Поиск аномалий в коде на языке программирования Kotlin с использованием статического анализа кода

Петухов Виктор

Магистрант второго курса обучения

Кафедра Компьютерных технологий

Актуальность

- Аномалии в коде, которые могут повлечь проблемы в производительности, возникают не только по вине пользователя языка программирования
- Часть проблем может заключаться в неоптимальной кодогенерации или некорректных оптимизациях в компиляторе языка
- Разработчикам языка Kotlin необходимо исследование, позволяющее выявить такие проблемы
- Пользователям языка необходим инструмент, позволяющий обнаруживать исходный код с потенциальными проблемами производительности на заданном проекте

Существующие решения по оценке производительности программ

- Профайлеры — инструменты для анализа производительности во время исполнения программы
- Инструменты, использующие статические анализаторы, основанные на эвристиках (например, performance issues в IDE)
- Performance-тесты (например, JMH — Java Microbenchmark Harness)

Постановка задачи

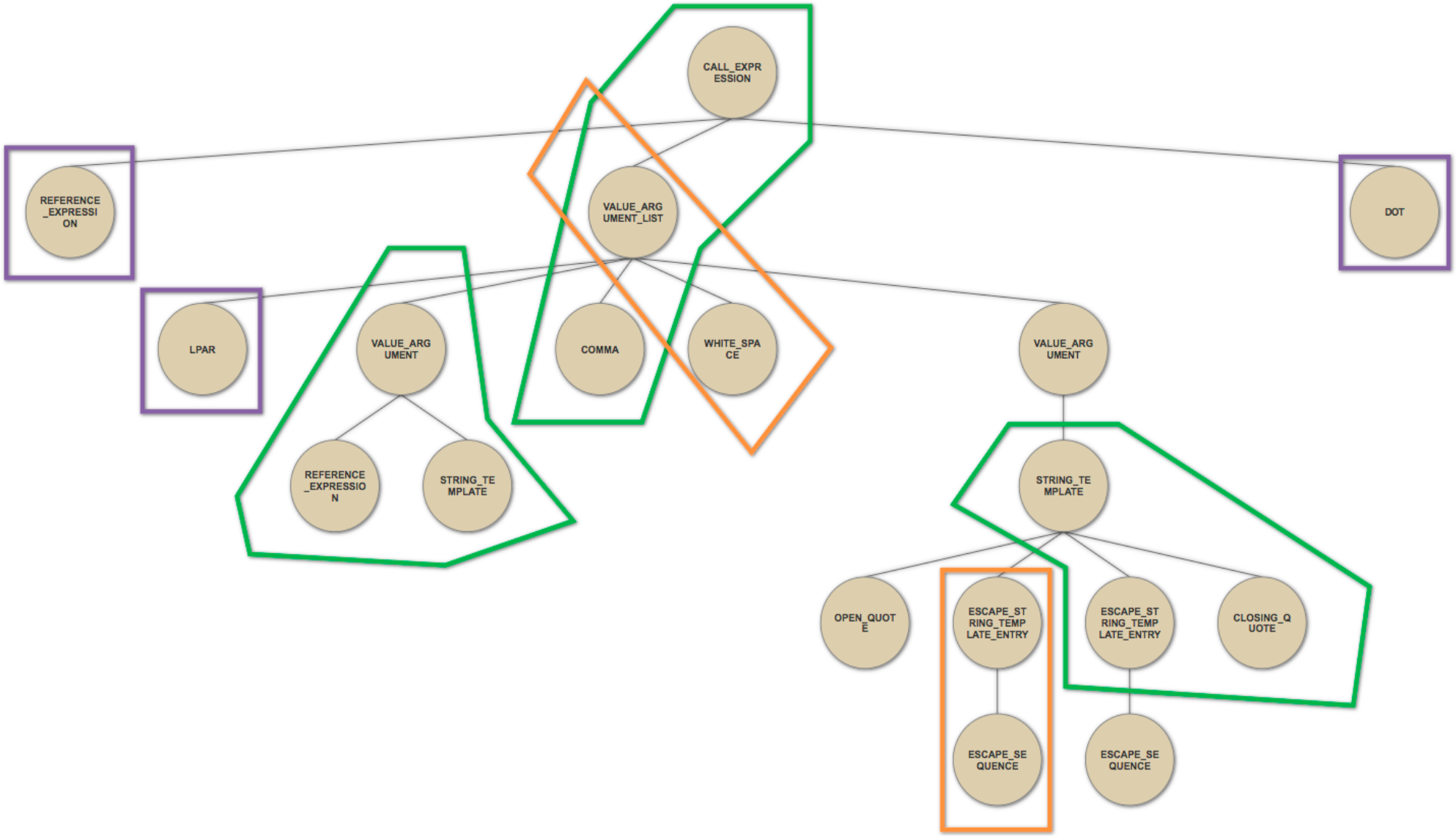
- Будем решать задачу обнаружения аномалий, уже поставленную в области машинного обучения
- Будем искать точечные аномалии
- Будем пробовать трактовать обнаруженные аномалии как проблемы производительности:
  - По причинам на стороне пользователя
  - По причинам на стороне компилятора (кодогенератора/оптимизатора)

Сбор данных

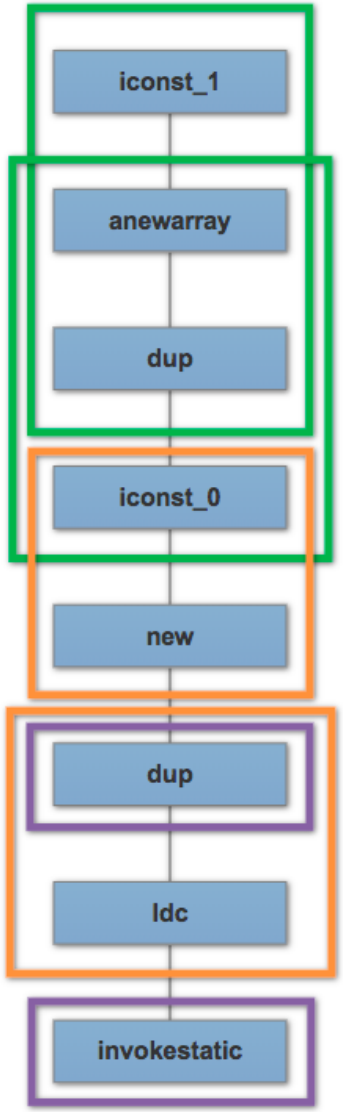
- В качестве набора данных для экспериментов были выбраны файлы с исходным кодом на Kotlin и JVM байт-кодом с ресурса Github
- Было собрано **930 тыс.** файлов с исходным кодом на Kotlin
- Было собрано **63 тыс.** файлов с JVM байт-кодом
- Файлы с JVM байт-кодом были сопоставлены файлам с исходным кодом

Подготовка данных. Задача факторизации дерева разбора и байт-кода

- Для алгоритмов машинного обучения требуются векторы — объекты с наборами признаков
- Встает задача факторизации — векторного представления дерева разбора и байт-кода
- Был выбран способ факторизации дерева путем разложения его на n-граммы (unigram, bigram и 3-gram)
- Для байт-кода (набора JVM-инструкций) был выбран аналогичный способ



Факторизация дерева разбора



Факторизация байт-кода

Результаты факторизации

- В результате по набору деревьев разбора на Kotlin было извлечено **11 тыс.** n-грамм
- По набору JVM байт-кода было извлечено **110 тыс.** n-грамм
- Таким образом, были составлены следующие наборы данных:
  - По деревьям разбора — **930 тыс.** примеров, **11 тыс.** признаков
  - По JVM байт-коду — **63 тыс.** примеров, **110 тыс.** признаков

Методы решения задачи обнаружения аномалий

- Метод опорных векторов с одним классом (OC SVM)
- Метод k-средних (k-means)
- Метод k-ближайших соседей (k-nearest neighbor)
- Статистические методы
- Репликаторные нейросети, автоэнкодеры

Автоэнкодеры. Обоснование выбора

- 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, нужно уметь сжимать информацию

Результаты

- По результатам работы автоэнкодера были вычислены евклидовы расстояния между входными и выходными векторами
- Расстояния трактовались как «степени аномальности»
- Аномалиями считались векторы, расстояния которых отклонялись на более чем 3-сигма (СКО) или 5-сигма
- Полученные аномалии были разделены на классы
- Был разработан инструмент, позволяющий запускать поиск аномалий на заданном проекте

Верификация результатов

- Для оценки качества результатов были привлечены эксперты — разработчики компилятора Kotlin
- Был разработан специальный веб-сервис, позволяющий давать оценку той или иной аномалии
- В результате планируется посчитать среднюю оценку по найденным аномалиям и тем самым оценить качество результатов
- Часть аномалий уже была включена в тесты на производительность компилятора Kotlin

Планы

- Запуск поиска аномалий на определенных срезах набора данных:
  - С заданными ограничениями на кол-во узлов / глубину в дереве разбора
  - С заданными ограничениями на объем JVM байт-кода
- Обучение классификатора на сгруппированном наборе аномалий для дальнейшей автоматической классификации и обнаружения новых классов
- Проведение аналогичного исследования в Kotlin/Native и Kotlin/JS

Спасибо за внимание!

Петухов Виктор

Кафедра Компьютерных технологий

i@victor.am, vapetukhov@yandex-team.ru