软件分析技术——指针分析

周子锐 2100011032
谷东润 2000013190
喻浩南 2000013124

我们基于 Tai-e 实现了有 field-sensitive 和 context-sensitive 的指针分析，并对一些 corner case 做了处理

• 主要算法:

首先介绍非 context sensitive 版本. 我们实现的指针分析基于 Anderson 算法, 主要维护了一个表示各个对象可能的指向关系的 PointerFlowGraph. 该图上的节点要么是一个指针, 要么是一个(可能)包含 field 的对象. 该图上一点向另一点连边当且仅当在我们的分析中认为一点所可能指向的对象也可能是另一点所指向的.因此我们希望求出这个图的一个传递闭包来得到每个指针所可能指向的对象.同时基于此图,我们可以对每一个指针维护它可能指向的对象的集合, 在这个过程中,我们还会动态构建 CallGraph 来辅助分析.

我们维护了一个 WorkList 来实现上述的求传递闭包的流程, 其包括了若干对等待传播的指针和其当前指向的对象集合. 分析的大致流程如下:

(1)对于每一个需要分析的 method. 提取出于指针分析相关的句子, 根据需求从 HeapModel 分配 object, 根据语句信息初步建立 PointerFlowGraph, 将需要首次传播的指针和其指向的对象集合加入 WorkList 中. 在这个过程中我们同时处理与静态类型相关的信息.

(2)开始迭代,每次迭代过程中都从 WorkList 中取出一个指针和其可能指向的对象集合,找到当前指向的对象集合与上一次迭代时记录的集合间的差异, 将该差异传递到该指针在 PointerFlowGraph 上的后继节点. 之后如果该指针是一个变量的话, 则根据该变量所在 method 的 LoadField 和 StoreField 语句来更新 PointerFlowGraph. 再处理该 method 内涉及到该变量的所有函数调用,包括进行参数传递,this 指针传递和返回值传递. 在这个过程中动态更新 CallGraph 以达到更加精准的确定可能出现的函数调用. 注意, 在这个过程中可能会出现新的需要分析的 method, 此时就需要重复上一步.

(3)输出所求的指针对应的信息,

对于 context sensitive 版本, 我们尝试了一些可能的策略, 包括不同层数的 callsite 和 object 克隆, 最终采用了基于两层 callsite 克隆的分析. 此时在原来的指针分析上所有的指针和分析创造的 object 都额外包含了上两层的函数调用信息.

• 优化

基于上面的算法, 我们进行了如下优化:

(1)支持更多的语句: 当前我们支持 New 语句, Cast 语句, Copy 语句, 静态对象和实例对象的 LoadField 和 StoreField 以及四种函数调用(InvokeStatic, InvokeSpecial, InvokeInterface, InvokeVirtual)的语义分析. 其逻辑通过框架自带的访问者模式实现.

(2)对于 Cast,通过额外在 PointerFlowGraph 的每条边上记录若干 filter 来避免不合法的类型转换的发生.

(3)支持简单的数组指针分析.

• 代码结构:

我们的分析的实现代码在\src\main\java 下,分为个两部分:

(1)文件夹 cipta 中包括一个没有 context sensitive 的分析, 其主要实现逻辑参见该目录下的 Solver.java.

(2)文件夹 cspta 中为基于上一条的 context sensitive 的分析, 其主要实现逻辑参见该目录下的 Solver.java. 这也是最后用于测试的版本.

• 小组成员分工

周子锐: 主要分析流程实现及优化实现

喻浩南: 边界情况测试,生成测试样例,提供部分理论支持

谷东润: 边界情况测试,部分优化实现,撰写报告