Экспериментальное исследование алгоритмов контекстно-свободной достижимости применительно к задачам статического анализа кода

Автор: Кутуев Владимир Александрович,
Научный руководитель: к. ф.-м. н., доцент Григорьев С. В.

Санкт-Петербургский государственный университет
Кафедра системного программирования

11 мая 2023г.

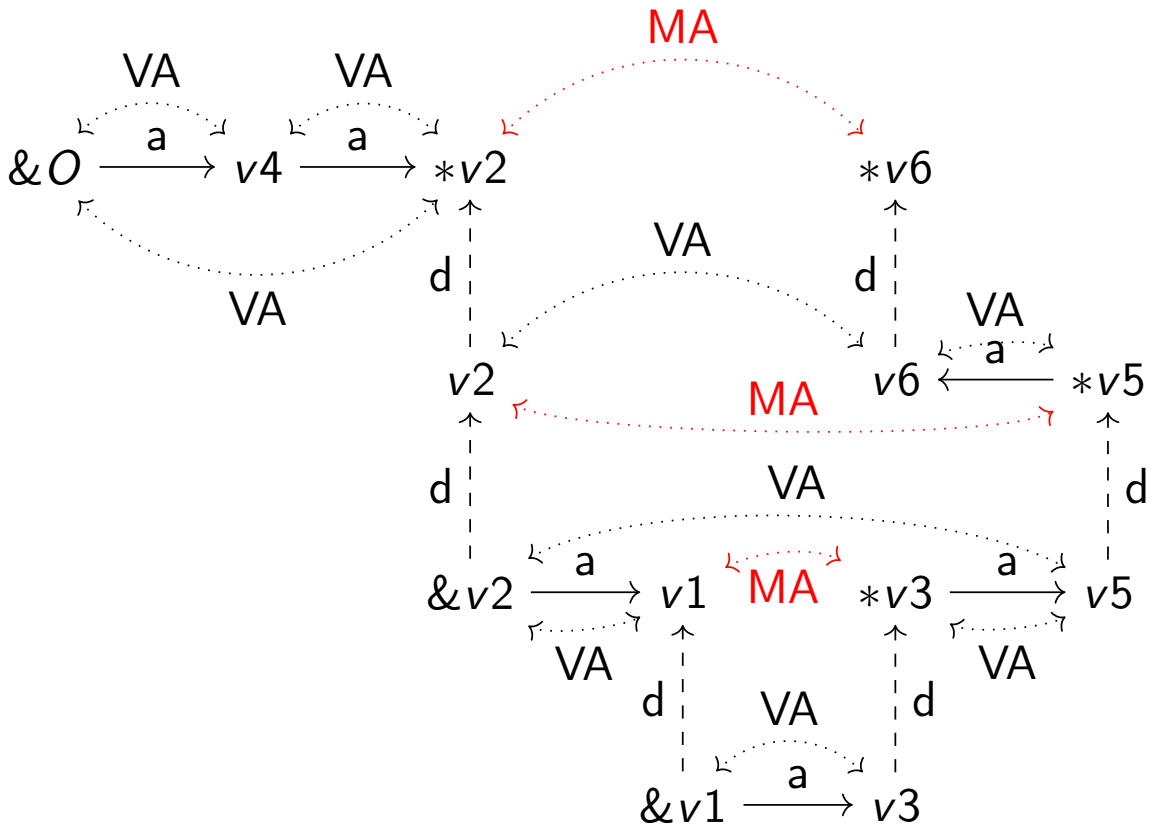
КС-достижимость

- Контекстно-свободная грамматика — $G = (\Sigma, N, P, S)$
  - Σ — множество терминальных символов
  - N — множество нетерминальных символов
  - P — множество продукций $\{A \to \alpha | A \in N, \alpha \in (\Sigma \cup N)^*\}$
  - $S \in N$ — стартовый
- Граф — $\mathcal{G} = (V, E, l)$
  - V — множество вершин
  - $E \subseteq V \times V$ — множество рёбер
  - $l : E \to L$
- $\{(v_1, v_n) : \exists p = (e_1, e_2, ..., e_n)) \in E^*, src(e_1) = v_1, dst(e_n) = v_n, l(e_1) \cdot ... \cdot l(e_n) \in L(G)\}$

Анализ псевдонимов

Программа

```
v1 = &v2;
v3 = &v1;
v4 = malloc(...);
*v2 = v4;
v5 = *v3;
v6 = *v5;
```



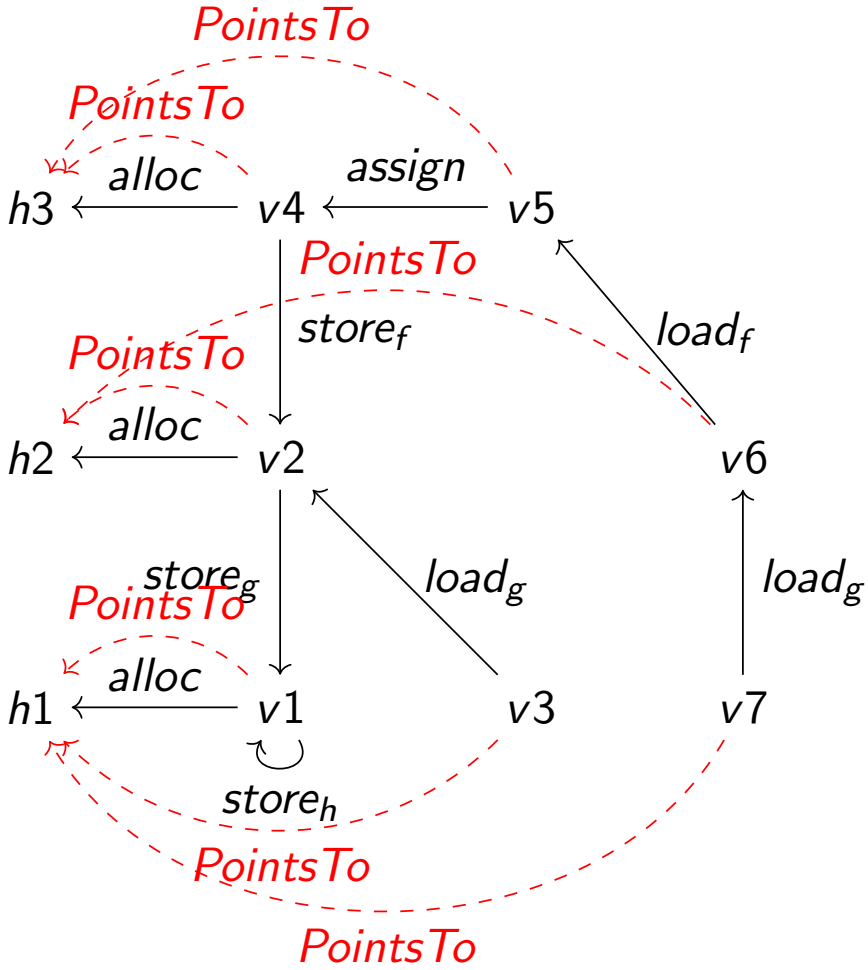
Грамматика, задающая анализ:

$MA \rightarrow \overline{d}\ VA\ d$

$VA \rightarrow (MA?\ \overline{a})^*\ MA?\ (a\ MA?)^*$

Points-to анализ, учитывающий поля

```
v1 = new Obj(); // h1
v2 = new Obj(); // h2
v4 = new Obj(); // h3
v5 = v4;
v1.h = v1;
v2.g = v1;
v4.f = v2;
v6 = v5.f;
v3 = v2.g;
v7 = v6.g;
```



Грамматика, задающая анализ:

$PointsTo \rightarrow (assign \mid load_f\ Alias\ store_f)^*\ alloc$

$Alias \rightarrow PointsTo\ \ \overline{FlowsTo}$ $\qquad \forall\ f \in Fields$

$FlowsTo \rightarrow \overline{alloc}\ (\overline{assign} \mid \overline{store_f}\ Alias\ \overline{load_f})^*$

Постановка задачи

Цель работы — экспериментально исследовать алгоритмы KC-достижимости в задаче статического анализа кода

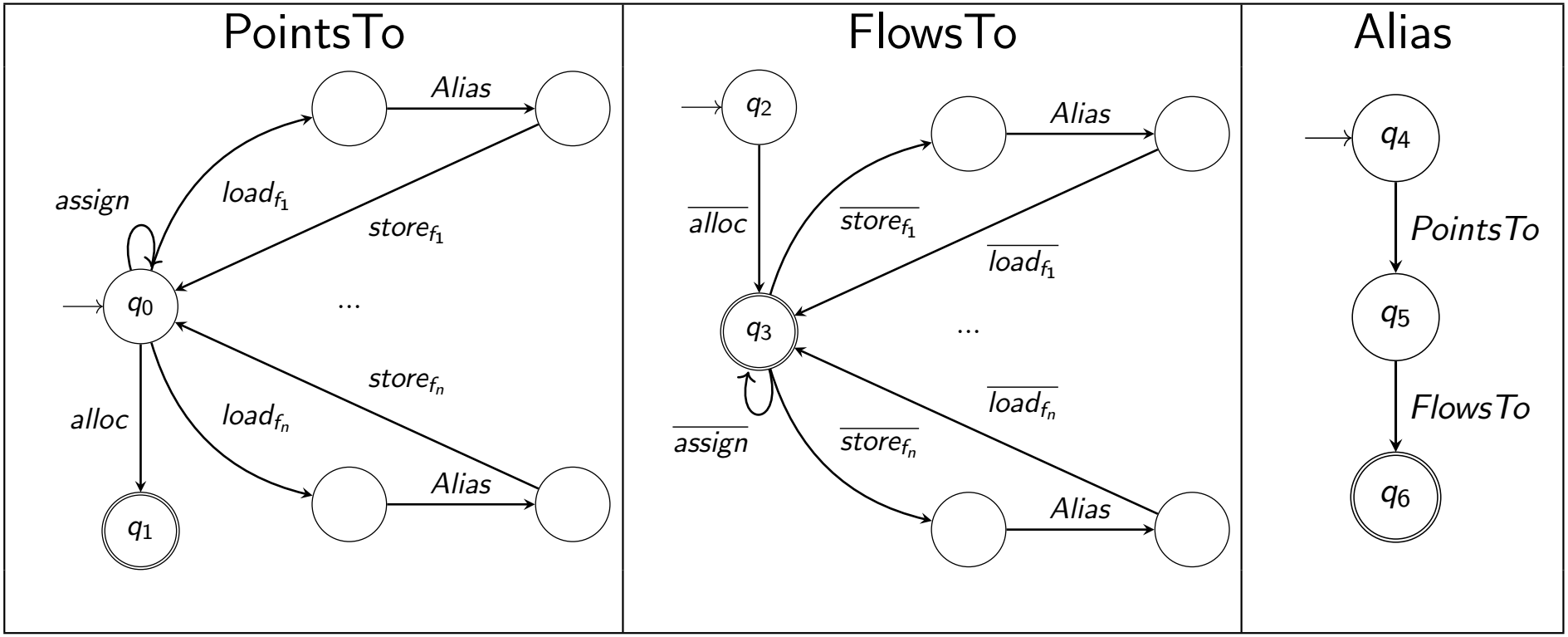
Задачи:

- Рассмотреть возможность применения алгоритмов, основанных на операциях линейной алгебры, для Points-to анализа, учитывающего поля, и предложить модификацию алгоритма, подходящую для этого анализа
- Оптимизировать реализации алгоритмов, основанных на операциях линейной алгебры
- Провести замеры производительности алгоритмов, основанных на операциях линейной алгебры, на графах, полученных по реальным программам, сравнить их с другими алгоритмами KC-достижимости

Алгоритмы, основанные на операциях линейной алгебры

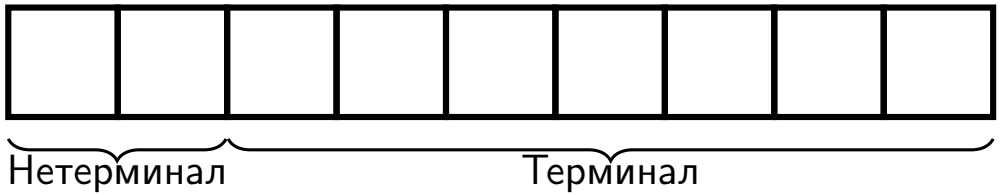
- Матричный алгоритм
  - Требует перевода грамматики в ослабленную нормальную форму Хомского, что значительно увеличивает её размер
  - Производительность зависит от размера грамматики
- Тензорный алгоритм
  - КС-язык задаётся рекурсивным автоматом
  - Рекурсивный автомат и граф представляются как композиция булевых матриц смежности для каждой метки на ребре

Рекурсивный автомат



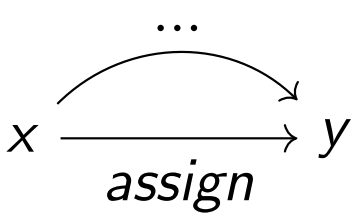
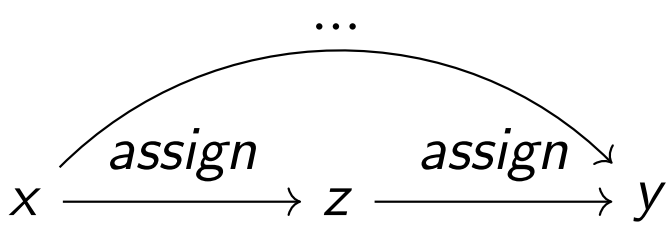
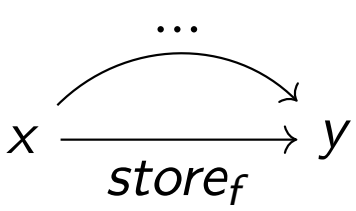
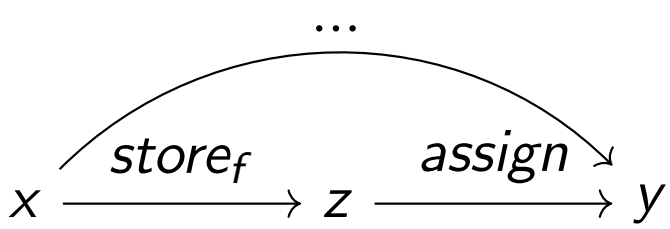
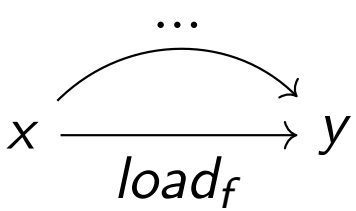
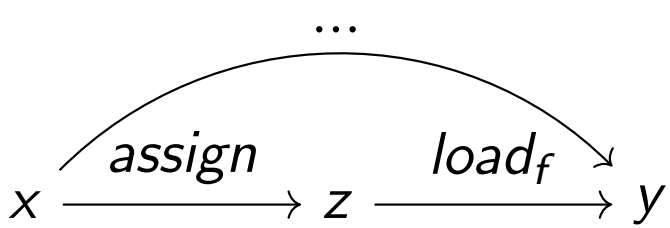
Представление с одним терминалом и одним нетерминалом

- Граф и рекурсивный автомат представляются матрицами смежности с целочисленными элементами
- Старшие биты содержат номер нетерминала, младшие — терминала



- Операция умножения элементов для произведения Кронекера
  - $times(x, y) = (x_{nonterm} = y_{nonterm} \neq 0 \ or \ x_{term} = y_{term} \neq 0)$

Адаптация графа

Исходные рёбра	Новая переменная	Результат
	z = y; x = z;	
	z = y; x.f = z;	
	z = y.f; x = z;	

Данные для экспериментов

- Для экспериментов были взяты графы из набора CFPQ_Data[1]
- Анализ псевдонимов
  - 5 небольших графов (число вершин — несколько тысяч)
  - 15 больших графов (число вершин — несколько миллионов)
- Points-to анализ, учитывающий поля
  - 10 средних графов (число вершин — десятки тысяч)
  - 4 больших графа (число вершин — сотни тысяч)

[1] https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_Data

Оптимизация реализации матричного алгоритма

- Реализация матричного алгоритма CFPQ_PyAlgo[2]
- BOOL.LOR_LAND
  - сложение — LOR (дизъюнкция)
  - умножение — LAND (конъюнкция)
- BOOL.ANY_PAIR
  - сложение — ANY (выбирает любой из переданных аргументов)
  - умножение — PAIR (возвращает 1, если оба операнда присутствующие в матрице элементы)

Граф	$\|V\|$	$\|E\|$	LOR_LAND (сек.)	ANY_PAIR (сек.)	Ускорение
wc	332	269	0,006	0,006	1,00
bzip2	632	556	0,021	0,022	0,95
pr	815	692	0,013	0,012	1,08
ls	1 687	1 453	0,051	0,045	1,13
gzip	2 687	2 293	0,038	0,030	1,26
apache	1 721 418	1 510 411	683,58	536,7	1,27
init	2 446 224	2 112 809	59,33	45,84	1,29

[2] https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_PyAlgo

Сравниваемые реализации

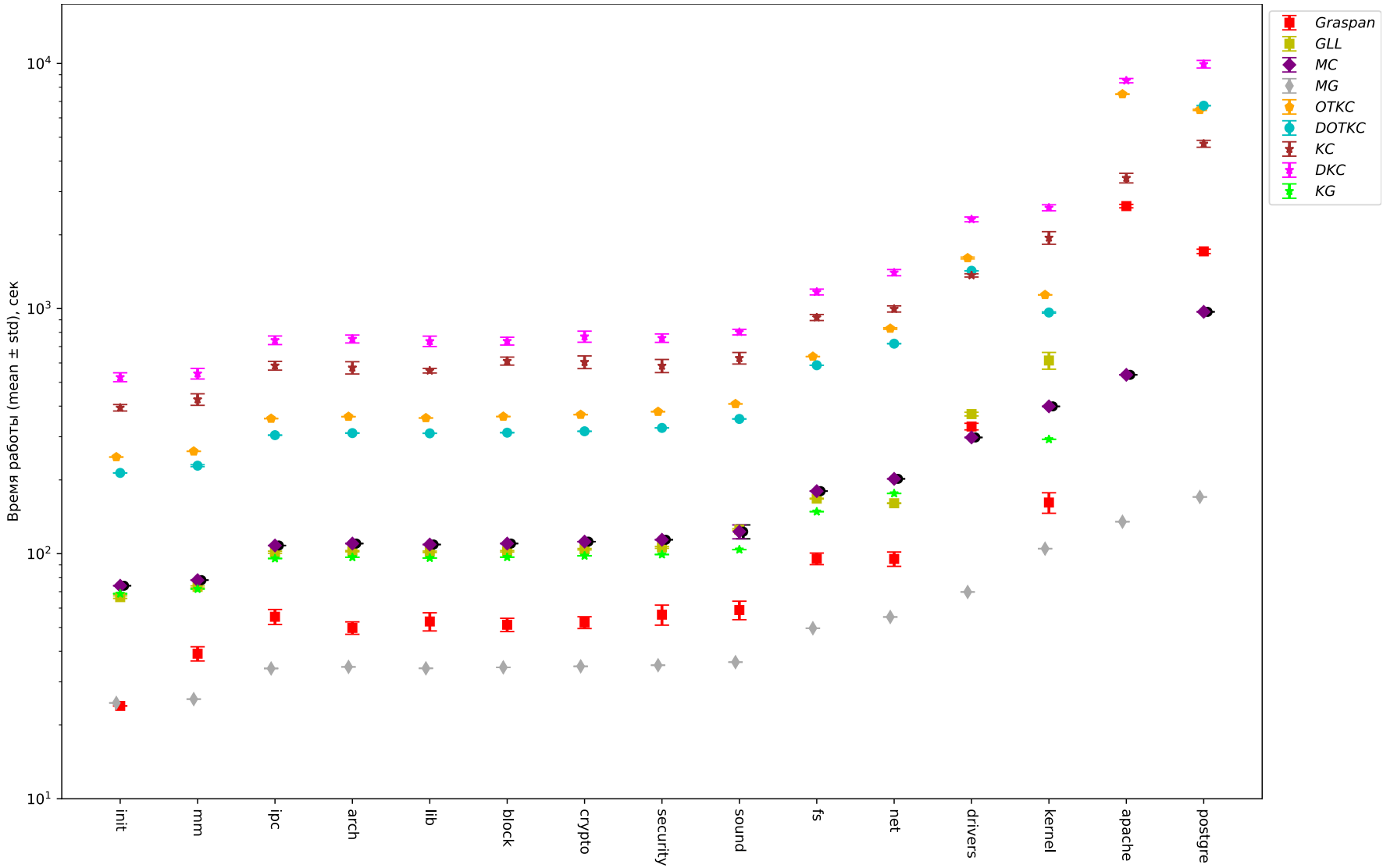
- Реализации матричного алгоритма из CFPQ_PyAlgo для CPU (**MC**) и GPU (**MG**)
- Реализация тензорного алгоритма из CFPQ_PyAlgo для CPU (**KC**) и GPU (**KG**), инкрементальная версия тензорного алгоритма для CPU (**DKG**)
- Реализация адаптированного для Points-to анализа, учитывающего поля, тензорного алгоритма (OTKC) и его инкрементальная версия (OTDKC);
- **Graspan**[3] (запускался только для анализа псевдонимов, так как эта реализация не поддерживает грамматики с большим количеством нетерминалов)
- **Gigascale**[4] (запускался только для Points-to анализа, учитывающего поля, так как эта реализация заточена под конкретную грамматику)
- **GLL**[5] (запускалась вариация с хранением графа в оперативной памяти)

[3] https://github.com/Graspan/Graspan-C

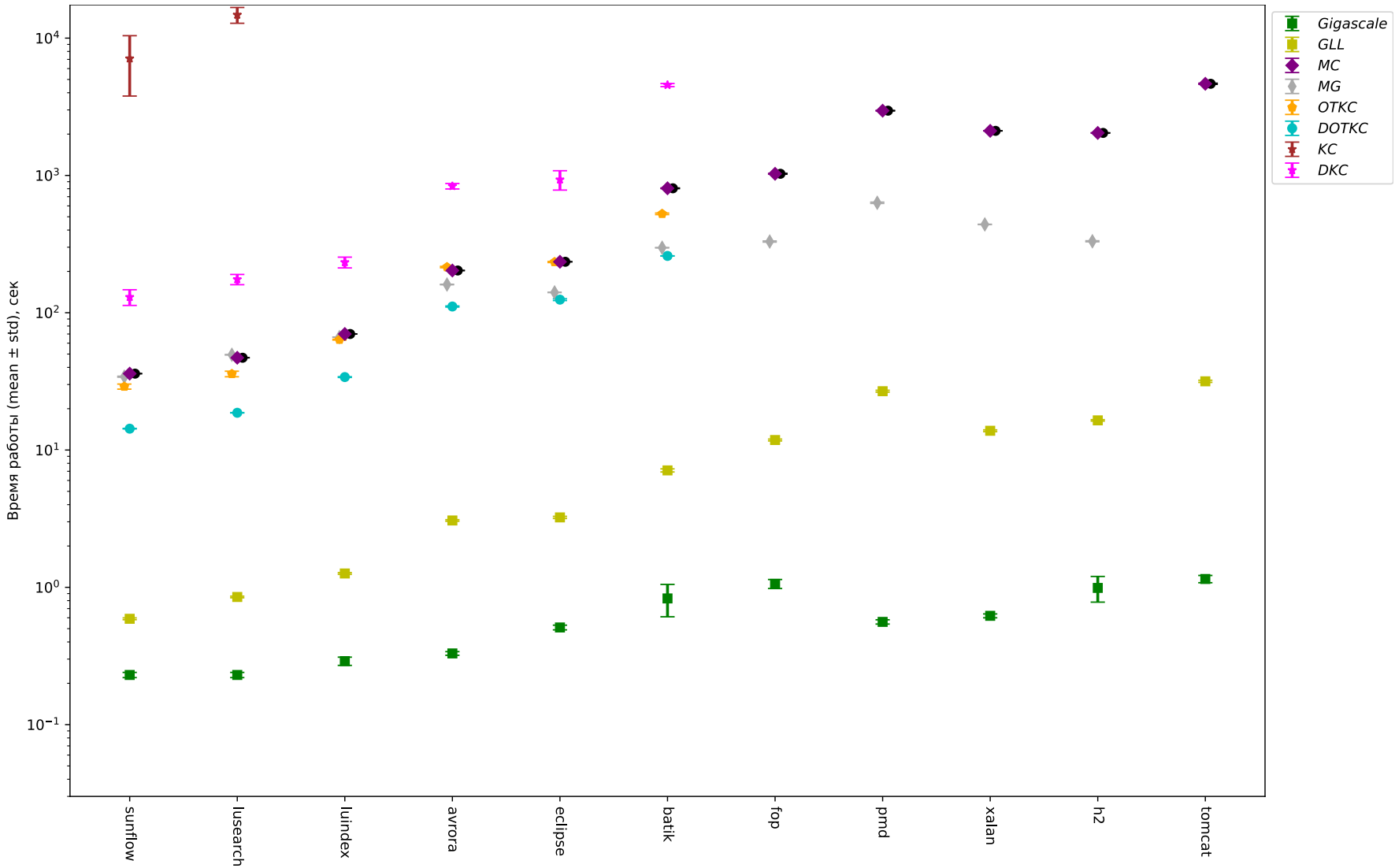
[4] https://bitbucket.org/jensdietrich/gigascale-pointsto-oopsla2015

[5] https://github.com/FormalLanguageConstrainedPathQuerying/GLL4Graph

Анализ псевдонимов: время работы



Points-to анализ, учитывающий поля: время работы



Результаты

- Для Points-to анализа, учитывающего поля, была предложена модификация тензорного алгоритма. Данная модификация была реализована в рамках библиотеки CFPQ_PyAlgo
- Оптимизирована реализация матричного алгоритма из библиотеки CFPQ_PyAlgo, эффективность оптимизации экспериментально проверена
- Проведены замеры производительности реализаций алгоритмов KC-достижимости