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.
Рецензент: старший преподаватель СПбПУ, Беляев М. А.

Санкт-Петербургский государственный университет
Кафедра системного программирования

КС-достижимость

- Область применения
  - Анализ RDF данных
  - Биоинформатика
  - Статический анализ кода
- Алгоритмы
  - Основанные на алгоритмах синтаксического анализа
    - LL
    - GLL
    - SYK
    - ...
  - Основанные на операциях линейной алгебры
    - Алгоритм, основанный на умножении матриц
    - Алгоритм, основанный на произведении Кронекера

Постановка задачи

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

Задачи:

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

Данные для экспериментов

- Для экспериментов были взяты графы из набора CFPQ_Data[1]
- Анализ псевдонимов
  - 5 небольших графов (число вершин — до нескольких 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	0,022	0,022	1,00
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	536,70	1,27
init	2 446 224	2 112 809	59,33	45,84	1,29

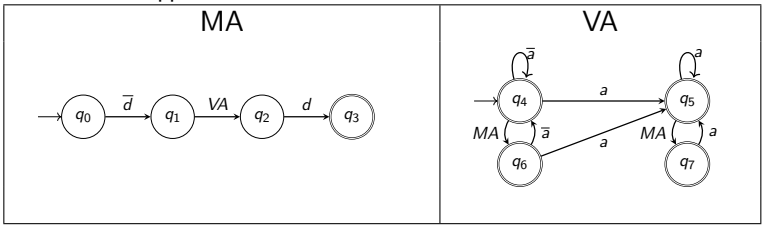
[2] https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_PyAlgo

Алгоритмы, основанные на операциях линейной алгебры

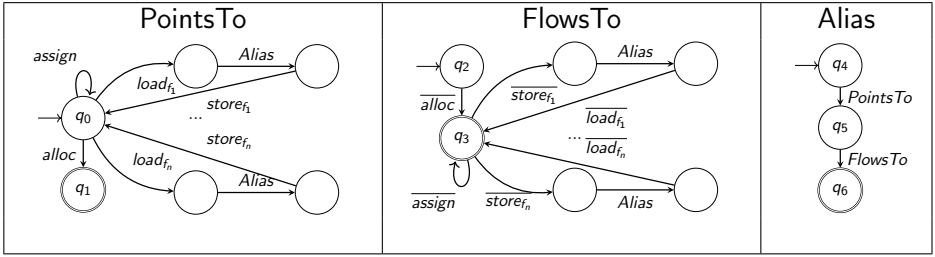
- Матричный алгоритм
  - Основан на умножении булевых матриц
  - Требует перевода грамматики в ослабленную нормальную форму Хомского, что значительно увеличивает её размер
  - Производительность зависит от размера грамматики
- Тензорный алгоритм
  - Основан на произведении Кронекера булевых матриц
 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

Рекурсивный автомат

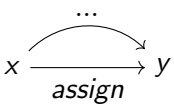
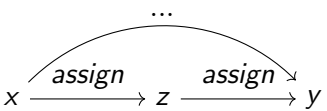
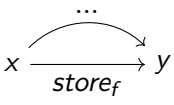
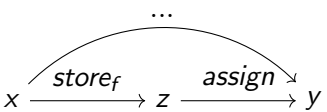
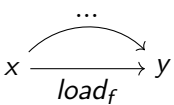
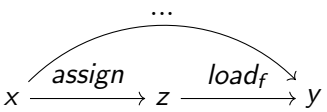
- Анализ псевдонимов



- Points-to анализ, учитывающий поля

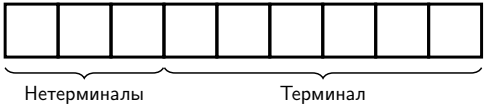


Адаптация графа

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

Представление с одним терминалом

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



- Операция умножения элементов для произведения Кронекера
  - $times(x, y) = (x_{nonterms} \& y_{nonterms} \neq 0 \ or \ x_{term} = y_{term} \neq 0)$
  - В матрице-результате много элементов будет *false*
    - Вычисление транзитивного замыкания потребует больше времени
    - Нужно больше памяти под его хранение
  - Необходима фильтрация результата произведения Кронекера

Фильтрация результатов

- Реализация произведения Кронекера из библиотеки SuiteSparse:GraphBLAS[3]
- Адаптированная реализация[4], не выделяющая память под значения *false*

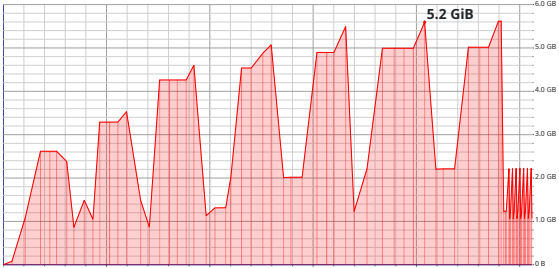


Рис.: Стандартная реализация

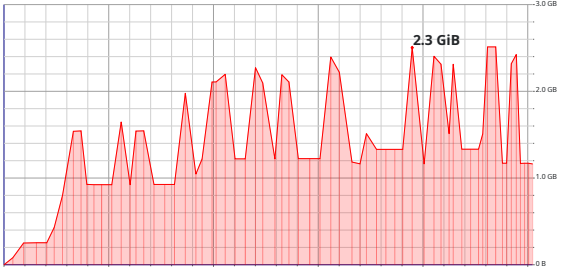


Рис.: Оптимизированная реализация

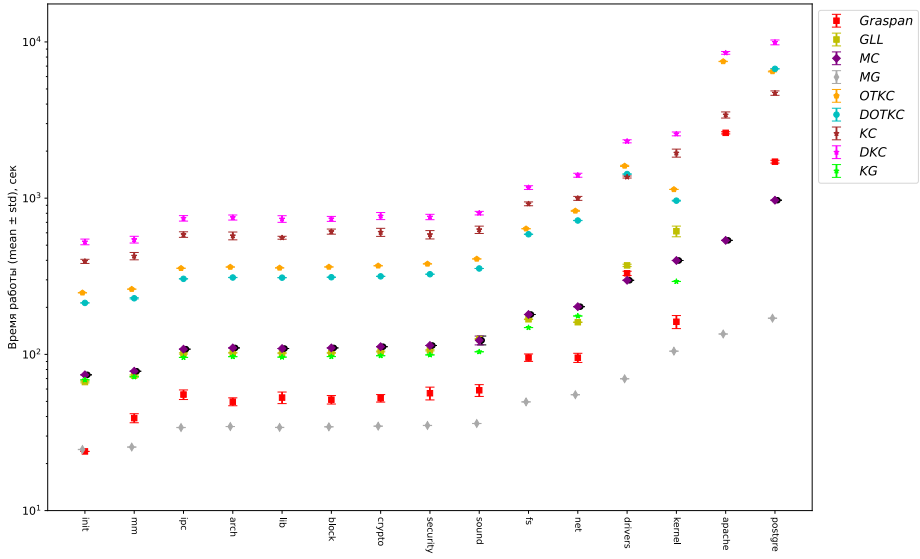
[3] https://github.com/DrTimothyAldenDavis/GraphBLAS
[4] https://github.com/vkutuev/GraphBLAS/tree/vkutuev/kron

Сравниваемые реализации

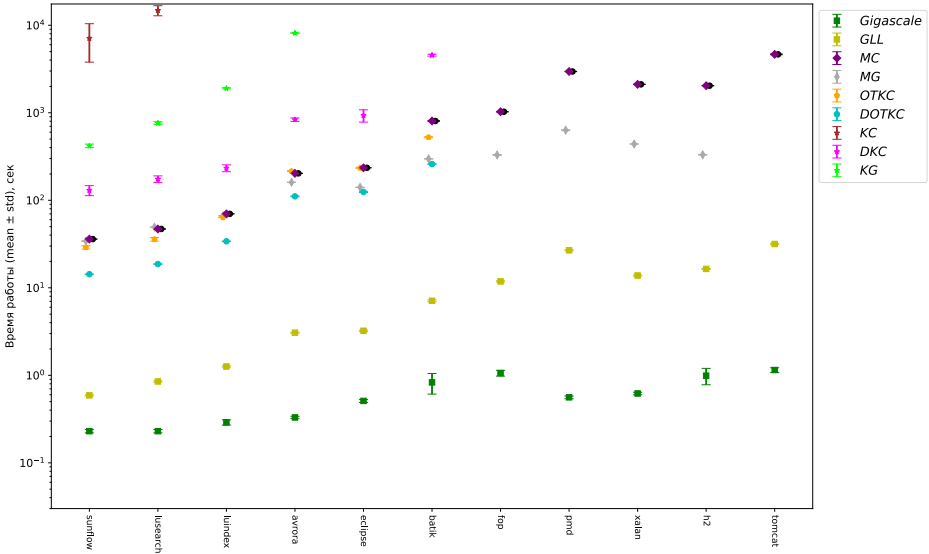
- Реализации матричного алгоритма из CFPQ_PyAlgo для CPU (**MC**) и GPU (**MG**)
- Реализация тензорного алгоритма из CFPQ_PyAlgo для CPU (**KC**) и GPU (**KG**), инкрементальная версия тензорного алгоритма для CPU (**DKC**)
- Реализация адаптированного для Points-to анализа, учитывающего поля, тензорного алгоритма (**OTKC**) и его инкрементальная версия (**OTDKC**);
- **GLL**[5] (запускалась вариация с хранением графа в оперативной памяти)
- **Graspan**[6] (запускался только для анализа псевдонимов, так как эта реализация не поддерживает грамматики с большим количеством нетерминалов)
- **Gigascale**[7] (запускался только для Points-to анализа, учитывающего поля, так как эта реализация заточена под конкретную грамматику)

[5] https://github.com/FormalLanguageConstrainedPathQuerying/GLL4Graph
[6] https://github.com/Graspan/Graspan-C
[7] https://bitbucket.org/jensdietrich/gigascale-pointsto-oopsla2015

Время работы: Анализ псевдонимов



Время работы: Points-to анализ, учитывающий поля



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

Ответы на вопросы и замечания

- Корректность
  - Анализ псевдонимов
    - ★ Program analysis via graph reachability[8] (Thomas Reps)
    - ★ Demand-Driven Alias Analysis for C[9] (Xin Zheng and Radu Rugina)
  - Points-to анализ, учитывающий поля
    - ★ Refinement-based context-sensitive points-to analysis for Java[10] (M. Sridharan and R. Bodik)
- Для графа $\mathcal{G} = \langle V, E, l \rangle$ и грамматики $G = \langle \Sigma, N, P, S \rangle$ вычислительная сложность
  - Матричный алгоритм — $O(|N||P||V|^5)$
  - Тензорный алгоритм — $O(|P|^3|V|^3/\log(|P||V|))$

[8] https://dl.acm.org/doi/10.5555/271338.271343
[9] https://dl.acm.org/doi/10.1145/1328897.1328464
[10] https://dl.acm.org/doi/10.1145/1133981.1134027