在程序分析时不考虑中断并发对程序状态的影响,则主程序中的很多执行路径将会被误认为不可执行,从而漏报潜在的缺陷;对于非对称的抢占和同步机制的缺乏,基于 happen-before 和 Lockset 的检测方法都无法适用;由于大量故意设计的共享访问,若以数据竞争的定义而进行检测的话,结果中会出现大量良性竞争假告警.

1.2 中断数据竞争缺陷模式

基于 97 个典型案例,本文从软件及运行环境、缺陷表象、发现手段、后果、处理方式等方面进行了系统的分析.提出 7 种航天嵌入式软件中断并发缺陷的模式,这些缺陷模式能够较准确地刻画什么样的数据竞争是真正有害的.

模式 1. 单变量访问序模式

同一变量的 3 次连续并发访问构成有害的数据竞争.3 次访问分别是 p 访问、c 访问和 r 访问,描述为 $p\rightarrow r\rightarrow c$,其中,$p$ 和 c 对应被抢占并发流中前后两次访问,r 对应另一个并发流中的访问.3 次内存访问可有 8 种组合情况,其中 4 种组合可能是有害的数据竞争缺陷,另外 4 种情况尽管可能是数据竞争,但不会构成缺陷.图 1 给出了 4 种访问序模式的示意图,具体说明如下:

- $R\rightarrow W\rightarrow R$:$p$ 访问和 c 访问预期读到相同的 x,而并发流 2 的抢占导致这种预期被破坏.在实际程序中,程序员往往对同一个并发流中的数据流存在一定假设;
- $W\rightarrow R\rightarrow R$:$c$ 访问预期读到 p 访问的赋值,当并发流 2 在之间抢占时,c 访问读到 r 的赋值.这与程序员的预期不符,往往是潜在的竞争;
- $R\rightarrow W\rightarrow W$:当 p 访问和 c 访问存在对 x 值的一致预期时,并发流 2 中的 r 访问会破坏这种一致性,从而导致数据竞争.典型情况是:p 访问为 x 相关的分支判断条件,而 c 访问为该分支下的赋值;
- $W\rightarrow R\rightarrow W$:这种模式发生在对同一连续数据区进行写入时,如数组、缓冲区等,在并发流 2 中进行读取可能会出现数据不完整的情况.

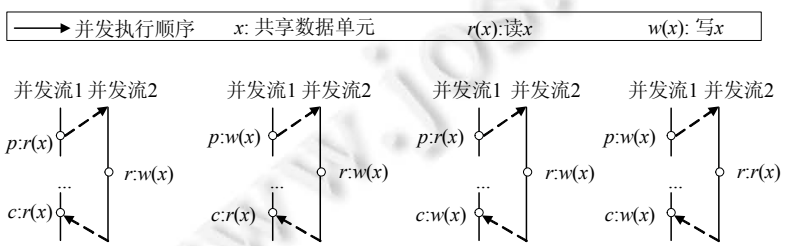

Fig.1 Single variable access order pattern

图 1 单变量访问序模式

模式 2. 多个关联变量访问一致性被破坏

程序中常常使用多个变量来共同表示同一个物理量或者一组关联的数据,对这些变量的访问往往是一致的,由于中断抢占破坏一组变量访问的一致性,会导致非预期的结果,如破坏数据完整性.如图 2 所示,程序在低优先级中断中完成数据接收后,对双缓冲区进行了切换,同时将 *g_timemark* 清 0 并设置校时标志,上述 2 个变量的值在软件设计中存在隐含的关联性.考虑以下特殊时序:当低优先级中断执行完毕 *g_timemark*=0;之后,被高优先级中断打断,由于 *g_timemark* 已清 0 且 *NewData_Flag_ForAjust* 标记未置位,进入校时函数后,会执行图 2

中的 if 语句真分支,*UTC_BJSec_GPS* 变量的值计算出现错误,导致返回的时差比实际多 1s.

```
低优先级中断                                    高优先级中断
g_timemark = 0;   //时间累计差值              //校时函数代码片段

                                               If (NewData_Flag_ForAjust==0)
                                               {
NewData_Flag_ForAjust=1;   //校时标志              UTC_BJSec_GPS=UTC_BJSec_GPS+g_timemark;
                                               }
```

Fig.2 Example: Breaking the consistency of multi-variable access

图 2 示例:多个关联变量访问一致性被破坏

模式 3. 多字节变量访问原子性被破坏

只有单条机器指令是具有天然原子性的,一行 C 代码往往对应多条指令.因此,对同一个变量的一次读写语句编译后是多条机器指令,当发生数据竞争时,会导致数据完整性被破坏.如在 Intel MCS-51 平台下,操作 *int* 类型的变量是通过两条指令来完成的,若两条指令之间被中断打断并发生了数据竞争,则会导致高低字节访问不一致.

模式 4. 与硬件的并行访问冲突

嵌入式软件与外部硬件可能同时操作一块共享数据(如 1553B 总线存储区),由于缺乏同步协议导致并行访问冲突,存取的数据完整性被破坏.这一缺陷模式属于并行范畴,无法通过程序分析发现.

模式 5. 重复加锁解锁

开关中断类似于多线程程序中的锁机制,可用于屏蔽中断处理,从而保证一段代码的原子性.这两个操作总是成对出现的.若在没有意识到此时已经处于锁保护状态下再次进行加锁解锁操作,会导致第 2 次解锁时候的代码未被保护,从而引发数据竞争.图 3 给出了重复加锁解锁模式的示例,程序使用加锁解锁把 1~100 的代码片段保护起来,期望这段代码的执行是原子的.在中间某个子函数中,程序员没有意识到外层函数已经进行了锁保护,再次对 10~90 的代码片段实施了加锁解锁,这使得 90~100 的代码片段失去保护,与第 1 次操作的预期不符,可能会引发数据竞争.

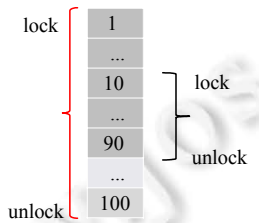

Fig.3 Double lock & unlock pattern

图 3 重复加锁解锁模式

模式 6. volatile 修饰符误用

volatile 修饰符是 C 语言中极容易被误解的特性,即使是嵌入式软件开发的专家也可能会在这个问题上出错.对 volatile 的误用体现在 3 个方面.

(a) 对外部设备端口、并发执行流之间共享的变量未使用 volatile,这导致一些变量的并发访问被编译优化掉,而出现数据错误.这类缺陷常被误认为是编译器错误导致;

(b) 误认为 volatile 访问与非 volatile 访问之间的执行顺序能够保持.编译优化会打乱 volatile 访问和非 volatile 访问之间的顺序,这种顺序与程序员在 C 代码中不一致,可能导致数据竞争.正确的做法是对需要保持顺序的变量都用 volatile 修饰;

(c) 误认为可以使用 volatile 保证原子性.volatile 可以用来说明对其他并发执行的计算可见,但并不保证对它访问的原子性.

模式 7. 共用寄存器组

这种缺陷模式仅发生在特定的嵌入式平台下.例如:对于 Intel MCS-51 平台,寄存器组用于指定主程序或某个中断处理程序中局部变量使用的一组寄存器.若不同的并发流指定了相同的寄存器组,当发生抢占时,就会出现寄存器之间的竞争,导致局部变量的值被破坏.

各个缺陷模式的统计见表 1,其中,模式 1~模式 3 占所有缺陷的 95%,是中断数据竞争主要来源;模式 4~模式 6 仅占比 5%,且数量都很少.针对模式 3,结合数据类型与机器字长,文献[3]提出的方法可以很好地解决;模式 4 属于系统设计导致的缺陷,需从设计层面解决;模式 5 一般通过基本的控制流分析和数据流分析即可静态检测;模式 6 通过相关编码规范来进行约束;模式 7 无法采用自动化的方法,一般通过代码审查即可有效识别.

Table 1 Bug patterns of interrupt data race and statistics

表 1 中断数据竞争的缺陷模式及其统计结果

序号	缺陷模式	缺陷数
1	单变量访问序模式	35
2	多个关联变量访问一致性被破坏	18
3	多字节变量访问原子性被破坏	39
4	与硬件的并行访问冲突	1
5	重复加锁解锁	1
6	volatile 使用不当	1
7	共用寄存器组	2

对于模式 1 和模式 2,目前尚缺乏有效的检测手段.本文重点针对模式 1 进行了研究,结合中断驱动型程序的并发特征,基于抽象解释提出了高效的检测方法.

2 检测方法与系统

针对缺陷模式 1,本文提出了一种高效的静态检测方法.该方法的关键在于将 3 次访问序模式的路径匹配问题转换为一个可达访问集的迭代求解问题,基于抽象解释框架设计并实现了工具 SpaceDRC,其中采用了多种分析策略来适应中断并发特征,从而提高缺陷检测精度:

(1) 扩展了抽象解释的迭代分析,以支持中断并发体系结构.即,分析时考虑中断并发对程序抽象状态的影响;

(2) 使用数值区间抽象域和线性约束求解支持路径敏感的分析,从而减少了自定义变量同步机制导致的误报;

(3) 采用了适合中断驱动型嵌入式软件的内存模型.

图 4 给出了 SpaceDRC 的架构和分析原理.SpaceDRC 采用 Java 语言开发,基于北京控制工程研究所开发的静态分析工具 SpecChecker 构建.SpecChecker 是专门针对航天嵌入式软件设计的静态分析工具,其语法分析前端采用 ANTLR 工具构建,能够进行 C 程序的预处理、词法/语法分析、符号表构建、类型检查、调用关系分析和控制流图构造,支持绝大多数嵌入式平台的 C 语言扩展,能够产生规范的中间表示,利于进行进一步的程序分析.图 4 灰色背景部分是本文在 SpecChecker 基础上进行的扩展,其中,抽象解释器支持中断并发、过程间、路径敏感和摘要机制,定义了一种与中断驱动型程序特征相符的内存模型,实现了整数区间抽象域和指针抽象域.SpaceDRC 实现了一个数百行代码的整数线性约束求解器来处理迭代分析过程中的简单分支约束,主要针对采用自定义标志变量进行并发同步的情况.

SpaceDRC 进行数据竞争检测分为以下几个步骤.

(1) 中间表示构建:产生后续分析所需的规范中间表示;

(2) 基于抽象解释的程序分析:采用数值区间抽象域、指针抽象域进行并发迭代分析,并计算可达访问集;

(3) 访问序模式匹配:在迭代过程中,根据可达访问集进行访问序模式匹配,获得数据竞争检测结果.

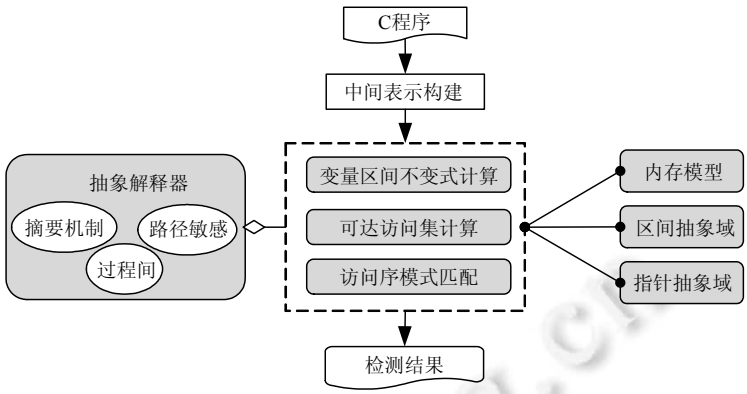

Fig.4 SpaceDRC system

图 4 SpaceDRC 系统

2.1 中间表示构建

给定软件源代码,SpaceDRC 对其进行语法分析、符号表系统构建、控制流分析,最终产生一种基于程序控制流图的中间表示.控制流图包含以下 4 种类型的结点:

- 赋值结点:$e=e'$,其中,e 和 e'是表达式;
- 调用结点:$e=f(e_1,\ldots,e_k)$,其中,e,e_1,e_k 都是表达式.当调用函数未使用返回值时,赋值左侧的 e 可能为空;
- 返回结点:return e,其中,e 是表达式;
- 假设结点:assume e,其中,e 是逻辑表达式,表明该结点的后继结点都符合约束(e==true);前端将 if 语句、while 语句、for 语句、switch 语句都转换为了假设结点.

各个结点中的所有表达式 e 都是无副作用的规范表达式,其定义说明见表 2.

Table 2 Canonical expressions generated by SpaceDRC frontend

表 2 SpaceDRC 前端产生的规范表达式

表达式	*e*:=*numconst*|*stringconst*|*addrconst*|*deref*|*fieldAccess*|*binaryExpr*
数值常量表达式	*numconst*:=*intconst*|*floatconst*|*doubleconst*
字符串常量表达式	*stringconst*:="*string*"
地址常量表达式	*addrconst*:=& *var*
解引用表达式	*deref*:=***e*
成员访问表达式	*fieldAccess*:=#*f*(*e*)
二元表达式	*binaryExpr*:=*e* op *e'*

其中,地址常量表达式中的 *var* 表示变量标识符,成员访问表达式中的 *f* 表示要访问的成员变量的标识符,二元表达式中的 op 表示 C 语言支持的二元操作符.

2.2 基于抽象解释的程序分析

本文设计了一个支持过程间分析、跨中断分析、路径敏感和摘要机制的抽象解释器,该抽象解释器采用 WTO(weak topological ordering)迭代求解,支持加宽和收窄算子的扩展.

本文主要的分析都是抽象解释的具体分析实例.与一般的抽象解释器不同:

- SpaceDRC 的抽象解释器针对中断并发进行了扩展,即,考虑中断并发对程序抽象状态的影响;
- 支持简单整数线性约束求解,因而分析是路径敏感的;
- 为了提高性能,抽象解释框架采用了摘要机制,即,对每个函数、每个中断的迭代求解结果进行摘要,避免多次迭代中重复分析.

根据中断驱动型嵌入式软件的特征,本文采用了一种简单且有效的内存模型,在变量区间不变式计算和可

达访问集计算中,仅使用数值区间抽象域和指针抽象域,因此保证了效率.

2.2.1 内存模型与程序抽象状态

我们定义了一套类型化的内存模型,支持嵌入式程序中的各种常用结构,如动态分配、基本类型变量、直接地址访问、字符串、区间偏移(数组和指针)、复合类型域.表 3 给出了 SpaceDRC 内存模型的定义.

Table 3 Memory model of SpaceDRC

表 3 SpaceDRC 采用的内存模型

	内存类型	定义	示例
内存区域	*Region*	*AllocRegion*|*VarRegion*|*AddrRegion*|*StringRegion*|*OffsetRegion*|*FieldRegion*	
动态分配	*AllocRegion*	(*type*)*malloc_i*	*int*p=malloc*(*sizeof*(*int*)); (*int*)*malloc_*0
变量	*VarRegion*	(*type*)*var*	*int x*; (*int*)*x*
直接地址访问	*AddrRegion*	(*type*)*address*	(*unsigned char volatile**)0*x*200000 (*unsigned char*)0*x*200000
字符串常量	*StringRegion*	*StringContent*	*char*str*="*hello*,*world*"; "*hello*,*world*"
区间偏移	*OffsetRegion*	*Region*+*offset*	*int arr*[10]; *arr*[5]→(*int*[10])*arr*+[5,5]
复合类型域	*FieldRegion*	*Region.f*	*struct point*{*int x*;*int y*}*p*; *p.x*→(*struct point*)*p.x*

基于上述内存模型,定义程序的抽象状态为内存区域到特定抽象值的映射.对于变量区间抽象域,为内存区域到区间值的映射.例如,⟨*i*,[0,2]⟩表示内存区域 *i* 的值为区间[0,2];对于指针指向抽象域,其抽象状态为内存区域到内存区域集合的映射.例如,⟨*p*,{*malloc*_0,*m*+[2,2]}⟩表明指针 *p* 指向 *malloc*_0 和 *m*[2];

在内存模型和程序抽象状态定义的基础上,SpaceDRC 通过抽象解释计算每个程序点处每个内存区域的值区间不变式.由于区间抽象域简单高效,并且足以刻画本文所关注的中断数据竞争采用的自定义标志变量,因此在实践中非常有效.

2.2.2 过程间分析和中断并发分析

SpaceDRC 采用的抽象解释框架支持过程间分析,并将过程间分析扩展到中断并发分析.对每个子过程的分析都采用摘要机制以提高效率.

假设 *iterator*(*f*,*i*)为迭代求解算法,其中,*f* 为被分析函数,*i* 为迭代的初始状态.

对中断触发和函数调用的处理如下:

- 中断触发

对于任意程序点 *p*、中断处理函数 *isr*,若当前迭代中 *p* 处的程序抽象状态为 S_1,E_{isr} 表示中断 *isr* 的使能约束,则以 $E_{isr}\sqcap S_1$ 作为中断服务函数 *isr* 的过程间迭代分析入口状态.若 $E_{isr}\sqcap S_1$ 不为⊥时,则以 $E_{isr}\sqcap S_1$ 作为初始状态对中断处理函数 *isr* 进行迭代分析;否则,表明中断无法触发,不在 *p* 点对 *isr* 进行迭代分析.*isr* 的出口节点状态为 S_2=*iterator*(*isr*,$E_{isr}\sqcap S_1$),则中断返回点状态为 $S_1\sqcup S_2$,这用于刻画中断并发的不确定性.

- 函数调用

对任意调用点 *c*,被调用函数为 *f*,若当前程序抽象状态为 S_1,则通过迭代分析获得函数 *f* 的出口状态为 S_2=*iterator*(*f*,S_1),则返回点的状态为 S_2.

SpaceDRC 对函数调用和中断并发的处理如图 5 所示.

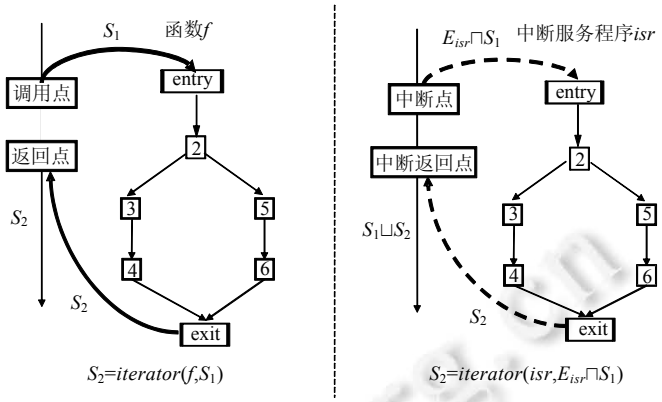

Fig.5 Handling of function calls and interrupt in SpaceDRC

图 5 SpaceDRC 对函数调用和中断并发的处理

2.2.3 可达访问集计算

本文提出了可达访问的概念,将检测并发路径中变量访问序模式的问题转换为一个迭代不动点求解问题.与可达定义(reaching definition)不同,可达访问的定义如下:

定义 1(可达访问). n_i为变量x在n_j的可达访问,当且仅当存在一条有效路径序列p,表示为$[n_1,n_2,\ldots,n_m]$,且在路径p中n_i为$n_j(1\leqslant i<j\leqslant m)$之前对$x$访问的最近结点,记为$n_i\rightarrow^x n_j$.

给定结点n,关于x的可达访问集表示为

$$\mathbb{R}_{x,n}=\{(n',\alpha)|n'\rightarrow^x n\}.$$

其中,α为n'结点对x的访问方式,$\alpha\in\{r,w,rw\}$.

可达访问集可以构造出变量的访问链,从而与访问序模式进行匹配,识别可疑数据竞争.

可达访问集的计算与可达定义类似,本文定义了一个可达访问集的抽象域,在抽象解释框架下,可以高效地求解得到每个结点、每个共享变量的可达访问集.

2.3 访问序模式匹配

访问序模式的匹配是在抽象解释迭代求解过程中进行的,这避免了不动点到达后程序状态的过度近似引入的误报.

设$(p,\alpha_p),(r,\alpha_r),(c,\alpha_c)$是对共享变量$x$的 3 次连续访问,即:

$$(p,\alpha_p)\in\mathbb{R}_{x,r},(r,\alpha_r)\in\mathbb{R}_{x,c}.$$

若p和c为主程序中的结点,r为中断服务程序中的结点,且$\alpha_p\rightarrow\alpha_r\rightarrow\alpha_c$满足访问序模式集中的某种模式,则认为这是一次有害数据竞争.

图 6 给出了在抽象解释迭代求解过程中进行访问序模式匹配的算法.其基本思想是:在对每个控制流结点进行分析时,通过当前获得的可达访问集追溯给定共享内存的长度为 3 的访问链,当访问链中第 2 次访问为远程访问(即,在高优先级并发流中访问)、并且与本文提出的 4 种访问序模式相匹配时,即认为该 3 次访问构成了一次可疑的有害数据竞争.

例如,图 7 所示程序来自于某卫星电源软件,我们对其进行了简化.其功能之一是:每秒钟通过端口 IN 对电压传感器进行遥测数据采集,经过计算后,将数据通过端口 OUTPUT 发送到地面.为了实现这一功能,设计师采用一个同步标志变量 *fgTv*28 来标记是否需要进行数据的处理,并通过一个每秒触发的定时中断(即,秒中断)对 *fgTv*28 进行设置(赋值 0*xAA*).主程序周期性地判断变量 *fgTv*28 的值,当值为 0*xAA* 时,表明需要进行下一秒的处理,于是对 *currentPower*28 数据计算并发送.由于软件每个周期要完成的计算任务较多,在设计程序的处理逻辑时,对 *currentPower*28 的计算和发送并没有连续进行,甚至并不处于同一个函数中,即,S_1和S_2中的处理并不是连

续的.然而这种设计是有缺陷的,设计师错误地假设了 S_1 和 S_2 的判断总是一致的.而当 S_1 不成立,秒中断触发,随后 S_2 成立时,就会将未经计算的 *currentPower*28 发送出去,导致在地面遥测看到连续两秒的数据是重复的.同时,由于该数据可能被用于反馈控制,数据错误的后果可能是很严重的.

```
1   算法. 访问序模式匹配算法 CheckDataRace.
2   输入:控制流结点 n;
3   输出:可疑数据竞争访问序集合 raceSet.
4   patternSet:={(r,w,r),(w,w,r),(r,w,w),(w,r,w)}
5   raceSet:=∅
6   for each accessed memory x in n do
7       αn:=access mode in n
8       for (r,αr)∈ℝx,n do
9           if r is a remote access then
10              for (p,αp)∈ℝx,r do
11                  if patternSet contains (αr,αp,αn) then
12                      raceSet ∪=(r,p,n)
13                  fi
14              od
15          fi
16      od
17  od
```

Fig.6 Algorithm of access order pattern matching

图 6 访问序模式匹配算法

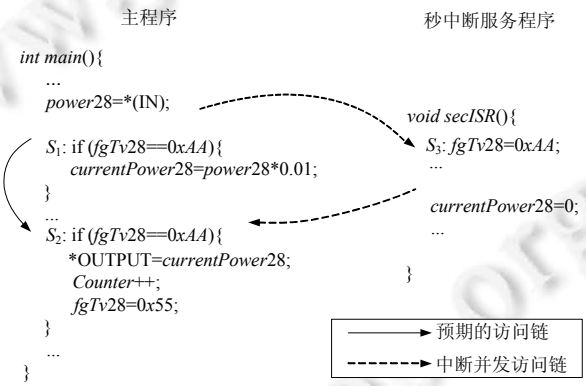

Fig.7 A Bug instance matched by pattern (*R*,*W*,*R*)

图 7 违反(*R*,*W*,*R*)访问序模式的数据竞争实例

上述案例是典型的中断数据竞争问题,可以用 3 次访问序模式来刻画,即,S_1,S_3,S_2 对 *fgTv*28 的访问构成了(*R*,*W*,*R*)模式.即,$S_1 \rightarrow^{fgTv28} S_3$,$S_3 \rightarrow^{fgTv28} S_2$.

3 实验评估与应用

本文从航天型号软件数据竞争案例集中选取了 4 个真实的中断驱动型嵌入式软件,所选取的软件版本中分别存在 1 个典型的中断数据竞争缺陷.这些缺陷都是经过单元测试、组装测试、确认测试和第三方测试之后被遗漏的,在实际工程中造成了严重的软件质量事故,是典型的难以被传统测试手段发现的并发缺陷.

所选取的被测软件都是安全关键的嵌入式软件,其中,2 个软件运行在 Intel MCS-8051 单片机平台,规模较小,是典型的主程序轮询式中断驱动型程序,用于关键敏感器数据采集和遥测下传;一个软件运行在 SPARC V7 系列的 TSC695 平台,规模相对较大,采用实时多任务操作系统与中断驱动的异构并发模式,主要功能是航天器的中央数据处理;另一个运行在 TI DSP c3x 平台,规模较大,采用主程序轮询和中断驱动的并发模式,用于 GPS

等数据的实时处理和计算.

3.1 实验结果及其分析

我们采用 SpaceDRC 工具对以上 4 个软件进行了分析,并与本文前期工作 H-RaceChecker[3]的结果进行了对比.分析所用的计算机配置为 Intel Core i5 3.20 双核、8GB 内存,两个工具都运行在 Java 平台,JVM 内存大小设置为 512MB.分析结果见表 4.

Table 4 Evaluation results of SpaceDRC

表 4 SpaceDRC 实验评估结果

软件	平台	规模/LOC	分析结果						
			共享变量	数据竞争结果		涉及共享变量		分析时间(ms)	
				SpaceDRC	H-RaceChecker	SpaceDRC	H-RaceChecker	SpaceDRC	H-RaceChecker
XWJ	8051	2 312	112	1(3)	1(212)	1	13	8	4
XWJ2	8051	2 849	20	1(9)	1(73)	9	15	178	157
CTU	TSC695	21 018	427	1(4)	1(57)	2	39	30	32
TDB	c3x	21 451	212	1(21)	1(396)	19	24	145	135

从表 4 的实验结果可以发现:

(1) SpaceDRC 非常高效地发现了所有存在的数据竞争缺陷.

SpaceDRC 通过变量访问序模式匹配在 4 个软件中分别发现了 3 个、9 个、4 个、21 个可疑数据竞争,这些数据竞争中包含了选取的 4 个典型缺陷.同时,SpaceDRC 进行数据竞争检测的时间仅需最多数百毫秒,如:对软件 CTU 进行分析的时间仅 30ms;最长的分析时间为软件 XWJ2,也仅为 178ms,这表明本文提出的分析方法非常高效.

(2) 与 H-RaceChecker 相比,SpaceDRC 的误报率显著降低.

以 XWJ 为例,H-RaceChecker 报告 212 次可疑竞争,而 SpaceDRC 仅报告 3 次.对于其他 3 个被测软件,误报次数同样有大幅度减少.这极大地减少了人工确认的工作量,进一步提高了工具的可用性.误报率显著降低的主要原因是:中断驱动型嵌入式程序中绝大多数的数据竞争是良性的,SpaceDRC 的检测针对有害数据竞争缺陷模式,而 H-RaceChecker 针对一般数据竞争,因而结果差异巨大.在分析效率方面,由于使用了更精确的分析,SpaceDRC 略低于 H-RaceChecker,但并没有量级的差别.

(3) SpaceDRC 的分析结果大大缩减了可疑共享变量.

在目前航天型号软件开发过程中,由于缺乏有效的针对中断数据竞争的工具,开发与测试团队多通过对共享变量的人工分析来发现中断数据竞争.由于软件设计的需要,程序中采用了大量的共享变量,例如在软件 CTU 中,共享变量多达 427 个,若测试人员逐一进行分析是非常耗时的.SpaceDRC 给出的可疑数据竞争结果将共享变量的关注缩小到非常少的变量集合,如:CTU 软件中,427 个共享变量中仅 2 个变量的访问存在可疑竞争;在软件 XWJ 中,112 个共享变量中仅 1 个有关.这对提高人工分析的效率有很大的帮助.

3.2 误报分析

SpaceDRC 检测结果存在一定的误报,本文对所有误报进行了分析,并按照误报类别进行举例讨论.由于在不同的被测软件中误报分布的表现差异较大,并无统计规律,因此本文并未对各种误报类别的分布情况进行数据分析.

- 误报情况 1. 数据竞争所涉及的访问序在实际执行中不可达.

图 8 是该类误报的典型样例.SpaceDRC 报告 S_1,S_2,S_3 这 3 处对变量 *onoffnum* 的访问构成了(*R*,*W*,*R*)模式,即:由于 CAN 中断中 S_2 的执行,使得在 S_3 处读取的 *onffnum* 值与 S_1 处所取值不一致.然而,S_2 所在的分支约束(*fgExecute*==0)与 S_1 所在的分支约束(*onoffnum*>=0 && *fgExecute*==1)相矛盾.在实际程序运行中,若中断在 S_1 语句后打断主程序,S_2 所在语句将不可达,因此该数据竞争为误报.

在中断驱动型程序中,采用标志变量进行同步是常用的设计模式,SpaceDRC 引入了区间抽象域和整数线

性约束求解,能够处理大多数的该类情况,从而避免误报.但图 8 所示:程序中,*ONOFFManage* 函数在 CAN 中断中的调用层次较深,且存在多次调用,SpaceDRC 采用了摘要来提高过程间分析的效率,这会引入对上下文抽象状态的过近似,从而无法裁剪这种不可行路径,导致误报.

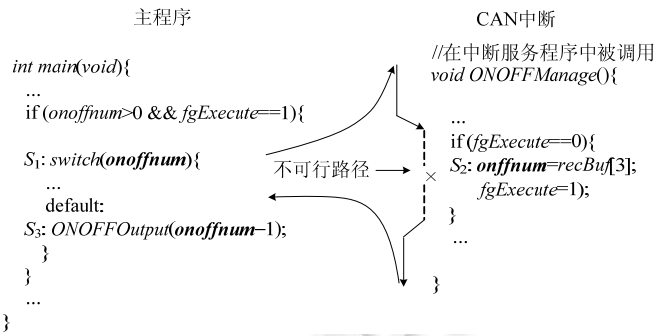

Fig.8 Fasle alarm related to infeasible path

图 8 不可行路径导致的误报

当 SpaceDRC 不使用函数摘要时,迭代分析常常难以终止.因此,我们在精度和效率进行了折衷,这是导致该类误报的本质原因.下一步,我们将探索更精确、高效的分析方法来消除该类误报.

- 误报情况 2. 数据竞争所涉及的中断并发时序不可能发生.

图 9 所示程序给出了一个典型案例.SpaceDRC 报告 S_1,S_2,S_3 这 3 处对 dataReady 的访问构成了(*R*,*W*,*R*)模式.假设该时序确定发生,则中断中对 dataReady 的设置在返回后将被 S_3 处语句清除,这将导致某一周期采集的数据不会被主程序处理.然而,该程序中隐含的时序约束是:1) 主程序每 100ms 循环一次;2) 中断每秒触发一次.这种时序约束,保证了中断每次采集(时刻为 t_1)的数据将在 100ms 内被主程序处理(时刻为 t_2,$t_2-t_1<100$ms),即,S_1 分支条件为 true 的情况.在此期间(t_1~t_2),秒中断并不会再次触发;在其他时间段,S_1 分支条件为 false,即使中断触发也无法构成(*R*,*W*,*R*)模式.因此,这是一个误报.

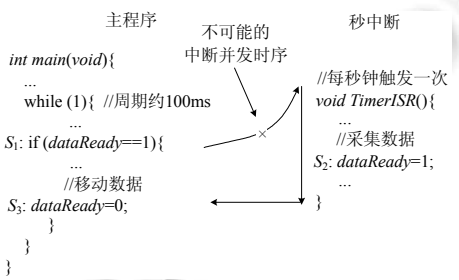

Fig.9 False alarm related to nonexistent interrupt interleaving

图 9 不可能的中断并发时序导致的误报

对于中断驱动型嵌入式软件的设计,硬件和外部系统的时序设计约束是需要考虑的关键因素.因此,软件的设计中往往隐含了一些时序约束.而这种约束并不体现在代码中,对于以源代码为唯一分析对象的工具,该类误报很难消除.未来可考虑结合由用户创建的系统时序模型进行精确检测.

- 误报情况 3. 良性的数据竞争.

本文提出的中断数据竞争缺陷模式较精确地刻画了真正有害的中断数据竞争,但由于中断数据竞争的 Ad-hoc 特征,仍然存在一些特殊的情况,即使符合访问序模式,但并不导致程序缺陷.

图 10 给出了一个典型案例.SpaceDRC 报告 S_1,S_2,S_3 符合(*R*,*W*,*R*)模式,即:当 S_1 对 *DCF_NUM* 的取值范围进

行判断后,程序在 S_3 以 *DCF_NUM* 作为循环上界对数组 *GNC_DCF_CLOSE*[10]进行访问,若在 S_1 和 S_3 之间被中断 *Int_GNC* 打断,*DCF_NUM* 值可能被更改,则 S_3 处的循环上界 *DCF_NUM* 取值可能并不在[1,10]范围内,这与 S_1 处的分支约束不一致并且可能会导致数组越界.然而,设计师考虑了这种数据竞争,并设计了容错机制,即:通过一个标志变量 *INGNC_setbit* 来标识高级中断中是否更改了 *DCF_NUM*,在 S_3 的每次循环中,都首先对 *INGNC_setbit* 进行了判断,当判定中断更改 *DCF_NUM* 时,表明在 S_1 之后发生了数据竞争,此时,低级中断直接返回.这种容错设计避免了数据竞争导致的数组越界,因此,图 10 的案例属于良性数据竞争.

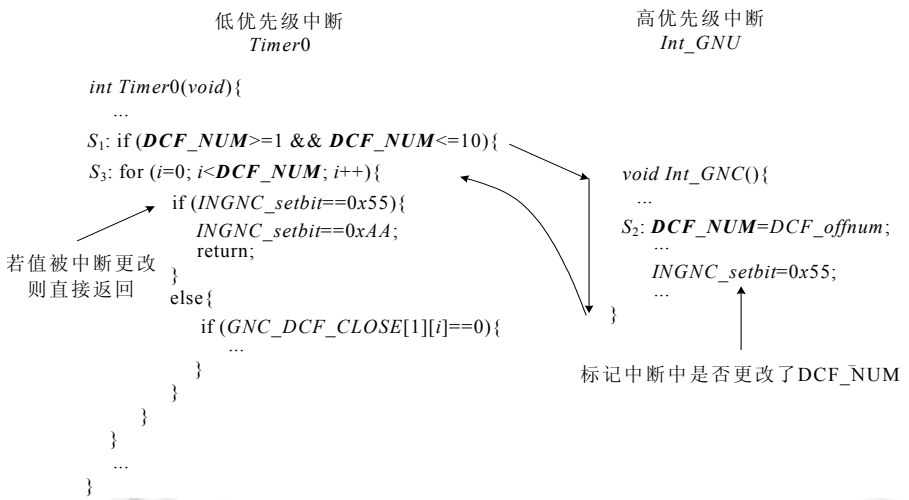

Fig.10 Benign interrupt data race example

图 10 良性数据竞争实例

在 SpaceDRC 的检测结果中,有相当一部分的误报属于良性数据竞争的范畴.对这类误报的人工确认相对容易,因为设计师早在程序设计时已经考虑到了这种情况.

3.3 工程应用情况

目前,SpaceDRC 已经在载人航天、探月工程等重点型号的关键软件研制以及空间飞行器软件第三方独立测试中进行了初步应用,有效弥补了当前工程实际中缺乏有效工具的不足.根据应用方的反馈,采用 SpaceDRC 工具之后,中断数据竞争专项分析的效率提高了至少 5 倍,并且降低了问题遗漏的可能.

4 相关工作

数据竞争检测方法一直是研究的热点,但大多数都针对多线程、多任务程序,主要包括基于 Lockset 的方法[5−7]、基于 happen-before 关系[6,9,10]的方法以及混合型方法[10,11].

在与中断有关的数据竞争检测方面,可见的成果并不多.Regehr 等人[12]提出了一种巧妙的验证中断驱动型程序并发正确性的思路,即:首先将中断驱动型程序转换为基于线程的程序;然后,用已有的针对多线程程序并发正确性验证的工具进行分析.但是该方法并不严格,需要进一步提供形式化语义表述.Henzinger[13]对 nesC 的检测方法进行了扩展,验证非原子的变量访问是安全的.Mercer 和 Jones[14]提出了一种结合 GDB 调试器的状态保存和恢复功能进行中断驱动型嵌入式代码模型检验的方法.

在中断驱动型程序的基础程序分析和验证技术方面,目前见诸文献的研究成果也较少.Palsberg 等人[15]提出了一个类型中断演算系统,利用静态类型系统保证堆栈边界,并且支持模块的类型检查,但该演算系统并不是一个完整通用的程序逻辑框架.Cooperider 提出的一种针对中断驱动型程序的并发数据流分析方法 ICD (interatomic concurrent data-flow)[16]是一个可自定义抽象域的数据流分析框架,并且对嵌入式程序中一些特有的语言特征都有支持.Kroening 等人[17]针对多重中断驱动型嵌入式程序提出了一种有效的形式化验证方法,即:

将程序转换为原子的内存读写事件;然后,将所有事件可能的交迭编码为符号化偏序关系,再用 SMT/SAT 求解来实现验证.该工作虽不针对中断数据竞争检测,但为本文工作进一步降低误报提供了新的思路.Liu 等人提出了一种针对中断驱动系统的领域建模语言 iDola[18],能够准确地描述复杂的中断处理机制,并能将模型转换为可验证的时间自动机和平台特定的 C 代码,为采用模型驱动开发方式构建中断驱动型系统提供了新的途径.文献[19]针对中断驱动系统提出了一种参数化的建模方式,通过自动地将参数化模型转换为时间自动机进行模型检验,支持共享资源完整性和子程序原子性的检测.与本文面向源代码模型不同,该文面向人工构建的中断并发时序模型,SpaceDRC 可考虑结合该模型进一步消除系统时序设计相关的误报.

文献[20–22]研究的问题与本文最接近.其中,文献[20]提出自动将并发中断程序转换为非确定性的顺序程序进行数据竞争检测,然后收集竞争发生的路径条件.在此基础上,通过有界模型检验将路径条件转换为 SMT 公式,然后判定该路径公式是否可满足从而消除误报.该文基于 CMBC 实现了原型工具,实验效果也较好.文献[21]实现了一个 Darco 的工具,充分利用中断驱动程序并发语义的特性,使用精确计算中断允许集结合中断副作用分析的方法进行了数据竞争检测.为了提高精确性,采用了流敏感、上下文敏感的过程间分析方法,并提出一个中断特征描述语言以对程序进行标注.Darco 对 17 850 行代码分析仅需要数秒,并且比传统的基于锁集的算法精确度提高 2.13 倍.文献[22]提出了一种基于控制流图的静态检测方法:首先,通过预处理分析检测程序中使用的共享资源和中断使能操作;然后构建简要控制流图,并进行抢占关系分析,得到可能的竞争关系执行序列.该方法可以帮助软件测试人员进行确认和修复.本文与这些研究的不同在于:根据中断驱动型程序的特点和中断数据竞争缺陷模式提出了针对性的检测方法,能够更精确地捕获有害数据竞争.

文献[3]是本文工作的基础,主要采用过程间数据流分析和启发式缺陷排序进行数据竞争检测,取得了不错的应用效果.与该项工作相比,本文做了大量的改进:一方面,通过系统的缺陷特征研究发现,绝大部分的中断数据竞争是良性的,因此,本文的检测方法面向的是有害缺陷模式,与前期工作仅针对数据竞争不同;另一方面,本文提出的检测方法基于抽象解释,引入了区间抽象域和简单整数线性约束求解,有效地处理了中断驱动型程序中大量的自定义同步变量,与前期工作采用数据流分析不同.

5 总 结

如何精确地检测中断数据竞争缺陷,是工业界和学术界研究的热点,但目前仍缺少适用的研究成果.一方面,常见的用于多线程数据竞争检测的方法,如锁集算法,几乎都基于特定同步机制,并不适用于中断数据竞争;另一方面,绝大多数中断数据竞争是良性的,因此,面向数据竞争定义的检测方法不可避免会出现大量误报.鉴于此,本文以航天嵌入式软件真实数据竞争案例为基础,系统地研究了中断数据竞争的特征,提炼出 7 种缺陷模式来准确刻画有害中断数据竞争;然后,针对其中最常见最易遗漏的单变量访问序模式,提出基于变量访问序模式的检测方法.该方法采用抽象解释计算检测所需的各种不变式,在迭代过程中将变量访问链与给定缺陷模式进行匹配实现数据竞争检测.本文的抽象解释采用了较简单的数值区间抽象域和适用于嵌入式软件的内存模型,考虑了过程间摘要和中断并发,因此在保证一定精度的情况下仍然非常高效.本文实现了中断数据竞争检测工具 SpaceDRC,并在 4 个真实航天嵌入式软件上进行了实验评估.结果表明,SpaceDRC 能够非常高效地检测出真正的数据竞争缺陷.同时,SpaceDRC 能够将中断数据竞争人工分析时对共享变量的关注集缩小到很小的范围.本文对 SpaceDRC 的误报情况进行了深入探讨,为进一步改进研究提供了基础.在多个航天型号软件研制中应用的结果表明,SpaceDRC 显著提高了中断数据竞争专项审查的效率.

References:

[1] Leveson NG, Turner C. An investigation of the Therac-25 accidents. IEEE Computer, 1993,16(7):18–41. [doi: 10.1109/MC.1993.274940]

[2] US-Canada power system outage task force. Final Report on the August 14. 2003 Blackout in the United States and Canada: Causes and Recommendations, 2004. http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/BlackoutFinal-Web.pdf

[3] Duan YH, Chen R. Heuristic static data race detection for interrupt-driven software. Computer Engineering and Design, 2013,34(1):140−145 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-7024.2013.01.027]

[4] Chen R, Guo XY, Duan YH, Gu B, Yang MF. Static data race detection for interrupt-driven embedded software. In: Proc. of the Int'l Conf. on Secure Integration and Reliability Improvement. IEEE Reliability Society, 2011. 47−52. [doi: 10.1109/SSIRI-C.2011.18]

[5] Pratikakis P, Foster J, Hicks M. LOCKSMITH: Context sensitive correlation analysis for race detection. In: Proc. of the ACM SIGPLAN 2006 Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 2006. 320−331. [doi: 10.1145/1133981.1134019]

[6] Li KQ, Chen SM, Zheng WJ. A dynamic detective algorithm based on lockset to solve multithreaded data race. Wuhan University Journal of Natural Sciences, 2000,46(3):289−292 (in Chinese with English abstract). [doi: 10.3321/j.issn:1671-8836.2000.03.008]

[7] Zhang LB, Zhang FX, Wu SG, Chen YY. A lockset-based dynamic data race detection approach. Chinese Journal of Computers, 2003,26(10):1217−1223 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2003.10.001]

[8] Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978,21(7):558−565. [doi: 10.1145/359545.359563]

[9] Savage S, Burrows M, Nelson G, Sobalvarro P, Anderson TE. Eraser: A dynamic data race detector for multi-threaded programs. ACM Trans. on Computer Systems, 1997,15(4):391−411. [doi: 10.1145/265924.265927].

[10] Wu P, Chen YY, Zhang J. Static data-race detection for multithread programs. Journal of Computer Research and Development, 2006,43(2):329−335 (in Chinese with English abstract). http://crad.ict.ac.cn/CN/Y2006/V43/I2/329

[11] O'Callahan R, Choi J. Hybrid dynamic data race detection. In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP 2003). 2003. 167−178. [doi: 10.1145/966049.781528]

[12] Regehr J, Cooprider N. Interrupt verification via thread verification. Electronic Notes Theoretical Computer Science, 2007,174:139−150. [doi: 10.1016/j.entcs.2007.04.002]

[13] Henzinger TA, Jhala R, Majumdar R, Sutre G. Software verification with Blast. In: Proc. of the 10th Int'l Workshop on Model Checking of Software (SPIN 2003). 2003. 235−239. [doi: 10.1007/3-540-44829-2_17]

[14] Mercer EG, Jones MD. Model checking machine code with the GNU debugger. In: Proc. of the SPIN Workshop on Model Checking of Software (SPIN 2005). 2005. 251−265. [doi: 10.1007/11537328_20]

[15] Palsberg J, Ma D. A typed interrupt calculus. In: Proc. of the 7th Int'l Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 2002). 2002. 291−310. [doi: 10.1007/3-540-45739-9_18]

[16] Cooprider N. Data-Flow analysis for interrupt-driven microcontroller software [Ph.D. Thesis]. University of Utah, 2008.

[17] Kroening D, Liang LH, Melham T, Schrammel P, Tautschnig M. Effective verification of low-level software with nested interrupts. In: Proc. of the 2015 Design, Automation & Test in Europe Conf. & Exhibition. 2015. 229−234.

[18] Han L, Zhang HH, Jiang Y, Song XY, Gu M, Sun JG. iDola: Bridge modeling to verification and implementation of interrupt-driven systems. In: Proc. of the Theoretical Aspects of Software Engineering Conf. (TASE). 2014. 193−200. [doi: 10.1109/TASE.2014.33]

[19] Zhou XY, Gu B, Zhao JH, Yang MF, Li XD. Model checking technique for interrupt-driven system. Ruan Jian Xue Bao/Journal of Software, 2015,26(9):2221−2230 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4713.htm [doi: 10.13328/j.cnki.jos.004713]

[20] Wu XG, Wen YJ, Chen LQ, Dong W, Wang J. Data race detection for interrupt-driven programs via bounded model checking. In: Proc. of the IEEE 7th Int'l Conf. on Software Security and Reliability-Companion (SERE-C). IEEE Computer Society, 2013. 204−210. [doi: 10.1109/SERE-C.2013.33].

[21] Huo W, Yu HT, Feng XB, Zhang ZQ. Static race detection of interrupt-driven programs. Journal of Computer Research and Development, 2011,48(12):2290−2299 (in Chinese with English abstract). http://crad.ict.ac.cn/CN/Y2011/V48/I12/2290

[22] Chen YJ, Shi JJ, Wang LZ, Li XD. Data race detection tool for interrupt-driven embedded system. Journal of Frontiers of Computer Science and Technology, 2015,9(8):914−912 (in Chinese with English abstract). [doi: 10.3778/j.jssn.1673-9418.1411052]

附中文参考文献:

[3] 段永颢,陈睿.基于启发式的静态中断数据竞争检测方法.计算机工程与设计,2013,34(1):140−145. [doi: 10.3969/j.issn.1000-7024.2013.01.027]

[6] 李克清,陈莘萌,郑无疾.一种基于锁集的多线程数据竞争的动态探测算法.武汉大学学报(自然科学版),2000,46(3):289−292. [doi: 10.3321/j.issn:1671-8836.2000.03.008]

[7] 章隆兵,张福新,吴少刚,陈意云.基于锁集合的动态数据竞争检测方法.计算机学报,2003,26(10):1217−1223. [doi: 10.3321/j.issn:0254-4164.2003.10.001]

[10] 吴萍,陈意云,张健.多线程程序数据竞争的静态检测.计算机研究与发展,2006,43(2):329−335. http://crad.ict.ac.cn/CN/Y2006/V43/I2/329

[19] 周筱羽,顾斌,赵建华,杨孟飞,李宣东.中断驱动系统模型检验.软件学报,2015,26(9):2221−2230. http://www.jos.org.cn/1000-9825/4713.htm [doi: 10.13328/j.cnki.jos.004713]

[21] 霍玮,于洪涛,冯晓兵,张兆庆.静态检测中断驱动程序的数据竞争.计算机研究与发展,2011,48(12):2290−2299. http://crad.ict.ac.cn/CN/Y2011/V48/I12/2290

[22] 陈园军,石浚菁,王林章,李宣东.中断驱动的嵌入式系统数据竞争检测工具.计算机科学与探索,2015. [doi: 10.3778/j.jssn.1673-9418.1411052]

陈睿(1984－),男,山西岚县人,博士生,工程师,CCF 会员,主要研究领域为程序静态分析,嵌入式软件测试.

杨孟飞(1962－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为控制计算机系统,嵌入式软件.

郭向英(1975－),男,高级工程师,CCF 会员,主要研究领域为嵌入式软件测试.