Санкт-Петербургский государственный университет

Кафедра Системного программирования

Гумин Егор Дмитриевич

Использование символьных конечных преобразователей для лексического анализа динамически формируемого кода

Курсовая работа

Научный руководитель:
ст. пр. Григорьев С. В.

Санкт-Петербург
2016

Оглавление

Введение

Существует подход к написанию программного кода, при котором код на одном языке программирования формируется программой на другом языке с помощью строковых операций, циклов и условных операторов. Такой код называется динамически формируемым. В качестве примера можно рассмотреть генерацию html-страниц в языке Javascript или формирования SQL-запросов в C# с помощью строковых операций. На листинге 1 представлен пример динамически формируемого кода.

```
1 private void Go(int cond){
2   string columnName = cond > 3 ? ''X'' : (cond < 0 ? ''Y'' : ''Z'');
3   string queryString =
4       ''SELECT name'' + columnName + '' FROM table'';
5   Program.ExecuteImmediate(queryString);}
```

Листинг 1: Пример динамически формируемого кода

Этот подход был достаточно распространен, поэтому существует большое множество программ, использующих его. Кроме того, иногда возникает необходимость написания такого кода (например, в случаях, когда ограничения по производительности не позволяют использовать ORM). Статический анализ такого кода, например, подсветка синтаксиса и диагностика ошибок, позволил бы существенно упростить его написание и поддержку.

Существует ряд инструментов [6, 2, 7] (подробно рассмотрены в работе [10]), позволяющих проводить статический анализ динамически формируемого кода. Однако эти инструменты обладают такими недостатками, как слабая расширяемость, ограниченная функциональность, применимость только к конкретным языкам. Для решения этих проблем [11] в рамках проекта YaccConstructor [14, 9], который посвящен исследованиям в области лексического и синтаксического анализа, реализована платформа для работы со встроенными языками. Модульность платформы позволяет легко добавлять дополнительную функци-

ональность и поддержку новых языков. статический анализ динамического кода производится в несколько шагов:

1. построение регулярной аппроксимации сверху множества значений динамически формируемого выражения;
2. лексический анализ;
3. синтаксический анализ;
4. семантический анализ.

Но механизм лексического анализа [10] в проекте YaccConstructor, использующий композицию конечных преобразователей, обладает недостаточной производительностью. Вероятная причина этой проблемы - разрастание конечных преобразователей на больших алфавитах, так как обработка большого количества дуг занимает много времени. Возможное решение - применение символьных конечных преобразователей [1], с помощью которых можно представить структуры данных более компактно. На данный момент единственной библиотекой для работы с символьными конечными преобразователями в рамках платформы .NET является библиотека Microsoft.Automata [13], которая активно поддерживается сообществом Microsoft Research, именно поэтому она и является предметом изучения. В рамках данной работы исследован вопрос о возможности увеличения производительности лексического анализа с помощью использования символьных конечных преобразователей из библиотеки Microsoft.Automata.

1. Постановка задачи

Целью данной работы является исследование применимости символьных конечных преобразователей для лексического анализа. Для ее достижения поставлены следующие задачи.

- Изучить возможности библиотеки Microsoft.Automata.
- Провести сравнение производительности алгоритма операции композиции над символьными конечными преобразователями в библиотеке Microsoft.Automata с производительностью операции композиции над конечными преобразователями в исследовательском проекте YaccConstructor.
- На основании полученных результатов сделать выводы о применимости библиотеки Microsoft.Automata для лексического анализа в проекте YaccConstructor.

2. Обзор

В данной главе определены основные понятия и проведен обзор основных применяемых формализмов, инструментов и технологий.

2.1. Конечные преобразователи

Конечный преобразователь [5] (Finite State Transducer) — формализм, напоминающий конечный автомат, который при переходе из состояния в состояние пишет символ в выходной поток. Конечный преобразователь можно задать шестеркой элементов: $\langle Q, \Sigma, \Delta, q_0, F, E \rangle$, где

- Q — множество состояний,
- Σ — входной алфавит,
- Δ — выходной алфавит,
- $q_0 \in Q$ — начальное состояние,
- $F \subseteq Q$ — набор конечных состояний,
- $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times Q$ — набор переходов.

Важнейшей для лексического анализа операцией над конечными преобразователями является операция композиции. Результат композиции двух конечных преобразователей — конечный преобразователь, работающий так, будто входной поток первого переправили во входной поток второго. Формальное определение дано ниже.

Композицией конечных преобразователей $T_1 = \langle Q_1, \Sigma_1, \Delta_1, q_{0_1}, F_1, E_1 \rangle$ и $T_2 = \langle Q_2, \Sigma_2, \Delta_2, q_{0_2}, F_2, E_2 \rangle$ является конечный преобразователь $T = \langle Q_1 \times Q_2, \Sigma_1, \Delta_2, \langle q_{0_1}, q_{0_2} \rangle, F_1 \times F_2, E \cup E_\varepsilon \cup E_{i,\varepsilon} \cup E_{o,\varepsilon} \rangle$, где

- $E = \{\langle \langle p, q \rangle, a, b, \langle p', q' \rangle \rangle \mid \exists c \in \Delta_1 \cap \Sigma_2 : \langle p, a, c, p' \rangle \in E_1 \wedge \langle q, c, b, q' \rangle \in E_2\}$
- $E_\varepsilon = \{\langle \langle p, q \rangle, a, b, \langle p', q' \rangle \rangle \mid \langle p, a, \varepsilon, p' \rangle \in E_1 \wedge \langle q, \varepsilon, b, q' \rangle \in E_2\}$

- $E_{i,\varepsilon} = \{\langle\langle p, q\rangle, \varepsilon, a, \langle p, q'\rangle\rangle \mid \langle q, \varepsilon, a, q'\rangle \in E_2 \wedge p \in Q_1\}$
- $E_{o,\varepsilon} = \{\langle\langle p, q\rangle, a, \varepsilon, \langle p', q\rangle\rangle \mid \langle p, a, \varepsilon, p'\rangle \in E_1 \wedge q \in Q_2\}$.

На рисунках 1- 3 представлен пример композиции конечных преобразователей. Результатом композиции конечного преобразователя T_1 с рисунка 1 с конечным преобразователем T_2 с рисунка 2 будет конечный преобразователь с рисунка 3.

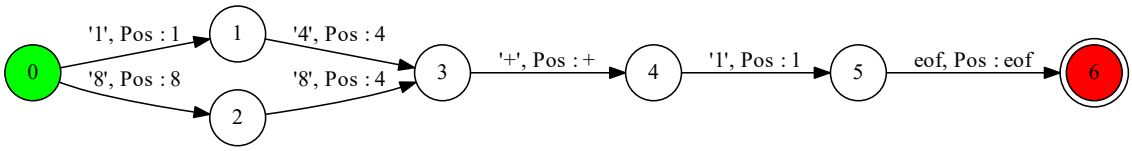

Рис. 1: Конечный преобразователь T_1

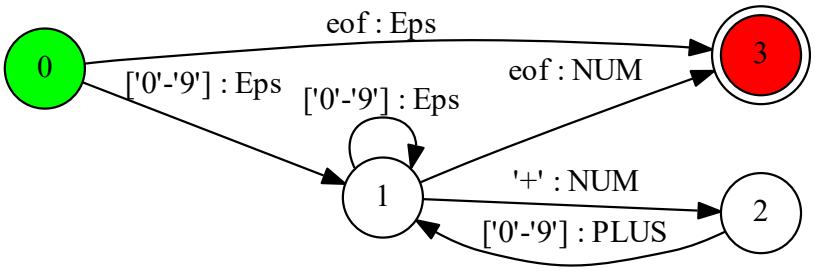

Рис. 2: Конечный преобразователь T_2

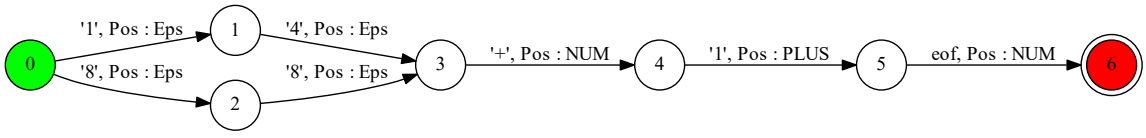

Рис. 3: Результат композиции конечных преобразователей T_1 и T_2

2.2. Символьные конечные преобразователи

При разработке лексических анализаторов часто возникает ситуация, когда из одного состояния в другое необходимы переходы сразу по нескольким символам. Преобразователи часто реализовывают как некоторые графы, и каждый переход представляется как дуга в этом графе. Соответственно, при разработке лексических анализаторов часто приходится добавлять много дублирующих ребер, например, если необходимо, по любой цифре перейти из состояния A в состояние B, нужно добавить 10 дуг (рисунок 4). Чем больше ребер в графе, тем больше расход памяти и ниже производительность.

Для борьбы с этой проблемой был предложен символьный конечный преобразователь (Symbolic Transducer) — конечный преобразователь, каждому переходу которого можно сопоставить не один символ, а формулу (например, регулярное выражение). Таким образом, в приведенном выше примере вместо десяти дуг будет одна дуга, которой соответствует регулярное выражение (рисунок 5), описывающее цифры. Такие преобразователи получаются гораздо более компактными.

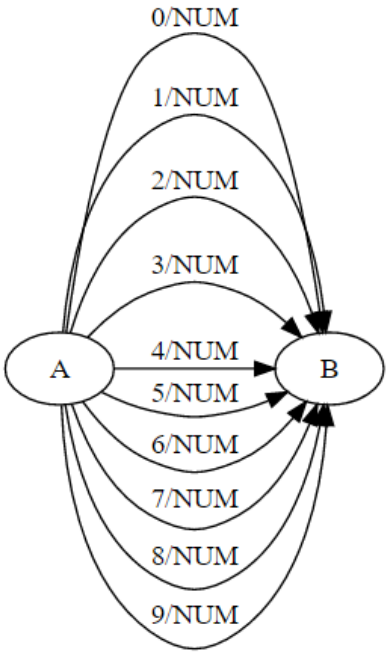

Рис. 4: Пример конечного преобразователя

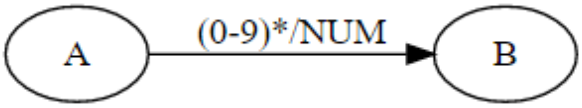

Рис. 5: Символьный конечный преобразователь, эквивалентный конечному преобразователю на рис. 4

Символьный конечный преобразователь можно задать шестеркой элементов: $\langle Q, \Sigma, \Delta, q_0, F, E \rangle$, где

- Q — множество состояний,
- Σ — входной алфавит,
- Δ — выходной алфавит,
- $q_0 \in Q$ — начальное состояние,
- $F \subseteq Q$ — набор конечных состояний,
- $E \subseteq Q \times (R(\Sigma) \cup \{\varepsilon\}) \times (\Delta \cup \{\varepsilon\}) \times Q$ — набор переходов, $R(X)$ — некоторая формула (в нашем случае — регулярное выражение) над множеством X.

2.3. Лексический анализ

Основной задачей лексического анализа является выделение лексем во входном потоке с сохранением их привязки к исходному коду. При лексическом анализе статического кода поток символов линеен, но в случае анализа встроенного кода он линейным не является. Поэтому применяется следующий подход.

1. В некоторой точке программы по множеству значений строкового выражения строится конечный автомат M, аппроксимирующий его сверху.
2. По конечному автомату M строится конечный преобразователь.

3. Производится композиция конечного преобразователя M с конечным преобразователем, являющимся лексическим анализатором языка, на котором написано строковое выражение.

4. По результирующему преобразователю строится конечный автомат над лексемами, который и является токенизированным входом.

Сложность операции композиции зависит от количества ребер, символьные преобразователи обычно содержат меньшее их количество, чем эквивалентные конечные преобразователи, что позволяет увеличить производительность композиции.

2.4. Библиотека Microsoft.Automata

Microsoft.Automata — .NET-библиотека, содержащая реализации конечных автоматов, конечных преобразователей, операции над ними и средства их анализа. Библиотека состоит из следующих модулей.

- Automata — основной модуль с реализацией автоматов и преобразователей, содержащий в себе такие элементы, как:
  - синтаксические анализаторы грамматик для построения по ним автоматов;
  - синтаксические анализаторы регулярных выражений, которые используются при построении преобразователей;
  - различные структуры данных, например, BDD;
  - символьные автоматы и операции над ними;
  - символьные преобразователи и операции над ними Z3 [8].
- Automata.Z3 — модуль, обеспечивающий интеграцию реализованных формализмов с SMT-решателем Z3 [15, 3].

- Bek — модуль, позволяющий работать с преобразователями и автоматами, используя язык Bek Z3 [12, 4]. Язык Bek - предметно-ориентированный язык для написания и анализа конечных преобразователей, работающих со строками, который является аналогом регулярных выражений для автоматов. Программы на Bek позволяют ответить на вопросы «Выводят ли две программы одну и ту же строку», «Может ли эта программа вывести некоторую строку?», «Что будет, если выполнить композицию программ?».

  На листинге 2 приведен пример программы на языке Bek. Программа проходит по строке, используя переменную c, чтобы итерироваться по символам и булеву переменную b, чтобы отслеживать, был ли предыдущий символ экранирующим и добавляет экранирующий символ перед одинарными или двойными кавычками, если он пропущен.

```
1  program sanitize(t);
2      string s;
3      s := iter(c in t){b := false;}{
4              case (!(b) && ((c == '\'') || (c == '\''))) :
5                  b := false;
6                  yield ('\\');
7                  yield (c);
8              case (c == '\\') :
9                  b := !(b);
10                 yield (c);
11             case (true) :
12                 b := false;
13                 yield (c);
14             };
15     return s;
```

Листинг 2: Пример программы на языке Bek

3. Окружение для тестирования и экспериментальное исследование

В данной главе описывается реализация окружения для тестирования производительности операции композиции на символьных конечных преобразователях в библиотеке Microsoft.Automata и конечных преобразователях в YaccConstructor. Также приведено описание процесса тестирования и сделаны выводы о применимости символьных преобразователей из библиотеки Microsoft.Automata в исследовательском проекте YaccConstructor.

При изучении возможностей библиотеки, проведенном в рамках обзора, было установлено, что полная интеграция проекта YaccConstructor и библиотеки Microsoft.Automata затруднительна. По этой причине для оценки производительности библиотеки Microsoft.Automata было необходимо создать окружение для тестирования, обеспечивающее возможность генерации тестов по одинаковым данным для разнородных сред: инструмента YaccConstructor и библиотеки Microsoft.Automata.

3.1. Окружение для тестирования

Предложенная архитектура среды для тестирования изображена на рис. 6, цветом обозначены модули, добавленные или модифицированные в рамках данной работы.

Окружение состоит из:

- генератора конечных автоматов для создания на их основе эквивалентных преобразователей для инструмента YaccConstructor и библиотеки Microsoft.Automata;
- генератора конечных преобразователей;
- генератора символьных конечных преобразователей;
- лексического анализатора в форме конечного преобразователя;

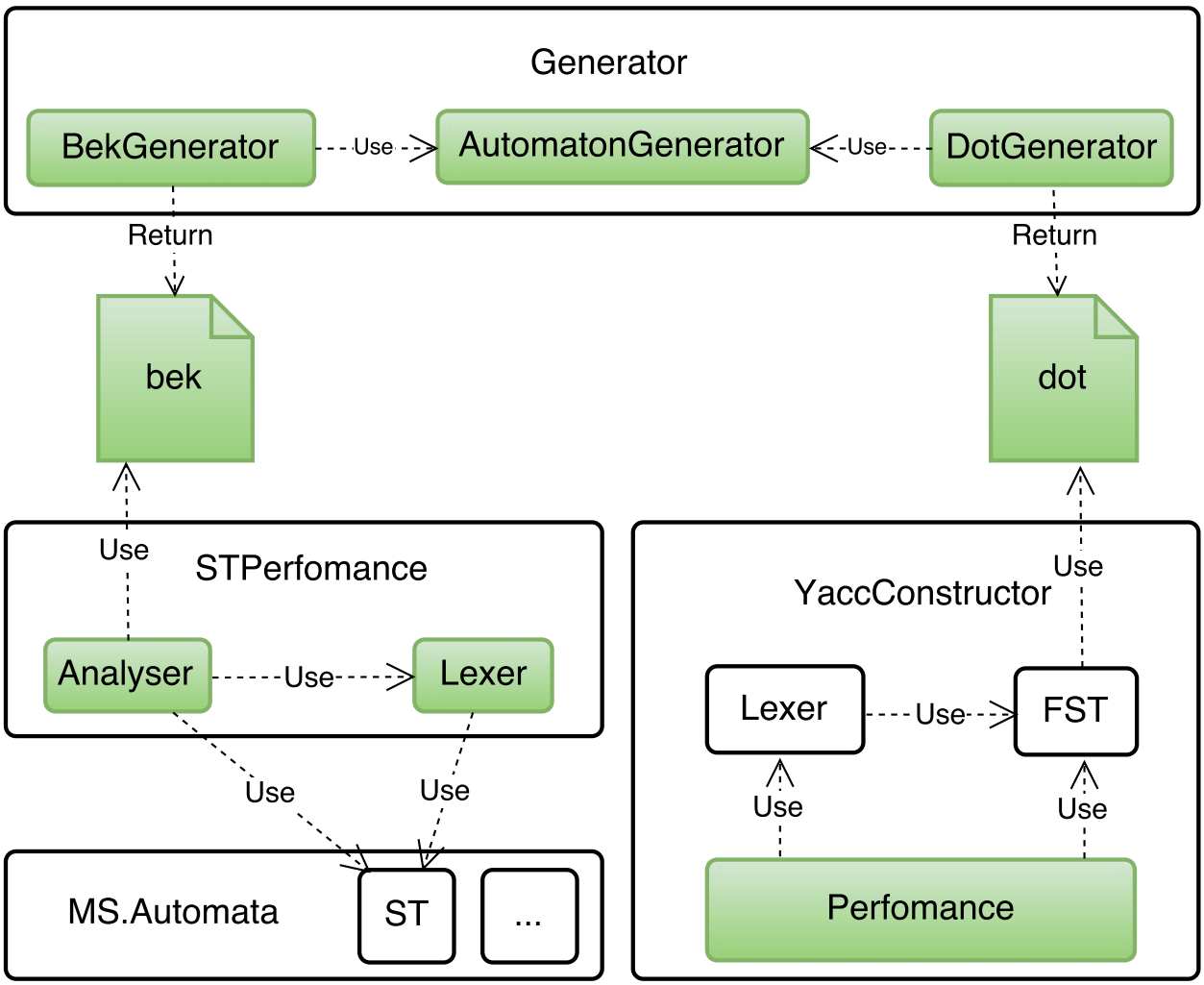

Рис. 6: Архитектура среды для тестирования

- лексического анализатора в форме символьного конечного преобразователя;
- модуля для анализа производительности операции композиции в проекте YaccConstructor;
- модуля для анализа производительности операции композиции в библиотеке Microsoft.Automata.

Подробное описание компонентов приведено далее.

3.2. Лексический анализатор

Для сравнения времени работы операции композиции на символьных конечных преобразователях и конечных преобразователях было решено реализовать лексический анализатор арифметических выражений, аналогичный анализатору, который уже был интегрирован в проект YaccConstructor, так как язык арифметических выражений является достаточно показательным примером и включается практически в любой язык программирования. Лексический анализатор может быть представлен в виде символьного конечного преобразователя и используется для анализа производительности библиотеки Microsoft.Automata. Реализация данного лексического анализатора выполнена на языке Bek.

3.3. Генераторы преобразователей

Генератор конечных автоматов (обозначен как AutomatonGenerator на рисунке 6) генерирует случайные конечные автоматы с указанным количеством дуг и вершин. Результат его работы — файл с промежуточным представлением конечного автомата, который далее интерпретируется генераторами конечных преобразователей (обозначены как BekGenerator и DotGenerator на рисунке 6), которые на его основе строят программы, представляющие эквивалентные преобразователи, записанные на языках Bek и DOT соответственно.

3.4. Тестирование производительности композиции в проекте YaccConstructor на конечных преобразователях

За тестирование производительности операции композиции на конечных преобразователях в проекте YaccConstructor отвечает модуль AbstractLexer.Interpreter.Perfomance (обозначен как Perfomance на рисунке 6). По dot-файлам лексического анализатора арифметических выражений и dot-файлам, сгенерированными модулем DotGenerator строятся конечные преобразователи. Затем производится композиция пре-

образователя, построенного по лексическому анализатору с каждым из тестовых преобразователей. Тест многократно повторяется и для каждого теста берется среднее значение затраченного времени.

3.5. Тестирование производительности композиции в библиотеке Microsoft.Automata на символьных конечных преобразователях

За тестирование производительности операции композиции на символьных конечных преобразователях в библиотеке Microsoft.Automata отвечает модуль PerfomanceTest (обозначен как STPerfomance на рисунке 6). По bek-файлам лексического анализатора арифметических выражений и bek-файлам, сгенерированными модулем BekGenerator строятся символьные конечные преобразователи. Затем производится композиция символьного преобразователя, построенного по лексическому анализатору с каждым из тестовых преобразователей. Тест многократно повторяется и для каждого теста берется среднее значение затраченного времени.

3.6. Экспериментальное исследование

С помощью описанного окружения был поставлен следующий эксперимент.

1. Сгенерированы пять конечных автоматов, характеристики которых указаны в таблице 1.
2. По автоматам сгенерированы программы на языке DOT.
3. По автоматам сгенерированы программы на языке Bek.
4. Проведена композиция сгенерированных конечных преобразователей из dot-файлов с конечным преобразователем, представляющим лексический анализатор языка арифметических выражений

из проекта YaccConstructor (операция повторялась 100 раз, в качестве результирующего значения взято среднее).

5. Проведена композиция сгенерированных символьных конечных преобразователей из bek-файлов с символьным конечным преобразователем, представляющим лексический анализатор языка арифметических выражений, аналогичный анализатору из YaccConstructor (операция повторялась 100 раз, в качестве результирующего значения взято среднее).

В результате эксперимента получены данные, которые представлены на рисунке 7.

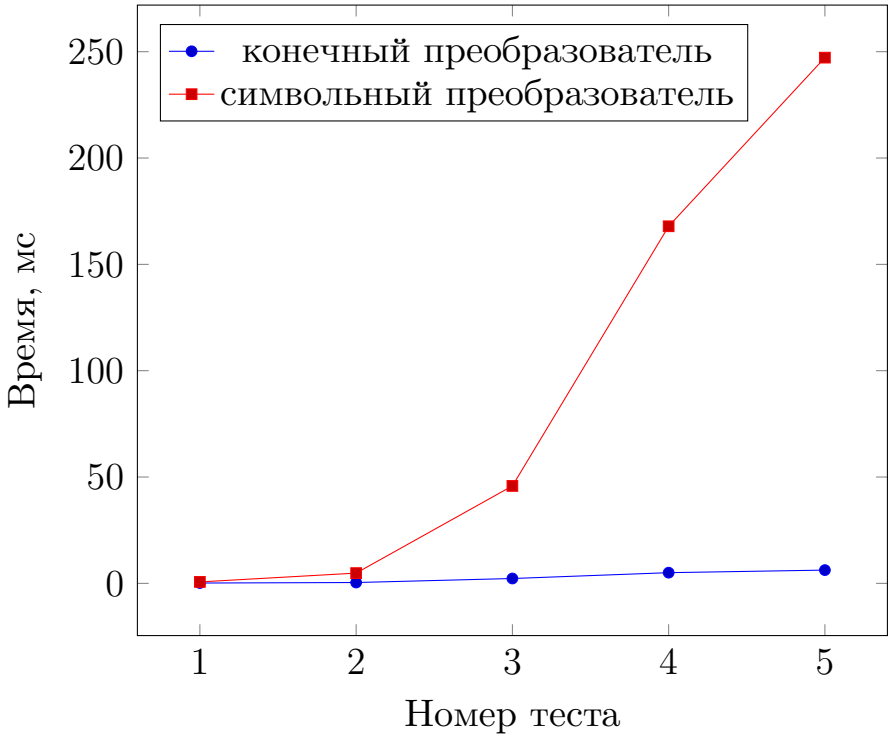

Рис. 7: Результаты измерения скорости работы операции композиции на символьных преобразователях и конечных преобразователях

Тест	Ребра	Вершины
1	1	2
2	6	5
3	25	20
4	50	5
5	60	50

Таблица 1: Размеры сгенерированных автоматов

Эксперимент показал, что операция композиции над символьными конечными преобразователями в библиотеке Microsoft.Automata работает гораздо медленнее, чем операция композиции над конечными преобразователями в проекте YaccConstructor. В связи с этим было принято решение проанализировать производительность самой библиотеки.

Анализ горячих точек показал, что при выполнении задач лексического анализа большую часть времени библиотека тратит на обращения к ядру библиотеки Z3. Поскольку в наших задачах отсутствует необходимость в использовании Z3, было решено в качестве эксперимента отключить некоторые затратные проверки условий, чтобы сократить количество обращений к ядру. Эксперимент показал, что таким образом можно как минимум увеличить производительность операции композиции на 20% на некоторых тестах (время работы операции композиции на модифицированной библиотеке отражено на рисунке 8), что показывает, что адаптация библиотеки к нашей задаче может способствовать увеличению производительности, но этого недостаточно для получения оптимальных результатов.

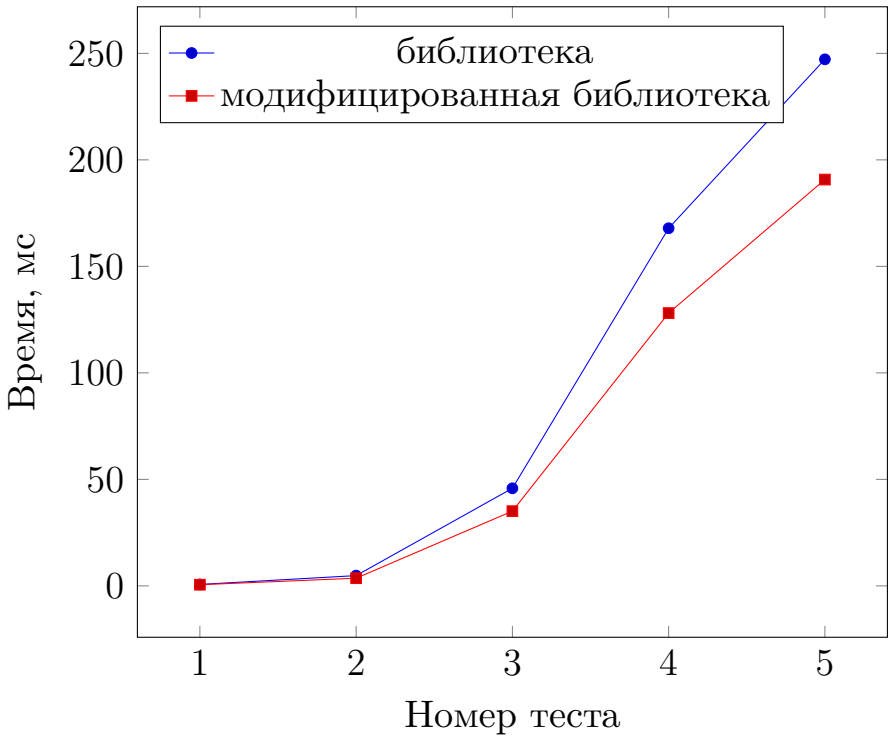

Рис. 8: Результаты измерения скорости работы операции композиции на символьных преобразователях с модифицированной и обычной версиями библиотеки

В результате экспериментов были сделаны выводы, что библиотека Microsoft.Automata, разработанная для решения задач из области формальной верификации, недостаточно производительна для лексического анализа. Большое количество обращений к ядру Z3 перекрывает все достоинства, приобретаемые от использования оптимального формализма и структур данных. Таким образом, применение библиотеки Microsoft.Automata в инструменте YaccConstructor для лексического анализа в настоящий момент неоптимально. Использование же символьных конечный преобразователей для лексического анализа все еще выглядит перспективным и подлежит дальнейшему исследованию.

Заключение

В рамках выполнения данной работы были получены следующие результаты.

- Исследованы возможности библиотеки Microsoft.Automata.
- Проведено сравнение производительности алгоритма операции композиции над символьными конечными преобразователями в библиотеке с производительностью операции композиции над конечными преобразователями в проекте YaccConstructor.
- На основании полученных результатов сделаны выводы о необходимости написания собственной реализации символьного конечного преобразователя.
- Результаты работы представлены на конференции «Современные технологии в теории и практике программирования», тезисы опубликованы в сборнике материалов конференции.

Код генераторов автоматов и преобразователей, а также код, тестирующий производительность символьных преобразователей в библиотеке Microsoft.Automata можно найти на сайте `https://github.com/GuminEgor/Automata`. Код, тестирующий производительность конечных преобразователей в проекте YaccConstructor можно найти на сайте `https://github.com/GuminEgor/YaccConstructor`. В указанных репозиториях автор принимал участие под учетной записью GuminEgor.

В дальнейшем необходима реализация собственной версии символьного конечного преобразователя, адаптированного под задачи лексического анализа, тестирование его производительности и, в случае удовлетворительных результатов, его интеграция в проект YaccConstructor.

Список литературы

[1] Symbolic transducers : Rep. / Technical Report MSR-TR-2011-3, Microsoft Research ; Executor: Nikolaj Bjørner, Margus Veanes : 2011.

[2] Christensen Aske Simon, Møller Anders, I.Schwartzbach Michael. Precise Analysis of String Expressions / Proc. 10th International Static Analysis Symposium (SAS). — Springer-Verlag: Berlin, 2003. — June. — P. 1–18.

[3] De Moura Leonardo, Bjørner Nikolaj. Z3: An efficient SMT solver // Tools and Algorithms for the Construction and Analysis of Systems. — Springer, 2008. — P. 337–340.

[4] Fast and precise sanitizer analysis with BEK / Pieter Hooimeijer, Benjamin Livshits, David Molnar et al. // Proceedings of the 20th USENIX conference on Security / USENIX Association. — 2011. — P. 1–1.

[5] Hanneforth Thomas. Finite-state Machines: Theory and Applications Unweighted Finite-state Automata / Institut fur Linguistik Universitat Potsdam.

[6] An Interactive Tool for Analyzing Embedded SQL Queries / Aivar Annamaa, Andrey Breslav, Jevgeni Kabanov, Varmo Vene / Programming Languages and Systems. — Springer: Berlin, 2010. — P. 131–138.

[7] Minamide Yasuhiko. Static approximation of dynamically generated web pages / In Proceedings of the 14th International Conference on World Wide Web, WWW '05. — ACM, 2005. — P. 432–441.

[8] Symbolic finite state transducers: Algorithms and applications / Margus Veanes, Pieter Hooimeijer, Benjamin Livshits et al. // ACM SIGPLAN Notices. — 2012. — Vol. 47, no. 1. — P. 137–150.

[9] Кириленко Я.А, Григорьев С.В., Д.А. Авдюхин. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. — 2013. — no. 174. — P. 94 – 98. — URL: http://ntv.spbstu.ru/telecom/article/T3.174.2013_11/.

[10] Полубелова Марина Игоревна. Лексический анализ динамически формируемых строковых выражений.

[11] С. Григорьев. Синтаксический анализ динамически формируемых программ : Ph. D. thesis / Григорьев С. ; Санкт-Петербургский государственный университет. — 2016.

[12] Сайт проекта Bek. — URL: http://research.microsoft.com/en-us/projects/bek/.

[13] Сайт проекта Mircosoft.Automata. — URL: http://research.microsoft.com/en-us/projects/automata/.

[14] Сайт проекта YaccConstructor. — URL: https://github.com/YaccConstructor/.

[15] Сайт проекта Z3. — URL: https://github.com/Z3Prover/z3/.