Семинар

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

Предметная область

```
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'а

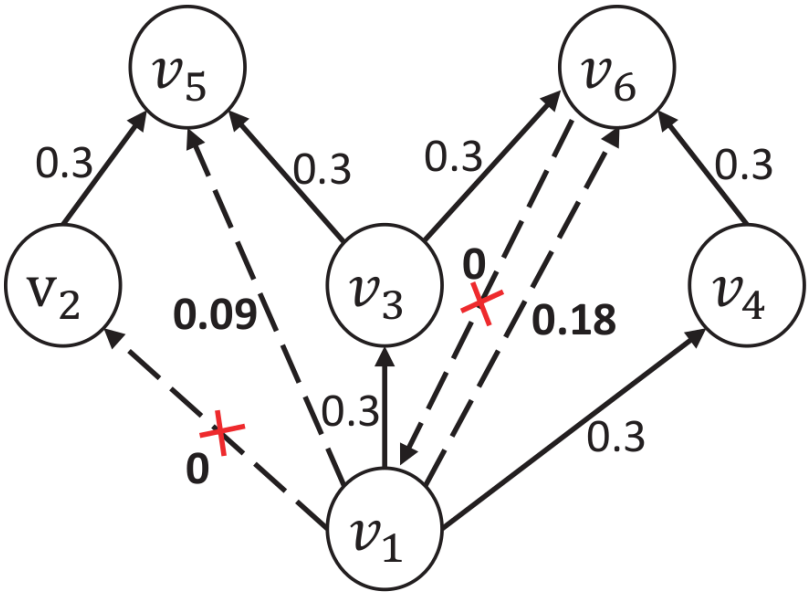
Структура презентации

- 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

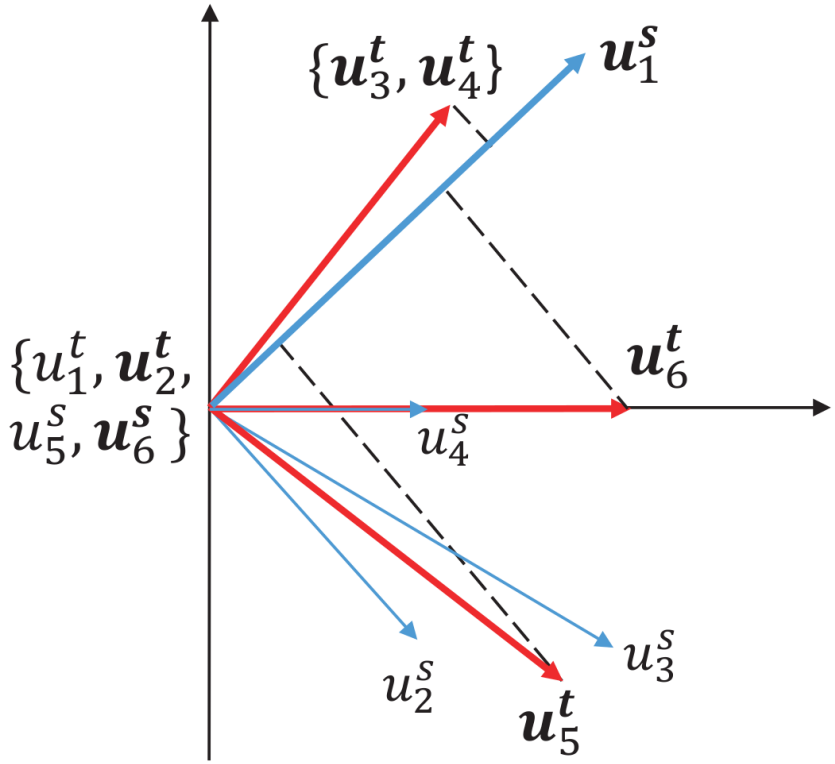
Graph Embedding

- Построение представления графа в векторном пространстве выбранной размерности
  - Вершины графа — это вектора
  - Можно использовать в задачах реконтсрукции графа, рекомендации соседей, предстказания связей и т.д.
- Проблемы:
  - Необходимо сохранить "важные" свойства графа
  - Реальные графы является ориентированными и обладают ассиметричной транзитивностью

Graph Embedding: Идея



(a) Directed graph



(b) Graph embedding

Figure: HOPE: Asymmetric Transitivity Preserving Graph Embedding

Graph Embedding: Постановка задачи

- Ориентированный граф $G = \langle V, E \rangle$, $V = \{v_1, ..., v_N\}$, $|V| = N$, $e_{ij} = (v_1, v_2) \in E$
- 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-том 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 emdedding'a, вектора u_i^s, u_i^t соответсвуют вершине графа v_i
- **Хотим приблизить матрицу S оптимально в следующем смысле $min \| S - U^s * {U^t}^T \|_F^2$, где $\| \bullet \|_F$ - норма Фробениуса**

Graph Embedding: High-order Proximity Matrix

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

Graph Embedding: Approximation of High-Order Proximity

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

Graph Embedding: Итоги

- 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 оценками ошибки приближения. Значение ошибки можно уменьшить, увеличив значение параметра K

Context-free Path Querying

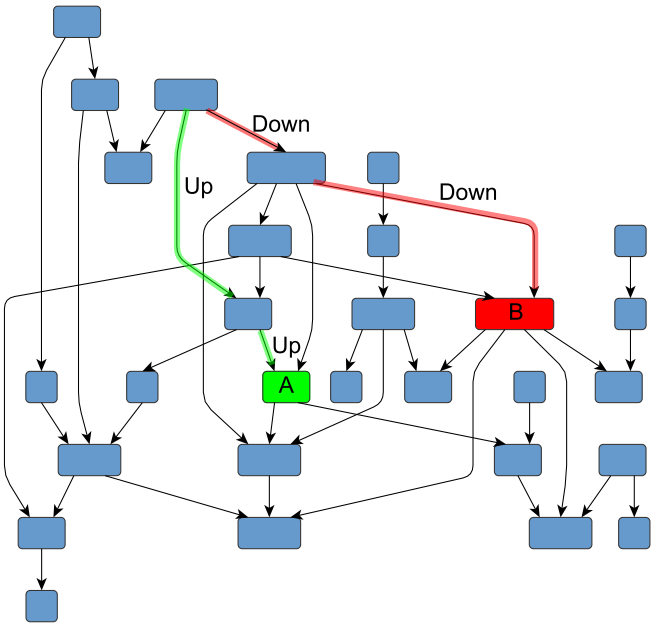


Figure: Example of a graph

Навигация в графе:

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

Context-free Path Querying: Применение

- Графовые базы данных[1]
- Анализ RDF данных[2]
- Биоинформатика[3]
- Статический анализ кода[4]

---

[1] Querying Graph Databases, Pablo Barceló Baeza, https://doi.org/10.1145/2463664.2465216

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

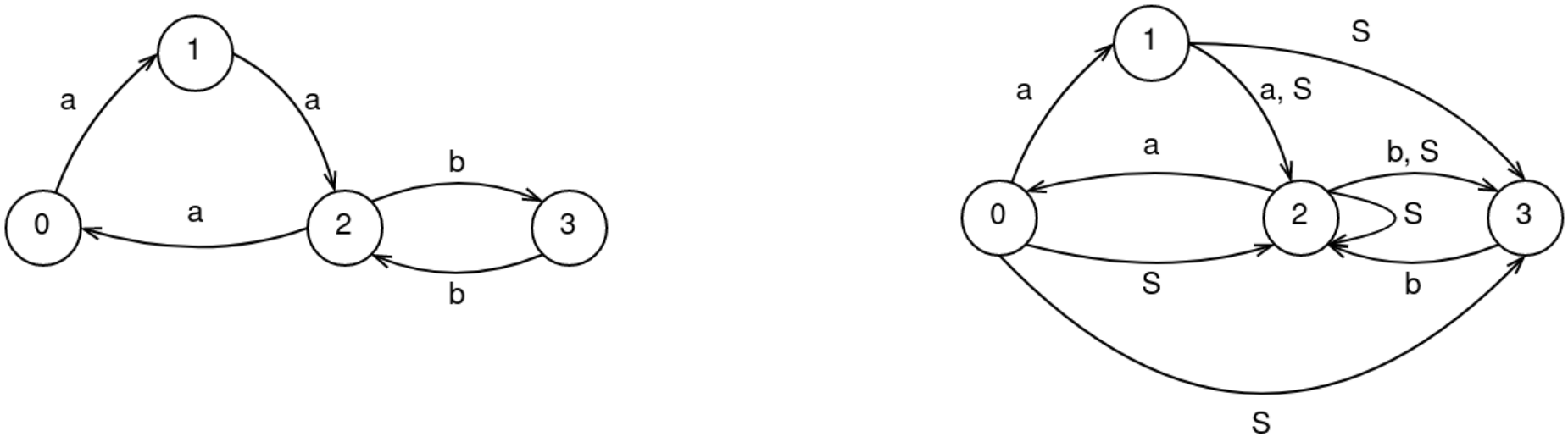
[3] Quantifying variances in comparative RNA secondary structure prediction, James Anderson, Adám Novák, Zsuzsanna Sükösd et al., https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-149

[4] Fast 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 меткаи $\mathcal{G} = \langle V, E, L \rangle$
- $\omega(\pi) = \omega(v_0 \xrightarrow{l_0} v_1 \xrightarrow{l_1} \cdots \xrightarrow{l_{n-2}} v_{n-1} \xrightarrow{l_{n-1}} v_n) = l_0 l_1 \cdots l_{n-1}$
- КС Грамматика $G = \langle \Sigma, N, P, S \rangle$
- Язык $L(G) = \{w \mid S \to^*_G w\}$
- 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 $S \to aSb \mid ab$
- Резульат (Семантика достижимости): ребра с меткой S
- Резульат (Семантика всех путей): $\{1 \xrightarrow{a} 3, 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 2, 1 \xrightarrow{a} 2 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{b} 3 \xrightarrow{b} 2 \xrightarrow{b} 3 \xrightarrow{b} 2, \ldots\}$

Context-free Path Querying: Существующие алгоритмы

- 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 ээфективные алгоритмы для вычисления запросов на двух основных вычислительных платформах

Code Embedding: Предыстория

- Развитие техник embedding'а в области обработки и анализа естественных языков (word2vec, и т.д.)
- Развитие техник code embedding'а для построения представления программ (code2vec, и т.д.)

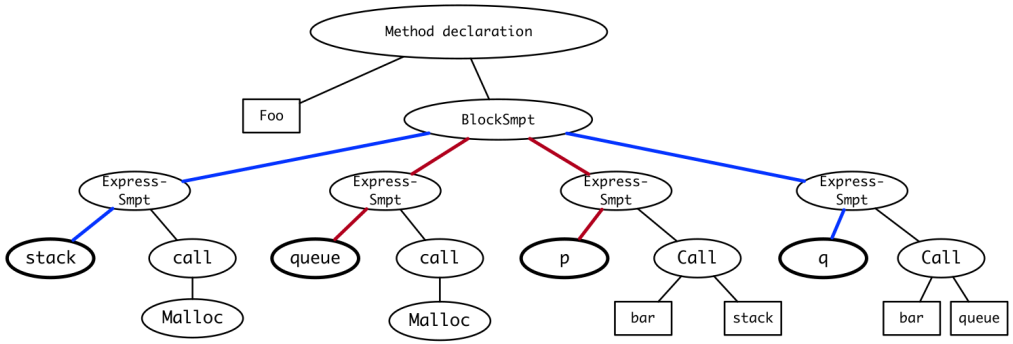
Code Embedding: Мотивация

- Развитие техник embedding'а в обрабокте естественных языков
- Code embedding'и в качестве представления программ
- Методы машинного обучения для анализа программ по новому представлению

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

- Существующие интсрументы
  - Code2vec
  - Code2seq
  - Word2vec-like ...
- Недостатки
  - Не учитывают меж-процедурное взаимодействие
  - Не учитывают псевдонимы (ссылки)
  - Не учитывают ассиметричную транзитивность программ

Flow2Vec

- Новый алгоритм, предложенный в статье *Flow2Vec: Value-Flow-Based Precise Code Embedding*[5]
- Учитывает ранее указанные недостатки
- Использует Intermediate Representation (IR), Interprocedural Value-Flow Graph (IVFG) и CFPQ для построяния value-flow reachability матриц

[5] Ссылка: https://dl.acm.org/doi/10.1145/3428301

Flow2Vec: Алгоритм

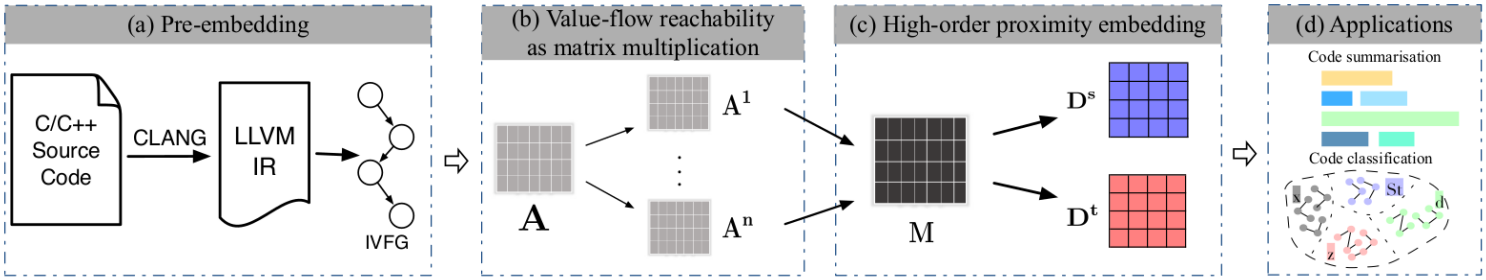


Figure: Flow2vec idea

- Шаг 1: Построение IR, IVFG, исходных матриц с call/return и value-flow информацией
- Шаг 2: Value-flow reachability через CFPQ
- Шаг 3: Построение emdedding'а высокого порядка
- Шаг 4: Применение построенного emdedding'а в пользовательских приложениях

Шаг 1: Pre-embedding (1)

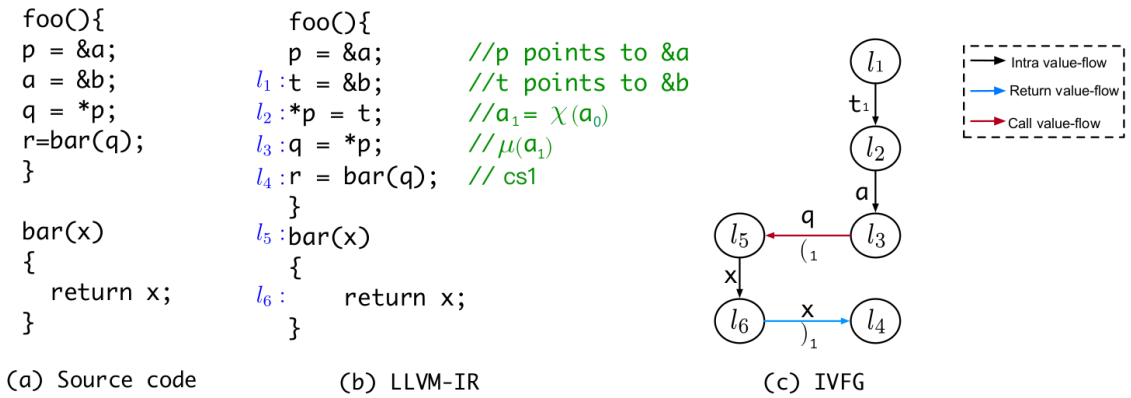


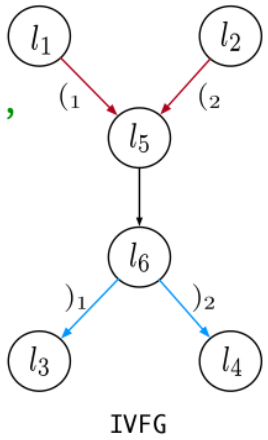
Figure: C Code fragment, its LLVM-IR and IVFG

- LLVM-IR в качестве промежуочного 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

Шаг 1: Pre-embedding (2)

```
   foo(){
l1 :    stack = malloc(..);
l2 :    queue = malloc(..);
l3 :    p = bar(stack);//cs1
l4 :    q = bar(queue);//cs2
        //operations on 'stack'
         via pointer 'p'
        . . .
   }

l5 :bar(x){
   //initialization for
   objects that 'x' points to
l6 :    return x;
   }
```



IVFG

(a) Code fragment and IVFG

$$\begin{array}{c} \\ l_1 \\ l_2 \\ l_3 \\ l_4 \\ l_5 \\ l_6 \end{array} \begin{array}{c} \begin{array}{cccccc} l_1 & l_2 & l_3 & l_4 & l_5 & l_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & (_1 & 0 \\ 0 & 0 & 0 & 0 & (_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 &)_1 &)_2 & 0 & 0 \end{bmatrix} \end{array}$$

(b) Call/return and value-flow matrix

Figure: Pre-embedding example

Шаг 2: Value-flow reachability via matrix multiplication

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

$\mathbf{A^1}$ $\mathbf{A^2}$ $\mathbf{A^3}$

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

(a) Matrix A^1 (b) Matrix A^2 (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

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

$$\left[\begin{array}{ccc:ccc} -0.51 & -0.49 & -0.69 & 0 & 0 & 0 \\ 0.51 & -0.49 & -0.69 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.33 & -0.79 & 0 & 0 & 0 \\ 0 & 0.71 & -0.58 & 0 & 0 & 0 \end{array}\right] \quad \left[\begin{array}{ccc:ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -0.51 & 0.49 & -0.69 & 0 & 0 & 0 \\ 0.51 & 0.49 & -0.69 & 0 & 0 & 0 \\ 0 & -0.71 & -0.58 & 0 & 0 & 0 \\ 0 & -0.33 & -0.79 & 0 & 0 & 0 \end{array}\right]$$

$\mathbf{D^s}$ $\mathbf{D^t}$

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

Figure: Embedding step

Шаг 4: Application scenarios

Souce code (A)		Souce code (B)	

`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;`
`}`
		`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 SUCESS;`
		`}`	

Ground truth	resize to the minimal	Ground truth	parse and execute command
Code2Vec	isMinimal	Code2Vec	check
Code2Seq	Expand size	Code2Seq	parse
Flow2Vec	reset minimal memory	Flow2Vec	parse command and execute

Figure: Code summarization example

- Code classification
- Code summarization

Что мы можем предложить

- Анализ Java программ
- Интсрументы для построения графов
  - **WALA**[6]
  - **Soot**[7]
- Построение value-flow reachability матриц
- Построение emdebbing'a

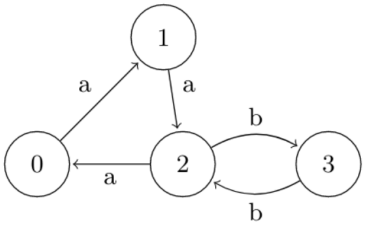
[6] ссылка: https://github.com/wala/WALA
[7] ссылка: https://github.com/soot-oss/soot

CFPQ

- Матричный Алгоритм Рустама Азимова[8]
  - Семантика достижимости
  - Семантика одного пути
  - Семантика всех путей
  - *Можем модифицировать полукольцо, чтобы считать кол-во уникальных путей*
- Алгоритм на основе произведения Кронекера и РА[9]
  - Семантика достижимости и всех путей
  - Итеративное извлечение путей
  - *Но! На выходе нечто большее, чем просто граф*

[8] ссылка: https://dl.acm.org/doi/10.1145/3398682.3399163
[9] ссылка: https://link.springer.com/chapter/10.1007/978-3-030-54832-2_6

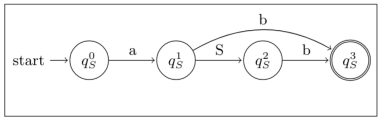
Kronecker CFPQ



(a) Input directed graph $\mathcal{G}$

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

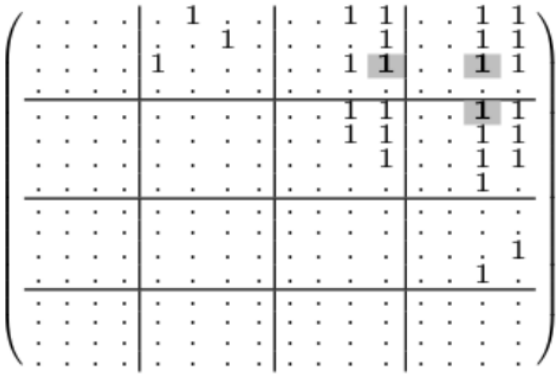
(b) $\mathcal{G}$ adjacency matrix



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

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

(d) RSA adjacency matrix



(e) Result index

Figure: Kronecker CFPQ brief example

Заключение

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

Дополнительно

- 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