

ITMO UNIVERSITY

Поиск аномалий в исходном Kotlin-коде и байт-коде

Петухов Виктор

Университет ИТМО
Кафедра Компьютерных технологий

Предметная область

- ✓ Аномалия — нетипичный по тем или иным меркам код

- ✓ Разработчики Kotlin хотят видеть экзотические примеры использования языка для разных целей

Объекты анализа

- ✓ PSI дерево разбора Kotlin-кода
- ✓ JVM байт-код (from Kotlin)
- ✓ JVM back-end IR (разрабатывается)
- ✓ Kotlin/Native IR (уже есть, можно пробовать)
- ✓ LLVM-код (from Kotlin, можно пробовать)
- ✓ Javascript (from Kotlin, можно пробовать)

3/26

Потенциальная польза аномалий на PSI

- ✓ Можно смотреть на решаемую задачу и делать выводы о недостатках в дизайне языка, о надобности разработки новых конструкций

- ✓ На таком коде компилятор языка либо не изучен, либо малоизучен => можно использовать аномалии как тесты компилятора (например, на производительность)

Потенциальная польза аномалий на байт-коде

- ✓ Польза в составе условных аномалий
- ✓ Условная аномалия — аномалия на байт-коде, но не аномалия на PSI, и наоборот
- ✓ Можно пробовать давать оценку работе кодогенератора и оптимизаций компилятора

Потенциальная польза условных аномалий по байт-коду

- ✓ Негативный случай: Было сгенерировано больше байт-кода, чем ожидалось
 - Возможные проблемы в кодогенерации

- ✓ Позитивный случай: Удачная реализация некоторой конструкции языка
 - Для большого кол-ва низкоуровневых действий удалось придумать элегантную конструкцию в языке
 - Тоже интересно смотреть: можно перенять опыт реализации таких конструкций

- ✓ Нужно научиться отделять

Потенциальная польза условных аномалий по PSI

- ✓ Сложное PSI свернулось в простой байт-код
- ✓ Вероятно, можно говорить об удачно сработавших оптимизациях компилятора
- ✓ Можно пробовать отслеживать эффективность оптимизаций, проводя на таких примерах регрессионное тестирование компилятора

Постановка задачи

- ✓ Аномалию сложно описать в терминах классических алгоритмов, «нетипичность» может быть выражена в самых разных свойствах
- ✓ Будем решать задачу обнаружения аномалий, уже поставленную в ML
- ✓ Предварительно будем осуществлять векторизацию исходного и байт-кода
- ✓ Будем собирать два набора данных: для простых аномалий и условных

Сбор данных

- ✓ GitHub — одно из крупнейших хранилищ открытого Kotlin-кода — будем использовать его для сбора кода
- ✓ 47 тыс. репозиториев с main language = Kotlin
- ✓ Байт-код собираем из релизных файлов репозитория
- ✓ Результаты:
 - Для малого датасета: собрано **40 тыс.** файлов с исходным кодом и **8 тыс.** файлов с байт-кодом
 - Для большого датасета: собрано **930 тыс.** файлов с исходным кодом на Kotlin и **63 тыс.** файлов с байт-кодом

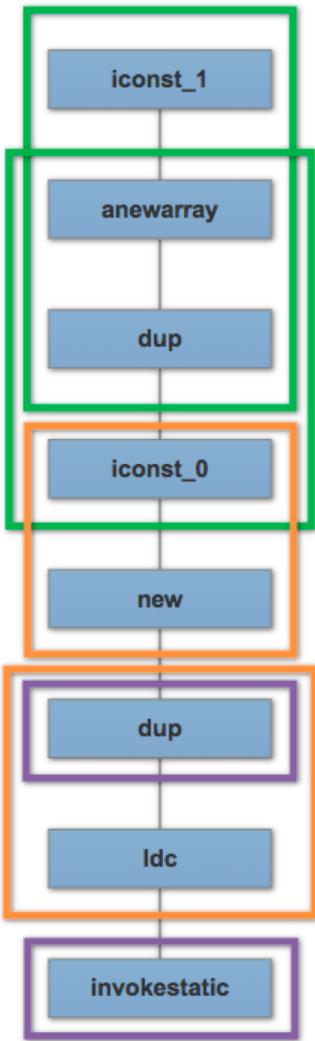
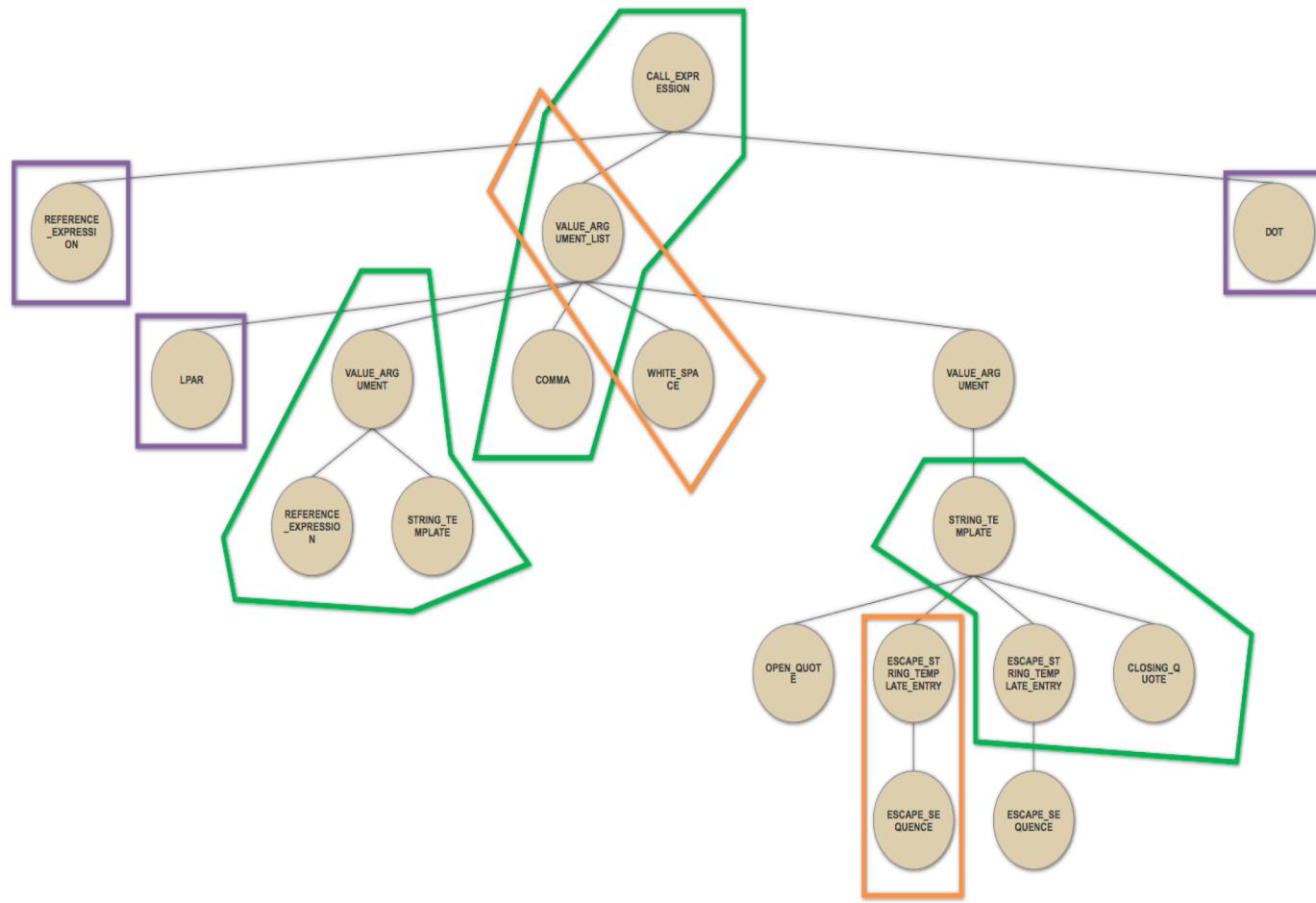
Сопоставление исходного кода и байт-кода

- ✓ Для дальнейшего формирования условных аномалий нужно иметь сопоставленные файлы исходного кода файлам с байт-кодом

- ✓ Был разработан соответствующий инструмент:
 - Достает из мета-информации class-файлов имя пакета и исходника
 - Достает из PSI имя пакета
 - Сопоставляет имена пакетов и файлов

Векторизация кода

- ✓ Векторное представление исходного и байт-кода будем формировать путем разложения на n-граммы (uni-, bi- и 3-граммы)
- ✓ Автокодирование позволяет извлекать неявные признаки, которые может быть сложно извлечь вручную и интерпретировать
- ✓ Результаты:
 - Для большого датасета: **11 тыс.** n-gram по PSI и **111 тыс.** n-gram по байт-коду
 - Для малого датасета: **5 тыс.** n-gram по PSI и **79 тыс.** n-gram по байт-коду



Векторизация байт-кода

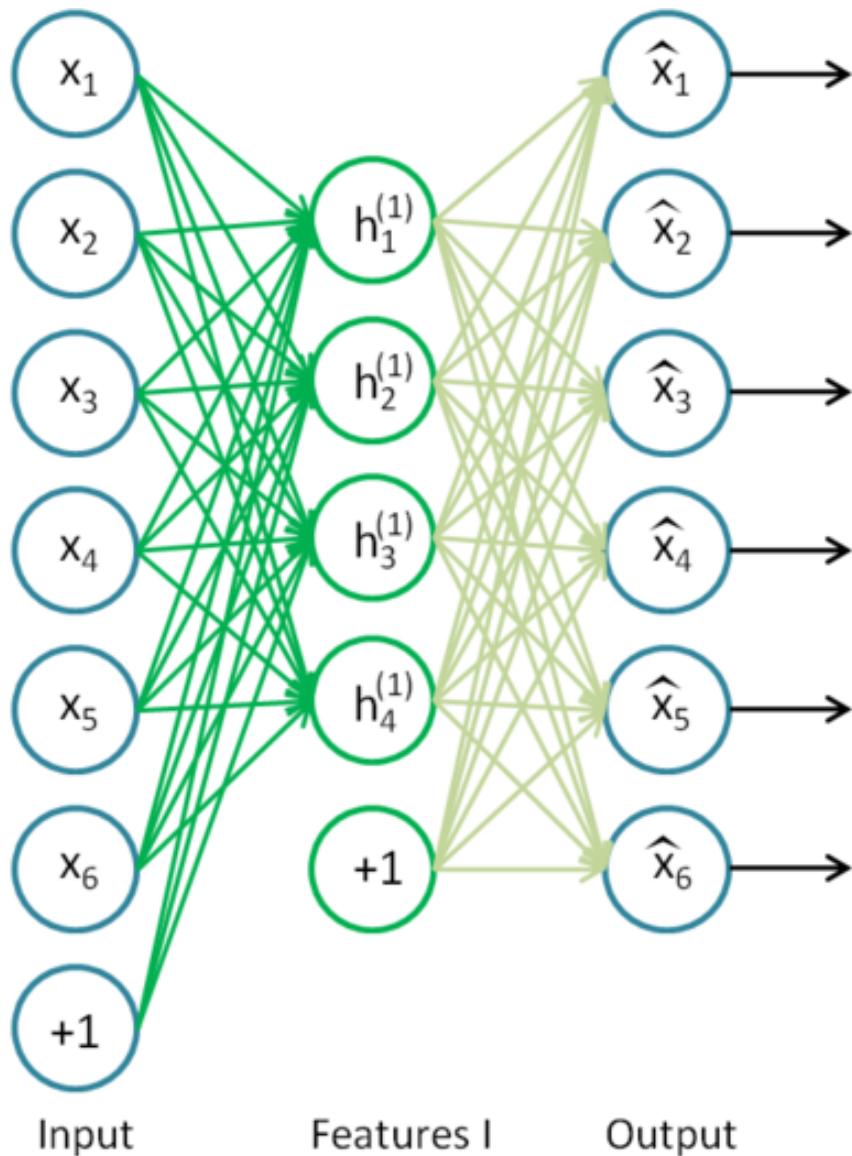
Векторизация
байт-кода

Методы решения задачи обнаружения аномалий

- ✓ Метод опорных векторов с одним классом (OC SVM)
- ✓ Методы, основанные на кластеризации (например, k-means)
- ✓ Метод k-ближайших соседей (k-nearest neighbor)
- ✓ Статистические методы
- ✓ Репликаторные нейросети, автоэнкодеры

Автоэнкодер

- ✓ Для решения задачи был выбран автоэнкодер:
 - Обучение без учителя (примеров аномалий изначально нет)
 - Не двоичный выход — ошибку восстановления можно трактовать как «степень аномальности»
 - Встроенные механизмы сокращения размерности — можно формировать для этого отдельные слои в архитектуре сети: признаков очень много



- ✓ Простая архитектура — один скрытый слой
- ✓ Несколько экспериментов с коэффициентом сжатия **0.8** и **0.5**

Отбор признаков

- ✓ На байт-коде n-грамм очень много, нужно сокращать размерность
- ✓ Был разработан инструмент для отсечения ТОПа частотного списка n-грамм
- ✓ Предположительно, такие n-граммы должны вносить минимальный вклад в формирование аномалий
- ✓ Результаты:
 - По большому датасету: отобрано **25 тыс.** n-грамм
 - По малому датасету: отобрано **2.8 тыс.** n-грамм

Результаты

- ✓ По результатам работы автоэнкодера были вычислены евклидовые расстояния между входными и выходными векторами
- ✓ Расстояния трактовались как «степени аномальности»
- ✓ В качестве аномалий были отобраны примеры, у которых расстояние от входного вектора до выходного отклонялось от среднего более чем на 3-сигма или 5-сигма
- ✓ Итого было получено аномалий:
 - По большому датасету: **4924** по PSI и **456** по байт-коду
 - По малому датасету: **362** по PSI и **151** по байт-коду

Классификация аномалий

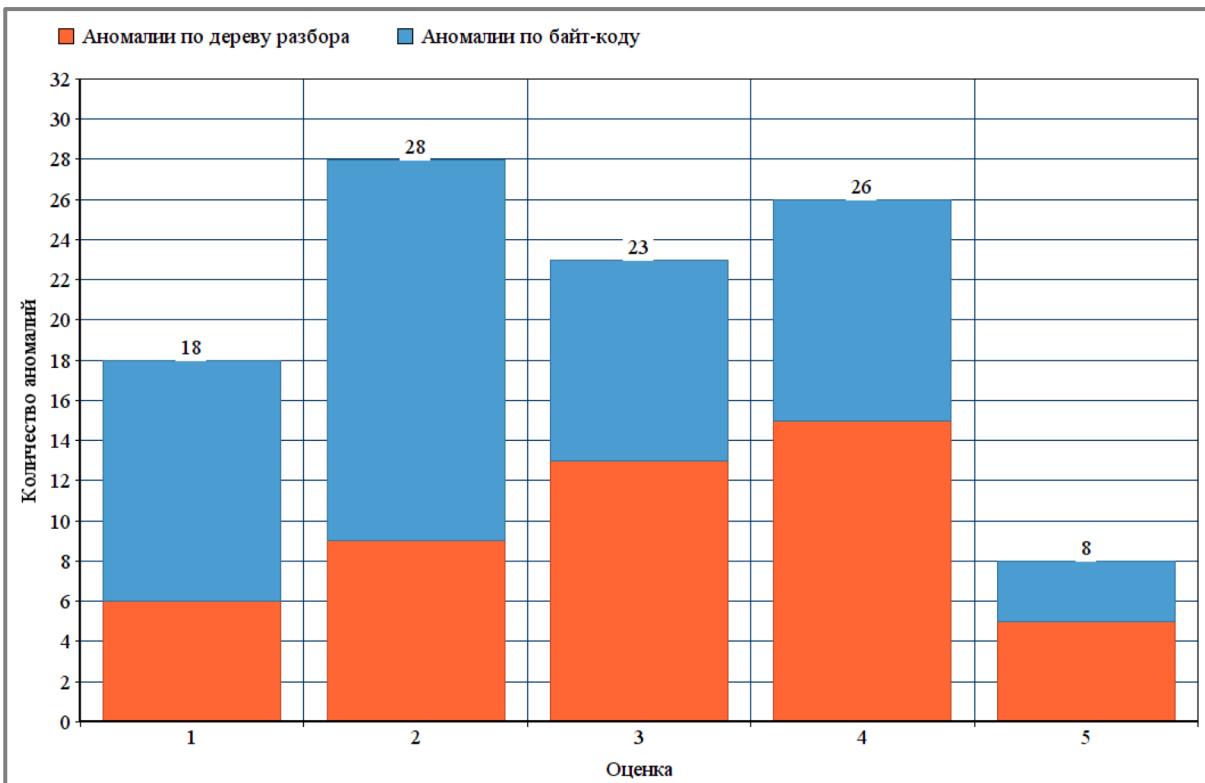
- ✓ Найденные аномалии по малому датасету были вручную классифицированы

- ✓ По **513** аномалиям было выделено **30** классов:
 - В **25** классах присутствовали аномалии по байт-коду
 - В **10** классах присутствовали аномалии по PSI
 - В **10** классах присутствовали аномалии по PSI и байт-коду



Сбор экспертных оценок

- ✓ Была разработана функциональность на веб-сайте для оценивания классов и аномалий разработчиками Kotlin



- ✓ Среднее по классам — **2.462**
- ✓ Среднее по аномалиям — **2.786**
 - По PSI — **3.083**
 - По байт-коду — **2.527**
- ✓ **12** классов с оценками 4 и 5
- ✓ **34** аномалии с оценками 4 и 5

20/26

Пример аномалии из класса Many generic parameters по дереву разбора с оценкой 5

```
@Suppress("UNCHECKED_CAST")
fun <V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20, E> concreteCombine(p1: Promise<V1, E>,
                                                                 p2: Promise<V2, E>,
                                                                 p3: Promise<V3, E>,
                                                                 p4: Promise<V4, E>,
                                                                 p5: Promise<V5, E>,
                                                                 p6: Promise<V6, E>,
                                                                 p7: Promise<V7, E>,
                                                                 p8: Promise<V8, E>,
                                                                 p9: Promise<V9, E>,
                                                                 p10: Promise<V10, E>,
                                                                 p11: Promise<V11, E>,
                                                                 p12: Promise<V12, E>,
                                                                 p13: Promise<V13, E>,
                                                                 p14: Promise<V14, E>,
                                                                 p15: Promise<V15, E>,
                                                                 p16: Promise<V16, E>,
                                                                 p17: Promise<V17, E>,
                                                                 p18: Promise<V18, E>,
                                                                 p19: Promise<V19, E>,
                                                                 p20: Promise<V20, E>):Promise<Tuple20<V1, ... (and 19 similar parameters)
val deferred = deferred<Tuple20<V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20>, E>()
val results = AtomicReferenceArray<Any?>(20)
val successCount = AtomicInteger(20)

fun createTuple(): Tuple20<V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V20> {
    return Tuple20(
        results.get(0) as V1,
        results.get(1) as V2,
        (and 19 similar expressions)
```

Пример аномалии из класса Long enumerations по дереву разбора с оценкой 5

```
@Suppress("PLATFORM_CLASS_MAPPED_TO_KOTLIN")
    override fun predicate(fromType: Type, toType: Type): Boolean {
        val fromErased = fromType.erasedType()
        return when {
            String::class.isAssignableFrom(fromType) -> when (toType) {
                String::class.java,
                Short::class.java, java.lang.Short::class.java,
                Byte::class.java, java.lang.Byte::class.java,
                Int::class.java, Integer::class.java,
                Long::class.java, java.lang.Long::class.java,
                Double::class.java, java.lang.Double::class.java,
                Float::class.java, java.lang.Float::class.java,
                BigDecimal::class.java,
                BigInteger::class.java,
                CharSequence::class.java,
                ByteArray::class.java,
                Boolean::class.java, java.lang.Boolean::class.java,
                File::class.java,
                URL::class.java,
                URI::class.java -> true
                else -> {
                    val toErased = toType.erasedType()
                    toErased.isEnum
                }
            }
            CharSequence::class.isAssignableFrom(fromType) -> when (toType) {
                CharSequence::class.java,
                String::class.java,
                ByteArray::class.java -> true
                else -> false
            }
            Number::class.isAssignableFrom(fromType) -> when (toType) {
                Short::class.java, java.lang.Short::class.java,
                Byte::class.java, java.lang.Byte::class.java,
                Int::class.java, Integer::class.java,
                (and 9 more similar when statements)
```

Формирование условных аномалий

- ✓ Всем аномалиям по байт-коду были сопоставлены «степени аномальности» по PSI и наоборот
- ✓ Были вычислены разности «степеней аномальности» и получен отсортированный по ним список
- ✓ Как условные аномалии были помечены примеры с отклонением от СКО/4 («голова» и «хвост» списков):
 - 6 условных аномалий по байт-коду
 - 32 условных аномалии по PSI

Пример условной аномалии по байт-коду 1

```
class ConfigModel(val configs: Config) : ViewModel() {  
    val ip = bind { configs.ipProperty }  
    val DataBase = bind { configs.dataBaseProperty }  
    val rootUser = bind { configs.rootUserProperty }  
    val password = bind { configs.passwordProperty }  
    val tableName = bind { configs.tableNameProperty }  
    val entityName = bind { configs.entityNameProperty }  
    val entityPackage = bind { configs.entityPackageProperty }  
    val mapperPackage = bind { configs.mapperPackageProperty }  
    val servicePackage = bind { configs.servicePackageProperty }  
}
```



```
{  
    "getIp" : [ "aload_0", "getfield", "areturn" ],  
    "getDataBase" : [ "aload_0", "getfield", "areturn" ],  
    "getRootUser" : [ "aload_0", "getfield", "areturn" ],  
    "getPassword" : [ "aload_0", "getfield", "areturn" ],  
    "getTableName" : [ "aload_0", "getfield", "areturn" ],  
    "getEntityName" : [ "aload_0", "getfield", "areturn" ],  
    "getEntityPackage" : [ "aload_0", "getfield", "areturn" ],  
    "getMapperPackage" : [ "aload_0", "getfield", "areturn" ],  
    "getServicePackage" : [ "aload_0", "getfield", "areturn" ],  
    "getConfigs" : [ "aload_0", "getfield", "areturn" ],  
    "<init>" : [ "aload_1", "ldc", "invokestatic", "aload_0", ... (4428 instructions)  
}
```

Планы

- ✓ Автоматическая классификация аномалий, фокус на обнаружение новых классов
- ✓ Проведение экспериментов с оценкой влияния привнесенных в язык фич (с перспективной разработки тестов на такой системе)
- ✓ Проведение экспериментов по отслеживанию эффективности оптимизаций компилятора
- ✓ Проведение экспериментов с анализом не расстояний между векторами, а векторов разностей (и с классификацией после через DBScan), и попробовать выявлять происхождение аномалий (n-граммы)
- ✓ Проведение аналогичного исследования в Kotlin/Native, Kotlin/JS и на IR



ITMO UNIVERSITY

Спасибо за внимание!

Петухов Виктор

Кафедра Компьютерных технологий

Университет ИТМО

i@victor.am

<http://victor.am>

Санкт-Петербург, 2018