**MUVI:Automatically Inferring Multi-Variable Access Correlation and Detecting Related Semantic and Concurreny Bugs**

**摘要**

在多种软件Bug中，并发错误是一个比较难检测的一个问题，本文主要是通过代码分析来检测多个变量在交互过程中产生的并发错误。 多变量引起的并发错误主要分为两种：

1. 相关的变量没有同时被更新（inconsistent updates）
2. 相关变量在进行存取操作时，没有被同时保护（multi-variable Inconsistency）

**之前的研究**

本文出现之前，已经有不少检测并发错误的方案（比较典型的有lockset data race detector,race detector at a coarse granularity,Atomicity violation bug detector），但都无法处理多变量的问题，究其原因，都是因为没能很好的检测出变量之间的一致性关系，在什么情况下是需要保证原子性而在什么情况下不需要。围绕着这两点，本文开始了研究

**具体方案**

**静态程序分析与数据挖掘技术结合**

**变量关联方式**

变量主要是通过以下四种方式进行关联:

1. 相互之间的约束关系(Constraint Specification)
2. 同一个事物的不同表现方法(Different Representation)
3. 表现同一个事物不同方面(Different Aspects)
4. 代码实现方法的要求(implementation demand)

这就揭示了多变量情况下,变量之间的操作的关联行为的复杂性,并不是简简单单的同时更新.

**Access Together**

**通过源代码距离(在代码中的相隔行数)来衡量两个变量之间的access关系,以函数为最小单位进行观察,在同一个函数内部,两个access之间的代码距离要是比设定的MaxDistance小,那么就认为这两个accee是together的,指的一提的是,这个MaxDistance是可以调整的.**

**Access Correlation**

**用Ai(v)来表示对于变量v的某次操作.如果 A1(x) ⇒A2(y), 当且仅当 A1(x) 和 A2(y)在MinSupport的时间内同时出现,而且两者同时出现的概率至少有MinConfidence这么高,那么就认定x和y存在Access Correlation,其中MinSupport和MinConfidence都是可以根据参数调整的**

Access Correlation通过以下的步骤进行推断:

1. Acess信息收集(Access information collection)，具体步骤： 通过解析源代码来收集每个函数的变量访问信息, 包括变量被访问的集合,被访问类型以及在代码中出现的位置,这些信息被存放 在 Acc\_Set 数据库中。MUVI 只考虑全局变量和结构体,通过访问类型(读或写)来对不同关联类型进行分类,通过在源代码中出现的位置(文件名和在代码中出 现的行数)来判断两次访问的同时性,访问来自函数本身还是它调用的函数为之 后的相关性关系修剪做支持。
2. 访问模式(Access Pattern)分析，具体步骤： 通过频繁集挖掘技术 FPclose 处理 Acc\_Set 数据库中的 数据,找到频繁同时出现的变量集,将这些结果作为相关性变量候选集。在频繁 集挖据技术中,将一组集合作为输入,找到出现频率大于 MInSupport 阈值的子 集。
3. Correlation的生成,筛选,排序，具体步骤： 利用 Acc\_Set 数据中的详细信息,产生和修 剪相关性集合,并对不同类型的相关性进行排名。

有以下四种情况关联关系会被去除

1. 如果一种关系(C)在函数(supporter) 中出现的次数不少于 MInSupport,定义为 support(C),但是满足这种关系的函 数小于 support 阈值;

2. 如果 support(C)/support(A1(x))<MInConfidence 阈值;

3. 直接 supporter 的数量小于 MinDirectSupport 阈值;

4. 一些常见的变量 stdout 和 stderr 等产生的关联性。

分析:

1. 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. 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
3. 存在漏报的现象,原因是  1.有些真正的相关性由于数量太少，进行排序的时候被放到了后面导致被忽略了
4. 2.有些相关性只有在特定的情况下才会成立

**Inconsistent update bug**

**通过代码分析找出相关变量之间不一致的更新**

对于任意的 write(x)⇒AnyAcc(y) correlation, 我们以这个为准则进行判断:

所有只更新x而不对y进行修改的函数,都被视为可能引起Inconsistent update bug的候选.

分析:

1. 确实能够发现新的而且是真的bug
2. 存在误报,原因是  1.特例，有些情况下，相关的变量确实不需要同时更新
3. 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
4. 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
5. 存在漏报,原因是  1.有些真正的相关性没有被分析出来
6. 2.一些真正的bug在correlation产生的bug报告里面被排的很后，因此被忽略了

**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。

分析: 1. 成功找到了多变量的data race bug 2. 存在误报,原因是

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）

1. 存在漏报,原因是  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**

**对于初始的算法只增加了很少的性能负荷,给lock-set增加的负荷是5.9%-40%,给happens-before增加的负荷是1%-21%.**

**得出的结论**

1.并不是任何两个变量之间都存在Access Correlation

2.关联的变量需要根据情况被相应地更新或者使用

**研究扩展点**

1.能够研究如何检测更多类型的并发错误(例如read inconsistency等)

2.修改correlation分析和bug detector以得到更加准确代码分析

3.支持动态分析

4.用来分析更多的现实项目以便于变得更加实用