Семинар

Flow2Vec: Value-Flow-Based Precise Code Embedding

Applications of CFPQ analysis

Егор Орачев

JetBrains Research, Лаборатория языковых инструментов
Санкт-Петербургский Государственный университет

5 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

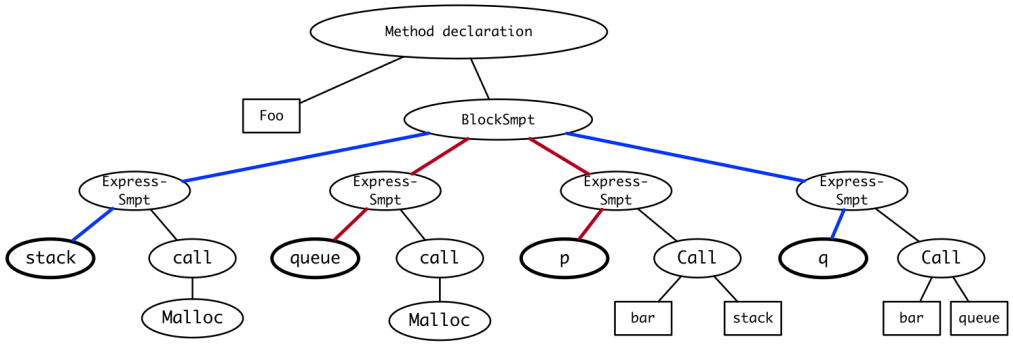
Code Embedding: Мотивация

- 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*[1]
- Учитывает ранее указанные недостатки
- Использует Intermediate Representation (IR), Interprocedural Value-Flow Graph (IVFG) и CFPQ для построяния value-flow reachability матриц

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

Flow2Vec: Алгоритм

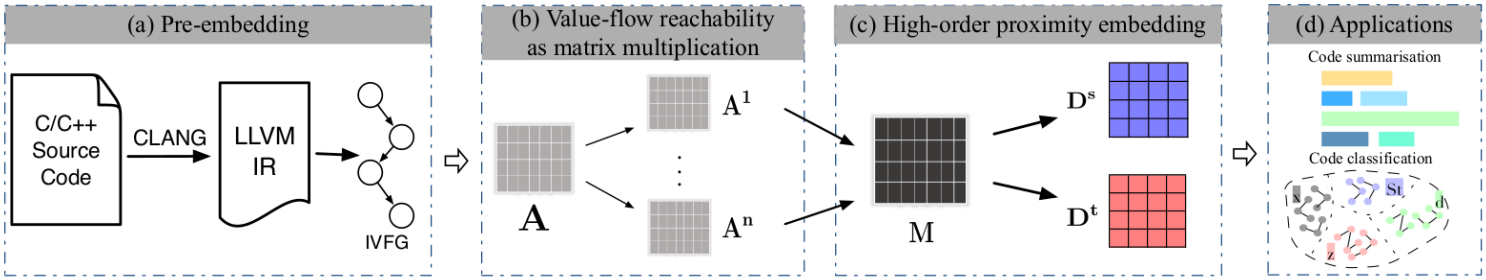


Figure: Flow2vec idea

- 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)

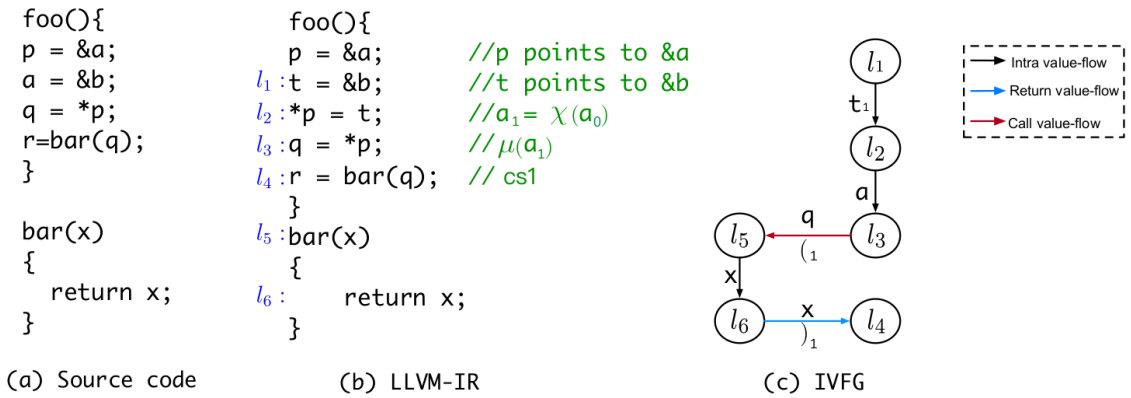


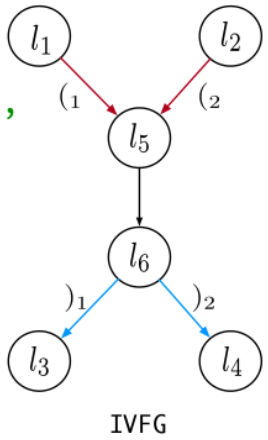
Figure: C Code fragment, its LLVM-IR and IVFG

- 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

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

$\mathbf{A^1}$

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

(a) Matrix A^1

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

$\mathbf{A^2}$

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

(b) Matrix A^2

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

$\mathbf{A^3}$

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

(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

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

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

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

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

Figure: Embedding step

Шаг 4: Application scenarios

Souce code (A)

```
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;
}
```

Souce code (B)

```
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
Code2Vec	isMinimal
Code2Seq	Expand size
Flow2Vec	reset minimal memory

Ground truth	parse and execute command
Code2Vec	check
Code2Seq	parse
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**[2]
  - **Soot**[3]
- Построение value-flow reachability матриц
- Построение emdebbing'a

[2] ссылка: https://github.com/wala/WALA
[3] ссылка: https://github.com/soot-oss/soot

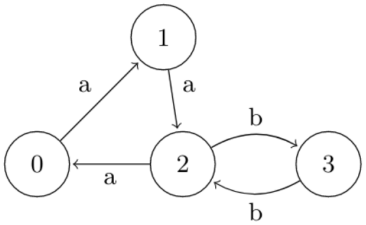
CFPQ

- Матричный Алгоритм Рустама Азимова[4]
  - Семантика достижимости
  - Семантика одного пути
  - Семантика всех путей
  - *Можем модифицировать полукольцо, чтобы считать кол-во уникальных путей*
- Алгоритм на основе произведения Кронекера и РА[5]
  - Семантика достижимости и всех путей
  - Итеративное извлечение путей
  - *Но! На выходе нечто большее, чем просто граф*

[4] ссылка: https://dl.acm.org/doi/10.1145/3398682.3399163

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

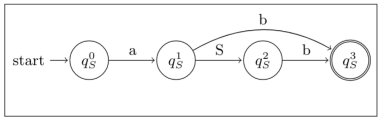
Kronecker CFPQ



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

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

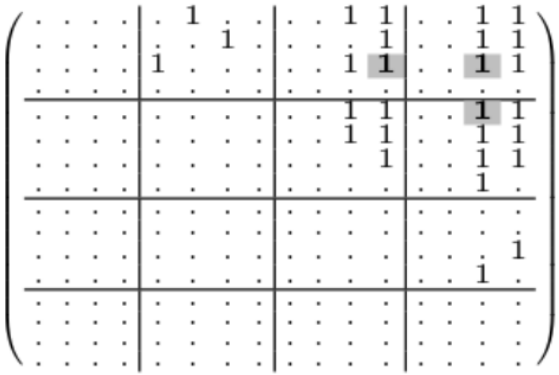
(b) $\mathcal{G}$ adjacency matrix



(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