

ITMO UNIVERSITY

Обнаружение проблем производительности в
программах на языке программирования Kotlin с
использованием статического анализа кода

Петухов Виктор, гр. М4238

Руководитель: Фильченков А. А., к. ф-м. н., доц. каф. КТ

Рецензент: Брыксин Т.А., к.т.н., доц. каф. системного программирования СПбГУ

Консультант: Поваров Н.И., аналитик, ООО «Интеллиджей ЛАБС»

Кафедра Компьютерных технологий

Предметная область

- ✓ Производительность в разработке ПО — одно из важнейших нефункциональных требований
- ✓ Часть проблем производительности может заключаться в компиляторе языка — неоптимальной кодогенерации или проблемах в оптимизациях
- ✓ Для их обнаружения разработчики компилятора Kotlin хотят видеть экзотические примеры использования языка (аномалии) с целью тестирования компилятора на них

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 области машинного обучения
- ✓ Данную задачу можно разделить на две подзадачи:
 - векторизация кода,
 - поиск аномалий.

Существующие решения

- ✓ Graph-based Mining of Multiple Object Usage Patterns
- ✓ Detecting Object Usage Anomalies
- ✓ A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors
- ✓ Детектор аномалий в программах на языке Kotlin

Цели исследования

Разработать решение, позволяющее на некотором наборе исходного кода на Kotlin и байт-кода осуществлять поиск аномалий

Провести ряд экспериментов, направленных на получение большего числа аномалий

Произвести сравнение полученных результатов между собой и с результатами похожей работы

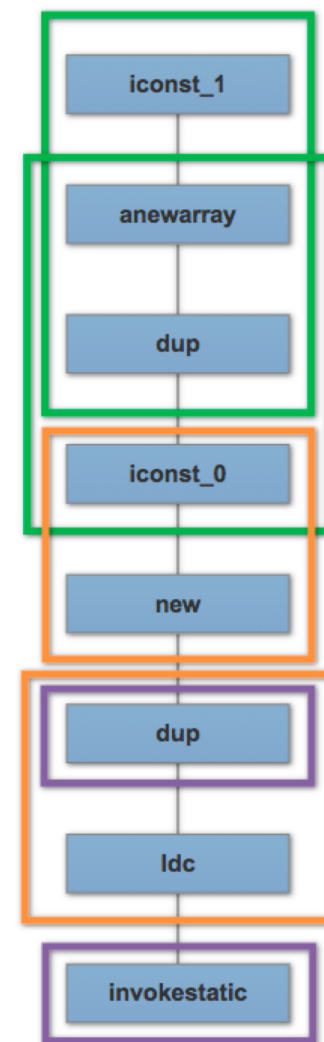
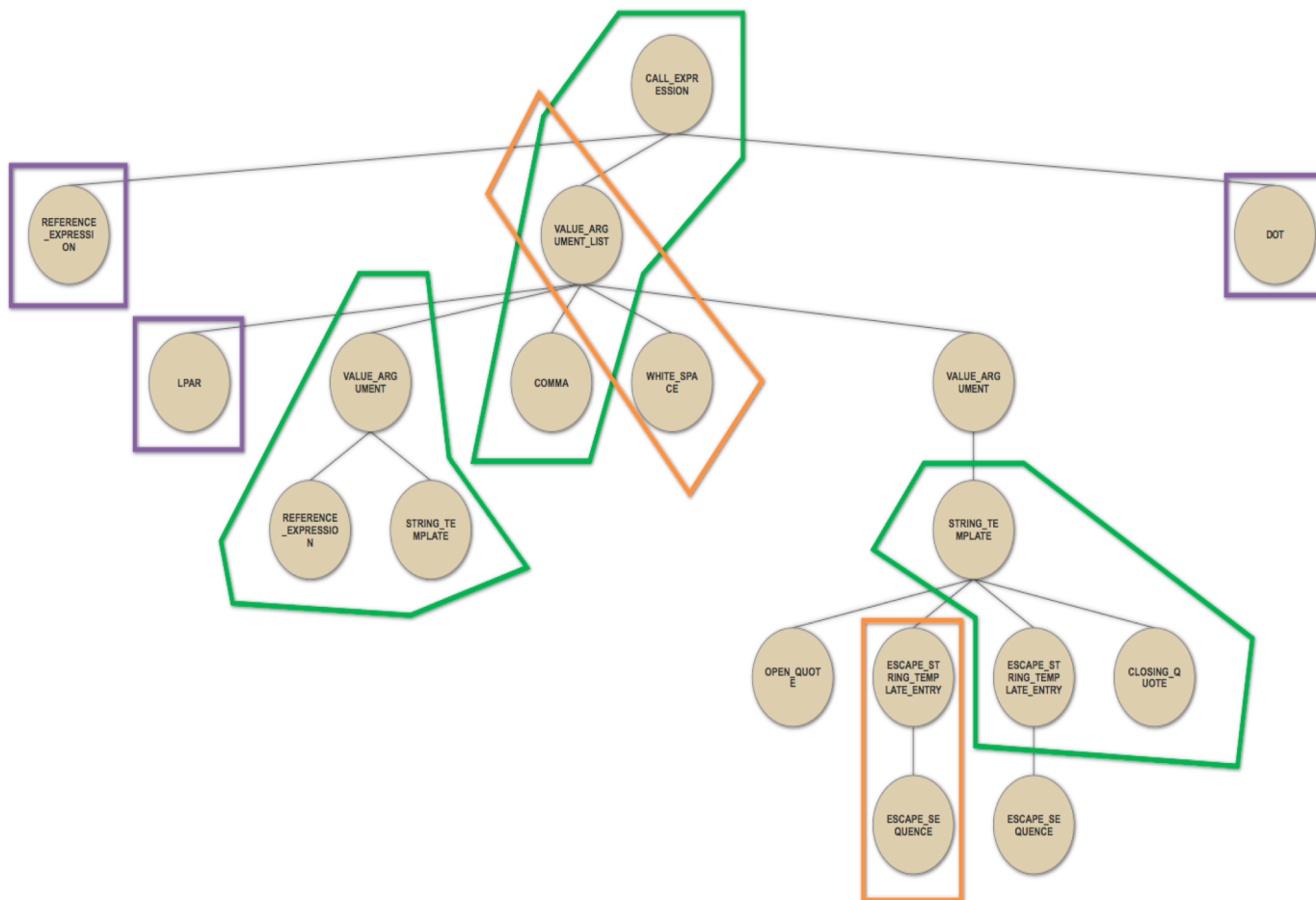
Получить экспертные оценки результатов

Научная новизна

- ✓ ...заключается в применении алгоритмов обнаружения аномалий для тестирования производительности компилятора и дальнейшего возможного обнаружения проблем в ней (путем поиска экзотических примеров использования языка)

Анализ задачи: векторизация кода

- ✓ Разные подходы к извлечению признаков:
 - ручное извлечение,
 - автокодирование.
- ✓ Будем использовать автокодирование путем разложения на n -граммы — оно позволяет извлекать неявные признаки:
 - Их сложно извлекать вручную и интерпретировать человеку;
 - Но они могут вносить существенный вклад в формирование аномалий.
- ✓ Произведем сравнение с ручным извлечением, используемым в похожей работе



Векторизация дерева разбора

Векторизация
байт-кода

Анализ задачи: обнаружение аномалий

- ✓ Существующие техники:
 - OC SVM,
 - local outlier factor (k-ближайших соседей),
 - изолирующий лес,
 - elliptic envelope (статистические методы),
 - автоэнкодеры.
- ✓ Будем пробовать использовать все перечисленные методы с последующим сравнением результатом
- ✓ Предположительно, часть методов использовать не удастся из-за очень большого числа признаков

Разработка решения

- ✓ Было разработано решение, выполняющее следующие задачи:
 - Сбор исходного кода на Kotlin и байт-кода с GitHub;
 - Получение дерева разбора и списка JVM инструкций;
 - Векторизация кода путем его разложения на униграммы, биграммы и триграммы;
 - Обнаружение аномалий.
- ✓ На выходе — список файлов с исходным кодом и байт-кодом, помеченных как аномалии (с численной оценкой аномальности)

Эксперимент: сбор данных

- ✓ Было собрано **930 тыс.** файлов с исходным кодом на Kotlin
- ✓ Было собрано **63 тыс.** файлов с JVM байт-кодом

Эксперимент: векторизация кода

Таблица 1. Количество извлеченных n-грамм для разных параметров

Тип исходных данных	Параметры	$n_{max} = 3$ $distance_{max} = 3$ тип = 1	$n_{max} = 3$ $distance_{max} = 3$ тип = 2	$n_{max} = 3$ $distance_{max} = 0$ тип = 2
		Вариант 1	Вариант 2	Вариант 3
Дерево разбора		4 тыс.	61 тыс.	11 тыс.
Байт-код		180 тыс.	180 тыс.	111 тыс.

- ✓ тип = 1 — учёт связей только типа «родитель-ребенок»
- ✓ тип = 2 — учёт произвольных связей
- ✓ *distance* — максимальное расстояние между двумя узлами в n-грамме (плавающее окно)

Эксперимент: поиск аномалий

- ✓ Из предложенных методов в связи с ресурсными ограничениями удалось использовать лишь автоэнкодер
- ✓ По результатам его работы были вычислены ошибки восстановления — евклидовы расстояния между входными и выходными векторами
- ✓ Данные расстояния трактовались как оценки аномальности
- ✓ Аномалиями считались примеры с оценками, отклоняющимися от среднего более чем на 3 СКО

Эксперимент: поиск аномалий

Таблица 2. Количество полученных аномалий для разных параметров с набором данных 1

Тип исходных данных	Пар-ры	$dim = 0.25$ набор данных = 1	$dim = 0.5$ набор данных = 1	$dim = 0.75$ набор данных = 1
Дерево-разбора		350	338	320
Байт-код		58	56	55

Таблица 3. Количество полученных аномалий для разных параметров с набором данных 2

Тип исходных данных	Пар-ры	$dim = 0.25$ набор данных = 2	$dim = 0.5$ набор данных = 2	$dim = 0.75$ набор данных = 2
Дерево-разбора		751	712	680
Байт-код		58	56	55

Таблица 4. Количество полученных аномалий для разных параметров с набором данных 3

Тип исходных данных	Пар-ры	$dim = 0.25$ набор данных = 3	$dim = 0.5$ набор данных = 3	$dim = 0.75$ набор данных = 3
Дерево-разбора		1505	1351	1306
Байт-код		126	120	120

Сравнение результатов

Таблица 5. Количество новых обнаруженных аномалий по результатам разных экспериментов

Набор аномалий	$N_d = 1$ $N_v = 1$	$N_d = 1$ $N_v = 2$	$N_d = 1$ $N_v = 3$	$N_d = 2$ $N_v = 1$	$N_d = 2$ $N_v = 2$	$N_d = 2$ $N_v = 3$	$N_d = 3$ $N_v = 1$	$N_d = 3$ $N_v = 2$	$N_d = 3$ $N_v = 3$
$N_d = 1, N_v = 1$		1 / 0	2 / 0	428 / 0	389 / 0	356 / 0	1150 / 65	998 / 60	955 / 59
$N_d = 1, N_v = 2$	11 / 2		0 / 0	412 / 2	372 / 0	340 / 0	1160 / 64	1105 / 59	965 / 58
$N_d = 1, N_v = 3$	28 / 3	18 / 1		429 / 3	390 / 1	357 / 0	1180 / 68	1026 / 58	970 / 61
$N_d = 2, N_v = 1$	27 / 0	1 / 0	2 / 0		4 / 0	5 / 0	725 / 65	589 / 60	542 / 59
$N_d = 2, N_v = 2$	27 / 2	2 / 0	2 / 0	35 / 2		2 / 0	780 / 64	631 / 59	582 / 58
$N_d = 2, N_v = 3$	26 / 3	2 / 1	3 / 0	66 / 3	30 / 1		815 / 68	660 / 58	614 / 61
$N_d = 3, N_v = 1$	5 / 3	7 / 6	5 / 3	29 / 3	13 / 6	10 / 3		4 / 0	9 / 1
$N_d = 3, N_v = 2$	3 / 2	8 / 1	5 / 2	11 / 2	8 / 1	11 / 2	150 / 7		40 / 1
$N_d = 3, N_v = 3$	1 / 1	3 / 0	16 / 1	13 / 1	12 / 0	12 / 1	190 / 6	5 / 1	

N_d — номер набора данных, N_v — номер набора параметров запуска автоэнкодера

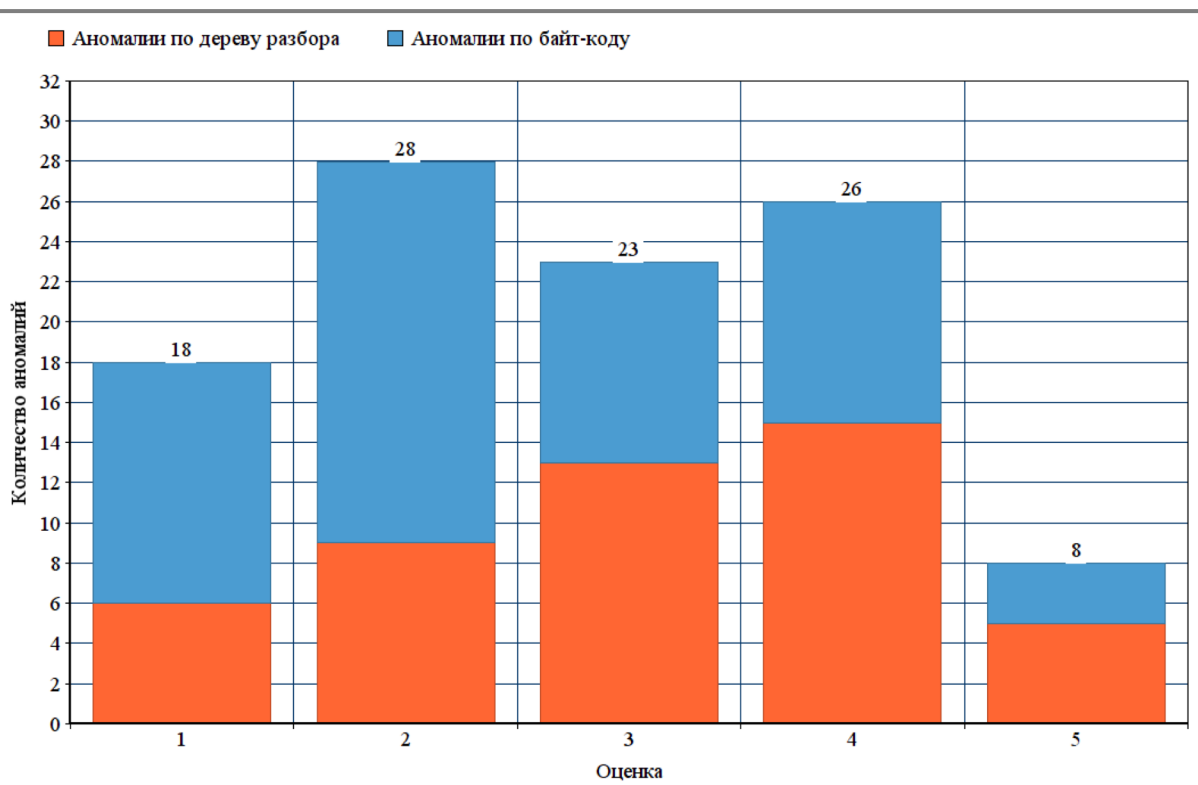
Классификация аномалий

- ✓ Из результатов всех экспериментов были отобраны все уникальные аномалии, получено:
 - **1587** аномалий по дереву разбора
 - **139** аномалий по байт-коду
- ✓ Было отобрано **513** наиболее показательных аномалий (удалены аномалии одного вида и дубликаты)
- ✓ Из полученных аномалий было выделено **30** классов^{*}

^{*} <http://victor.am> — список всех классов и аномалий

Сбор экспертных оценок

- ✓ По полученным аномалиям и классам были собраны экспертные оценки от разработчиков компилятора Kotlin
- ✓ Аномалии с оценками 4 и 5 были включены в тесты производительности компилятора



- ✓ 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

17/21

Сравнение с похожей работой

- ✓ В похожей работе использовалось:
 - Векторизация путем явного извлечения признаков;
 - local outlier factor, elliptic envelope и isolation forest.

W_1 — настоящая работа, W_2 — похожая работа, V — оценка

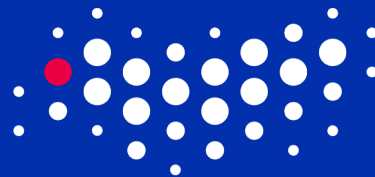
Название класса	W_1	W_2	V
Много веток в when-выражении	+	+	5
Много делегированных свойств	+	+	
Много типовых параметров	+	+	
Большая иерархия вызовов	+		4
Много if-выражений	+	+	
Большой набор констант	+		
Сложные аннотации функции		+	
Длинные цепочки вызовов	+		
Длинные перечисления (enum)	+		
Необычные кодовые конструкции		+	
Сложная иерархия <i>enum</i> -классов	+		
Много аннотаций с кв. скобками	+		
Много схожих вызовов	+	+	
Много функций на уровне файла	+		
Сложная структура кода	+	+	3
Большое тело функции	+	+	
Длинные строковые литералы	+		
Много конструкций try-catch		+	
Много присваиваний	+		
Много послед. арифм. выражений	+		
Много параметров у функции	+		

Название класса	W_1	W_2	V
Много операторов !!	+		2
Много схожих фрагментов		+	
Много inline-функций	+		
Много опер. конкатенации	+		
Много ссылок на класс		+	
Много лямбда-выражений		+	
Много throw-операторов		+	
Много reified тип. парам.	+		
Сложные лог. выражения	+		
Много циклов	+		
Большие литералы-коллекции	+		
Много локальных переменных		+	
Много вложенных функций		+	1
Большой companion-объект	+		
Длинный init-метод	+		
Много null-безопасных вызовов	+		
Много преобразований типов		+	
Много пустых стр. литералов		+	
Много строковых литералов	+	+	
Много строковых литералов		+	
Много вложенных структур	+		
Код из руководств по Kotlin		+	

Выводы

- ✓ Не смотря на невысокую оценку многих аномалий и классов, предложенный подход оказался вполне пригоден для поиска полезных кодовых аномалий

- ✓ Дальнейшая работа может быть направлена:
 - на автоматическую классификацию для отсеечения неинтересующих классов и фокусирования на интересующих;
 - на объединение двух подходов к векторизации и обнаружению аномалий (в настоящей и похожей работе).



ITMO UNIVERSITY

Спасибо за внимание!

Петухов Виктор

Студент группы М4238

Кафедра Компьютерных технологий

Санкт-Петербург, 2018