

Flow2Vec: Value-Flow-Based Precise Code Embedding

Applications of CFPQ analysis

Егор Орачев

JetBrains Research, Лаборатория языковых инструментов
Санкт-Петербургский Государственный университет

22 марта 2021

```
1 int speed(int input) {  
2     int x, y, k;  
3     k = input / 100;  
4     x = 2;  
5     y = k + 5;  
6  
7     while ( x < 10 ) {  
8         x++;  
9         y = y + 3;  
10    }  
11  
12    if ((3*k + 100) > 43) {  
13        y++;  
14        x = x / ( x - y );  
15    }  
16  
17    return x;  
}
```

Figure: Code fragment

Статический анализ кода

- Interprocedural data flow analysis
- Program slicing
- Pointer analysis
- Shape analysis
- Code classification
- Code summarization

Представление программы

- Проблема: эффективность анализа зависит от того, насколько "хорошим" является используемое представление программы
- Варианты:
 - ▶ Абстрактное синтаксическое дерево
 - ▶ Представление в промежуточном языке
 - ▶ Граф потока данных
 - ▶ Граф потока управления
 - ▶ Граф вызовов
 - ▶ Представление программы в виде embedding'a

- Code2vec: метод и его описание
- Построение embedding'а для ориентированного графа
- Запросы с контекстно-свободными (КС) ограничениями
- Flow2vec: метод и его описание
- Flow2vec: практическое применение
- Наши исследования в области вычисления КС запросов
- Дальнейшее направление исследований

Code2vec: Code Embedding

- Code2vec: построение code embeddings для представление фрагментов кода в виде числовых векторов фиксированной длины
- Идея:
 - 1 Представить фрагмент кода в виде абстрактного синтаксического дерева
 - 2 Сделать случайную выборку путей из дерева
 - 3 Получить векторное представление путей
 - 4 Суммировать полученные представления с весами и получить финальный вектор
 - 5 Использовать данный вектор для анализа программы
 - 6 Выполнять шаги 3 и 4 одновременно

Code2vec: Мотивационный пример

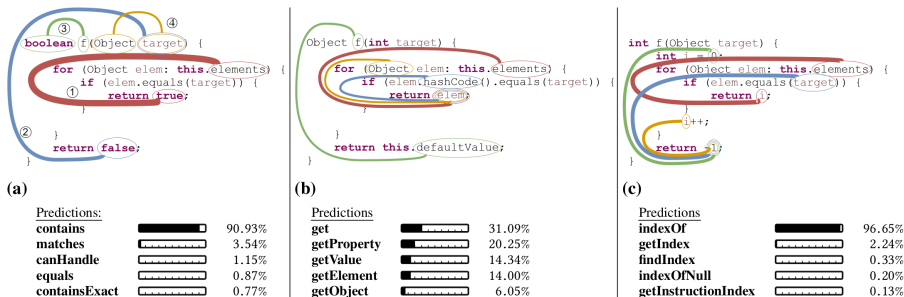


Figure: An example for three methods that albeit having have a similar syntactic structure can be easily distinguished by our model; our model successfully captures the subtle differences between them and manages to predict meaningful names. Each method portrays the top-4 paths that were given the most attention by the model. The widths of the colored paths are proportional to the attention that each path was given.

- Абстрактное синтаксическое дерево (АСТ) $\mathcal{C} = \langle N, T, X, s, \delta, \phi \rangle$,
 $\delta : N \rightarrow (N \cup T)^*$, $\phi : T \rightarrow X$
- Путь в дереве $p = n_1 d_1 \dots n_k d_k n_{k+1}$, $n_1, n_{k+1} \in T$, $n_2, \dots, n_k \in N$,
 $d_i \in \{\uparrow, \downarrow\}$
- Путь-контекст $\langle x_s, p, x_t \rangle$, $x_s, x_t \in X$, $x_s = \phi(\text{start}(p))$,
 $x_t = \phi(\text{end}(p))$, для $x = 7$ путь-контекст
 $\langle x, \text{NameExpr} \uparrow \text{AssignExpr} \downarrow \text{IntegerLiteralExpr}, 7 \rangle$
- Множество меток Y , P множество АСТ путей, извлекаемых из датасета

Code2vec: Модель (1)

- Что будем обучать:
 - ▶ embedding путей $path_vocab \in \mathbb{R}^{|P| \times d}$
 - ▶ embedding значений $value_vocab \in \mathbb{R}^{|X| \times d}$
 - ▶ полносвязный слой $W \in \mathbb{R}^{d \times 3d}$
 - ▶ вектор внимания $\mathbf{a} \in \mathbb{R}^d$
 - ▶ embedding меток $tags_vocab \in \mathbb{R}^{|Y| \times d}$
- Параметр $d \in \mathbb{N}$ - размерность embedding'а, подбирается эмпирически

Code2vec: Модель (2)

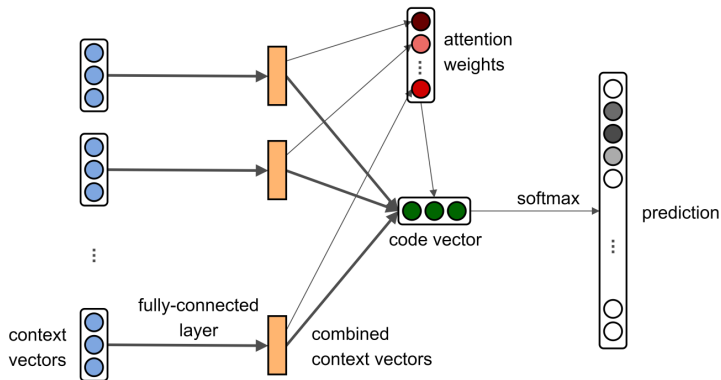


Figure: The architecture of our path-attention network. A full-connected layer learns to combine embeddings of each path-contexts with itself; attention weights are learned using the combined context vectors, and used to compute a code vector. The code vector is used to predicts the label.

Code2vec: Code as Bag of Path-Contexts

- Множество всех путей P
- Фрагмент кода $C = \langle N, T, X, s, \delta, \phi \rangle$
- $T_{pairs} = \{(t_i, t_j) \mid t_i, t_j \in termNodes(C), i \neq j\}$
- $Rep = \{\langle x_s, p, x_t \rangle \mid p \in P \text{ и } \exists (t_i, t_j) \in T_{pairs} : \text{start}(p) = x_s, \text{end}(p) = x_t, \text{ где } x_s = \phi(t_i), x_t = \phi(t_j)\}$
- Мешок путей-контекстов $\mathcal{B} = \{b_1, \dots, b_n\}, b_i \in Rep$

Code2vec: Path-Attention Model

- Контекстный вектор $c_i = \text{embedding}(\langle x_s, p_j, x_t \rangle) = [\text{value_vocab}_s, \text{path_vocab}_j, \text{value_vocab}_t] \in \mathbb{R}^{3d}$
- Комбинированный контекстный вектор $\tilde{c}_i = \tanh(W * c_i) \in \mathbb{R}^d$
- Веса внимания $\alpha_i = \frac{\exp(c_i^T \cdot \mathbf{a})}{\sum_{j=1}^n \exp(c_j^T \cdot \mathbf{a})}$
- Вектор фрагмента кода $\mathbf{v} = \sum_{i=1}^n \alpha_i \tilde{c}_i$
- Предсказание $y_i \in Y, q(y) = \frac{\exp(\mathbf{v}^T \cdot \text{tags_vocab}_i)}{\sum_{i=1}^{|Y|} \exp(\mathbf{v}^T \cdot \text{tags_vocab}_i)}$

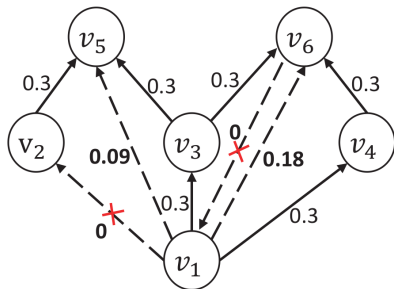
- Можем представить пути в АСТ дереве фрагмента кода в виде набора векторов
- Используя взвешенную сумму, можем представить весь фрагмент кода как вектор
- Полученный вектор можем использовать в дальнейшем для анализа

HOPE: Graph Embedding

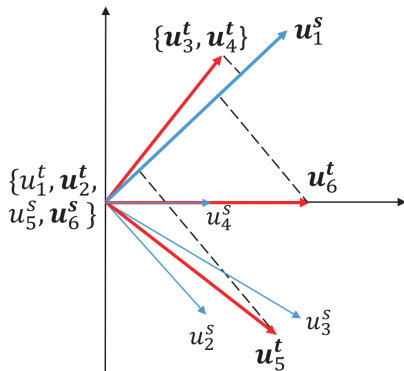
- Построение представления графа в векторном пространстве выбранной размерности
 - ▶ Вершины графа — это вектора
 - ▶ Можно использовать в задачах реконструкции графа, рекомендации соседей, предсказания связей и т.д.
- Проблемы:
 - ▶ Необходимо сохранить "важные" свойства графа
 - ▶ Реальные графы являются ориентированными и обладают ассиметричной транзитивностью
- Решение: Алгоритм High-Order Proximity preserved Embedding (HOPE)¹

¹Asymmetric Transitivity Preserving Graph Embedding, Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu, <https://doi.org/10.1145/2939672.2939751>

HOPE: Идея



(a) Directed graph



(b) Graph embedding

Figure: HOPE: Asymmetric Transitivity Preserving Graph Embedding

HOPE: Постановка задачи

- Ориентированный граф $G = \langle V, E \rangle$, $V = \{v_1, \dots, v_N\}$, $|V| = N$, $e_{ij} = (v_1, v_2) \in E$
- Матрица смежности графа $A \in \mathbb{R}^{N \times N}$, a_i - i -ая строка матрицы, A_{ij} - элемент матрицы в i -ой строке и j -ом столбце
- Матрица $S \in \mathbb{R}^{N \times N}$ - матрица близости графа
- Embedding матрицы $U = [U^s, U^t]$, $U^s, U^t \in \mathbb{R}^{N \times K}$, $K \in \mathbb{N}$ - размерность embedding'a, вектора u_i^s, u_i^t соответствуют вершине графа v_i
- Хотим приблизить матрицу S оптимально в следующем смысле $\min \|S - U^s * U^{tT}\|_F^2$, где $\|\bullet\|_F$ - норма Фробениуса

HOPE: High-order Proximity Matrix

- Построение матрицы близости S , которая сохранит "важные" свойства графа для дальнейшего анализа
- Варианты выбора S :
 - ▶ Индекс Катца (англ. Katz index):
 $S^{Katz} = \sum_{i=1}^{\infty} \beta^i A^i$, где $\beta \in (0, 1)$ - фактор ослабления
 - ▶ Rooted PageRank
 - ▶ Common Neighbors
 - ▶ Adamic-Adar
- Только Katz index и Rooted PageRank сохраняют глобальную ассиметричную транзитивность графа

HOPE: Approximation of High-Order Proximity

- $S = S^{Katz} = \sum_{i=1}^{\infty} \beta^i A^i$,
 $S^{Katz} = \beta A * S^{Katz} + \beta * A$, $S^{Katz} = (I - \beta A)^{-1} * \beta A$
- $S = \sum_{i=1}^N \sigma_i v_i^s v_i^t{}^T$ используя SVD разложение, где $\{\sigma_1, \dots, \sigma_N\}$ сингулярные значения в порядке убывания
- $U^s = [\sqrt{\sigma_1} * v_1^s, \dots, \sqrt{\sigma_K} * v_K^s]$
- $U^t = [\sqrt{\sigma_1} * v_1^t, \dots, \sqrt{\sigma_K} * v_K^t]$
- Ошибка аппроксимации: $\|S - U^s * U^t{}^T\|_F^2 = \sum_{i=K+1}^N \sigma_i^2$
- Относительная ошибка аппроксимации:
$$\frac{\|S - U^s * U^t{}^T\|_F^2}{\|S\|_F^2} = \frac{\sum_{i=K+1}^N \sigma_i^2}{\sum_{i=1}^N \sigma_i^2}$$

- Для ориентированного графа G можем построить embedding, который сохраняет ассиметричную транзитивность графа
- Для построения требуется матрица близости высокого порядка S , которая сохраняет глобальную ассиметричную транзитивность графа
- Полученный embedding $U = [U^s, U^t]$ обладает имеет доказанные оценки ошибки приближения. Значение ошибки можно уменьшить, увеличив значение параметра K

Context-free Path Querying

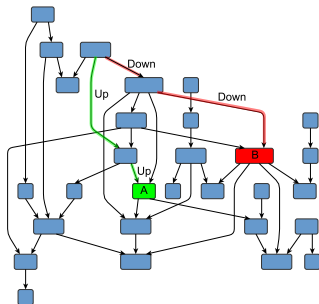


Figure: Example of a graph

Навигация в графе:

- Находятся ли вершины A и B на одном уровне иерархии?
- Существует ли путь вида $Up^n Down^n$?
- Найти все такие пути $Up^n Down^n$, которые начинаются в вершине A

Context-free Path Querying: Применение

- Графовые базы данных²
- Анализ RDF данных³
- Биоинформатика⁴
- Статический анализ кода⁵

²Querying Graph Databases, Pablo Barceló Baeza,
<https://doi.org/10.1145/2463664.2465216>

³Context-Free Path Queries on RDF Graphs, Xiaowang Zhang, Zhiyong Feng, Xin Wang et al., <https://arxiv.org/abs/1506.00743>

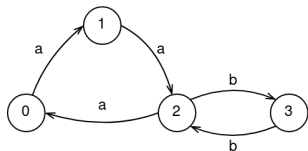
⁴Quantifying variances in comparative RNA secondary structure prediction, James Anderson, Adám Novák, Zsuzsanna Sükösd et al.,
<https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-149>

⁵Fast Algorithms for Dyck-CFL-Reachability with Applications to Alias Analysis, Qirun Zhang, Michael R. Lyu, Hao Yuan, Zhendong Su,
<https://doi.org/10.1145/2499370.2462159>

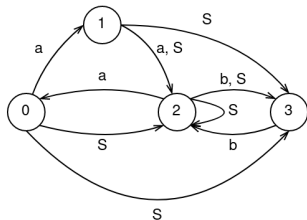
Context-free Path Querying: Терминология

- Ориентированный граф с метками $\mathcal{G} = \langle V, E, L \rangle$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \dots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \dots l_{n-1}$
- КС Грамматика $G = \langle \Sigma, N, P, S \rangle$
- Язык $L(G) = \{w \mid S \xrightarrow{*}_G w\}$
- Семантика достижимости: $R = \{(u, v) \mid \exists u\pi v : \omega(\pi) \in L\}$
- Семантика всех путей: $\Pi = \{u\pi v \mid \omega(\pi) \in L\}$

Context-free Path Querying: Пример



(a) Input graph



(b) Result graph

Figure: CFPQ Example

- Грамматика $S \rightarrow aSb \mid ab$
- Результат (Семантика достижимости): ребра с меткой S
- Результат (Семантика всех путей): $\{1 \xrightarrow{a} 3, 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 2, 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2, \dots\}$

- Алгоритмы, основанные на различных техниках парсинга (CYK, LL, LR, etc.)
- **Алгоритмы, основанные на линейной алгебре**
 - ▶ Матричный алгоритм Рустама Азимова
 - семантика: достижимости, одного пути, всех путей
 - платформа: CPU, GPU
 - ▶ Алгоритм на основе пересечения графа и рекурсивного автомата через произведения Кронекера
 - семантика: достижимости, одного пути, всех путей
 - платформа: CPU, GPU

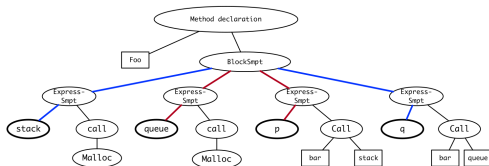
Context-free Path Querying: Итоги

- Для анализа графа можем задавать запросы с КС ограничениями
- Для каждой конкретной задачи можем выбирать различную семантику запроса
- Имеем эффективные алгоритмы для вычисления запросов на двух основных вычислительных платформах

Code Embedding: Проблемы

```
foo(){  
    stack = malloc(..);  
    queue = malloc(..);  
    p = bar(stack); //cs1  
    q = bar(queue); //cs2  
}
```

(a) Foo function



(b) Ast for foo

Figure: Spurious paths example

Недостатки существующих инструментов:

- Не учитывают межпроцедурное взаимодействие
- Не учитывают псевдонимы (ссылки)
- Не учитывают асимметричную транзитивность программ

Flow2Vec: Value-Flow-Based Precise Code Embedding

- Flow2vec: построение code embeddings для представления фрагмента кода как части всей программы с учетом межпроцедурных потоков данных и множества вызовов функций
- Идея:
 - 1 Построить промежуточное представление программы в терминах простых операций
 - 2 Построить граф потоков данных
 - 3 Используя КС запросы вычислить количество потоков данных между каждым выражением в программе (в ее промежуточном представлении)
 - 4 Построить матрицу близости высокого порядка
 - 5 Вычислить на ее основе embedding для графа программы

Flow2Vec: Алгоритм

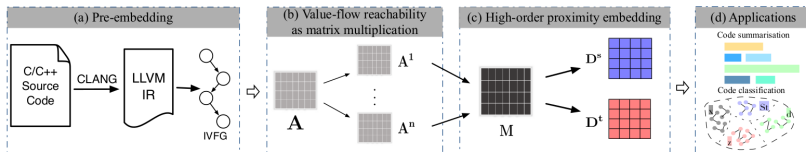


Figure: Flow2vec idea

- Шаг 1: Построение LLVM-IR, IVFG, исходных матриц с call/return и value-flow информацией
- Шаг 2: Value-flow reachability через CFPQ
- Шаг 3: Вычисление матрицы близости высокого порядка и построение embedding'a
- Шаг 4: Применение построенного embedding'a в пользовательских приложениях

Шаг 1: Pre-embedding (1)

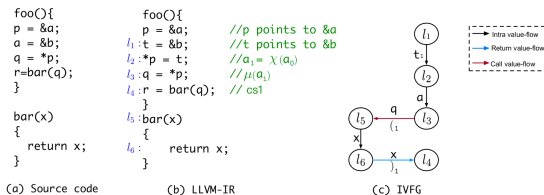
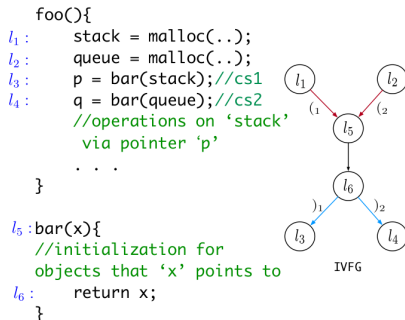


Figure: C Code fragment, its LLVM-IR and IVFG

- LLVM-IR в качестве промежуточного представления
- Построение IVFG на основе LLVM-IR программы
- $\mathcal{V} = \mathcal{O} \cup \mathcal{P}$, два типа переменных
- $t \xrightarrow{v} t', v \in \mathcal{V}$ def-use отношение
- $t \xrightarrow{p} t', p \in \mathcal{P}$ direct value-flow отношение

War 1: Pre-embedding (2)



(a) Code fragment and IVFG

| | l_1 | l_2 | l_3 | l_4 | l_5 | l_6 |
|-------|-------|-------|-------|-------|-------|-------|
| l_1 | 0 | 0 | 0 | 0 | $(1$ | 0 |
| l_2 | 0 | 0 | 0 | 0 | $(2$ | 0 |
| l_3 | 0 | 0 | 0 | 0 | 0 | 0 |
| l_4 | 0 | 0 | 0 | 0 | 0 | 0 |
| l_5 | 0 | 0 | 0 | 0 | 0 | 1 |
| l_6 | 0 | 0 | $)_1$ | $)_2$ | 0 | 0 |

(b) Call/return and value-flow matrix

Figure: Pre-embedding example

Шар 2: Value-flow reachability via matrix multiplication

$$\begin{bmatrix} 0 & 0 & 0 & 0 & (1 & 0) \\ 0 & 0 & 0 & 0 & (2 & 0) \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 &)_1 &)_2 & 0 & 0 \end{bmatrix}$$

A^1



$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

(a) Matrix A^1

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 * (1) \\ 0 & 0 & 0 & 0 & 0 & 1 * (2) \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 *)_1 & 1 *)_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A^2



$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) Matrix A^2

$$\begin{bmatrix} 0 & 0 & (1*1*)_1 & -(1*1*)_2 & 0 & 0 \\ 0 & 0 & -(2*1*)_1 & (2*1*)_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A^3



$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) Matrix A^3

Figure: Context-sensitive value-flow reachability

Шаг 3: High-order proximity embedding

$$\mathbf{M} = \sum_{h=1}^3 (\beta \cdot \mathbf{A})^h$$

$$\begin{bmatrix} 0 & 0 & 0.512 & 0 & 0.8 & 0.64 \\ 0 & 0 & 0 & 0.512 & 0.8 & 0.64 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.64 & 0.64 & 0 & 0.8 \\ 0 & 0 & 0.8 & 0.8 & 0 & 0 \end{bmatrix}$$

$$\mathbf{M}(\beta = 0.8)$$

(a) High-order proximity matrix

$$\mathbf{M} \approx \mathbf{D}^s \cdot \mathbf{D}^{t\top}$$

$$\begin{bmatrix} -0.51 & -0.49 & -0.69 & 0 & 0 & 0 \\ 0.51 & -0.49 & -0.69 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.33 & -0.79 & 0 & 0 & 0 \\ 0 & 0.71 & -0.58 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.51 & 0.49 & -0.69 & 0 & 0 & 0 \\ 0.51 & 0.49 & -0.69 & 0 & 0 & 0 \\ 0 & -0.71 & -0.58 & 0 & 0 & 0 \\ 0 & -0.33 & -0.79 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}^s$$

$$\mathbf{D}^t$$

(b) Embedding vectors (K -factor is 3)

Figure: Embedding step

Шар 4: Application scenarios

| Source code (A) | | Source code (B) | |
|---|--|---|--|
| <pre>int _____(dict *d){ int minimal; minimal = d->ht[0].used; if (minimal<INITIAL_SIZE) minimal=INITIAL_SIZE; return dictExpand(d, minimal); } int dictExpand(dict *d, unsigned long size){ dictht n; unsigned long realsize = _dictNextPower(size); n.table = zcalloc(realsize*sizeof(dictEntry*)); return DICT_OK; }</pre> | | <pre>SIG _____(char* inp){ if (!is_valid(inp)) return FAULT; CMD *cmd = parse(inp); return process(cmd); } SIG process(CMD *cmd){ Executor *e = get_glb_executor(); if (e->is_full) return FULL; e->execute(cmd); return SUCCESS; }</pre> | |
| Ground truth | resize to the minimal | Ground truth | parse and execute command |
| Code2Vec | <code>isMinimal</code> | Code2Vec | <code>check</code> |
| Code2Seq | <code>Expand</code> size | Code2Seq | <code>parse</code> |
| Flow2Vec | <code>reset</code> <code>minimal</code> memory | Flow2Vec | <code>parse</code> <code>command</code> and <code>execute</code> |

Figure: Code summarization example

- Code classification
- Code summarization

Что мы можем предложить

- Анализ Java программ
- Инструменты для построения графов
 - ▶ **WALA**⁶
 - ▶ **Soot**⁷
- Построение value-flow reachability матриц
- Построение emdebbling'a

⁶ссылка: <https://github.com/wala/WALA>

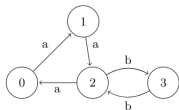
⁷ссылка: <https://github.com/soot-oss/soot>

- Матричный Алгоритм Рустама Азимова⁸
 - ▶ Семантика достижимости
 - ▶ Семантика одного пути
 - ▶ Семантика всех путей
 - ▶ *Можем модифицировать полукольцо, чтобы считать кол-во уникальных путей*
- Алгоритм на основе произведения Кронекера и PA⁹
 - ▶ Семантика достижимости и всех путей
 - ▶ Итеративное извлечение путей
 - ▶ *Но! На выходе нечто большее, чем просто граф*

⁸ссылка: <https://dl.acm.org/doi/10.1145/3398682.3399163>

⁹ссылка: https://link.springer.com/chapter/10.1007/978-3-030-54832-2_6

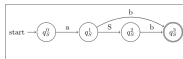
Kronecker CFPQ



(a) Input directed graph \mathcal{G}

$$\begin{pmatrix} \cdot & \{a\} & \cdot & \cdot \\ \cdot & \cdot & \{a\} & \cdot \\ \{a\} & \cdot & \cdot & \{b\} \\ \cdot & \cdot & \{b\} & \cdot \end{pmatrix}$$

(b) \mathcal{G} adjacency matrix



(c) RSA for $S \rightarrow ab \mid aSb$

$$\begin{pmatrix} \cdot & \{a\} & \cdot & \cdot \\ \cdot & \cdot & \{S\} & \{b\} \\ \cdot & \cdot & \cdot & \{b\} \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

(d) RSA adjacency matrix

$$\left(\begin{array}{ccc|ccc|ccc} \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & 1 \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 & 1 \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} \right)$$

(e) Result index

Figure: Kronecker CFPQ brief example

- Flow2Vec: связь анализа кода, CFPQ, и методов машинного обучения
- CFPQ для всех путей
- Вычисления на GPU
- Что с этим делать?

- Почта: egororachyov@gmail.com
- Материалы презентации:
 - ▶ Flow2Vec: Value-Flow-Based Precise Code Embedding, Yulei Sui, Xiao Cheng, Guanqin Zhang, Haoyu Wang, ссылка: <https://dl.acm.org/doi/10.1145/3428301>
 - ▶ Context-Free Path Querying with Single-Path Semantics by Matrix Multiplication, Arseniy Terekhov, Artyom Khoroshev, Rustam Azimov, Semyon Grigorev, ссылка: <https://dl.acm.org/doi/10.1145/3398682.3399163>
 - ▶ Context-Free Path Querying by Kronecker Product, Egor Orachev, Ilya Epelbaum, Rustam Azimov, Semyon Grigorev, ссылка: https://link.springer.com/chapter/10.1007/978-3-030-54832-2_6