Машинное обучение для анализа исходного кода

16 декабря 2020 г.

ИСП РАН

Зачем анализировать исходный код

ИСП РАН

- Совместно с текстами на естественном языке
  - генерация сообщений к коммитам, комментариев (docstrings), документации
  - поиск кода по запросу на естественном языке
- Поиск похожего кода, автодополнение
- Поиск и автоматическое исправление багов/ошибок сборки
- Автоматизация code review и багтрекеров (например, назначение ревьюера)
- Оптимизация запуска тестов в CI (приоритизация сценариев)
- ...

code2seq: Generating Sequences from Structured Representations of Code (2018)

Идея

ИСП РАН

Генерация описания программного кода на естественном языке

Существующее решение seq2seq — код представляется в виде последовательности токенов

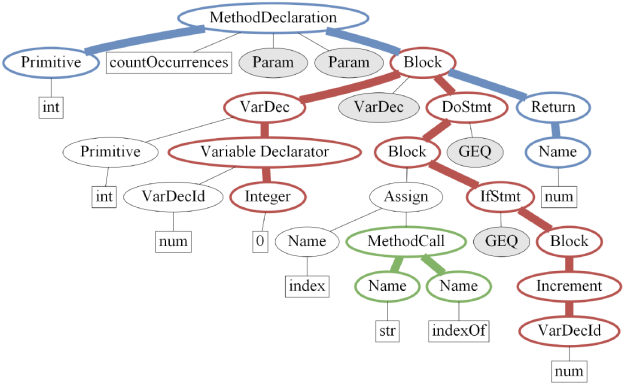
Минус: теряется структурная информация (синтаксис, семантика)

Предлагаемое решение: работать с *абстрактным синтаксическим деревом*

Полезные свойства АСД

ИСП РАН

- Конечное число вершин
- Единственное представление кода
- Позволяет работать с любым языком программирования



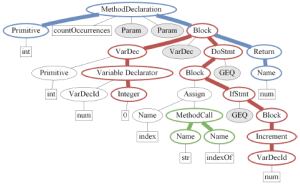
Векторное представление кода

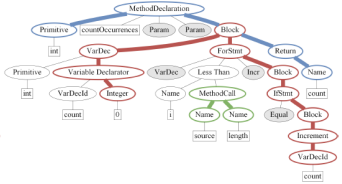
ИСП РАН

Множество путей между листовыми вершинами АСД

```
int countOccurrences(String str, char ch) {
    int num = 0;
    int index = -1;
    do {
        index = str.indexOf(ch, index + 1);
        if (index >= 0) {
            num++;
        }
    } while (index >= 0);
    return num;
}
```

```
int countOccurrences(String source, char value) {
    int count = 0;
    for (int i = 0; i < source.length(); i++) {
        if (source.charAt(i) == value) {
            count++;
        }
    }
    return count;
}
```





Множество путей $x = (x_1, \ldots, x_k)$, где $x_i = (v_{i_1}, \ldots, v_{i_{l_i}})$ — последовательность вершин в пути

Архитектура модели

ИСП РАН

Модель реализует стандартную архитектуру NMT.

Энкодер преобразует последовательность путей АСД (x) в последовательность векторов $z = (\mathbf{z}_1, ..., \mathbf{z}_k)$ фиксированной длины.

Декодер генерирует последовательность выходных токенов $y = (y_1, ..., y_m)$, по одному на каждом шаге декодирования.

На каждом этапе декодирования вероятность следующего токена зависит от предыдущих, она представима в виде:

$$p(y_1, ..., y_m | x_1, ..., x_n) = \prod_{j=1}^{m} p(y_j | y_{<j}, \mathbf{z}_1, ..., \mathbf{z}_n)$$

Энкодинг

ИСП РАН

Закодированный путь x имеет вид:

$$\mathbf{h}_1, \ldots, \mathbf{h}_l = \mathrm{BiLSTM}(\mathbf{E}^{\mathrm{nodes}}_{v_1}, \ldots, \mathbf{E}^{\mathrm{nodes}}_{v_l})$$

$$\mathrm{encode_path}(v_1, \ldots, v_l) = [\mathbf{h}^{\rightarrow}_l; \mathbf{h}^{\leftarrow}_1],$$

где $\mathbf{E}^{\mathrm{nodes}}$ — матрица эмбеддинга (множество типов вершин АСД конечно)

Кроме того, токенизируются и кодируются значения листовых вершин (camelCase, snake_case, kebab-case):

$$\mathrm{encode_token}(t) = \sum_{s \in \mathrm{split}(t)} \mathbf{E}^{\mathrm{subtokens}}_s$$

Результат энкодинга

ИСП РАН

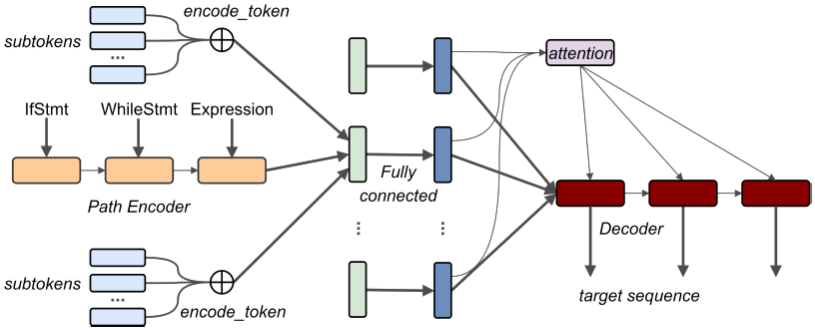
$$\mathbf{z} = \tanh(\mathbf{W}_{\text{in}}[\text{encode_path}(v_1, \ldots, v_l); \text{encode_token}(\text{value}(v_1)); \text{encode_token}(\text{value}(v_l))]),$$

где $\mathbf{W}_{\text{in}}$ — матрица размера $(2d_{\text{path}} + 2d_{\text{token}}) \times d_{\text{hidden}}$

Архитектура сети

Стартовое состояние декодера: $\mathbf{h}_0 = \frac{1}{k}\sum_{i=1}^{k} \mathbf{z}_k$

Присутствует Attention слой, отвечающий за фильтрацию релевантных путей



Эксперимент *A*

Code summarization in Java:

```
public boolean ____ ① (Set<String> set,
                       String value) {
  for (String entry : set) {
  ②
    if (entry.equalsIgnoreCase(value))
  ③       return true;
      }
    }
    return false;
  }
```

contains ① ignore ② case ③

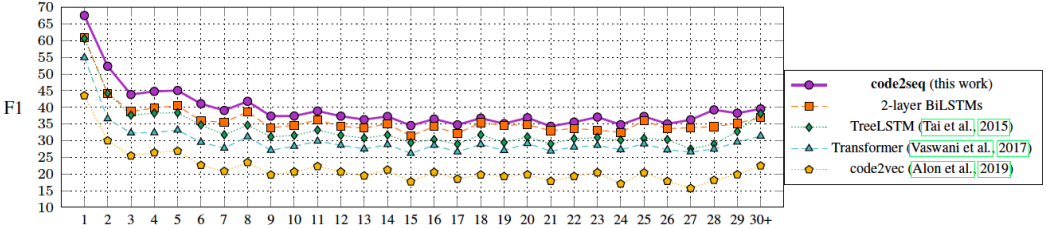
Предсказывается название метода

Результаты эксперимента *A*

ИСП РАН

Model	Java-small			Java-med			Java-large		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
ConvAttention (Allamanis et al., 2016)	50.25	24.62	33.05	60.82	26.75	37.16	60.71	27.60	37.95
Paths+CRFs (Alon et al., 2018)	8.39	5.63	6.74	32.56	20.37	25.06	32.56	20.37	25.06
code2vec (Alon et al., 2019)	18.51	18.74	18.62	38.12	28.31	32.49	48.15	38.40	42.73
2-layer BiLSTM (no token splitting)	32.40	20.40	25.03	48.37	30.29	37.25	58.02	37.73	45.73
2-layer BiLSTM	42.63	29.97	35.20	55.15	41.75	47.52	63.53	48.77	55.18
TreeLSTM (Tai et al., 2015)	40.02	31.84	35.46	53.07	41.69	46.69	60.34	48.27	53.63
Transformer (Vaswani et al., 2017)	38.13	26.70	31.41	50.11	35.01	41.22	59.13	40.58	48.13
code2seq	**50.64**	**37.40**	**43.02**	**61.24**	**47.07**	**53.23**	**64.03**	**55.02**	**59.19**
Absolute gain over BiLSTM	+8.01	+7.43	+7.82	+6.09	+5.32	+5.71	+0.50	+6.25	+4.01

Зависимость от размера примера кода



Эксперимент *B*

ИСП РАН

Code captioning in C#:

```
void Main() {
    string text = File.ReadAllText(@"T:\File1.txt");
    int num = 0;

    text = Regex.Replace(text, "map", delegate (Match m) {
        return "map" + num++;
    }));
    File.WriteAllText(@"T:\File1.txt", text);
}
```

replace① **a string**② **in a text**③ **file**③

Результаты эксперимента *B*

ИСП РАН

Model	BLEU
MOSES† (Koehn et al., 2007)	11.57
IR†	13.66
SUM-NN† (Rush et al., 2015)	19.31
2-layer BiLSTM	19.78
Transformer (Vaswani et al., 2017)	19.68
TreeLSTM (Tai et al., 2015)	20.11
CodeNN† (Iyer et al., 2016)	20.53
code2seq	**23.04**

Датасет — набор пар (вопрос, ответ) со StackOverflow на C#

Выбор k

ИСП РАН

Для приведенных результатов $k = 200$ (количество путей в одном АСД), такое значение выбрано экспериментально. Среднее количество же 220.

Значения $k > 300$ не дают выигрыш в результате, исключая редкие большие примеры. $k < 100$ существенно снижает эффективность модели. Кроме того, $k = 200$ оптимально для хранения на GPU.

Исследование архитектуры сети

ИСП РАН

Model	Precision	Recall	F1	ΔF1
code2seq (original model)	**60.67**	**47.41**	**53.23**	
No AST nodes (only tokens)	55.51	43.11	48.53	-4.70
No decoder	47.99	28.96	36.12	-17.11
No token splitting	48.53	34.80	40.53	-12.70
No tokens (only AST nodes)	33.78	21.23	26.07	-27.16
No attention	57.00	41.89	48.29	-4.94
No random (sample k paths in advance)	59.08	44.07	50.49	-2.74

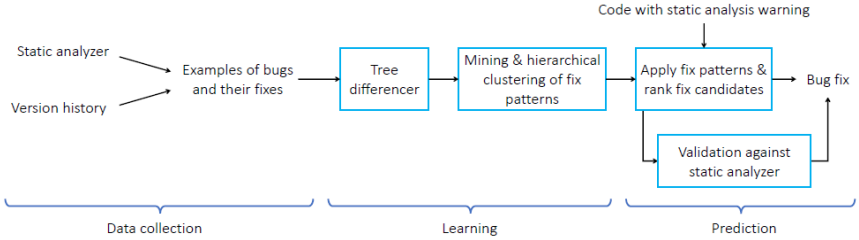
Getafix: Learning to Fix Bugs Automatically (2019)

Концепция

ИСП РАН

- Обработка дефектов в разных версиях кода (по истории)
- Отслеживание конкретных модификаций кода, исправляющих дефекты
- Создание базы правил исправлений в виде преобразований АСД
- Подбор подходящего правила для новых дефектов и автоматическое исправление
- Чем больше история, тем более полная и точная база правил

Схема работы



- Инструмент поиска дефектов должен предоставлять тип дефекта, позицию
- Эксперту предлагается наиболее подходящий корректный вариант исправления

АСД и их трансформация

ИСП РАН

Узел АСД — метка (тип), значение, набор дочерних узлов.

Трансформация АСД — изменение структуры узлов + отображение узлов и их компонентов.

Трансформации узлов:

- Добавление — создание новых дочерних узлов
- Удаление — удаление некоторых дочерних узлов
- Перемещение — перенос дочернего узла с потерей отображения между родительскими узлами
- Обновление — изменение значения узла

Нотации АСД

ИСП РАН

$$Tree = Label \times Value \times \bigcup_{k \in \mathbb{N}} (Location \times Tree)^k$$

$$\mathrm{x = y + 2 \Rightarrow Assign(x : Name, + : BinEx(y : Name, 2 : Literal))}$$

$$Edit = Tree \times Tree \times \mathcal{P}^{Mapping}$$

$$Mapping = TreeRef \times TreeRef \times \{mod, unmod\}$$

$$\mathrm{x = y + 2 \mapsto x = 3 + y}$$

$$\Downarrow$$

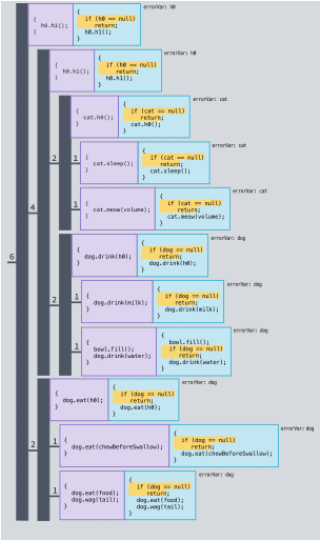
$$\mathsf{Assign}_0(\mathrm{x : Name_1}, \mathsf{+} : \mathsf{BinEx}_2(\mathsf{y} : \mathsf{Name}_3, \mathsf{2} : \mathsf{Literal})) \mapsto$$

$$\mathsf{Assign}_0(\mathrm{x : Name_1}, \mathsf{+} : \mathsf{BinEx}_2(\mathsf{3} : \mathsf{Literal}, \mathsf{y} : \mathsf{Name}_3))$$

Алгоритм агломеративной кластеризации

ИСП РАН

- Pick a pair e_1, e_2 of edit patterns from W.
- Generalize the edit patterns using anti-unification, which yields $e_3 = \text{antiUnify}(e_1, e_2)$.
- Remove e_1 and e_2 from W, and add e_3 to W instead.
- Set e_3 as the parent of e_1 and e_2 in the hierarchy.



Ранжирование правил

ИСП РАН

Для правила p, исправляющего ошибку при применении на позиции z относительно позиции ошибки по базе правил H:

$$s_{\text{prevalence}} = \frac{|\{\texttt{instances of } p \texttt{ in } H\}|}{|H|}$$

$$s_{\text{location}} = \frac{|\{\texttt{instances of } p \texttt{ at location } z \texttt{ in } H\}|}{|\{\texttt{instances of } p \texttt{ in } H\}|}$$

$$s_{\text{specialized}} = \frac{|\{\texttt{all AST nodes}\}|}{|\{\texttt{all AST nodes that match } p\}|}$$

Максимальное значение $s_{\text{prevalence}} \cdot s_{\text{location}} \cdot s_{\text{specialized}}$ — наиболее подходящий кандидат.

Практические результаты (язык Java)

ИСП РАН

Статистика по типам ошибок

Категория ошибки	Количество	Top-1	Top-5
NullPointerException	804	94 (12%)	156 (19%)
BoxedPrimitiveConstructor	260	236 (91%)	238 (92%)
ClassNewInstance	21	6 (29%)	11 (52%)
DefaultCharSet	55	28 (51 %)	42 (76%)
OperatorPrecedence	49	6 (12%)	30 (61%)
ReferenceEquality	79	22 (28%)	53 (67%)
Итого	1268	381 (30%)	526 (41%)

Статистика использования исправлений

Исправление применено автоматически	42%
Исправление не из Top-1	10%
`assert` вместо исправления	10%
Исправление модифицировано с сохранением семантики	9%
Удален код с ошибкой	10%
Исправление модифицировано	19%

Hoppity: learning graph transformations to detect and fix bugs in programs (2020)

Введение

ИСП РАН

Задача

Поиск и автоматическое исправление дефектов в программах на языке JavaScript.

Мотивация:

- Сложность и размер кодовых баз $\Rightarrow$ много дефектов
- Существующие инструменты ориентированы на конкретные ошибки/проекты
- Отсутствие end-to-end инструментов для языка JavaScript

Особенности анализа языка JavaScript

ИСП РАН

- Динамическая типизация
- Особенности языка, порождающие специфические ошибки
  - Доступ к неопределенным полям (undefined properties)
  - Некорректное использование операторов сравнения (`!=` vs `!==`)
  - Проблемы с областью видимости переменных (var)
- Отсутствие хорошего инструментария (tooling) для поиска ошибок

Особенности анализа языка JavaScript

ИСП РАН

```
function clearEmployeeListOnLinkClick(){
  document.querySelector("a").addEventListener("click",
    function(event){
      document.querySelector("ul").InnerHTML = "";
    }
  );
}
```

(a) `InnerHTML` should have been `innerHTML`.

```
if (matches) {
  return {
    episode: Number(matches.groups.episode),
    hosts: matches.groups.hosts.split(/([,&]+|\sand\s)/).
              map(el => S(el).trim().s)
  };
}
```

(b) Highlighted parentheses should have been removed.

```
module.exports = function (grunt) {
  grunt.initConfig({
    execute: {...}, copy: {...}, checktextdomain: {...}
    wp_readme_to_markdown: {...}, makepot: {...}})
  ...
  grunt.registerTask('default',['wp_readme_to_markdown'
    ,'makepot','execute','checktextdomain'])
};
```

(c) `copy` function should have also been included in the highlighted list.

```
export default {
  computed: {
    level () {
      return dictMap.skillLevel[
        parseInt((this.value === 0 ? 1 : this.value)/20)];
    }
  },...
}
```

(d) `parseInt` should have been removed because `===` implies `this.value` is an integer.

Подход

ИСП РАН

Идея

Научиться для данного внутреннего графового представления программы с ошибкой выдавать последовательность необходимых преобразований на уровне графа (graph-to-graph)

Отличия от существующих решений:

- Одна модель и для поиска, и для исправления ошибок
- Поддержка нескольких типов ошибок в одной модели
- Поддержка более сложных преобразований (с добавлением/удалением выражений)

Формальная постановка задачи

ИСП РАН

Задача структурного обучения (structured prediction) на графовом представлении программ.

Даны пары $(g_{\text{bug}}, g_{\text{fix}})$, где g_{bug} — граф программы с ошибкой; g_{fix} — граф программы без ошибки.

Модель для предсказания до T шагов преобразования графа

$$p(g_{\text{fix}}|g_{\text{bug}};\theta) = p(g_1|g_{\text{bug}};\theta)p(g_2|g_1;\theta)\ldots p(g_{\text{fix}}|g_{T-1};\theta)$$

Представление программы

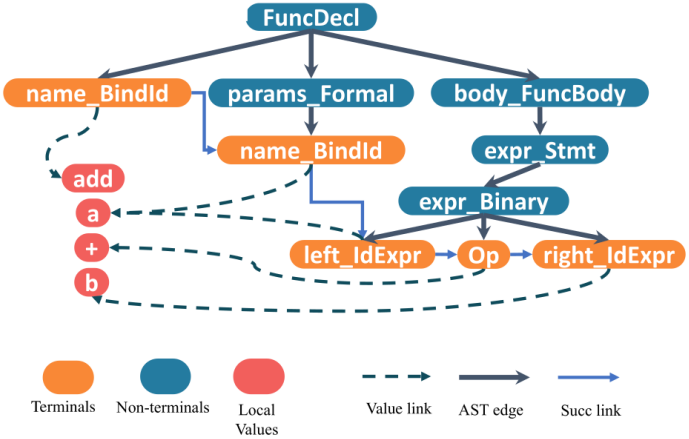
ИСП РАН

- Граф программы — расширенное АСД
- `SuccToken` ребра для соединения листовых узлов между собой
- Value-узлы со значениями для листовых вершин
- `ValueLink` ребра для соединения листовых узлов с value-узлами

Value-узлы и `ValueLink` нужны для представления и преобразования кода, которые не зависят от конкретных имен.

Представление программы: пример

ИСП РАН




```
function add(a) { a + b; }
```

Векторное представление графов

ИСП РАН

- Дано АСД $g = (V, E)$, где $v \in V$ — множество вершин, и E — множество ребер разных типов
- K — общее число различных типов ребер в графе
- $f(g) \to (\mathbb{R}^d, \mathbb{R}^{|V| \times d})$, для d-мерного представления графа $\vec{g}$ и вершин $\vec{v}$
- Для параметризации $f(\cdot)$ — графовая сеть, вариация Graph Isomorphism Network (GIN)

Графовая сеть

ИСП РАН

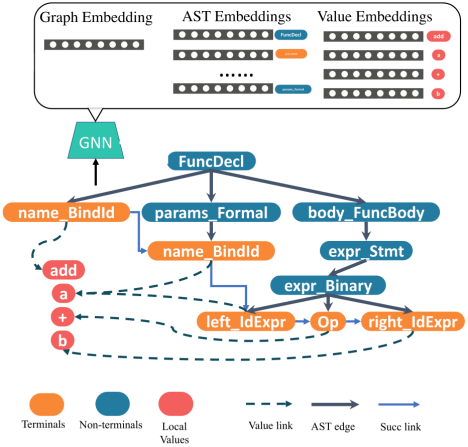
Вариация GIN

$$\mathbf{h}_v^{(l+1),k} = \tanh\Big(\sum_{u \in \mathcal{N}^k(v)} \mathbf{W}_1^{l,k} \mathbf{h}_u^{(l)}\Big), \quad \forall k \in 1, 2, \ldots, K$$

$$\mathbf{h}_v^{(l+1)} = \tanh(\mathbf{W}_2^l [\mathbf{h}_v^{(l+1),1}, \mathbf{h}_v^{(l+1),2}, \ldots, \mathbf{h}_v^{(l+1),K}] + \mathbf{h}_v^{(l)})$$

- $\mathbf{W}_1^{l,k} \in \mathbb{R}^{d \times d}$, $\mathbf{W}_2^l \in \mathbb{R}^{dK \times d}$ — параметры модели
- $\mathcal{N}^k(v)$ — множество соседей вершины v по ребру типа k
- Вложение $\vec{v} = \mathbf{h}_v^{(L)}$, где L — общее число итераций распространения
- Вложение $\vec{g}$ — среднее max pooling агрегаций $\mathbf{h}_v^l, \forall l \in 0, 1, \ldots, L$

Графовая сеть




```
function add(a) { a + b; }
```

Трансформации графов: операции

ИСП РАН

Один шаг трансформации — одна из 5 операций:

1. **ADD** — добавление узла в АСД
2. **DEL** — удаление узла из АСД
3. **REP_VAL** — замена значения в узле АСД
4. **REP_TYPE** — замена типа узла АСД
5. **NO_OP** — окончание цепочки трансформаций

Все операции основаны на одном и том же наборе низкоуровневых *примитивов*.

Трансформации графов: примитивы

ИСП РАН

Три низкоуровневых примитива:

1. Примитив Location
2. Примитив Value
3. Примитив Type

Контроллер

Вектор $\vec{c} \in \mathbb{R}^d$ (d — размер пространства вложений), в котором закодированы глобальное состояние и история трансформаций.

Примитив Location

ИСП РАН

- Отвечает за поиск узла (участка исходного кода) для трансформации
- Разные программы имеют разное количество узлов
- Используется специальный тип сетей — pointer networks (Vinyals et al. 2015)
- После получения вложений вершин $\{\vec{v}\}_{v \in V}$, узел выбирается согласно

$$\mathrm{loc}(\vec{c}, g) = \arg\max_{(v \in V)} \vec{v}^T \vec{c}$$

Примитив Value

ИСП РАН

- Присваивает конкретное значение листовому узлу АСД
- Использует механизм внимания (attention), который позволяет выбирать:
  - Локальные значения из текущего файла (Local Value Table V_{val})
  - Глобальные значения, которые наиболее часто встречаются в листовых вершинах для конкретного языка (Global Value Dictionary D_{val})
- Значение выбирается согласно:

$$\text{val}(\vec{c}, g) = \arg \max_{(t \in D_{\text{val}} \cup V_{\text{val}})} \vec{t}^T \vec{c}$$

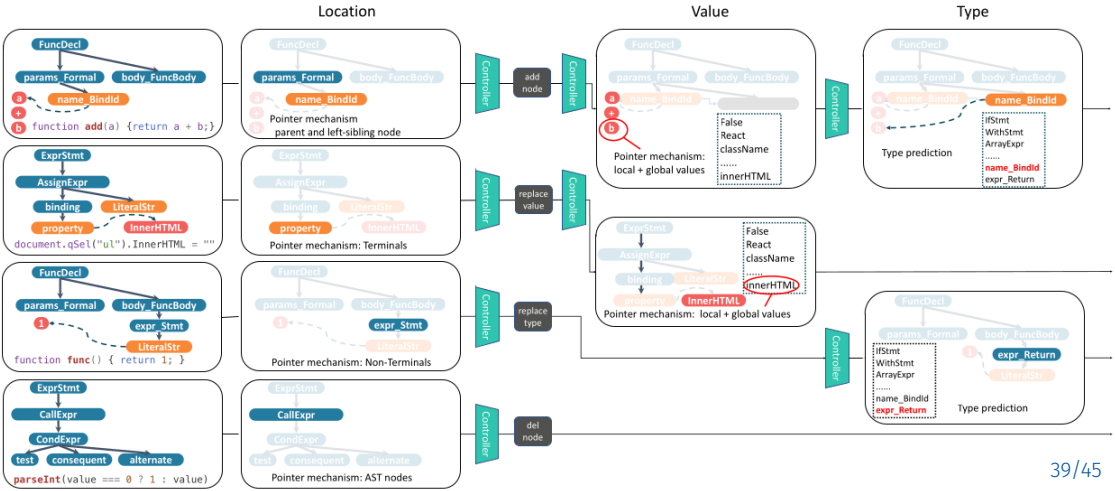
Примитив Type

ИСП РАН

- Присваивает тип нетерминальному узлу АСД
- Множество типов конечно для конкретного языка $\Rightarrow$ задача многоклассовой классификации
- Множество типов дополнительно сокращается за счет синтаксических проверок АСД (валидность по правилам вывода из грамматики)

Трансформации графов: операции и примитивы

ИСП РАН



Трансформации графов: шаг t

ИСП РАН

Даны граф g_{t-1}, его вложение $\vec{g}_{t-1}$ и вектор макро-контекста $\vec{C}_{M_{t-1}}$

1. Обновить макро-контекст: $\vec{C}_{M_t} = \mathrm{LSTM}(\vec{g}_{t-1} | \vec{C}_{M_{t-1}})$
2. Выбрать узел v для трансформации: $\mathrm{loc}(\vec{C}_{M_t}, \vec{g}_{t-1})$
3. Выбрать операцию e_t для данных v_t и $\vec{C}_{M_t}$ (для NO_OP останов)
   - в статье не говорится как, вероятно, softmax слой :)
4. Вычислить микро-контекст для операции: $\vec{c}_{m_t} = \mathrm{LSTM}(\vec{e}_t | \mathrm{LSTM}(\vec{v}_t | \vec{C}_{M_t}))$
5. Применить операцию e_t с помощью микро-контекста $\vec{c}_{m_t}$ (используется для получения val), получить граф g_t

Фаза обучения: датасет

ИСП РАН

- Датасет $\mathcal{D} = \{(g^{(i)}_{\text{bug}}, g^{(i)}_{\text{fix}})\}_{i=1}^{|\mathcal{D}|}$ с Github $\sim$ 500 тыс. пар
- SHIFT AST и JSON diff для извлечения последовательности трансформаций АСД
- Три варианта по количеству трансформаций:
  1. `OneDiff` — исправления с ровно 1 трансформацией
  2. `ZeroOneDiff` — исправления с 0 или 1 трансформацией
  3. `ZeroOneTwoDiff` — исправления с 0, 1 или 2 трансформациями
- Дополнильная фильтрация АСД с количеством узлов >500

Фаза обучения

ИСП РАН

- Цель — максимизация правдоподобия исправления:

$$\max_{\theta} \mathbb{E}_{[(g_{\text{bug}}, g_{\text{fix}}) \sim \mathcal{D}]} p(g_{\text{fix}} | g_{\text{bug}}; \theta)$$

- Функция потерь — сумма cross entropy loss для каждой трансформации
- Adam с $\beta_1 = 0.9, \beta_2 = 0.99$, начальный learning rate 10^{-3}, batch size 10 и 3 эпохи
- Модули для векторного представления графов, модули для операций и модуль с контроллером (LSTM) оптимизируются совместно

Фаза предсказаний

ИСП РАН

- Поиск максимума: $G = \arg\max_{g_{\text{fix}}} p(g_{\text{fix}}|g_{\text{bug}}; \theta)$
- Пространство поиска очень большое $\Rightarrow$ используется лучевой поиск (beam search)
- Размер множества поиска ограничен параметром B: шириной лучевого поиска; в работе пробовали $B = 1$ и $B = 3$
- На выходе отранжированный список из Top-B наиболее вероятных трансформации

Результаты

ИСП РАН

	ADD	REP_TYPE	REP_VAL	DEL	total
train	6,473	1,864	251,097	31,281	290,715
validate	790	245	31,357	3,957	36,349
test	796	233	31,387	3,945	36,361

Table 1: Statistic of OneDiff dataset. See appendix for more information of other dataset.

	Total		*Location*		*Operator*	*Value*		*Type*	
	Top-3	Top-1	Top-3	Top-1	Top-1	Top-3	Top-1	Top-3	Top-1
TOTAL	**26.1**	14.2	35.5	20.4	34.4	52.3	29.1	76.1	66.7
ADD	52.9	39.2	69.6	51.4	70.6	65.7	55.1	76.8	68.5
REP_VAL	23.4	11.9	33.3	18.5	31.7	53.0	28.8	-	-
REP_TYPE	71.7	52.4	73.0	52.8	79.4	-	-	74.7	61.0
DEL	39.6	24.8	44.0	27.5	45.8	-	-	-	-
Random	.08	.07	2.28	1.4	27.7	.01	.01	.27	0

Table 2: Evaluation of model on the OneDiff dataset: accuracy (%).

Результаты

ИСП РАН

	Total		*Location*		*Operator*	*Value*		*Type*	
	Top-3	Top-1	Top-3	Top-1	Top-1	Top-3	Top-1	Top-3	Top-1
`ZeroOneTwoDiff`	40.8	29.7	18.9	3.9	30.3	35.0	6.5	38.6	3.4
`ZeroOneDiff`	51.6	34.5	27.1	5.5	35.6	45.4	10.4	73.9	58.9
`OneDiff`	**26.1**	14.2	35.5	20.4	34.4	52.3	29.1	76.1	66.7
Random	.08	.07	2.28	1.4	27.7	.01	.01	.27	0

Table 7: Evaluation of models on each dataset. The Random model is evaluated on the `OneDiff` dataset and is shown for comparison.