На правах рукописи

**Григорьев Семён Вячеславович**

# Синтаксический анализ динамически формируемых программ

Специальность 05.13.11 —
«Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Автореферат

диссертации на соискание учёной степени

кандидата физико-математических наук

Санкт-Петербург — 2015

Работа выполнена на кафедре системного программирования математико-механического факультета Санкт-Петербургского государственного университета

Научный руководитель: кандидат физико-математических наук, доцент
**Кознов Дмитрий Владимирович**

Официальные оппоненты: **,**

**Ицыксон Владимир Михайлович,**
кандидат технических наук, доцент
ФГБОУ ВПО “Санкт-Петербургский государственный политехнический университет”

Ведущая организация: Федеральное государственное бюджетное учреждение науки “Институт системного программирования Российской академии наук” (ИСП РАН)

Защита состоится на заседании диссертационного совета Д 212.232.51 на базе Санкт-Петербургского государственного университета по адресу: 198504, Санкт-Петербург, Петродворец, Университетский пр., 28, математико-механический факультет, ауд. 405.

С диссертацией можно ознакомиться в Научной библиотеке Санкт-Петербургского государственного университета по адресу: 199034, Санкт-Петербург, Университетская наб., д. 7/9.

Автореферат разослан

Ученый секретарь
диссертационного совета
Д 212.232.51, д.ф.-м.н., профессор Демьянович Юрий Казимирович

# Общая характеристика работы

## Актуальность темы

Взаимодействие различных компонент приложений часто реализуется с помощью встроенных языков, то есть приложение, созданное на одном языке, генерирует код на другом языке и передаёт этот код на выполнение в соответствующее окружение. Примерами могут служить динамические SQL-запросы к базам данных в Java-коде или формирование HTML-страниц в PHP-приложениях. Генерируемая программа собирается таким образом, чтобы в момент выполнения результирующий фрагмент кода (строка) представлял собой корректное выражение на соответствующем языке. Такой подход весьма гибок, так как позволяет использовать для формирования таких фрагментов кода различные строковые операции (replace, substring и т.д.) и комбинировать код из различных источников (например, учитывать текстовый ввод пользователя, что часто используется для задания фильтров при конструировании SQL-запросов). Необходимо отметить, что такой подход не имеет дополнительных накладных расходов, присущих, например, ORM-технологиям, и это позволяет достигать высокую производительность.

Однако динамическое формирование кода программ часто происходит посредством конкатенации в циклах, ветках условных операторов или рекурсивных процедурах, что приводит к множеству возможных вариантов значений для каждого выражения в момент выполнения. При этом фрагменты кода на встроенных языках воспринимаются компилятором исходного языка как простые строки, не подлежащие дополнительному анализу, что приводит к высокой вероятности возникновения ошибок во время выполнения программы. В худшем случае такие ошибки не приведут к прекращению работы приложения, что явно указало бы на проблему, но целостность данных при этом может оказаться нарушенной. Более того, например, при наличии в коде приложения встроенных SQL-запросов нельзя, не проанализировав все динамически формируемые выражения, точно ответить на вопрос о том, с какими элементами базы данных не взаимодействует система, и удалить их. При переносе такой системы на другую СУБД необходимо гарантировать, что для всех динамически формируемых выражений значение в момент выполнения будет корректным кодом на языке новой СУБД. Кроме того, при создании приложений распространённой практикой является использование интегрированных сред разработки, производящих подсветку синтаксиса и автодополнение, сигнализирующих о синтаксических ошибках, предоставляющих возможность рефакторинга. Всё это значительно упрощает процесс разработки и отладки приложений и полезно не только для основного языка, но и для встроенных языков. Для решения данных задач

необходимы инструменты, проводящие статический анализ динамически формируемых программ.

### Степень разработанности темы

Анализу динамически формируемых строковых выражений посвящены работы таких зарубежных учёных как Кюнг-Гу Дох (Kyung-Goo Doh), Ясухико Минамиде (Minamide Yasuhiko), Андерс Мёллер (Anders Møller), а также отечественных учёных — А.А. Бреслава и других. Вопросы проверки корректности динамически формируемых выражений достаточно подробно изучены, однако актуальным представляется проведение более сложных видов статического анализа, требующих построения структурного представления динамически формируемого кода. То есть требуется механизм синтаксического анализа динамически формируемых выражений, позволяющий строить лес вывода для всех возможных значений соответствующего выражения.

Методы обобщённого синтаксического анализа, лежащие в основе данной работы, изложены в трудах таких учёных как Масару Томита (Masaru Tomita), Элизабет Скотт (Elizabeth Scott) и Адриан Джонстон (Adrian Johnstone) из университета Royal Holloway (Великобритания), Ян Рекерс (Jan Rekers, University of Amsterdam), Элко Виссер (Eelco Visser) и других.

Так же важной является разработка компонентов, упрощающих создание новых инструментов для решения конкретных задач. Данных подход хорошо исследован в области разработки компиляторов, где широкое распространение получили генераторы анализаторов и пакеты стандартных библиотек.

В работах отечественных учёных М.Д. Шапот и Э.В. Попова, а так же зарубежных учёных Антони Клеви (Anthony Cleve), Жан-Люк Эно (Jean-Luc Hainaut), Йост Виссер (Joost Visser) и других рассматриваются различные аспекты реинжиниринга систем, использующих встроенные SQL-запросы, однако не формулируется общего метода реинжиниринга таких систем. Разработка такого метода является актуальной задачей.

Существует также ряд инструментов для работы с динамически формируемыми выражениями: Alvor и IntelliLang, предоставляющие поддержку встроенных языков в интегрированных средах разработки, JSA и PHPSA, позволяющие искать ошибки в выражениях на встроенных языках, SQLWays, поддерживающий трансформацию выражений на встроенных языках, SAFELI — инструмент статического анализа, предназначенный для определения возможности SQL-инъекций в Web-приложениях и некоторые другие. Однако, эти инструменты имеют существенные ограничения по функциональности: не поддерживают часто встречающиеся на практике сложные способы формирования строковых

выражений (ветвления, циклы), решают только одну узкую задачу (например, проверку корректности, поиск уязвимых конструкций) и т.д.

Таким образом, актуальной является задача дальнейшего исследования синтаксического анализа динамически формируемых строковых выражений, а так же возможностей построения структурного представления динамически формируемого кода. Кроме этого важным является решение вопросов практического применения средств анализа динамически формируемого кода: упрощение разработки инструментов анализа и создание методов их применения в реинжиниринге программного обеспечения.

## Цель диссертационной работы

Целью данной работы является создание подхода к статическому синтаксическому анализу динамически формируемых выражений, позволяющего обрабатывать произвольную регулярную аппроксимацию без потери точности, а также разработать технологию для создания инструментов статического анализа динамически формируемых программ, реализующей данный алгоритм.

## Результаты, выносимые на защиту

1. Разработан алгоритм синтаксического анализа динамически формируемых выражений, позволяющий обрабатывать произвольную регулярную аппроксимацию множества значений выражения в точке выполнения, реализующий эффективное управление стеком и гарантирующий конечность представления леса вывода. Доказана завершаемость и корректность предложенного алгоритма при анализе регулярной аппроксимации, представимой в виде произвольного конечного автомата без $\varepsilon$-переходов.
2. Создана архитектура инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений.
3. Разработан метод анализа и обработки встроенного программного кода в проектах по реинжинирингу информационных систем.

## Методы исследования

В работе используются методы лексического и синтаксического анализов. В частности, применяется алгоритм обобщённого восходящего синтаксического анализа RNGLR, созданный Элизабет Скотт (Elizabeth Scott) и Адриан Джонстон (Adrian Johnstone) из университета Royal Holloway (Великобритания). Для компактного хранения леса вывода использовалась структура данных Shared Packed

Parse Forest (SPPF), которую предложил Ян Рекерс (Jan Rekers, University of Amsterdam).

Доказательство завершаемости и корректности предложенного алгоритма проводилось с применением теории формальных языков, теории графов и теории сложности алгоритмов. Приближение множества значений динамически формируемого выражения строилось в виде регулярного множества, описываемого с помощью конечного автомата.

## Научная новизна

На текущий момент существует несколько подходов к анализу динамически формируемых строковых выражений. Некоторые из них, такие как JSA, предназначены только для проверки корректности выражений, основанной на решении задачи о включении одного языка в другой. Выполнение более сложных видов анализа, трансформаций или построения леса разбора не предполагается. В работах А. Бреслава и Кюнг-Гу Дох (Kyung-Goo Doh) рассматривается применение механизмов синтаксического анализа для работы с динамически формируемыми выражениями, однако не решается вопрос эффективного представления результатов разбора. Предложенный в диссертации алгоритм предназначен для синтаксического анализа динамически формируемых выражений и построения компактной структуры данных, содержащей деревья вывода для всех корректных значений выражения. Это позволяет как проверять корректность анализируемых выражений, так и проводить более сложные виды анализа, используя деревья вывода, хранящиеся в построенной структуре данных.

Большинство существовавших готовых инструментов для анализа динамически формируемых строковых выражений (JSA, PHPSA, Alvor и т.д.), как правило, предназначены для решения конкретных задач для определённых языков. Решение новых задач или поддержка других языков с помощью этих инструментов зачастую затруднены ввиду ограничений, накладываемых архитектурой и возможностями используемого алгоритма анализа. В рамках работы предложена архитектура инструментального средства, включающего предложенный алгоритм и позволяющего упростить создание новых инструментов для анализа динамически формируемых выражений на любых языках программирования (различные диалекты SQL, HTML, JSON и т.д.).

## Практическая ценность

На основе полученных в работе научных результатов был разработан инструментарий (Software Development Kit, SDK), предназначенный для создания

средств статического анализа динамически формируемых выражений. В данный инструментарий входят следующие компоненты: генератор лексических анализаторов, генератор синтаксических анализаторов, библиотеки времени выполнения, реализующие соответствующие алгоритмы анализа, набор интерфейсов и вспомогательных функций для реализации конечного инструмента. Набор генераторов позволяет по описанию лексики и синтаксиса языка строить синтаксический и лексический анализатор, обрабатывающий аппроксимацию множества значений динамически формируемого выражения на соответствующем языке, представленную в виде произвольного конечного автомата $\varepsilon$-переходов.

Данный инструментарий позволяет автоматизировать создание лексических и синтаксических анализаторов при разработке программных средств, использующих регулярную аппроксимацию для приближения множества значений динамически формируемых выражений. Инструментарий может использоваться для решения задач реижиниринга — изучения и инвентаризации систем, поиска ошибок в исходном коде, автоматизации трансформации выражений на встроенных языках. Также данный инструментарий может использоваться при реализации поддержки встроенных языков в интегрированных средах разработки.

Разработанный метод реинжиниринга встроенного программного кода основан на использовании инструментария в качестве генератора для создания лексического и синтаксического анализатора для динамически формируемых выражений по соответствующим спецификациям. В случае динамического SQL могут быть переиспользованы ранее разработанные спецификации. Построение регулярной аппроксимации выделяется в отдельный шаг и производится с помощью анализов, реализованных для обработки основного кода. После завершения синтаксического разбора, анализ леса проводится в основном с помощью тех же методов, что и анализ основного кода, что достигается за счёт идентичности структур деревьев. Данная методика может быть переиспользована для работы с произвольными встроенными текстовыми языками.

## Реализация и внедрение результатов исследования

С использованием разработанного инструментария было реализовано расширение к инструменту ReSharper (компания ООО “ИнтеллиДжей Лабс”, Россия), предоставляющее поддержку встроенного T-SQL в проектах на языке программирования C# в среде разработки Microsoft Visual Studio. Была реализована следующая функциональность: статическая проверка корректности выражений и подсветка ошибок, подсветка синтаксиса и подсветка парных элементов. Исходный код разработанного инструментария и расширения досту-

пен в репозитории по адресу `https://github.com/YaccConstructor/YaccConstructor`.

Так же было выполнено внедрение результатов работы в промышленный проект по переносу хранимого SQL-кода с MS-SQL Server 2005 на Oracle 11gR2 (ЗАО “Ланит-Терком”, Россия). Исходная система состояла из 850 хранимых процедур и содержала около 3000 динамических запросов на 2,7 млн. строк хранимого кода. Более половины динамических запросов были сложными и формировались с использованием от 7 до 212 операторов. При этом, среднее количество операторов для формирование запроса — 40.

## Апробация работы

Основные результаты работы были доложены на ряде научно-практических конференциях: SECR-2012, SECR-2013, SECR-2014, TMPA-2014, Parsing@SLE-2013, Рабочий семинар “Наукоемкое программное обеспечение” при конференции PSI-2014. Доклад на SECR-2014 награждён премией Бертрана Мейера за лучшую исследовательскую работу в области программной инженерии. Разработка инструментальных средств на основе предложенного алгоритма была поддержана Фондом содействия развитию малых форм предприятий в технической сфере (программа УМНИК).

## Публикационная активность

Результаты диссертации изложены в 7 научных работах из которых 3 [1–3] опубликованы в журналах из списка ВАК, 3 [4,5,7] индексируются scopus (работа [4] индексируется scopus, работы [5, 7] будут проиндексированы scopus в 2016 году). Работы [1–7] написаны в соавторстве. В [1] С. Григорьеву принадлежит реализация ядра платформы YaccConstructor. В [2, 3] и [5] Григорьеву С. принадлежит постановка задачи, формулирование требований к разрабатываемым инструментальным средствам, работа над текстом. В [4] автору принадлежит идея, описание и реализация анализа встроенных языков на основе RNGLR алгоритма. В [6] Григорьеву С. принадлежит реализация инструментальных средств, проведение замеров, работа над текстом. В работе [7] автору принадлежит алгоритм синтаксического нализа динамически формируемого кода.

**Объем и структура работы**

Диссертация состоит из введения, шести глав, заключения и списка литературы. Полный объем диссертации **125** страницы текста с **26** рисунками и **8** таблицами. Список литературы содержит **87** наименований.

## Содержание работы

Во **введении** обосновывается актуальность исследований, выполненных в рамках данной диссертационной работы, приводится обзор научной литературы по изучаемой проблеме, формулируется цель, ставятся задачи работы, описывается научная новизна и практическая значимость представляемой работы.

В **первой главе** проводится обзор области исследования. Рассматриваются подходы к анализу динамически формируемых строковых выражений и соответствующих инструментов. Описывается алгоритм обобщённого восходящего синтаксического анализа RNGLR, использованный в работе. Также описываются проекты YaccConstructor и ReSharper SDK, использованные в качестве технологий реализации результатов диссертации. На основе проведённого обзора можно сделать следующие выводы.

- Проблема анализа строковых выражений актуальна в нескольких областях: поддержка встроенных языков в интегрированных средах разработки; оценка качества кода, содержащего динамически формируемые строковые выражения; реинжиниринг программного обеспечения.
- Большинство существующих технологических средств поддерживают конкретный внешний и встроенный языки и, как правило, решают только одну задачу (например, поиск ошибок). При этом, они плохо расширяемы, как в смысле поддержки других языков, так и в смысле решения новых задач. Полноценные средства разработки инструментов статического анализа динамически формируемых выражений, упрощающие создание решений для новых языков, отсутствуют.
- Для эффективного решения задач анализа строковых выражений необходимо структурное представление динамически формируемого кода, однако на текущий момент отсутствует законченное решение, позволяющего строить деревья вывода для динамически формируемых выражений.

Во **второй главе** задача синтаксического анализа динамически формируемых выражений формализуется следующим образом: *для данной однозначной контекстно-свободной грамматики* $G = \langle T, N, P, S \rangle$ *и детерминированного конечного автомата без* $\varepsilon$*-переходов* $M = (Q, \Sigma, \delta, q_0, q_f)$ *такого, что* $\Sigma \subseteq T$*,*

*необходимо построить конечную структуру данных $F$, содержащую деревья вывода в $G$ всех цепочек $\omega \in L(M)$, корректных относительно грамматики $G$, и не содержащую других деревьев.* Иными словами, необходимо построить алгоритм $\mathbb{P}$ такой, что

$$(\forall \omega \in L(M))(\omega \in L(G) \Rightarrow (\exists t \in \mathbb{P}(L(M),G))AST(t,\omega,G))$$

$$\wedge(\forall t \in \mathbb{P}(L(M),G))(\exists \omega \in L(M))AST(t,\omega,G).$$

Здесь $AST(t,\omega,G)$ — это предикат, который истинен, если $t$ является деревом вывода $\omega$ в грамматике $G$.

Так как $\mathbb{P}$ игнорирует ошибки, то будем называть его алгоритмом *ослабленного* (relaxed) синтаксического анализа регулярной аппроксимации динамически формируемого выражения.

Далее описывается алгоритм, решающий поставленную задачу: алгоритм синтаксического анализа регулярного множества на основе RNGLR, строящий конечную структуру данных, содержащую деревья вывода для всех цепочек анализируемового множества. Далее доказывается ряд вспомогательных утверждений, необходимых для доказательства основных утверждений о корректности предложенного алгоритма.

ОПРЕДЕЛЕНИЕ 1. *Корректное дерево* — это упорядоченное дерево со следующими свойствами.

1. Корень дерева соответствует стартовому нетерминалу грамматики $G$.
2. Листья соответствуют терминалам грамматики $G$. Упорядоченная последовательность листьев соответствует некоторому пути во входном графе.
3. Внутренние узлы соответствуют нетерминалам грамматики $G$. Дети внутреннего узла (для нетерминала $N$) соответствуют символам правой части некоторой продукции для $N$ в грамматике $G$.

ЛЕММА. *Пусть обрабатывается внутренний граф $\mathcal{G} = (V,E)$. Тогда для каждого ребра GSS $(v_t, v_h)$ такого, что $v_t \in V_t.processed$, $v_h \in V_h.processed$, где $V_t \in V$ и $V_h \in V$, терминалы ассоциированного поддерева соответствуют некоторому пути из вершины $V_h$ в $V_t$ в графе $\mathcal{G}$.*

Сформулированы и доказаны три теоремы о завершаемости и корректности предложенного алгоритма.

ТЕОРЕМА 1. *Алгоритм завершает работу для любых входных данных.*

ТЕОРЕМА 2. *Любое дерево, извлечённое из SPPF, является корректным.*

ТЕОРЕМА 3. *Для каждой строки, соответствующей пути $p$ во входном графе, и выводимой в эталонной грамматике $G$, из SPPF может быть извлечено корректное дерево $t$. То есть $t$ будет являться деревом вывода цепочки, соответствующей пути $p$, в грамматике $G$.*

В **третьей главе** описывается инструментальный пакет YC.SEL.SDK, разработанный автором работы на основе алгоритма, описанного во второй главе. YC.SEL.SDK предназначен для разработки инструментов анализа динамически формируемых выражений, поддерживающих процесс, схема которого представлена на рисунке 1. Описывается архитектура и особенности реализации компонентов отвечающих зя выделение точек интереса, построение регулярной аппроксимации множества значений динамически формируемого выражения, проведения лексического и синтаксического анализа. Также описывается YC.SEL.SDK.ReSharper — обёртка над YC.SEL.SDK, позволяющая создавать расширения к ReSharper для поддержки встроенных языков.

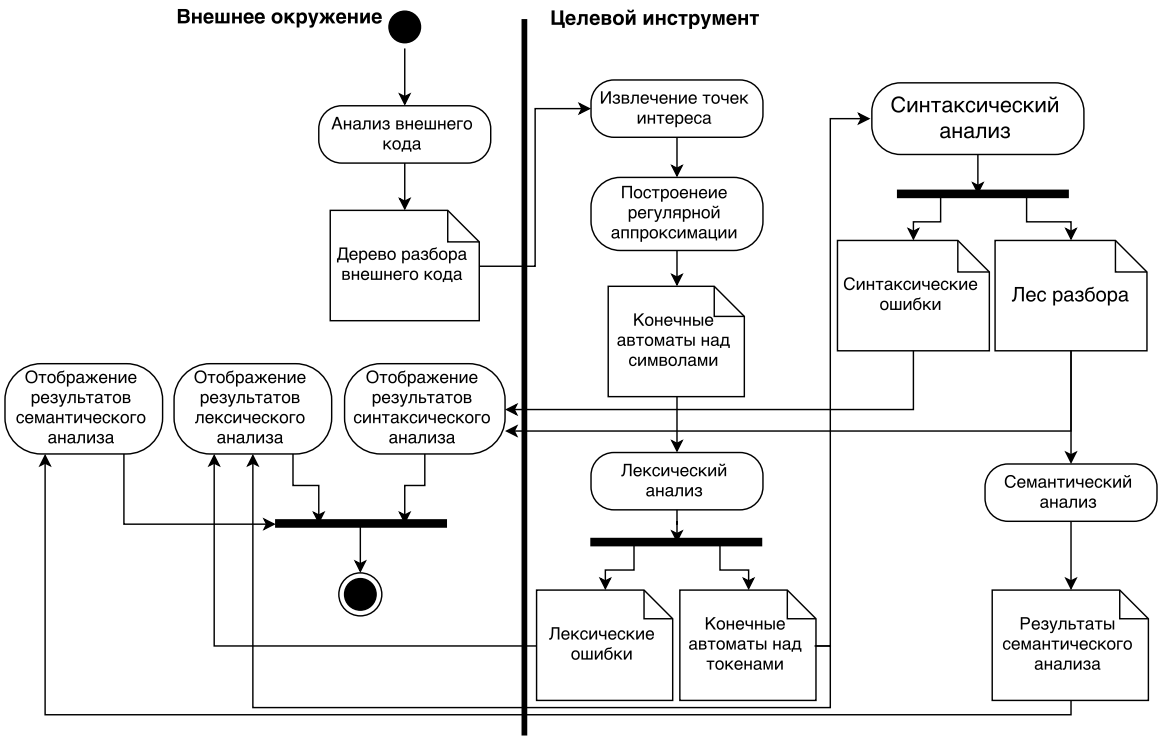


Рис. 1: Пороцесс обработки динамически формируемых выражений

В **четвёртой главе** описывается метод реинжиниринга встроенного программного кода, основные шаги которого представлены на рисунке 2. Данный метод позволяет сформулировать требования к конкретным инструментам обработки встроенного программного окда, необходимым для обработки конкретной информационной системы.

В **пятой главе** приводятся результаты экспериментального исследования YC.SEL.SDK.

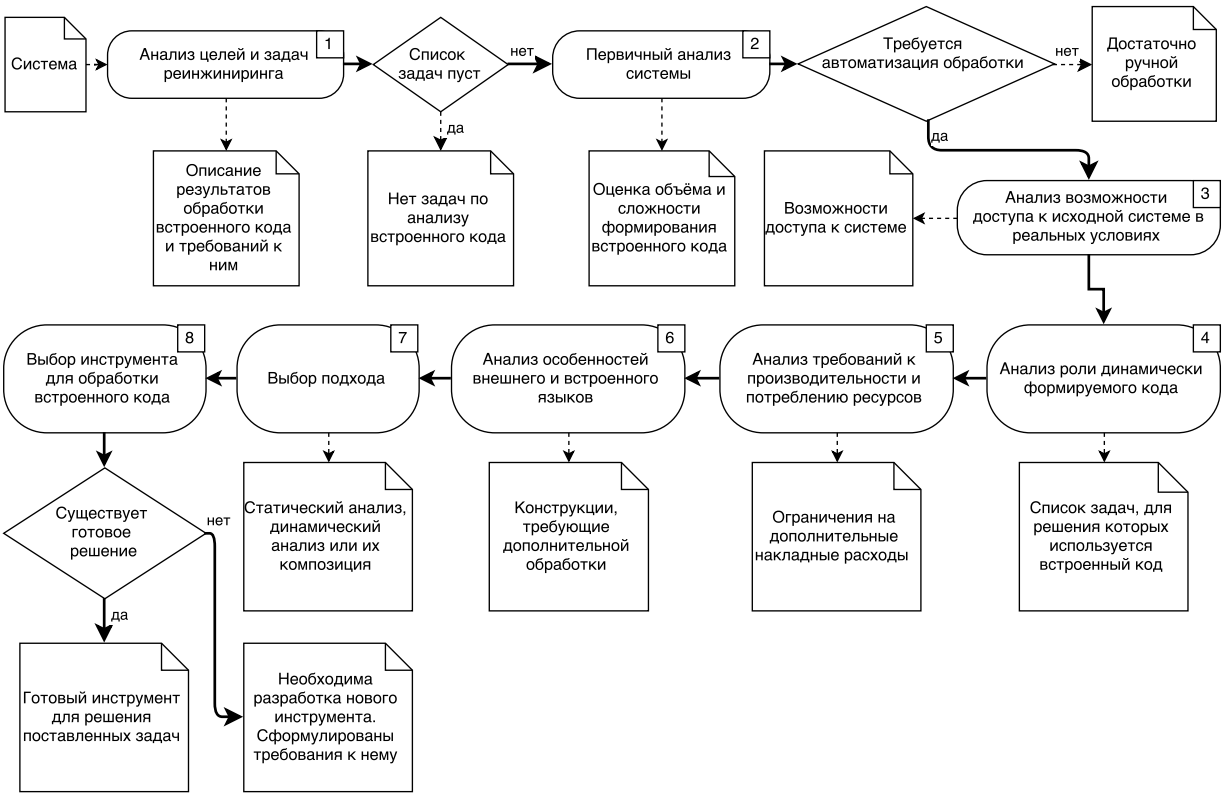


Рис. 2: Основные шаги метода обработки встроенных языков и их результаты

Реализованный инструментарий был апробирован в рамках промышленного проекта по миграции базы данных с MS-SQL Server 2005 на Oracle 11gR2, что позволило оценить предложенную архитектуру и протестировать отдельные компоненты инструментария на реальных данных.

Обрабатываемая система состояла из 850 хранимых процедур и содержала около 2,6 миллионов строк кода. В ней присутствовало 2430 точек выполнения динамических запросов, 75% этих запросов могли принимать более одного значения, при их формировании использовалось от 7 до 212 операторов, среднее количество операторов для формирования запроса равнялось 40.

Алгоритм успешно завершил работу на 2188 входных графах из 2430, аппроксимирующих множества значений запросов. Ручная проверка входных графов, на которых алгоритм завершался с ошибкой, показала, что они действительно не содержали ни одного выражения, корректного в эталонном языке. Причиной этого была либо некорректная работа лексического анализатора, либо наличие в выражениях конструкций, не поддержанных в существующей грамматике. Так как лексический анализатор и грамматика были полностью заимствованы из оригинального проекта, то наличие этих ошибок не является недоработками алгоритма синтаксического анализа. В дальнейшем часть найденых ошибок была исправлена.

Общее время синтаксического анализа составило 27 минут, из них 13 минут было затрачено на разбор графов, не содержащих ни одного корректного выражения, и 4 минуты на обработку графа, прерванную по таймауту. На анализ двух графов было затрачено более 2 минут. Распределение входных графов по промежуткам времени, затраченным на анализ, приведено на рисунке 3.

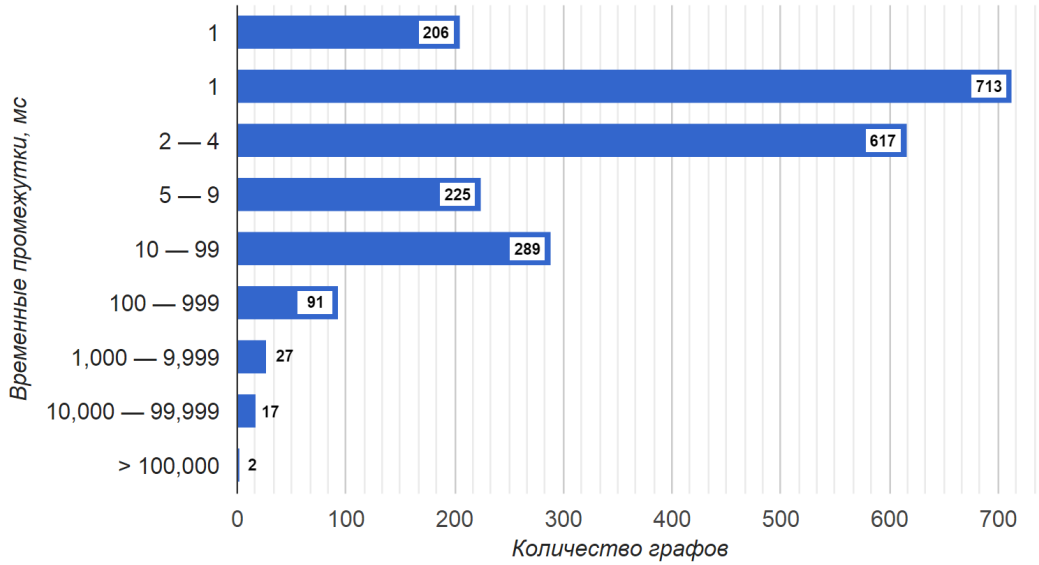


Рис. 3: Распределение запросов по времени анализа

Также было проведено сравнение производительности компоненты синтаксического анализа YC.SEL.SDK с инструментом Alvor. Данный инструмент реализует подход, близкий к представленному в работе: независимые шаги анализа, что позволяет легко выделить синтаксический анализ, который основан на GLR-алгоритме. Существенным отличием является то, что Alvor не строит деревья вывода. Важным для успешного проведения измерений является то, что исходный код Alvor опубликован, что позволяет модифицировать его таким образом, чтобы измерять параметры выполнения конкретных методов.

Так как Alvor не предоставляет платформы для простой реализации поддержки новых языков, то для сравнения было выбрано подмножество языка SQL, общее для Alvor и реализованного в рамках апробации инструмента.

В результате сравнения было установлено, что на линейном входе Alvor показывает лучшую производительность, однако на входных данных, содержащих большое количество ветвлений и циклов производительность Alvor существенно хуже (до 1000 раз).

**Шестая глава** содержит результаты сравнения и соотнесения полученных результатов с существующими аналогами.

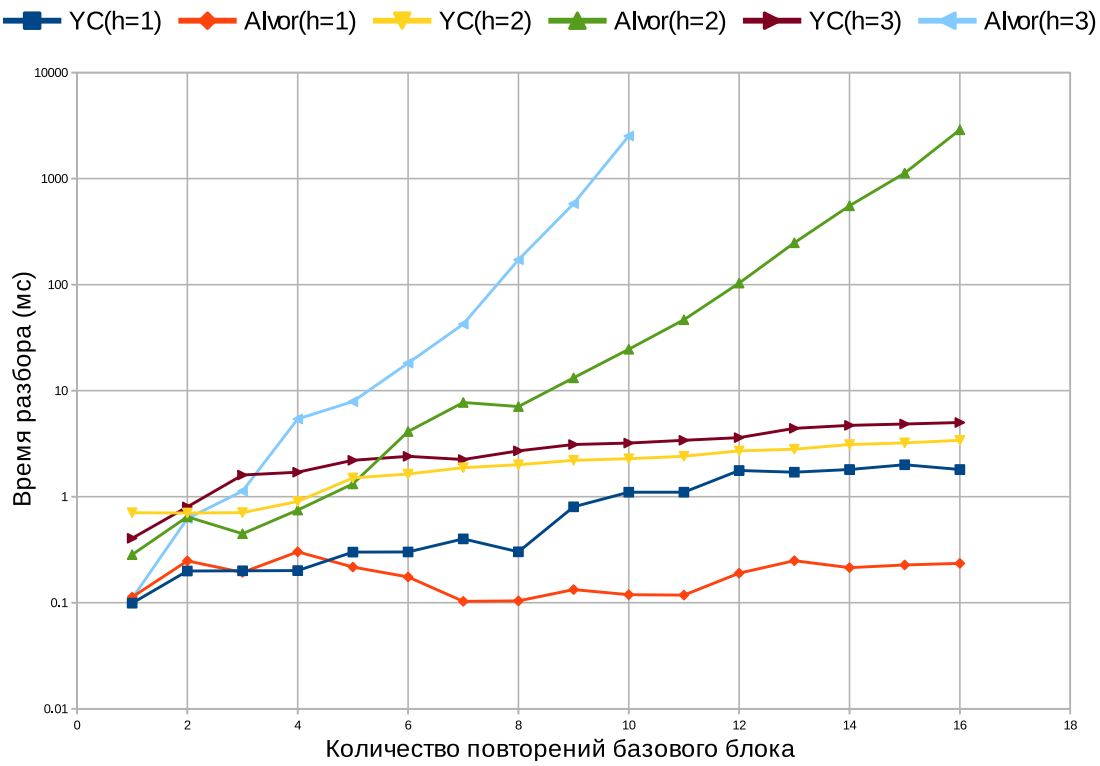


Рис. 4: Сравнение производительности Alvor и синтаксического анализатора на базе YC.SEL.SDK

1. Разработанный алгоритм синтаксического анализа динамически формируемых выражений является единственным алгоритмом, обрабатывающим регулярную аппроксимацию, строящим конечное представление леса разбора.
2. Созданная архитектура позволяет предоставить первую платформу для разработки средств анаиза динамически формируемого кода.
3. Метод реинжиниринга встроенного программного кода сформулирован впервые.

В **заключении** приведены основные результаты работы.

1. Разработан алгоритм синтаксического анализа динамически формируемых выражений, позволяющий обрабатывать произвольную регулярную аппроксимацию множества значений выражения в точке выполнения, реализующий эффективное управление стеком и гарантирующий конечность представления леса вывода. Доказана завершаемость и корректность предложенного алгоритма при анализе регулярной аппроксимации, представимой в виде произвольного конечного автомата без $\varepsilon$-переходов.
2. Создана архитектура инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений.

3. Разработан метод реинжиниринга встроенного программного кода в проектах по реинжинирингу информационных систем. Данный метод применён в проекте компании ЗАО “Ланит-Терком” по переносу информационной системы с MS-SQL Server на Oracle Server, для чего реализованы соответствующие программные компоненты.

Кроме того, был реализован инструментальный пакет для разработки средств статического анализа динамически формируемых выражений. Код опубликован на сервисе GitHub:
`https://github.com/YaccConstructor/YaccConstructor`. На его основе реализован плагин для ReSharper.

## Публикации автора по теме диссертации


1. Григорьев С. В. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем / Кириленко Я. А., Григорьев С. В., Авдюхин Д. А. // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета информатика, телекоммуникации, управление. — 2013. — Т. 3, № 174. — С. 94–98.

2. Григорьев С. В. Инструментальная поддержка встроенных языков в интегрированных средах разработки / Григорьев С. В., Вербицкая Е. А., Полубелова М. И. и др. // Моделирование и анализ информационных систем. — 2014. — Т. 21, № 6. — С. 131–143.

3. Григорьев С. В. Обобщенный табличный LL-анализ / Григорьев С. В., Рагозина А. К. // Системы и средства информатики. — 2015. — Т. 25, № 1. — С. 89–107.

4. Grigorev S. GLR-based abstract parsing / Grigorev S., Kirilenko I. // In Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR ’13). — 2013. — P. 1–9.

5. Grigorev S. String-embedded language support in integrated development environment / Grigorev S., Verbitskaia E., Ivanov A. et al. // In Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR ’14). — 2014. — P. 1–11.

6. Grigorev S. From Abstract Parsing to Abstract Translation / Grigorev S., Kirilenko I. // Proceedings of the Spring/Summer Young Researchers’ Colloquium on Software Engineering. — 2014. — P. 1–5.

7. Grigorev S. Relaxed Parsing of Regular Approximations of String-Embedded Languages / Verbitskaia E., Grigorev S., Avdyukhin D. // Preliminary Proceedings of the PSI 2015: 10th International Andrei Ershov Memorial Conference. — 2015. — P. 1–12.