软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, [doi: 10.13328/j.cnki.jos.000000]

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

# HarmonyFlow:基于方舟 Panda IR 的 HarmonyOS 应用静态分析框架 *

王越[1], 陈森[1], 李佳洺[1], 党文婧[1], 柴仁超[1], 石子跃[1], 黎立[2]

[1](天津大学 智能与计算学部,天津 300350)

[2](北京航天航空大学 软件学院,北京 100191)

通讯作者: 陈森, E-mail: senchen@tju.edu.cn

**摘 要**: 随着鸿蒙生态系统的快速发展,鸿蒙应用的安全问题逐渐成为研究重点.在安卓领域,已有多种成熟的静态分析框架广泛应用于安全检测任务.然而,针对鸿蒙应用的静态分析框架尚处于初步发展阶段.OpenHarmony 社区正在基于鸿蒙应用 ArkTS 源代码开展静态分析,但在实际的安全检测任务中,应用源代码往往难以获取,限制了其适用范围.为缓解上述问题,我们提出了一种基于方舟中间表示(Panda IR)的鸿蒙应用静态分析框架.该框架提供了方舟 Panda IR 的基本信息接口,设计了适应 ArkTS 语法特性的字段敏感指针分析算法,并实现了与指针分析交互的拓展分析接口.具体来说,我们对 Panda IR 中的 318 条指令进行了语义分类和处理,进一步定制化设计了指针流图.为了支持 ArkTS 语法特性,我们新增了指向集合传播规则,对特殊调用的相关语义进行准确建模.此外,我们基于指针分析结果优化了过程间数据依赖关系并提供了别名分析能力.我们从 ArkTS 语法特性覆盖性,指针分析精度和指针分析速度三个方面对 HarmonyFlow 进行了实验评估.实验结果表明,HarmonyFlow 可以正确处理 ArkTS 的关键语法,在 9 个开源鸿蒙应用上调用边识别的精确率和召回率分别为 98.33%和 92.22%,在 35 个真实鸿蒙应用上的平均运行时间为 96 秒.

**关键词**: 静态分析框架;鸿蒙应用;指针分析;方舟字节码;ArkTS 语言

中图法分类号: TP311

中文引用格式: 王越,陈森,李佳洺,党文婧,柴仁超,石子跃,黎立.HarmonyFlow:基于方舟 Panda IR 的 HarmonyOS 应用静态分析框架.软件学报,2025,32(7).http://www.jos.org.cn/1000-9825/0000.htm

英文引用格式: Wang Y, Chen S, Li JM, Dang WJ, Chai RC, Shi ZY, Li L. HarmonyFlow: A Static Analysis Framework for HarmonyOS Applications Based on Ark Panda IR. Ruan Jian Xue Bao/Journal of Software, 2025 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

## HarmonyFlow: A static analysis framework for HarmonyOS applications based on Ark Panda IR

WANG Yue[1], CHEN Sen[1], LI Jia-Ming[1], DANG Wen-Jing[1], CHAI Ren-Chao[1], SHI Zi-Yue[1], LI Li[2]

[1](College of Intelligence and Computing, Tianjin University, Tianjin 300350, China)

[2](School of Software, Beihang University, Beijing 100191, China)

**Abstract**: With the rapid development of the HarmonyOS ecosystem, security issues related to HarmonyOS applications have gradually become a key research focus. In the Android domain, various mature static analysis frameworks have been widely applied to security detection tasks. However, static analysis frameworks for HarmonyOS applications are still in the early stages of development. The OpenHarmony community is currently working on static analysis based on the source code of HarmonyOS applications using ArkTS. However, in practical security detection tasks, obtaining application source code is often difficult, which limits the applicability of this approach. To alleviate this problem, we propose a static analysis framework for HarmonyOS applications based on the Ark Intermediate Representation (Panda IR). This framework provides basic information interfaces for Panda IR, designs a field-sensitive pointer analysis algorithm that adapts to ArkTS syntax features, and implements extended analysis interfaces that interact with pointer analysis. Specifically, we categorized and processed 318 instructions in Panda IR semantically and further customized the design of the pointer flow graph. To support ArkTS syntax features, we introduced new propagation rules for pointer sets and accurately modeled the semantics of special calls.

* 基金项目: 国家自然科学基金(62472309)

收稿时间: 2025-02-25; 修改时间: 2025-05-06; 采用时间: 2025-05-26

In addition, based on the pointer analysis results, we optimized inter-procedural data dependencies and provided alias analysis capabilities. We conducted an experimental evaluation of HarmonyFlow in terms of three aspects: coverage of ArkTS syntax features, pointer analysis accuracy, and pointer analysis speed. Experimental results show that HarmonyFlow can correctly handle key ArkTS syntax. The accuracy and recall rates for call-edge identification in 9 open-source HarmonyOS applications were 98.33% and 92.22%, respectively, and the average runtime for 35 real-world HarmonyOS applications was 96 seconds.

**Key words**: Static analysis framework; HarmonyOS application; Pointer analysis; Ark Bytecode; ArkTS Language

鸿蒙操作系统作为国产操作系统的代表,凭借自研的方舟编译器,提供了包括编译器前端、工具链和运行时在内的关键组件,支撑着手机、平板、电视、汽车和智能穿戴设备等多种终端的应用与服务运行.其跨平台能力使开发者能够在不同设备上快速部署应用,推动了生态系统的快速发展.然而,随着鸿蒙系统在各类设备上的广泛使用,鸿蒙应用的安全性问题也日益成为关注的焦点.

静态分析是代码优化和安全分析等下游任务的基础,对缓解鸿蒙应用的安全问题具有重要意义.在安卓平台,经过长期发展,已经形成了一些成熟的静态分析框架,例如 Soot[1] 、Wala[2] 、Doop[3] 、Qilin[4] 和 Tai-e[5] 等.这些框架主要提供了中间表示(Intermediate Representation,IR)抽象、控制流图(Control Flow Graph)构建、指针分析(Pointer Analysis)以及别名分析(Alias Analysis)等基本功能,并广泛应用于安卓应用的静态分析场景.然而,这些框架的设计和实现主要针对安卓应用的 Java 源代码或 Java 字节码,面向鸿蒙应用的静态分析框架仍缺乏系统性的研究[6].

静态分析框架的输入主要包括源代码和字节码两种形式.OpenHarmony 开源社区正在基于鸿蒙应用的 ArkTS 源代码开展分析,提出了面向 ArkTS 语言的静态分析框架 ArkAnalyzer[7] .然而,在实际的安全检测任务中,应用的源代码往往不可获取,因此我们选择针对鸿蒙应用的方舟字节码进行分析.在静态分析中,选取适合的 IR 作为分析对象至关重要.不同静态分析框架通常采用特定的 IR,例如 Soot 基于 Jimple IR,Wala 基于 Wala IR.在我们的研究中,选择将鸿蒙应用字节码反汇编得到的 Panda IR 作为分析对象.Panda IR 作为方舟编译器代码优化过程中的中间表示,能够提供丰富的语义信息,也有助于增强我们的工作与方舟编译器代码优化流程之间的关联性.

在基于方舟 Panda IR 实现静态分析的过程中,我们面临以下三个主要挑战:1)**Panda IR 信息处理的复杂性**:指针分析需要聚焦于程序中的内存访问指令和函数调用指令,例如 New、Assign、Load、Store、Call 和 Return 等.然而,Panda IR 采用了独特的节点海(Sea of Nodes)[8] 结构,并经过了方舟编译器的优化处理.这种优化使得难以直接在 IR 指令与源代码语义之间建立映射关系,从而增加了提取关键指令信息的难度.将这些信息建模为指针流图(Pointer Flow Graph)成为一项复杂且至关重要的任务.2)**ArkTS 语法特性的复杂性**:鸿蒙应用开发在很大程度上依赖于 ArkTS 中的语言特性,例如函数对象和回调、作用域链与闭包等机制,以实现事件驱动的编程模式.这些特性会显著影响程序的控制流和数据流,为分析精确性带来极大挑战.为了保证静态分析能够准确捕捉程序行为,需要针对这些特性进行逐一适配和处理,从而提升分析的全面性和精确性.3)**静态分析框架的可拓展性**:静态分析框架需要具备良好的可拓展性,以支持集成下游分析任务,尤其是与指针分析交互的工作.设计一种机制能够有效提供指针分析的中间结果,并在此基础上构建下游分析任务,是提升静态分析框架功能性和易用性的关键.

本文首次提出了基于鸿蒙应用 Panda IR 的静态分析框架——**HarmonyFlow**,框架主要包含以下三个核心模块:Panda IR 信息处理模块、针对 ArkTS 语法的指针分析模块和拓展分析模块.1)Panda IR 信息处理模块:我们对 Panda IR 的 318 种指令进行了系统分析,筛选出与内存访问和函数调用相关的关键指令.在此基础上,提出了一套基于回溯的指令语义建模规则,结合指令间依赖关系及指令内部属性,逐一提取必要的语义信息.随后,针对关键指令构建局部指针流图,并整合为完整的指针流图结构.该结构具备通用性,既可作为字段敏感分析的基础数据结构,也为上下文敏感分析提供了有效的程序抽象.此外,指针流图融合了 ArkTS 的语言特性,例如将函数建模为特殊对象类型,以适应其函数作为一等对象且可包含字段的特性.2)针对 ArkTS 语法的指针分析模块:针对 ArkTS 特有的回调机制和显式绑定调用等语法特性,我们设计并补充了指向集合的传播规则,以确保对复杂

语法特性的精确建模.这些规则在现有面向 Java 的安卓平台指针分析算法中尚属空白,缓解了分析能力的不足.3) 拓展分析模块:在完成基础指针分析之后,HarmonyFlow 提供了针对变量指向集合的查询接口,以及调用指令目标函数的解析能力.基于上述结果,我们进一步实现了通用的别名分析能力.同时,结合指针分析所构建的精确调用图,并考虑 ArkTS 支持可修改外部变量的多样化闭包机制,我们优化了词法变量间的数据依赖建模.通过分析 Panda IR 中词法变量的作用域信息,并在调用图中回溯检索相关变量的最近修改位置,实现了对闭包语义下数据依赖关系的精准建模.总体来说本文的主要贡献如下:

- 通过对 Panda IR 的深入理解,提取了 Panda IR 中的控制流图和数据依赖关系,分类并处理了与内存访问和函数调用相关的关键指令.
- 基于 Panda IR 实现了字段敏感的指针分析算法,适配了 ArkTS 语言的语法特性,并同步构建了程序的调用图(Call Graph).
- 在指针分析的基础上,提供了别名分析接口,并优化了 Panda IR 的过程间数据依赖关系,提供了指令间的数据依赖关系接口.
- 针对鸿蒙应用首次提出了静态分析框架——**HarmonyFlow**,并在合成数据集、开源数据集和真实数据集上评估了其准确性与效率.HarmonyFlow 已成功落地应用.

本文第 1 节介绍静态分析框架和指针分析的相关研究.第 2 节介绍关于方舟 Panda IR 和 ArkTS 语法特性的基础知识.第 3 介绍本文的静态分析框架设计.第 4 节在合成测试用例、开源鸿蒙应用和真实鸿蒙应用上评估指针分析算法的准确性和效率.第 5 节讨论 HarmonyFlow 框架设计与实现中的局限性和发展方向.第六节对本工作进行总结.

## 1 静态分析相关工作

我们回顾代表性的静态分析框架,展示针对鸿蒙应用设计静态分析框架的重要性.关注基于不同 IR 和编程语言语法特性设计指针分析算法的工作,将 HarmonyFlow 置身于此研究背景,启发下文的框架设计思路.

### 1.1 静态分析框架

长期以来,静态分析框架主要应用于 Java 和 C/C++语言.近年来,针对 Rust 和 JavaScript 等语言的静态分析研究也逐渐兴起.首先介绍针对 Java 语言的静态分析框架[9] .Soot[1] 旨在优化 Java 字节码,设计了 BAF、Jimple IR 和 GRIMP 三种中间表示,并在 Spark[10] 和 Paddle[11] 模块中拓展了指针分析.Wala[2] 的核心目标是静态和动态的程序分析,基于 Wala IR 实现了过程间数据流分析和切片工具等.Doop[3] 基于 Soot 的 Jimple IR,通过 DataLog 语言实现了高效的静态分析工具.Qilin[4] 基于 Soot 提出了变量级别的上下文敏感指针分析算法.Tai-e[5] 根据以上框架遵循 HBDC(利用经典的最佳设计)原则开发了易用的高效系统.然后介绍针对 C/C++语言的静态分析框架.Phasar[12] 提出了基于 LLVM 的静态分析框架,可以自动化解决数据流问题.SVF 框架[13] 基于指针分析的指向信息实现了精确的过程间静态值流分析,便于客户端应用程序的跨过程边界分析值流(例如内存泄漏检测).最后,李等人[14] 还开发了针对 Rust 语言的指针分析和调用图构建工具.JSAI[15] 提供用户可选择分析敏感度的 Javascript 静态分析平台.综上,针对鸿蒙应用的静态分析框架亟待提出.

### 1.2 指针分析算法

指针分析是静态分析的基础工作,Soot 中提供了 SPARK 上下文不敏感指针分析模块和 Paddle 上下文敏感指针分析模块,WALA 中提供了 Andersen[16] 风格的流不敏感指针分析框架,Qilin 在 Soot 的基础上开发了细粒度上下文敏感指针分析框架.现有工作主要关注指针分析精度与速度之间的平衡,致力于通过增强敏感性[17] 提升分析精度,通过选择性上下文敏感分析[18] 和增量指针分析[19] 等加速分析速度.为了适应 Panada IR 设计和 ArkTS 语法特性,本文需要关注中间表示和源代码特性对指针分析影响的工作.

首先讨论中间表示对指针分析影响的工作.Guo 等人[20] 提倡在低级中间表示上进行指针分析,认为高级中间表示上的指针分析过程未考虑链接库和运行时加载代码,而且已有别名信息传播到低级中间表示时存在偏

差,准确性不及在低级别中间表示上重新进行指针分析.于是,Guo 等人提出了第一个在汇编级别上运行的上下文敏感、部分流敏感的指针分析方法.Prakash 等人[21] 发现在 Doop 同等精度的调用点敏感指针分析下,针对 Jimple IR 和 Wala IR 的指向分析结果在指向集合和效率方面存在差异.也就是说,不同的中间表示形式会对指针分析算法的设计和性能产生影响.

然后讨论源代码特性对指针分析影响的工作.当前不存在针对 ArkTS 语言的指针分析工作,由于 ArkTS 语言是 JavaScript(JS)语言的超集,因此我们关注针对 JS 语言的指针分析工作.Jang 等人[22] 提出了首个针对 JS 的指针分析方法.具体来说,他们提出了一种基于集合约束的分析框架,能够生成和解决对象动态特性(例如属性添加或更新)的约束.通过定义一种约束生成规则,分析器可以在 JS 程序中追踪属性的读写操作,进而计算准确的指向集合.Sridharan 等人[23] 分析了 JS 语言无类型声明、动态创建属性和函数参数的灵活性对指针分析结果的影响,针对 JS 的对象模型开发了字段敏感的指针分析方法.Feldthaus 等人[24] 提出了一种可扩展的基于字段的流分析方法,用于构建 JavaScript 程序的近似调用图.该分析方法通过对相同属性名的字段进行抽象,显著减少了需要分析的抽象对象数量,适用于现代集成开发环境(IDE)实现代码导航.

## 2 基础知识

### 2.1 Panda IR中间表示

方舟字节码是 ArkTS 编译后的二进制产物,其可以被进一步反汇编为可读的 Panda IR.Panda IR 被设计为节点海形式,具有全新的语法结构.下文介绍节点海 IR 的设计理念和 Panda IR 的具体信息.

**节点海 IR 设计理念**:节点海是控制流图与静态单赋值形式结合的进一步优化.其主要特点在于更清晰地表达程序的依赖关系,同时消除了控制流图中一些不必要的约束.

在传统 IR 中,控制流通过控制流图进行表示,而 IR 本身则侧重于表示数据流.以 LLVM IR[25] 为例,其控制流图由基本块节点和有向跳转边组成.基本块为连续的三地址码序列,控制流只能从该序列的起始指令进入,并且只能从该序列的最后一条指令退出.从基本块 A 到基本块 B 之间通过有向跳转边连接,跳转关系成立的条件是:存在有条件或无条件跳转从 A 到 B,或者 B 是 A 的紧邻块且 A 的最后一条指令不是无条件跳转.与此同时,LLVM IR 的 SSA 形式能够直观地体现值的定义与使用关系,从而清晰地描述程序的数据流信息.

节点海 IR 通过一个数据结构同时表示控制流和数据流,并放宽了指令间的顺序约束.在图 1 所示的示例程序中,第 4 行的语句中,变量 b 仅依赖于 a,因此不需要放在循环体内.在传统编译器中,这一步通常需要先分析出与循环无关的变量,然后将该语句移出循环.而在采用节点海数据结构的情况下,变量 b 一开始并未被归属于特定的基本块,因此也无需专门进行代码重排或移动.

```
1 int foo(int a){
2     int sum = 0;
3     for(int i = 0; i< 10; i++){
4         int b = a*2;
5         sum += b;
6     }
7 }
```

图 1 节点海 IR 中浮动特性展示的源代码示例

为了实现这种浮动特性,节点海 IR 设计了特殊的语法结构[26].在语法结构层面,节点被分为数据计算节点和控制流执行节点.每个节点根据其输入和操作定义一个值.数据计算节点的输入来源于其前驱节点,输出值则被所有后继节点所使用.控制流执行节点包括 Jmp、If 和 Return 三类,其输入和输出代表程序的控制流执行顺序.特别地,控制流执行 Region 节点和数据计算 Phi 节点均具有多个有序输入,Region 的实际输入与 Phi 的实际输入一一映射,从而同步了 IR 中的控制流与数据流表示.上述语法结构的差异需要我们深入理解并加以处理,以便更好地适应指针分析算法的需求.

**Panda IR 的具体实现:**方舟编译器提供了逐级遍历方舟字节码文件中模块、类和函数的接口.通过遍历函

数结构,可以访问其包含的 Panda IR 指令节点.Panda IR 被实现为节点海结构.IR 中每条指令由操作码(即指令名称)和指令参数列表组成.指令的参数可以包括寄存器(前驱指令节点)、立即数、String ID、Method ID 和 Literal ID 等.除此之外,部分指令中使用累加器作为默认参数.Panda IR 通过指令的寄存器参数确定其直接前驱指令,从而维护了指令之间的控制和数据依赖关系.

为了充分包含鸿蒙应用信息,Panda IR 中还存在全局变量、模块(module)命名空间和模块变量、词法环境和词法变量、补丁变量四种值存储方式.指令可以使用这四种储值位置中的值作为入参.全局变量是一个存储在全局唯一的映射中的变量,其键值为全局变量的名称,值为全局变量的值.全局变量可通过全局相关的指令进行访问.模块命名空间是指模块中的变量、函数和类所处的作用域,而模块变量是定义在模块级别的变量,这些变量在模块的整个命名空间中是可用的.此外,为了兼容 ArkTS 中的作用域链和闭包特性,Panda IR 通过词法环境相对层级编号和词法变量顺序索引二元组来表示一个词法变量.方舟编译器仍在不断演进发展中,Panda IR 也在优化更新,基于 Panda IR 的静态分析框架能更快适应方舟编译器的更新迭代过程.

### 2.2 ArkTS指针分析语言特性

ArkTS 是鸿蒙应用的主要开发语言,支持与 JS/TS 的高效互操作,并兼容 JS/TS 生态.TS 是 JS 的超集,通过在 JS 的基础上添加静态类型定义扩展了 JS 的语法.ArkTS 在保持 TS 基本语法风格的同时,进一步通过规范加强静态检查和分析,从而在开发阶段就能发现更多潜在错误,提升代码的健壮性,并优化运行性能.方舟编译器支持 ArkTS、JS 和 TS 源代码输入,并通过编译优化将其转换为统一的 Panda IR.为了有效地进行针对鸿蒙应用的静态分析,我们对 ArkTS 的语法特性进行了详细总结与分析.

**静态语言特性**:JS 是典型的动态语言,其动态特性体现在两个主要方面:变量声明无需绑定类型,类型与具体的值关联;对象的属性和方法可以在运行时动态增加、删除或修改,并支持通过反射动态访问和遍历属性.TS 在保留 JavaScript 动态语言特性的同时,还引入了结构化类型系统(Structural Typing)增强类型安全性.在变量类型方面,TS 默认通过类型推断为变量分配类型,类型确定后不可更改,但可支持 Any 类型.在对象布局方面,TS 对对象的属性和方法施加了严格的类型约束,不允许新增或删除未定义的属性,但可以通过索引签名动态添加属性.ArkTS 采用名义化类型系统(Nominal Typing),通过规范约束了 TS 中过于灵活而影响开发正确性或者给运行时带来不必要额外开销的特性,其对象字面量必须标注类型,不支持在运行时更改对象布局,也不支持 TS 中的 Structural Typing.由于纯血鸿蒙应用基于 ArkTS 语言开发,因此我们主要基于 ArkTS 的静态语言特性设计静态分析框架.

**函数对象和回调**:ArkTS 中的每个函数都是作为一个内部对象被维护和运行的.通过函数对象的性质,可以方便的将一个函数赋值给一个变量或者函数作为参数传递.将函数作为参数传递,或者将函数赋值给其他变量是所有事件机制的基础,例如回调函数的使用.此外,借助于函数的原型对象,可以很方便的修改和扩充 Function 类型的定义,如增加属性和方法等.特别的,ArkTS 为函数对象定义了两个方法:Apply 和 Call,它们的作用是将函数绑定到另一个对象上运行,实现显式绑定调用.这种特殊性为我们的指针分析工作带来了挑战.

**作用域链与闭包**:作用域链与闭包是 ArkTS 的重要特性之一.ArkTS 中包含两种作用域:函数作用域和全局作用域.全局作用域是唯一的,例如浏览器中的 Window 对象;每个函数则具有自己的函数作用域,函数内部声明的变量和参数共享同一作用域.函数嵌套即在一个函数中定义另一个函数,嵌套函数形成闭包.由于闭包特性,内嵌函数能够访问外部函数的参数和变量,而外部函数无法访问内嵌函数的参数和变量,从而形成作用域链.访问变量时,ArkTS 会从当前作用域的本地变量和参数开始,逐层向上遍历作用域链,直到全局作用域.这里关于作用域的讨论对于精确的数据依赖分析至关重要.

## 3 HarmonyFlow 框架设计

图 2 展示了 HarmonyFlow 静态分析框架的总体设计思路.首先在 3.1 节介绍我们如何从 Panda IR 中提取基础信息并处理关键指令,然后在 3.2 节介绍我们的指针分析过程,依次说明我们构建指针流图、设计传播规则、实现算法框架和适配 ArkTS 语法的全过程,最后在 3.3 节说明静态分析框架提供的拓展分析模块.

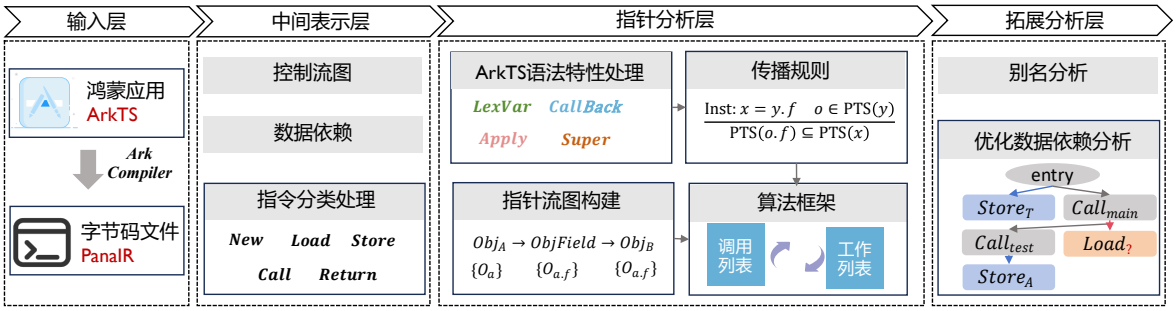



图 2 HarmonyFlow 框架设计概览

### 3.1 Panada IR信息处理

#### 3.1.1 基础信息提取

HarmonyFlow 旨在对 Panda IR 进行高效的静态分析,将其集成到方舟编译器中时会面临高耦合性和低效率的问题,因此需要单独实现 Panda IR 的文件解析和信息提取.方舟编译器由编译工具链和运行时两部分组成,编译工具链将源代码编译为方舟字节码,运行时负责执行生成的字节码.为了实现 Panda 字节码文件(.abc 文件)的解析,我们借鉴方舟运行时中承载字节码的 ArkCompiler File 组件,提取其中的模块、类、函数和全局变量等信息,并进一步提取每个函数中的节点海形式的 Panda IR.

为提高框架的通用性,我们基于 Panda IR 构建了基本块级别和指令级别的过程内控制流图,并根据数据依赖关系构建了指令间的 Def-Use 链.此外,针对作用域链和闭包问题,我们注意到 Panda IR 中的词法变量操作指令(StLexVar 和 LdLexVar)的参数表示相对词法环境层级,这使得直接判断操作的具体变量变得困难.为此,我们提供了接口,能够获取与 LdLexVar 指令对应的所有 StLexVar 指令.

#### 3.1.2 指令分类和处理

Panda IR 包含 318 条指令,需要对指令进行分类处理.在指针分析过程中,重点关注五类指令:NEW、LOAD、STORE、CALL 和 RETURN.这些指令与内存操作和函数调用相关.我们逐一分析所有指令,筛选出这五类关键指令,并从每条关键指令的参数和上下文中提取必要信息.

由于 Panda IR 设计为节点海结构,具有独特的语法和语义,增加了指令分类和处理的难度.对于一个指令操作,需要向上追溯多条指令获得.在指令语义处理过程中,我们首先将 ArkTS 源代码映射到 Panda IR 的一条指令,然后通过解析该指令的参数和 Def-Use 链向上检索必要信息,向上检索的过程的终止条件为:1)获得目标信息.2)检索到已经处理过的指令;3)目标信息为指针分析预期结果.

图 3 展示了一个字段敏感的函数调用示例,其左侧为源代码,右侧为相应的简化 Panda IR.例子中,第 16 行的 x.foo()调用了类 B 的 foo 方法.具体而言,函数 test 首先调用函数 getA,后者实例化类 A 并返回该实例.类 A 的构造函数在隐式创建类 B 的实例后,将该实例赋值给 obj 属性.随后,test 函数通过 p.obj 引用类 B 的实例,并调用其 foo 方法.在解析此调用时,需依次确定 x 指向对象的类型(即类 A 的实例),随后解析其 obj 属性的类型(类 B 的实例),最终解析到 foo 方法作为被调函数.此示例体现了跨类实例化与字段引用的解析流程,是字段敏感分析的典型情况.

根据图 3 可以理解 Panda IR 的结构组成.Panda IR 对每个模块的代码抽象为一个入口函数 func_main_0,该函数包含全局的类声明、函数声明以及其他语句.在 Panda IR 中,类 A 和类 B 的定义分别表示为独立的函数,其内部包含类的成员变量和成员方法.类似的,源代码中的函数 getA 和 test 也被表示为 Panda IR 中的函数,其内部由具体的指令构成.对于这些指令,主要关注以下核心属性:前驱指令(inputs),用于表示当前指令依赖的输入;字符串(string),用于标识相关的符号或常量;词法环境层级(lexEnv),表示指令所属的词法作用域;词法变量(lexVar),用于标识当前指令涉及的变量.

表 1 描述了我们针对方舟字节码关键指令的语义处理过程.以 StObjByName 指令为例,我们需要获得其左操作数对象、右操作数对象和字段名称.左操作数对应第 11 行的 Paramter 指令,其为 Class 类的第 2 个参数,Panda IR 将第二个参数作为函数或类的 this 指针,即左操作数为 $O_a$.右操作数对应第 13 行的 NewObj 指令,这个指令处理后的结果为 $O_b$.字段名称可以根据 StObjByName 指令的 String 字段获得,其值为“obj”.于是,这里 StObjByName 的语义为将 $O_b$ 储存到 $O_a$ 的“obj”字段.

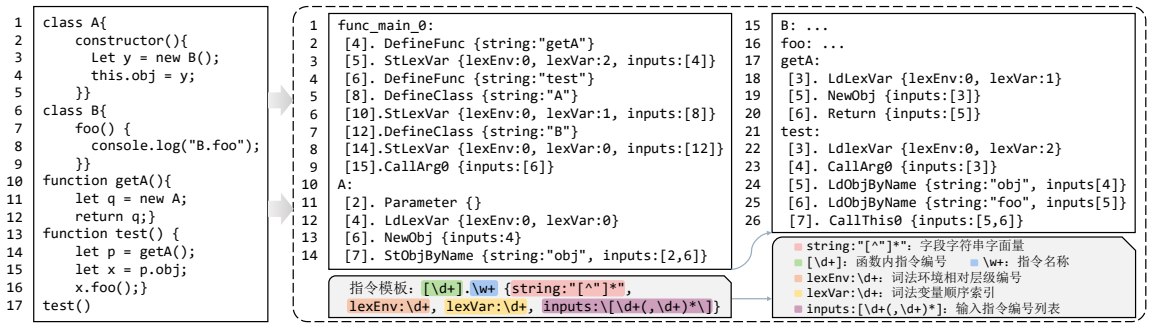



图 3 Panda IR 程序中的字段敏感性分析示例

### 3.2 针对ArkTS的指针分析算法

指针分析是最基本的静态程序分析之一,几乎所有其他分析都是建立在它之上的.下文介绍基于 Panda IR 设计的指针分析算法设计,算法实现为 Andersen 风格的字段敏感算法.我们首先构建指针流图,作为指针分析的基础数据结构.然后介绍指向集合传播规则,定义如何在指针流图上传播指向信息.最后介绍算法框架,说明以何种顺序完成传播过程.此外,我们特别说明了算法针对 ArkTS 语法的特殊支持.

#### 3.2.1 指针流图构建

指针流图表示程序中有哪些指针以及它们可能的传播关系.指针流图的节点是指针,代表变量或对象的域成员;边是传播关系,指针的指向集合需要被传播到其所有的后继节点. 本文的指针流图设计不仅继承了传统指针分析图结构的抽象能力和可拓展性,还针对 Panda IR 结构和 ArkTS 语言做出了一些列关键调整.一方面,我们提出了一种指令粒度的局部图构建策略,通过关键指令的识别和连接,提升了图的紧凑性和精度,有效支持字段敏感和上下文敏感指针分析;另一方面,我们将函数统一建模为具备字段属性的对象节点,从而支持函数作为一等对象的建模与追踪.

**节点设计**:程序执行时会分配大量的堆对象,如何对堆对象建模是指针分析中的重要问题.通过分析 ArkTS 中涉及堆操作的语法,我们将堆对象分为四类,并相应的实现为四类指针.指针的实现充分近似了每种堆对象的数据结构形式.1)对象指针(Object):它包括通过关键字 new 显式分配的所有对象(例如 new T() ).该指针通过 NewObj 指令标识.2)对象字段指针(ObjectField):它通常由对象引用与字段名称共同确定,指向对象的特定字段或属性.例如 obj.field 表示指向 obj 对象中 field 字段的指针.该指针通过其输入指令(包含其对象信息)和字段名称标识.3)数组元素指针(ArrayElement):它通过数组引用和元素的索引来确定,指向数组中的特定元素.例如, arr[i] 表示指向数组 arr 中第 i 个元素的指针.该指针通过其输入指令(包含其数组信息)和索引数值标识.4)静态字段指针(StaticField).静态字段指针指向在词法作用域中声明的全局变量以及类或函数中的静态成员.静态成员包括类的静态属性和方法,以及函数内部定义的静态变量.该指针通过一个二元组<词法环境相对层级编号, 词法遍历顺序索引>唯一标识这些变量在词法作用域中的位置和访问路径.

**边设计**:节点之间通过有向的传播边连接.指针 x 到指针 y 的传播关系表示指针节点 x 的指向集合可能传递到指针节点 y 的指向集合.

**表 1** 指令语义处理和局部指针流图构建规则

| 源代码 | 指令语义处理 | 局部指针流图 |
|---|---|---|
| **Line13:let q = new A()** | 对象类型$O_a$依赖LdLev指令获得<br>Line 18: LdLexVar{< 0,1 >}<br>**Line 19: NewObj {(Line18)}** | $Object_{Line19:NewObj}$ $\{O_a\}$ |
| **Line18: let x = p. obj** | 对象$O_a$依赖指针分析结果获得<br>Line 23: CallArg0{}<br>**Line 24: LdObjByName{"obj", (Line23)}** | $Object_{Line23:CallArg0}$<br>$ObjectField_{Line23\ \#obj}$<br>$Object_{Line24:LdObjByName}$ |
| **Line4: this. obj = y** | Line11：左操作数对象为$O_a$依赖Panda IR规定<br>Line13：右操作数对象为$O_b$依赖New指令获得<br>Line 11: Parameter<br>Line 13: NewObj<br>**Line 14: StObjByName{"obj", (Line13, Line11)}** | $Object_{Line13:NewObj}$ $\{O_b\}$<br>$Object_{Line11:Parameter}$ $\{O_a\}$<br>$ObjectField_{Line11\ \#obj}$ |
| **Line12: function getA()** | Line 2: DefineFunc{"getA"}<br>**Line 3: StLexVar{<0,2>, (Line2)}** | $Object_{Line2:DefineFunc}$ $\{O_{getA}\}$<br>$StaticField_{<0,2>}$ |
| **Line17: p = getA()** | **Line 22: LdLexVar{<0,2>}** | $StaticField_{<0,2>}$<br>$Object_{t3:LdLexVar}$ |

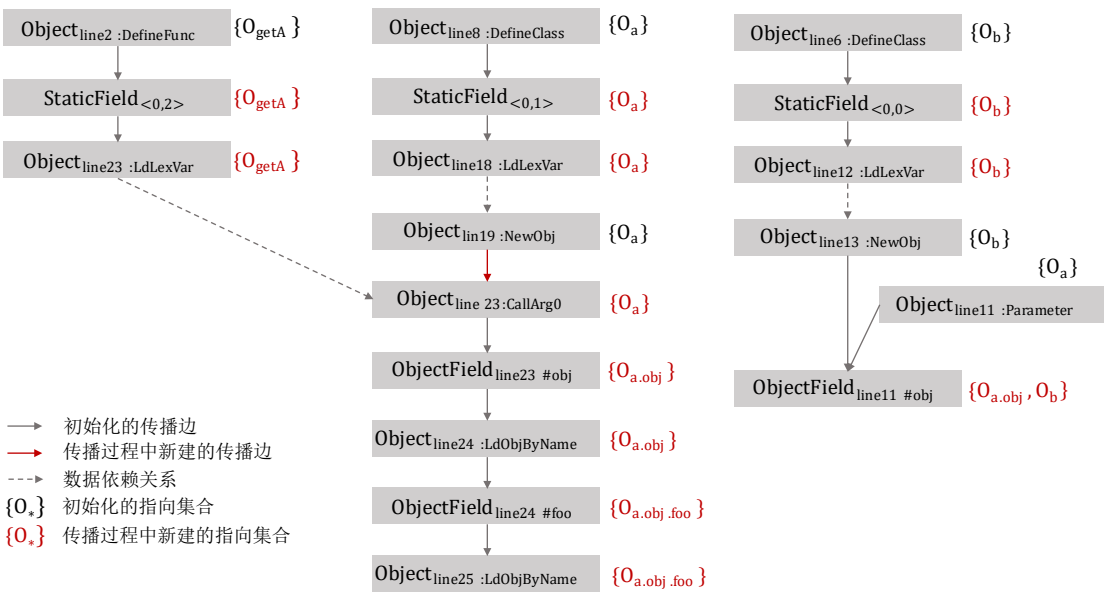



图 4 Panda IR 程序字段敏感性分析的指针流图示例

**图构建过程**:针对不同类型的指令,我们基于指令语义处理的结果设计了相应的规则创建局部指针流图的节点和边.如表 1 所示,对于创建新对象的指令类型,以 NewObj 为例,我们基于当前指令创建 Object 指针,局部的传播关系和指向关系为 Object:{ $O_a$ };对于引入别名关系的指令,以单个对象加载的 LoadObjByName 指令为例,建立 $Object_{src} \rightarrow ObjectField \rightarrow Object_{to}$ 的传播关系,基于此可以将 $Object_{src}$ 对象的目标字段的指向集合传递到 $Object_{to}$ 中;同样的,对于 StoreObjByName 指令,建立 $Object_{src} \rightarrow ObjectField$ 和 $Object_{to} \rightarrow ObjectField$ 两条传播关系,从而可以将 $Object_{src}$ 对象的指向集合传递到 ObjectField 中,这里通过 $Object_{to}$ 确定 ObjectField 的对象类

型.对于操作静态字段如全局变量的指令,例如一对 StLexVar 和 LdLexVar 指令,局部的传播关系和指向关系为 $\text{Object}_{\text{src}} \to \text{StaticField} \to \text{Object}_{\text{to}}$,基于此可以联系针对词法变量的储存和加载操作.类似的,我们针对不同类型的语句,设计了多种局部图构建规则,完成了针对 Panda IR 的指针流图构建,作为指针分析的基础.

通过表 1 展示的局部指针流图构建规则,可构建完整的指针流图,如图 4 所示.图中灰色部分为初始化的指针流图,包含灰色的指针节点和灰色实线的传播边.此外,针对创建新对象的节点,还初始化了其指向集合.后续的指针分析算法将在这个图上进行.图中的红色部分代表传播过程中生成的新增内容(传播规则将在后续章节详细说明),包括新识别的调用边和更新后的指向集合.传播完成后,通过指令 $\text{Object}_{\text{line25:LdObjByName}}$ 的指向集合 $\{\text{O}_{\text{a.obj.foo}}\}$,可以推导出调用路径.首先通过 $\text{ObjectField}_{\text{line11\#obj}}$ 的指向集合 $\{\text{O}_{\text{a.obj}}, \text{O}_{\text{b}}\}$ 可确定 $\text{O}_{\text{a.obj}}$ 属于类 B 的实例.基于此,可以进一步调用类 B 的 foo 函数.该过程展示了指针流图在解析字段敏感调用时的作用,通过逐步确定对象及其字段的类型和指向关系,完成对调用链的精准解析.

3.2.2 传播规则设计

针对内存影响和函数调用指令,我们分别设计了对应的传播规则,用于精确建模指针流图中各节点的指向关系,指针流图在图结构的遍历与规则执行过程中不断传播和汇聚指向集,直到达到稳定状态.在经典 Andersen 算法的通用传播框架基础上,我们针对 ArkTS 的语言特性,包括回调函数、显式绑定调用、函数作为对象属性等复杂语法结构,逐一进行了扩展建模,确保分析结果在精度与覆盖范围上的可用性.我们为上述语义特性定义了相应的传播规则,以形式化方式描述指向集的传递与更新过程.下文将给出这些传播规则的形式化定义.

设 V, H, T, M, F, C 和 I 分别表示变量集、堆抽象、类型、方法、字段、类和指令,定义以下符号:

- $\text{PTS}: \text{V} \cup \text{H} \times \text{F} \to \text{P}(\text{H})$
- $\text{MethodOf}: \text{I} \to \text{M}$
- $\text{ClassOf}: \text{V} \to \text{C}$
- $\text{ParentOf}: \text{C} \to \text{C}^{\text{Super}}$
- $\text{Inst}: \text{M} \to \text{P}(\text{I})$
- $\text{Dispatch}: \text{M} \times \text{H} \to \text{M}$
- $\text{HeapAbs}: \text{I} \times \text{T} \to \text{H}$

这里的 PTS 记录了对象、静态字段和常量的指向信息集合,MethodOf 给出了包含语句的函数,ParentOf 给出当前类的父类,ClassOf 给出变量的类型,Inst 返回了函数中的指令集合,Dispatch 用来解析对目标方法的调用,HeapAbs 定义了对于一个对象的堆抽象规则.

以下给出了形式化的指向集传播规则,这些规则支持动态调用图构建.

$$[\text{NEW}] \quad \frac{\text{I}: \text{x} = \text{newT} \quad \text{o} = \text{HeapAbs(I, T)}}{\text{o} \in \text{PTS(x)}} \tag{1}$$

$$[\text{LOAD}_{\text{object}}] \quad \frac{\text{I}: \text{x} = \text{y.f} \quad \text{o} \in \text{PTS(y)}}{\text{PTS(o.f)} \subseteq \text{PTS(x)}} \tag{2}$$

$$[\text{STORE}_{\text{object}}] \quad \frac{\text{I}: \text{x.f} = \text{y} \quad \text{o} \in \text{PTS(x)}}{\text{PTS(y)} \subseteq \text{PTS(o.f)}} \tag{3}$$

$$[\text{CALL}] \quad \frac{\text{I}: \text{x} = \text{a}_{\text{o}}.\text{f}(\text{a}_1,...,\text{a}_{\text{r}}) \quad \text{o} \in \text{PTS}(\text{a}_0) \quad \text{m}' = \text{Dispatch(f,o)}}{\text{o} \in \text{PTS}(\text{this}^{\text{m}'}) \quad \forall \text{i} \in [1,\text{r}]: \ \text{PTS}(\text{a}_{\text{i}}) \subseteq \text{PTS}(\text{para}_{\text{i}}^{\text{m}'}) \quad \text{PTS}(\text{ret}^{\text{m}'}) \subseteq \text{PTS(x)}} \tag{4}$$

$$[\text{CALL}_{\text{Super}}] \quad \frac{\begin{array}{c}\text{I}: \text{x} = \text{Super.f}(\text{a}_1,...,\text{a}_{\text{r}}) \quad \text{m=MethodOf(I)} \quad \text{c=ClassOf}(\text{this}^{\text{m}}) \\ \text{c}' = \text{ParentOf(c)} \quad \text{m}' = \text{Dispatch(f,c}') \quad \text{o} \in \text{PTS}(\text{this}^{\text{m}})\end{array}}{\text{o} \in \text{PTS}(\text{this}^{\text{m}'}) \quad \forall \text{i} \in [1,\text{r}]: \ \text{PTS}(\text{a}_{\text{i}}) \subseteq \text{PTS}(\text{para}_{\text{i}}^{\text{m}'}) \quad \text{PTS}(\text{ret}^{\text{m}'}) \subseteq \text{PTS(x)}} \tag{5}$$

$$[\text{CALL}_{\text{Apply}}] \quad \frac{\text{I}: \text{x} = \text{a}_0.\text{f}(\text{a}_{\text{apply}},\text{a}_1,...,\text{a}_{\text{r}}) \quad \text{o} \in \text{PTS}(\text{a}_{\text{o}}) \quad \text{m}' = \text{Dispatch(f,o)} \quad \text{o}' \in \text{PTS}(\text{a}_{\text{apply}})}{\text{o}' \in \text{PTS}(\text{this}^{\text{m}'}) \quad \forall \text{i} \in [1,\text{r}]: \ \text{PTS}(\text{a}_{\text{i}}) \subseteq \text{PTS}(\text{para}_{\text{i}}^{\text{m}'}) \quad \text{PTS}(\text{ret}^{\text{m}'}) \subseteq \text{PTS(x)}} \tag{6}$$

$$[\mathrm{CALL}_{\mathrm{Back}}] \quad \frac{\mathrm{I}:\mathrm{a}_{\mathrm{callback}}(\ )\quad \mathrm{m}=\mathrm{MethodOf(I)}\quad \mathrm{m}^{'}=\mathrm{Dispatch}\left(\mathrm{parameter}^{\mathrm{m}}\right)}{\forall \mathrm{i}\in[1,\mathrm{r}]:\ \mathrm{PTS}(\mathrm{a}_{\mathrm{i}})\subseteq \mathrm{PTS}(\mathrm{para}_{\mathrm{i}}^{\mathrm{m}^{'}})} \tag{7}$$

如公式(1)到(7)所示,在内存访问方面,$o\in H$ 是 HeapAbs 创建的堆抽象对象. $\mathrm{LOAD}_{\mathrm{object}}$ 和 $\mathrm{STORE}_{\mathrm{object}}$ 是已有的标准规则,这里我们进一步拓展支持了数组元素.在函数调用方面,CALL 指令表示标准的对象方法调用,我们对 ArkTS 中的父类调用( $\mathrm{CALL}_{\mathrm{Super}}$ )、显式绑定调用( $\mathrm{CALL}_{\mathrm{Apply}}$ )和回调( $\mathrm{CALL}_{\mathrm{Back}}$ )等调用方式进行了适配.对于父类调用,实际调用的方法是当前函数的 this 指针指向对象的父类方法.在显式绑定调用中,参数列表中指定了被调函数的 this 指针,因此在被调函数的 this 指针与相应参数之间建立了传播边.对于回调,被调函数作为当前函数的参数,通过当前函数的参数来解析目标函数.

### 3.2.3 基于工作列表和调用列表的算法框架

在工作列表层面,我们借鉴了 Andersen[16] 提出的增量工作列表算法.在找到新的被调函数时,我们一方面解析其所有指令构建该函数的指针流图,另一方面通过参数和返回值将该函数的指针流图连接到整个程序的总体指针流图中.完成指针流图构建后,我们将指针流图新增的待传播指针节点添加到工作列表中.然后算法按照传播规则逐一将工作列表中指针节点的指向集合传播到其后继节点中.

在调用列表层面,我们设计了基于深度优先搜索的调用图遍历算法,用于对整个程序的函数进行指针分析.为了避免无限循环,我们对已解析的调用边不再进行处理,即当某一调用点重复调用同一被调函数时,该函数将不再被处理.深度优先遍历能够确保符合程序实际的函数执行顺序,从而保证指针分析结果的真实性和可靠性.

算法 1 给出了我们指针分析算法的整体伪代码.指针流图 PFG 维护为指针节点映射到其后继节点的集合.PTS 表示每个指针节点可能指向的对象实例集合.算法首先通过 VisitGraph 函数构建指针流图,然后通过 SolveConstraint 函数进行指向集合的传播.在第 11 行,针对 LoadObj 指令的处理,展示了 3.2.1 节中指针流图构建的具体方法.第 33 行 PropPTS 函数列举了典型传播规则.第 26 行和第 30 行则实现了基于工作列表和调用列表的传播过程.

HarmonyFlow 整体算法流程

输入:待测试鸿蒙应用字节码文件(.abc 文件)
输出:指向集合 PTS,调用边 CallInst2CalleeFunc

```
1.  Function AliasAnalysis(abcfile)
2.      ∀(p)∈P:PTS(p)←⟨∅⟩
3.      ∀(p)∈P:PFG(p)←⟨∅⟩
4.      WorkList ← CallList ← ReachCallEdge ←⟨∅⟩
5.      VisitGraph(m_entry)
6.      SolveConstraints( )
7.  Function VisitGraph(m)
8.      for I: NewObj ∈ Inst(m) do
9.          Pointer_object = HeapAbs (I)
10.         PTS( Pointer_object )∪= { Pointer_object }
11.     for I: LoadObj ∈ Inst(m) do
12.         Pointer_Object_src = HeapAbs ( I.input [0])
13.         Pointer_ObjectField = HeapAbs ( I.input [0] , I.string )
14.         Pointer_Object_to = HeapAbs ( I )
15.         PFG ( Pointer_Object_src )∪= { Pointer_ObjectField }
```

```
16.      PFG ( Pointer Object_Field ) ∪= { Pointer Object_to }
17.    ...
18.    for p : Pointer do
19.      if (PTS(p) ≠ ⟨∅⟩ ) and (PFG(P) ≠ ⟨∅⟩ )do
20.        AddWorkList(p)
21.    for I : Call ∈ Inst(m) do
22.      Index = GetIndex(currentCallInst)
23.      CallList.insert(Index + 1, I)
24.  Function SolverConstraints()
25.    while WorkList ≠ ∅ or CallList ≠ ∅ do
26.      if WorkList ≠ ∅ do
27.        Pointer_src = Poll(WorkList)
28.        for each Pointer_to = PFG(Pointer_src)
29.          PropPTS(Pointer_src,Pointer_to)
30.      else if CallList ≠ ∅ do
31.        CallInst = Poll(CallList)
32.        currentCallInst = CallInst;  HandleCall(currentCallInst)
33.  Function ProPTS(src, to)
34.    for each alias in PTS(src)
35.      if alias. Inst == to. Inst and to. type == ObjectField do
36.        Pointer_ObjectField = HeapAbs(alias.Inst, to.string)
37.        PTS(to) ∪= { Pointer ObjectField }
38.      else
39.        PTS(to) ∪= { alias}
40.      ...
41.      AddWorkList(to)
42.  Function HandleCall(call)
43.    for each alias in PTS(call.input[0])
44.      curClass = GetClass(alias);  funcName=GetFunc(call)
45.      m^callee = DisPatch(curclass, funcname, call)
46.      if(call, m^callee ) ∉ ReachCallEdge do
47.        PropPTS(alias, this^m')
48.        PFG(Pointer_input) ∪= {Pointer_parameter_i | i ∈ [1,r]}
49.        PFG(Pointer_ret) ∪= Pointer_call
50.        ReachCallEdge ∪ = {(call,m^callee )}
51.        VisitGraph(m^callee)
```

算法 1　指针分析整体伪代码

3.2.4　针对 ArkTS 语法的特殊支持

为了保证静态分析的全面性,我们不仅支持基础的面向对象语法,而且兼容 ArkTS 中的语法特性.

**基础面对对象语法**:关于 Load 和 Store 类型指令,我们广泛处理了对象字段、数组元素和全局变量的内存

操作,建立了定制化的传播链,并设计了特定的传播规则.关于 Call 指令,我们逐一适配了对象方法调用、全局方法调用、父类函数调用等多种调用方式,给出了相应的传播规则.

**ArkTS 语法特性**:在 2.2 节,我们总结了 ArkTS 的静态语言特性、函数对象和回调特性、作用域链与闭包问题.这里函数对象和回调特性对指针分析算法设计有显著影响.具体来说,增添了回调和显式绑定调用两种调用方式,需要针对的适配.

```
// 回调函数调用
1 function test(callback) {
2   console.log("test");
3   callback();
4 }
5 function foo() {
6   console.log("foo");
7 }
8
9 test(foo);
```

```
// 显式绑定调用
1 function func1() {
2  this.p = "func1-";
3  this.A = function(arg) {console.log(this.p + arg); };}
4 function func2() {
5  this.p = "func2-"; }
6 let obj1 = new func1();
7 let obj2 = new func2();
8 obj1.A("byA");
9 obj1.A.apply(obj2, ["byA"]);
```

图 5　回调和显式绑定示例

如图 5 所示,我们简要介绍适配方法.针对回调的情况,首先对函数声明指令 DefineFunc 创建 Object 指针,并将函数对象添加到指向集合中.由于过程间指针分析会在第 8 行 test(foo)的实参和第 1 行 function test(callback)的形参之间建立传播边,因此传播后 callback 形参指向函数对象.进一步,根据上文的 $\text{CALL}_{\text{Back}}$ 传播规则可以成功解析此函数调用.针对显式绑定调用,我们修改了 function 函数 this 指针的传播关系.对于第 10 行未显式绑定调用的情况,我们直接将接收对象 obj1 的指向集合传播到 function 的 this 指针中.对于第 11 行显式绑定调用的情况,我们将参数 obj2 的指向集合传播到 function 的 this 指针中.

### 3.3　拓展分析接口

在完成基础指针分析之后,HarmonyFlow 构建了统一的指向信息查询与调用关系解析能力,为后续静态分析任务提供了通用接口支持.更进一步,我们实现了两个核心扩展接口:(1)别名分析接口,用于动态追踪变量间的潜在别名关系;(2)指令数据依赖接口,结合调用图与作用域信息,精确建模闭包场景下的变量修改与使用路径.

#### 3.3.1　别名分析接口

基于指针分析的指向结果,我们实现了指令间的别名分析接口.指令间的别名分析旨在确定不同指令操作的内存位置是否可能引用相同的内存地址,这对于静态分析工具至关重要.通过指针分析得到的指向信息,我们可以明确哪些指令操作可能存在别名关系,从而帮助识别潜在的内存访问冲突或数据依赖.

具体而言,别名分析依赖于指针分析所提供的指向信息,首先通过追踪每个指令的操作数及其内存访问模式,结合指针分析结果,确定是否存在不同指令指向同一内存位置.如果两个或多个指令涉及相同的内存地址,则可以推测这些指令之间存在别名关系.

#### 3.3.2　指令数据依赖接口

数据依赖分析的实现是下游安全检测任务的重要基础.在本研究中,我们重点关注并解决 Panda IR 中词法变量依赖关系不准确的问题.此外,我们设计了接口用于确定特定指令的数据依赖前驱指令.

方舟 Panda IR 中仅存在函数内的 Def-Use 链,缺失过程间数据依赖信息.因此我们基于指针分析构建的调用图,在调用函数的形参和被调函数的实参之间创建数据依赖边,并且在被调函数的返回指令和调用函数的调用指令之间创建数据依赖边.

ArkTS 的闭包机制导致每个函数都有自己的词法环境,内层函数可以访问外层函数中的变量.Panda IR 给定一个 LdLex 指令,可能找到多个 SdLex 指令,这样是不够精确的.因此,我们根据调用语句和词法变量修改语句的执行顺序确定了 LdLex 指令对应的最新的 StLex 指令.如图 6 实验分析所示,首先收集 3.2.3 节指针分析过程中深度优先调用语句列表,表示为树状图.然后将待分析词法变量的 LoadLex 语句和 StoreLex 语句按照调用顺序和语句顺序插入到相应位置.最后将整个将修改后的树节点按照前序遍历排序,LoadLex 指令左侧最近的

StoreLex 指令为最新一次针对词法变量的修改.

```
1  function func1(){ pg = new A(); }
2  function func2(){ pg = new B(); }
3  function func3(){ pg = new C(); }
4  var pg = new T();
5  function test() {
6      func1();
7      pg.foo();
8      func3(); }
9  function main() {
10     test();
11     func3(); }
12 main();
```

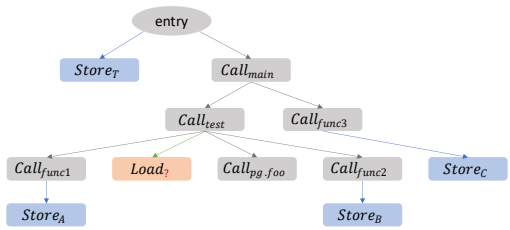



图 6 优化数据依赖分析示例

## 4 实验分析

为了评估 HarmonyFlow 在鸿蒙应用上的指针分析与调用图构建的精确性和效率,我们分别在自主构建的测试用例和真实鸿蒙应用上进行了实验评估.本文提出了以下三个研究问题(RQs)以指导实验设计:

- RQ1: HarmonyFlow 对 ArkTS 语法特性的支持效果如何?
- RQ2: HarmonyFlow 在开源鸿蒙应用中调用边识别的精度如何?
- RQ3: HarmonyFlow 在真实鸿蒙应用上的运行效率如何?

**环境设置**.HarmonyFlow 基于 GCC 11.2.0 开发,由大约 4000 行代码组成,可以直接分析方舟编译器生成的方舟字节码文件(.abc 文件).方舟编译器为 OpenHarmony v3.2 Beta5 的内置编译器.所有实验均在一台搭载 Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz 的服务器上完成,操作系统为 Ubuntu20.04.

**实验指标**.实验指标的设计旨在全面评估静态分析框架的有效性,重点关注精度和效率两个方面.指针分析是最基本的静态程序分析之一,几乎所有其他分析都依赖于指针分析[27].于是,实验主要评估指针分析调用边识别的精度和效率.在识别精度方面,我们主要关注调用边数、精确率和召回率.调用边数用于衡量工具识别调用关系的能力,反映工具发现的调用边数量;精确率用于衡量工具识别的调用边中真实调用边所占的比例,体现分析的精确度;召回率则反映工具在所有真实调用边中成功识别的比例,反映工具的健全性.为评估效率,我们以运行时间为主要指标,并进一步探讨软件规模与运行时间之间的关系,以分析框架在不同规模应用上的性能表现.这些指标共同构成了对静态分析框架适用性和稳定性的全面评价标准.

### 4.1 RQ1: ArkTS语法特性覆盖性

**表 2** ArkTS 语法特性覆盖结果

| 测试用例 | 调用边数量 | 误报数量 | 漏报数量 |
|---|---|---|---|
| TestArray | 2 | 0 | 0 |
| TestDict | 2 | 0 | 0 |
| TestPara1 | 7 | 0 | 0 |
| TestPara2 | 9 | 0 | 0 |
| TestPara3 | 14 | 0 | 0 |
| TestReturn1 | 7 | 0 | 0 |
| TestReturn2 | 8 | 0 | 0 |
| TestInherit | 10 | 0 | 0 |
| TestIndirect1 | 11 | 0 | 0 |
| TestIndirect2 | 4 | 0 | 0 |
| TestGlobal | 7 | **1** | 0 |
| TestField1 | 8 | **1** | 0 |
| TestField2 | 12 | 0 | 0 |
| TestStatic | 2 | 0 | 0 |
| TestSuper | 8 | 0 | 0 |
| TestFuncObj | 6 | 0 | 0 |
| TestAnonFunc | 2 | 0 | 0 |
| TestLexVar | 6 | 0 | 0 |
| TestCallBack | 2 | 0 | 0 |

**实验数据**.为了评估 HarmonyFlow 对于 ArkTS 的基础面向对象语法和语法特性是否全部支持,我们和合作单位共同构建了 19 个包含各种语法结构的测试用例.经统计,测试数据集合包含了针对对象字段、数组元素、字典元素和全局变量的内存操作,还包含了对象方法调用、全局方法调用、父类函数调用和绑定调用等调用过程.总体来说,该测试集包含了我们精心构造的 127 条调用边,我们人工标注了这些调用边,以便评估我们工具的实现效果.

**实验结果**.如表 2 所示,对于 19 个人工构建的测试用例,我们能够完整处理 3.2.4 节提及的语法特性并正确识别 127 条调用边.然而,在 TestGlobal 和 TestField 两个测试用例中均出现了一次误报,这是由算法的流不敏感性导致的.在源代码中,依次执行了 p=new A()、p.obj=new B()和 p.obj.foo().算法一方面正确解析到调用类 B 的 foo 函数,另一方面却先将类 A 的 obj 属性解析为类 G,再解析到调用类 G 的 foo 函数,从而导致误报.

### 4.2 RQ2: 开源鸿蒙应用上的分析精度

**实验数据**.为了评估 HarmonyFlow 在鸿蒙应用上的运行精度与效率,我们从 OpenHarmony 的 Gitee 仓库[28] 中收集了 9 个开源鸿蒙应用.这些应用具有以下显著特点:代表性:这些应用由 OpenHarmony 官方提供,旨在帮助开发者熟悉 OpenHarmony SDK 提供的 API 和开发流程,全面涵盖了 ArkTS 组件集、媒体与原生特性、窗口与电话等系统特性、分布式功能、多线程等语言基础库.时效性:这些应用基于 OpenHarmony 最新的 Stage 模型开发,采用面向对象的编程范式,使得代码具备较高的可读性、易维护性和良好的可扩展性,适用于复杂应用的开发需求.通过选取这些样例,我们能够全面测试 HarmonyFlow 在不同应用场景和特性组合下的表现,从而验证其适配性、精度和性能的实际效果.

由于软件应用的代码规模较大,审计全部代码以标注调用边存在较高的成本,因此我们采用抽样检测的方法评估准确率和召回率.具体而言,我们从 HarmonyFlow 输出的调用边中随机抽取 20 条(若总数少于 20 条,则选择所有调用边),通过人工核查其正确性以统计精确率.同时,从样本应用中随机抽取 20 条真实调用边,观察这些调用边是否被 HarmonyFlow 正确检测到,以统计召回率.此方法在保证统计结果可靠性的同时,有效降低了人工核查的工作量.

**实验设置**.由于 HarmonyFlow 静态分析框架专注于对鸿蒙应用的安全检测,因此我们不分析鸿蒙应用框架中底层自动触发的默认调用,也不关注官方库的调用行为.重点检测应用开发者编写的代码,确保其安全性.

**实验结果**.如表 3 所示,HarmonyFlow 在 9 个开源鸿蒙应用上的调用边识别精确率和召回率分别达到了**98.33%**和 **92.22%**.样本应用的代码复杂度显著高于人工合成的测试用例,平均调用边数量达 194 条,且不同样本应用的调用边数量变化较大.总体来看,调用边的误报比例较低,但漏报比例相对较高.误报的主要原因仍源于算法流不敏感的局限性.例如,在 FileEdit 应用中,某处调用点的接收对象为 PHI 指令,而 PHI 指令的两个输入分别对应不同的控制流路径,需要根据控制流信息确定接收对象的真实值.然而,算法未能区分这两种情况,而是将两个输入的值均作为接收对象,从而导致误报.在漏报分析中,我们发现样本应用的多模块特性是主要原因之一.一些跨模块的方法由于模块间依赖关系未能正确解析,导致调用边未被识别.在本文讨论部分的 5.2 节,提供了全面的结果分析和改进思路.

**表 3** 鸿蒙样本应用分析精度

| 样例应用 | 调用边数量 | 准确率 | 召回率 |
|---|---|---|---|
| VPNFoundation | 48 | 100% | 95% |
| CameraPage | 112 | 100% | 90% |
| DlpManager | 674 | 100% | 95% |
| FileEdit | 77 | 95% | 95% |
| MyphoneFilePage | 259 | 100% | 85% |
| ProcessMessage | 34 | 95% | 95% |
| UpdateWorkScheduler | 6 | 100% | 100% |
| VideoPlayer | 263 | 95% | 85% |
| VideoRecorder | 273 | 100% | 90% |

在分析过程中发现,鸿蒙样例应用中的动态调用较少,而通过事件机制触发的回调调用较为常见.以

MyPhoneFilePage 应用为例,经统计其包含 169 个回调调用、91 个动态调用和 2 个静态调用.回调调用的被调函数均为全局唯一的匿名函数,且 Panda IR 为每个匿名函数分配了唯一的函数名称.由于回调函数的唯一性,指针分析在回调调用中的误报率较低.然而,回调机制可能引发复杂的函数嵌套关系,从而导致变量作用域链中断,增加了漏报的可能性.动态调用作为面向对象语言的主要调用方式之一,也是指针分析误报的主要来源.在样例应用中,我们共检测到三处误报.这表明,尽管动态调用增加了指针分析的复杂性,但字段敏感的指针分析算法已能够较好地满足鸿蒙样例应用的分析需求,提供较高的精确性和实用性.

### 4.3 RQ3: 真实鸿蒙应用上的测试效率

指针分析的效率对于现代软件开发和优化至关重要[29] .在真实应用中,指针分析需要处理大量的内存访问和调用关系,直接影响编译器优化速度和静态分析工具性能.

**实验数据**:在 RQ2 中的鸿蒙应用代码规模较小,为了贴近真实生产需求,我们与合作单位共同收集了 35 个实际部署在鸿蒙操作系统上的应用字节码文件,以评估工具的运行效率.

**表 4** 鸿蒙真实应用分析效率

| 样例应用 | 应用大小 | 调用边数量 | 时间(秒) | 样例应用 | 应用大小 | 调用边数量 | 时间(秒) |
|---|---|---|---|---|---|---|---|
| AIlife | 10.55 MB | 10,687 | 66.30 | HwCompassOH | 216 KB | 126 | 0.07 |
| AILifeSve | 10.62 MB | 5,227 | 630.27 | HwHmosVAssitant-V | 1.66 MB | 4,911 | 31.43 |
| AppGalleryl | 2.12 MB | 5,595 | 20.00 | HwHmosVAssitant | 2.07 MB | 6,088 | 24.70 |
| AppGallery2 | 1.14 MB | 3,571 | 18.71 | HwSimToolkits | 250 KB | 973 | 0.20 |
| AppNotepad | 1.59 MB | 4,708 | 2.90 | HwStartupGuide | 424 KB | 553 | 0.98 |
| Browser | 2.15 MB | 5,895 | 12.33 | HwThemeManager | 2.32 MB | 6,672 | 32.414 |
| CalendarData | 210 KB | 512 | 0.17 | HwVmall | 2.39 MB | 6,188 | 316.192 |
| CallUI | 218 KB | 520 | 0.17 | Mms | 808 KB | 2,265 | 1.27 |
| CallUI-M | 95 KB | 191 | 0.08 | Music | 5.20 MB | 12,546 | 242.68 |
| CallUI-S | 146 KB | 344 | 0.17 | OUC | 1.05 MB | 3,022 | 5.70 |
| CeliaKeyboard | 1.8 MB | 5,194 | 12.12 | PetalClip | 1.64 MB | 4,556 | 19.27 |
| Camera | 870 KB | 2,881 | 1.36 | PhotosHm | 2.76 MB | 5,351 | 9.04 |
| Clock | 490 KB | 1,299 | 0.46 | Rocket | 3.54 MB | 3,329 | 4.25 |
| Contacts | 1.22 MB | 3,452 | 2.63 | Screenshot | 50 KB | 164 | 0.02 |
| Entry-default | 2.15 MB | 1,896 | 4.90 | Settings | 3.58 MB | 4,604 | 2.43 |
| Health | 4.66 MB | 9,757 | 1,519.00 | Tips | 1.27 MB | 3,145 | 1.69 |
| HongYan | 5.44 MB | 5,371 | 481.25 | Watt | 2.89 MB | 2,377 | 5.74 |
| HuaweiShare | 183 KB | 627 | 0.18 | | | | |

表 4 展示了 HarmonyFlow 在 35 个真实鸿蒙应用中的运行效率,包括应用大小、调用边数量以及运行时间.表中显示,这些应用的大小范围在 50KB 至 10.55MB 之间,分析得出的平均调用边数量为 3738 条,平均运行时间为 96 秒,能够满足下游静态分析任务的需求.从数据分析可以看出,应用大小与运行时间之间并未呈现严格的正相关关系.例如,Health 应用的大小为 4.66MB,但其运行时间却高达 1519 秒.通过反汇编分析发现,该应用导入了加密算法软件包,而加密算法中密集的数据处理语句显著增加了分析工具的时间开销.相比之下,调用边数量与运行时间的正相关性较强.例如,Health 和 Music 是运行时间较长的两个应用,分别为 1519 秒和 242.68 秒,这与它们较高的调用边数量(分别为 9757 和 12546)密切相关.总的来说,HarmonyFlow 在处理复杂应用时,调用边数量对运行时间的影响显著,尤其是循环调用和递归调用可能导致时间开销增加.同时,对于轻量级和中等规模的应用,HarmonyFlow 展现了较高的效率和稳定性.

## 5 讨 论

### 5.1 与安卓静态分析框架对比

与 HarmonyFlow 类似,安卓平台上也存在诸如 Soot、Wala、Doop、Qilin 和 Tai-e 等多个主流静态分析框架.然而,由于 Java 字节码与方舟字节码在语法结构与语义逻辑方面存在显著差异,现有分析框架难以实现对跨平台字节码文件的统一处理,致使无法进行直接的对比实验.为此,本文系统梳理了 Android 平台静态分析工具的发展脉络,从精确率、召回率与分析效率三个维度,对 HarmonyFlow 的优势与局限进行了深入分析.

在精确率方面,Soot、Doop 和 Qilin 通常通过统计识别出的调用边数量来衡量精确率.在默认召回率近似的假设下,调用边数量越少往往意味着更高的精确率.然而,最新研究 Tai-e 指出,现有工具在使用更全面的数据集时表现出召回率不一致,表明直接对比调用边数量未必能够准确反映精确率.因此,Android 平台静态分析框架缺乏可量化的精确率评价标准,难以与本文方法在数值层面进行直接比较.

指针分析中的敏感性设计原则具有平台无关性,下面在理论层面进行精确性对比.Soot 作为 Android 平台上最早的静态分析框架之一,首次处理 Java 字节码提出了 Jimple 中间表示.在此基础上,后续的多篇工作逐步增加了字段敏感性(SPARK)与上下文敏感性(Paddle),还引入了选择性上下文敏感机制(Qilin、Tai-e),以提升分析精确性.与之类似,HarmonyFlow 首次对方舟字节码进行抽象,设计了可建模关键指令的指针流图,并在此基础上实现了字段敏感指针分析算法.从理论上讲,HarmonyFlow 的字段敏感分析算法的精确性低于最新安卓工具的上下文敏感分析算法.不过,本文所提出的指针流图为鸿蒙应用中的指针分析提供了基础数据结构,并在官方开源的鸿蒙应用中实现了可接受的精确率(98.33%).随着鸿蒙生态系统不断发展,构建更大规模、更高复杂度的基准测试集,以及引入上下文敏感分析模块,将成为 HarmonyFlow 后续演进的重要方向.

在召回率层面,Tai-e 首次系统性地报告了各主流安卓静态分析工具的召回率,分别为:Soot 73.2%、Doop 68.4%、Qilin 83.5%、Tai-e 91.3%.本文提出的 HarmonyFlow 在同类实验设置下达到 92.22%的召回率,在数值上具有竞争力.一般而言,静态分析框架的召回能力取决于其对目标语言语法特性的支持程度.对 Android 工具而言,支持 Java 的反射机制是召回率提升的关键,Doop 因未充分解析反射信息而召回率最低.而在 ArkTS 语言中,分析的重点变为鸿蒙应用中广泛使用的回调机制的建模与支持.作为一门快速演进的新兴语言,ArkTS 未来可能引入更多独特的语法特性.因此,HarmonyFlow 需持续跟进语言层面的演进,确保语法支持的广度与深度.

在效率层面,考虑到 Android 平台 DaCapo 数据集与本文所用数据集在应用规模上的潜在差异,我们采用“单位兆字节的运行时间(s/MB)”作为评估指标.在相同硬件环境和分析敏感度下,Soot、Doop、Qilin 与 Tai-e 在字段敏感指针分析中的效率分别为:13.67s/MB、12.73s/MB、9.02s/MB 和 10.35s/MB;HarmonyFlow 在排除含加密模块的 Health 应用后效率为 13.65s/MB,与 Soot 接近,但低于其他后续工具.造成效率差异的主要原因在于分析架构设计的不同.Doop 采用声明式语言 Datalog 构建分析引擎,显著提高了执行效率;Qilin 与 Tai-e 则分别引入了增量式工作列表求解器和基于稀疏位集的指向集表示,进一步优化了性能.为提升 HarmonyFlow 的工程实用性和可扩展性,未来可参考上述优化手段,重构当前分析引擎,以获得更优的性能表现.

### 5.2 实验结果分析与改进思路

根据本文实验结果,HarmonyFlow 存在流敏感性不足和模块间依赖关系缺失的问题.下面我们详细讨论其成因和可能的解决方案.

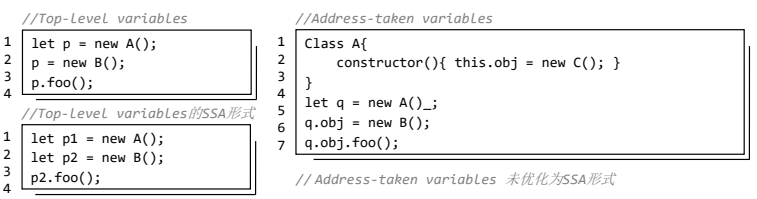



图 7 Panda IR 的 Partial SSA 优化

流敏感分析按照程序语句的执行顺序传播和更新变量的状态信息,因此能够区分变量在不同程序点具体取值.在图 7 所示的未经 SSA 优化的代码中,以顶级变量 p 为例,若采用流敏感分析,则其指向集合为$\{O_B\}$;而在流不敏感分析中,p 的指向集合则为$\{O_A,O_B\}$,这种不加区分的合并可能导致误报的产生.幸运的是,Panda IR 经过了方舟编译器的局部静态单赋值(Partial SSA)优化,对于不被指针引用的顶级变量(Top-level variables),已经将其多个定义拆分为多个单独变量实例,从而保证 IR 中每个变量只被赋值一次;而对于被指针引用的变量

(Address-token variables),由于结构复杂,保留其原有形式.针对 SSA 形式的顶级变量,本文的指针分析算法能够准确地将 p1 的指向集合推导为$\{O_A\}$,p2 为$\{O_B\}$,自然避免了误报.对于保留非 SSA 形式的被指针引用变量,需要我们对 Panda IR 做进一步优化.可参考稀疏流敏感指针分析(SFS)方法[30],引入 χ/υ 函数,将通过指针访问的变量转换为可追踪的“伪 SSA 形式”,再构造精细的 Def-Use 图,使这类变量也具备流敏感,有效提升指针分析的敏感度.

当前 HarmonyFlow 在处理跨模块调用时表现出不同程度的适应性,主要取决于模块的加载方式:静态加载或动态加载.对于静态加载场景,Panda IR 提供了较为完善的全局模块信息支持.具体而言,Panda IR 中的 ModuleRecord 能够维护类型与模块之间的映射关系,使得静态分析器可以基于接收对象的类型准确地推导其所属模块,并进一步解析跨模块的调用关系.因此,在静态加载情况下,HarmonyFlow 能够有效还原跨模块调用边.然而,在动态加载场景下,由于 Panda IR 在此类场景中并未预先维护类型与模块的映射信息,静态分析工具需依赖路径参数与字节码文件中模块名进行字符串层面的匹配.这种匹配方式高度依赖开发者实现细节,并缺乏统一规范,因此难以覆盖所有动态加载路径的变种.在模块名称被动态构造、路径混淆或加载逻辑较为复杂的情况下,字符串匹配策略容易失效,导致调用边识别失败.为提升分析的全面性与精确性,后续工作可考虑结合配置文件和资源路径分析,还原动态模块加载行为,从而进一步降低漏报率,提升调用图的完整性与准确性.

### 5.3 鸿蒙样本数据的局限性

目前,针对 Java 应用的静态分析已有广泛使用的 DaCapo 标准测试集[31].虽然 Arkanalyzer[7] 开源了鸿蒙应用的测试数据集,但未提供这些应用的调用边标签等关键程序分析指标.目前,我们采用抽样的方法手动评估调用边的准确性,但这种方式可能引入一定的偏差.此外,当前的工作主要面向 OpenHarmony 社区的开源鸿蒙应用,而对 HarmonyOS 和 Harmony NEXT 应用的兼容尚需进一步拓展.具体而言,OpenHarmony 是鸿蒙系统的底层内核,继承了方舟编译器,开发者可基于 OpenHarmony SDK 开发鸿蒙应用.HarmonyOS 是基于 OpenHarmony 与安卓(AOSP)开发的闭源手机操作系统,兼容安卓生态;而 Harmony NEXT 去除了对安卓(AOSP)的支持,代表了鸿蒙系统未来发展的方向.由于 HarmonyOS 和 Harmony NEXT 均为闭源环境,本研究基于 OpenHarmony 生态构建静态分析框架.

### 5.4 面向用户接口

HarmonyFlow 作为鸿蒙应用的基础静态分析框架,为用户提供了便捷易用的接口.在过程内分析方面,框架生成了基本块级别和指令级别的控制流图,并提供了指令间的 Def-Use 链,为进一步的数据流分析奠定了基础.在指针分析方面,框架提供指令的指向集合查询接口和调用图构建接口,同时实现了指令间的别名分析接口.基于这些指针分析能力,HarmonyFlow 优化了数据依赖关系,提供了更加精确的过程间 Def-Use 链,同时支持生成过程间控制流图.尽管如此,HarmonyFlow 的功能仍需进一步完善,以满足更多应用层面的分析需求.在数据流分析方面,仍可扩展实现包括定义可达性分析、活跃变量分析以及可用表达式分析等经典分析方法.在安全性检测方面,污点分析和并发错误检测是常见的拓展方向,这些功能的实现将进一步提升框架的实用性和适用范围.

### 5.5 框架更新迭代

OpenHarmony 生态仍在持续发展和演进的过程中,方舟编译器的 Panda IR 也在不断优化和完善,为开发者提供更高效的中间表示格式.HarmonyFlow 以 Panda IR 为主要分析对象,因此需要随着 Panda IR 的优化不断调整和升级,确保分析的准确性与效率能够满足新的技术需求.

此外,OpenHarmony SDK 也在持续拓展,加入了更多功能模块和新特性,以支持更丰富的应用场景.为了适应这些变化,HarmonyFlow 需要对新特性进行及时的兼容性更新和功能扩展,使其在分析能力上能够覆盖更多复杂的开发需求,同时为开发者提供更加全面的静态.

## 6 总 结

HarmonyFlow 旨在为鸿蒙应用提供基础的静态分析能力,为此,我们克服了 Panda IR 信息处理的复杂性、

ArkTS 语法特性的复杂性以及静态分析框架可扩展性等三大挑战.实验评估表明,HarmonyFlow 能够较全面地支持 ArkTS 的语法特性,并在鸿蒙应用的调用边识别任务中表现出较高的精度与运行效率.随着鸿蒙生态的持续发展和完善,HarmonyFlow 将在更多鸿蒙应用场景中进行优化和迭代,并将持续更新以长期支持不断演进的 OpenHarmony 系统.

**References**:


[1] Vallée-Rai R, Co P, Gagnon E, Hendren L, Lam P, Sundaresan V. Soot: A Java bytecode optimization framework. In: CASCON First Decade High Impact Papers. Toronto: IBM Canada Ltd, 2010. 214-224.[doi: 10.1145/1925805.1925818]

[2] IBM. WALA: T.J. Watson Libraries for Analysis. 2020. http://wala.sourceforge.net/

[3] Bravenboer M, Smaragdakis Y. Strictly declarative specification of sophisticated points-to analyses. In: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages and Applications (OOPSLA). New York: ACM Press, 2009. 243-262. [doi: 10.1145/1640089.1640108]

[4] He DJ, Lu JB, Xue JL. Qilin: A new framework for supporting fine-grained context-sensitivity in Java pointer analysis. In: 36th European Conference on Object-Oriented Programming (ECOOP 2022). Schloss Dagstuhl: Leibniz-Zentrum für Informatik, 2022. 30:1–30:29. [doi: 10.4230/LIPIcs.ECOOP.2022.30]

[5] Tan T, Li Y. Tai-e: A developer-friendly static analysis framework for Java by harnessing the good designs of classics. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2023. 1093–1105. [doi: 10.1145/3597926.3598120]

[6] Li L, Gao X, Sun HL, Hu CM, Sun XY, Wang HY, Cai HP, Su T, Luo XP, Bissyande. Software engineering for OpenHarmony: A research roadmap. ACM Computing Surveys, 2023. [doi: 10.1145/3720538]

[7] Chen HN, Chen DH, Yang YZ, Xu LY, Gao L, Zhou MY, Hu CM, Li L. ArkAnalyzer: The static analysis framework for OpenHarmony. arXiv:2501.05798, 2025. [doi: 10.48550/arXiv.2501.05798]

[8] Click C, Paleczny M. A simple graph-based intermediate representation. ACM Sigplan Notices, 1995,30(3):35-49. [doi: 10.1145/202530.202534]

[9] Li HF, Meng HN, Zheng HJ, Cao LQ, Li L. Context-sensitive pointer analysis for object-oriented programs. Journal of Software, 2022,33(1):78-101. http://www.jos.org.cn/1000-9825/6345.htm [doi: 10.13328/j.cnki.jos.006345]

[10] Lhoták O, Hendren L. Scaling Java points-to analysis using Spark. In: Knoop J, ed. Compiler Construction: International Conference on Compiler Construction (CC 2003). Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.153–169. [doi: 10.1007/3-540-36579-6_12]

[11] Lhoták O, Hendren L. Evaluating the benefits of context-sensitive points-to analysis using a BDD-based implementation. ACM Trans. Softw. Eng. Methodol., 2008,18(1):1–35. [doi: 10.1145/1391984.1391987]

[12] Schubert PD, Hermann B, Bodden E. Phasar: An inter-procedural static analysis framework for C/C++. In: Margaria T, Steffen B, eds. International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Cham: Springer International Publishing, 2019. 393–410. [doi: 10.1007/978-3-030-17465-1_22]

[13] Sui YL, Xue JL. SVF: interprocedural static value-flow analysis in LLVM. In: Bodík R, ed. Proceedings of the 25th International Conference on Compiler Construction (CC). New York: ACM Press, 2016. 265–266. [doi: 10.1145/2892208.2892235]

[14] Li W, He DJ, Gui YJ, Chen WG, Xue JL. A context-sensitive pointer analysis framework for Rust and its application to call graph construction. In: Pizlo F, ed. Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction (CC). New York: ACM Press, 2024. 60–72. [doi: 10.1145/3640537.3641574]

[15] Kashyap V, Dewey K, Kuefner EA, Wagner J, Gibbons K, Sarracino J, Wiedermann B, Hardekopf B. JSAI: A static analysis platform for JavaScript. In: Ernst M D, ed. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). New York: ACM Press, 2014. 121–132. [doi: 10.1145/2635868.2635904]

[16] Andersen LO. Program analysis and specialization for the C programming language [Ph.D. Dissertation]. Copenhagen: University of Copenhagen, 1994.

[17] An X, Jia XQ, Du HC, Xie YM. Structure-Sensitive Pointer Analysis for Multi-structure Objects. In: Chen G, ed. Proceedings of the 15th Asia-Pacific Symposium on Internetware. New York: ACM Press, 2024. 155–164. [doi: 10.1145/3671016.3671396]

[18] Li Y, Tan T, Møller A, Smaragdakis Y. A principled approach to selective context sensitivity for pointer analysis. ACM Transactions on Programming Languages and Systems (TOPLAS), 2020,42(2):1–40. [doi: 10.1145/3381915]

[19] Shen TQ, Wang XZ, Bin XR, Bu L. DDoop: An incremental points-to analysis framework based on differential Datalog solving. Journal of Software, 2024,35(6):2608-2630. https://link.cnki.net/doi/10.13328/j.cnki.jos.007100 [doi: 10.13328/j.cnki.jos.007100]

[20] Guo B, Bridges MJ, Triantafyllis S, Ottoni G, Raman E, August DI. Practical and accurate low-level pointer analysis. In: International Symposium on Code Generation and Optimization. IEEE, 2005. 291-302. [doi: 10.1109/CGO.2005.27]

[21] Prakash J, Tiwari A, Hammer C.PointEval: On the Impact of Pointer Analysis Frameworks. arxiv:1912.00429, 2019. [doi: 10.48550/arXiv.1912.00429]

[22] Jang D, Choe KM. Points-to analysis for JavaScript. In: Proceedings of the 2009 ACM Symposium on Applied Computing. 2009. 1930-1937. [doi: 10.1145/1529282.1529711]

[23] Sridharan M, Dolby J, Chandra S, Schäfer M, Tip F. Correlation tracking for points-to analysis of JavaScript. In: Hauptmann S, ed. ECOOP 2012 – Object-Oriented Programming: 26th European Conference, Beijing, China, June 11-16, 2012. Proceedings 26. Berlin: Springer Berlin Heidelberg, 2012. 435-458. [doi: 10.1007/978-3-642-31057-7_20]

[24] Feldthaus A, Schäfer M, Sridharan M, Dolby J, Tip F. Efficient construction of approximate call graphs for JavaScript IDE services. In: Proceedings of the 2013 35th International Conference on Software Engineering (ICSE). Los Alamitos: IEEE, 2013. 752-761. [doi: 10.1109/ICSE.2013.6606621]

[25] LLVM Foundation. LLVM Language Reference Manual. 2025. https://llvm.org/docs/LangRef.html

[26] Demange D, Fernández de Retana Y, Pichardie D. Semantic reasoning about the sea of nodes. In: Proceedings of the 27th International Conference on Compiler Construction. 2018. 163-173. [doi: 10.1145/3178372.3179503]

[27] Lhotak O, Smaragdakis Y, Sridharan M. Pointer analysis (dagstuhl seminar 13162). Dagstuhl Reports, 2013,3(4):91-113. [doi: 10.4230/DagRep.3.4.91]

[28] Openharmony. A comprehensive open source project for all-scenario, fully-connected, and intelligent era. 2025. https://gitee.com/openharmony

[29] Nikolić D, Stefanović D, Dakić D, Sladojević S; Ristić S. Analysis of the tools for static code analysis. In: 2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH). IEEE, 2021. 1-6. [doi: 10.1109/INFOTEH51037.2021.9400688]

[30] Hardekopf B, Lin C. Flow-sensitive pointer analysis for millions of lines of code. In: International Symposium on Code Generation and Optimization (CGO 2011). IEEE, 2011:289-298. [doi: 10.1109/CGO.2011.5764696]

[31] Blackburn SM, Garner R, Hoffmann C, Khang AM, McKinley KS, Bentzur R, Diwan A, Feinberg D, Frampton D, Guyer SZ, Hirzel M, Hosking A, Jump M, Lee H, Moss JEB, Phansalkar A, Stefanović D, VanDrunen T, von Dincklage D, Wiedermann B. The DaCapo benchmarks: Java benchmarking development and analysis. In: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. 2006.169-190. [doi: 10.1145/1167473.1167488]

## 附中文参考文献:

[9] 李昊峰, 孟海宁, 郑恒杰, 曹立庆, 李炼. 面向对象程序的上下文敏感指针分析研究. 软件学报, 2022,33(1):78-101. http://www.jos.org.cn/1000-9825/6345.htm [doi: 10.13328/j.cnki.jos.006345]

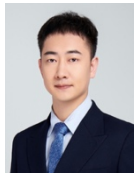
[19] 沈天琪, 王熙灶, 宾向荣, 卜磊. DDoop: 基于差分式 Datalog 求解的增量指针分析框架. 软件学报, 2024,35(6):2608-2630. https://link.cnki.net/doi/10.13328/j.cnki.jos.007100 [doi: 10.13328/j.cnki.jos.007100]

**王越**(2000－),男,硕士,主要研究领域为程序分析,漏洞验证.

**李佳洛**(2001－),女,硕士,主要研究领域为软件安全.

**柴仁超**(2002－),男,硕士,主要研究领域为程序分析,网络安全.

**黎立**(1994－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为移动软件工程,智能化软件分析与测试.



**陈森**(1990－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件安全,软件供应链安全.

**党文婧**(2000－),女,博士,CCF 学生会员,主要研究领域为网络空间安全,漏洞验证.

**石子跃**(2002－),男,硕士,主要研究领域软件测试,可信人工智能.