软件分析技术报告

关智超 杨晨阳 袁野

2019/12/1

1 算法简介

指针分析算法是 Anderson 风格的，即基于约束的指针分析方法。基本算法包含四条规则：

1. $a = \textbf{new}\ A()$，将 a 指向该处堆的位置。
2. $a = b$，将 a 的指向集合替换为 b 的指向集合。
3. $a = b.f$，先找到 b 所有可能的指向位置，然后对所有这些位置的域 f 的指向集合求并，用来替换 a 的指向集合。
4. $a.f = b$, 先找到 a 所有可能的指向位置。如果只有一个，将其域 f 的指向集合替换为 b 的指向集合。否则，对每个可能指向位置的域的指向集合，将其并上 b 的指向集合。

处理域敏感时，我们记录了一个全局的 memory，用以记忆各个 heap 可能的指向位置。处理过程间调用时，我们使用了 on-the-fly 的建图方法，从而保证较好的上下文敏感性、流敏感性。

算法整体框架基于 soot 提供的 ForwardFlowAnalysis 类，并对各个语句的变换函数进行了修改。这个框架提供了 worklist 算法，每个节点维护一个 local 的指向集合，通过判断指向集合流入节点后是否变换，不断维护需要更新的节点。

2 算法细节

2.1 上下文敏感与流敏感

首先，我们使用 Soot 提供的 ExceptionalUnitGraph 的接口，建立 main 函数的局部控制流图（含异常处理）。我们使用 Anderson 类的一个实例化对象记录本次调用的各种局部信息（如方法名、参数表、调用控制），同时我们使用 Anderson 类的静态成员（查询记录、分配记录、静态变量字典、函数调用栈）来记录本次分析的共有信息。

然后在每次函数调用时，我们使用函数方法建立局部控制流，new 一个新的 Anderson 对象进行分析，传入当前参数（包括可能存在的 this 指针）的指向关系，并维护返回值的赋值。特别地，我们需要额外判断：

1. 系统调用，系统调用内部分析十分复杂耗时，我们不处理系统调用对 heap 的影响，对系统调用的返回值置为全集。

2. 递归调用，静态分析无法知道递归深度，我们发现调用函数已经在调用栈时，会将参数涉及到所有 heap 的相应域设为当前的全集，并将返回值（如果有）设为全集。

3. 分析需要的 alloc 和 test，前者我们用来维护当前 allocID，后者我们遇到时更新查询记录，并将当前的指向内容赋予查询表的对应项。

2.2 域敏感

域敏感基于对全局内存分配的维护，成为内存分配表（MemoryTable）。在任何时候新建一个未通过 Alloc 的对象时，我们会为其分配一个负数的 id（这个 id 在测试时会被输出为 0）。每一个内存标号会记录一个域到 TreeSet<Integer> 的映射，代表该域所指向的位置。在 Soot 生成的 Jimple 中，每个域的访问均被拆解为多次的域的访问。因此，只需要在内存分配表中逐一进行获取，得到最终指向的内存空间。

2.3 其余细节

本部分主要涉及各类赋值语句的处理，我们分成左值/右值两部分介绍。

2.3.1 右值处理

new 语句 我们首先判断当前 new 语句是否重复出现，是则使用第一次出现时赋予的 ID，这个处理可以避免死循环。然后我们检查是否有可用的 allocID（即上一句是否调用 alloc 函数），如果没有则赋予一个负值的匿名 ID（这样可以区分不同的 new），然后将 ID 加入右值集合，并更新 memory 表项。

对于一维数组和多维数组，我们额外在 memory 中初始化 arrayIndex，便于后续查找。后续判断时，我们不区分 a[k] 具体为 a 中的哪一项。

局部变量 对于局部变量和类型转换，我们使用流入的局部映射，直接将其可能指向的所有位置赋给右值集合。

域和数组 对于域和数组，我们使用 memory 找到它们的可能指向的位置，并将其赋给右值集合。对于静态域，我们查找静态域表，并将查询结果赋值给右值集合。

函数调用 具体处理见 2.1，我们将返回值赋给右值集合。

参数 对于参数和 this 指针，我们查找参数表，并将查询结果赋值给右值集合。

其余表达式 对于 NULL 和算术/逻辑表达式，我们返回空集。

对于其余可能的表达式，我们返回全集。如果上述表达式中有返回空集的（除了 NULL 和算术/逻辑表达式），则将其置为全集。

2.3.2 左值处理

局部变量 建立左值的字符串到右值集合的映射。

域和数组 更新 memory 或静态域表。

3 代码结构

- MyPointerAnalysis：程序调用入口（来源于原程序包）；

- WholeProgramTransformer：负责进入分析算法和输出（来源于原程序包）；
- Anderson：主要分析算法，负责控制流的分析、初始化、交汇等操作，同时负责分发不同的语句类型（来源于程序包）；
- AnserPrinter：用于输出结果到文件（来源于原程序包）；
- StmtHandler：抽象类，对各种语句对处理均继承自该类；
- DefinitionHandler：继承自 StmtHandler，处理赋值语句；
- InvokeHandler：继承自 StmtHandler，处理函数调用语句；
- ReturnHandler：继承自 StmtHandler，处理返回语句；
- InvokeExprHandler：对于函数调用具体语句的处理，创建一个新的运行时环境（Anderson）；
- PointsToMap：继承自 HashMap<String, TreeSet<Integer»，用于记录在某一运行环境中，变量所指向的内存区域（由 MemoryTable）管理；
- MemoryTable：管理全局内存分配；
- MemoryItem：每一内存项，存储在 MemoryTable 中；
- ArrayHelper：对于数组处理的辅助类，所有成员函数及方法均为静态；
- FieldHelper：对于域相关处理的辅助类，所有成员函数及方法均为静态；