MUVI: Automatically Inferring Multi-Variable Access Correlations and Detecting Related Semantic and Concurrency Bugs

摘要：

在多种软件 bug 之中，语义和并发错误是最难检测的。

文章提出的方法，是通过代码分析来检测多变量相关交互过程中出现的问题：(1) inconsistent updates—相关的变量没有被一致地进行更新 (2) multi- variable concurrency bugs—相关联的存取操作在并发的程序中没有得到保护

MUVI (MUlti-Variable Inconsistency)自动检测程序中多变量的相关性存取 (access) 关系，同时发现其中的不一致更新操作和其他 concurrency bug

测试环境包括 Linux, Mozilla, MySQL 和 PostgreSQL。

MUVI 提出的背景：

1.动机：

软件的缺陷会很明显的降低软件的可依赖性。本文的研究主要在相关变量的更新和存取问题，用 variable access correlations 或者 variable correlations 来表示，意思是，相关的变量们是内在的被关联在一起的，他们的存在不是孤立的，只有在同时被读取的时候才能表达出一个完整的表述或者视图，相应的，他们在被更新时，也需要是同时被更新，而不能脱离了各自单独更新。

虽然这是一个很重要的问题，但是程序员却不会将 access correlation 写入到文档中，这导致一个问题，一旦程序交给了其他程序员，别的程序员无法意识到这么个关系，从而很容易就会破坏这个关系，出问题。更糟的是，因为这些变量仅仅是语义上相关，而不是数据上的依赖，传统的编译器无法分析出他们的相关性，从而无法检测这个问题。

这就会导致刚才提到的两个问题：(1) inconsistent updates—相关的变量没有被一致地进行更新 (2) multi- variable concurrency bugs—相关联的存取操作在并发的程序中没有得到保护

之前唯一一个解决 multi- variable concurrency bugs 的办法是，基于锁解决方法，一旦有相关变量在同一个 lock critical section 中被同时 access 了，那么他们将会被绑定到只能在同一个 critical section 中被 access，然而这个方法并不能起到作用，原因是片面的认为相关变量一定得同时被 access。

同时，现有的很多 concurrency bug 检测都没法对多相关变量进行检测:

- lock-set data race detectors ：只针对单一变量，无法检测多相关变量
- Simply doing race detection at a coarse granularity ：虽然可以针对共享的 object 进行检测，但是无法处理不同 object 中变量的相关，同时，也无法处理同一 object 内局部变量的相关。
- Atomicity violation bug detection tools ：考虑分检查 code region，能够检查到对于多个变量的 access，但是在推断出 code region 的时候，还是基于单一变量在考虑，因而还是没法解决多相关变量问题。

在所有相关变量的同步问题都正确的被处理了之后，出现上述 concurrency bug 的最主要的原因就是，本应在同一 atom region 的相关变量的操作被放到了不同的事务里处理了，也就是没有保证原子性。

最近的工作，例如 AtomicSet 和 Colorama 都注意到了这个问题，然而却提出要人工标识相关变量来解决这个问题，这会带来很巨大的人工代价。因此，很需要有能够自动化监测出相关变量并且找到其中的 concurrency bug 的工具，这就是这个工具的设计实现的动机。

通过静态程序分析和数据挖掘技术进行多变量相关性的推断

通过代码分析找出相关变量不一致更新的地方

改进两个经典的竞争检测方法 (lock-set 和 happens-before) ，用于检测多变量相关的数据竞争。

关于变量相关性的说明，变量基本通过以下的几种方式进行关联：

- Constraint Specification
- Different Representation
- Different Aspects
- Implementation-demand

两个变量之间的 access correlations 并不是指简单的同时更新，而是根据实现的细节情况具体分配

Access Together：使用源代码距离（代码相隔的行数）来衡量两个变量之间的 access 关系，同时以函数作为基础 uint 观察。也就是说，如果在同一个函数的两个操作（读或者写）之间的源代码距离比 MaxDistance 小，那么就认为他们两个是 together 的，同时 MaxDistance 是可以调整的

Access Correlation ： x 和 y 有操作相关性，如果 A1(x) ⇒A2(y), 当且仅当 A1(x) 和 A2(y) appear together at least MinSupport times and whenever A1(x) appears, A2(y) appears together with at least MinConfidence probability, where MinSupport and MinConfidence are tunable parame- ters, A1 and A2 can be, respectively, any of the three: read, write or AnyAcc。

相关性推断的步骤：

(1) Access Information Collection

(2) Access Pattern Analysis

(3) Correlation generation, pruning and ranking

Inconsistent update bug：

For any write(x)⇒AnyAcc(y) cor- relation, we examine the violations to it. All the functions that only update x without accessing y are treated as incon- sistent update bug candidates.

缺点：Of course, MUVI inconsistent update bug detection cannot solve all the multi-variable inconsistency problem. Since MUVI only considers access types (read or write) and not specific variable values, both false positives and false negatives could occur due to special variable values.

Multi-Variable Race Detection ：

由于经典的两种方法 Lock-Set 和 Happens-before 方法只能检测单变量，要实现对多变量的 data race 的检测，文章决定在这两种方法的基础上进行修改。

MUVI 的结果：

以抽取的样本来说，VariableAccessCorrelation 的误报率在 17%左右，主要的原因是（1）代码中的 macro 和 inline 函数需要特殊的方法进行处理；（2）某些变量是恰好才同时被更新，而不是真正的存在着相关性。

同样也会存在漏报的现象，有以下的原因：（1）有些真正的相关性由于数量太少，被忽略了；（2）有些相关性只有在特定的情况下才会成立。

Inconsistent Update Bug Detection ：确实能够发现新的而且是真的 bug，但是同时存在误报的现象由于以下的原因：（1）特例，有些情况下，相关的变量确实不需要同时更新；（2）前一步的相关性分析出了问题，分析出了不存在的相关性，被 detection 所使用导致出错；（3）变量确实违反了相关性进行更新，但是后续不存在对其的读取操作，不会造成错误。

漏报现象，主要有两个原因：（1）有些真正的相关性没有被分析出来；（2）一些真正的 bug 在 bug 报告里面被排的很后，因此被忽略了。。

Concurrency Bug Detection ：

误报主要有两个原因：（1）分析出了不存在的相关性；（2）无害的多变量竞争（只有在特殊的情况下才会有竞争）

漏报的原因：有些相关性只有在特定的程序上下文才能得出，而 MUVI 无法检测

性能：Our MUVI extension only adds a small percentage of extra overhead on the original race detec- tors: 5.9%-40% for lock-set; and 1%-21% for happens-before。

结果分析：access correlations do not exist between any two random variables the correlated variables should be either used or updated consistently.

总结----可以扩展的点：

(1) Detect other types of multi-variable related bugs, such as read inconsistency, multi-variable atomicity violation bugs, etc.

(2) Improve MUVI correlation analysis and bug detec- tion accuracy via better code analysis.

(3) Extend MUVI to analyze dynamic traces to get run-time correlation.

(4) Evaluate more real-world applications.