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 Санкт-Петербургского государственного университета

Научный руководитель:	кандидат физико-математических наук, доцент **Кознов Дмитрий Владимирович**
Официальные оппоненты:	**Марчук Александр Гурьевич,** доктор физико-математических наук, профессор, федеральное государственное бюджетное учреждение науки Институт систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук, директор
	Ицыксон Владимир Михайлович, кандидат технических наук, доцент, федеральное государственное автономное образовательное учреждение высшего образования “Санкт-Петербургский политехнический университет Петра Великого”, исполняющий обязанности заведующего кафедрой
Ведущая организация:	Федеральное государственное бюджетное учреждение науки Институт системного программирования Российской академии наук (ИСП РАН)

Защита состоится _____ г. в _____ часов 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, и на сайте `http://spbu.ru/science/disser/`.

Автореферат разослан _____ 20____ года

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

Общая характеристика работы

Актуальность темы исследования

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 высокой производительности.

Однако динамическое формирование программ часто происходит с помощью операций конкатенации в циклах, условных операторах или рекурсивных процедурах, что приводит к множеству возможных вариантов значений для каждого выражения, что затрудняет их анализ. Поэтому фрагменты кода на встроенных языках воспринимаются компилятором исходного языка как простые строки, 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, выполняющих подсветку синтаксиса и автодополнение, сигнализирующих о синтаксических ошибках, предоставляющих возможность рефакторинга. Эти возможности значительно упрощают процесс разработки и отладки приложений и полезны не только для основного языка, но и для встроенных языков. Таким образом, для решения данных задач необходимы инструменты, проводящие статический анализ динамически формируемых программ.

Степень разработанности темы исследования

Существуют классические исследования, посвященные разработке компиляторов — работы А. Ахо, А. Брукера, С. Джонсона, А.Н. Терехова, В.О. Сафонова, Б.К. Мартыненко и др. Однако содержащиеся там алгоритмы синтаксического анализа, как и методы обобщённого синтаксического анализа, лежащие в основе данной работы и исследованные такими учёными 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 другими, не могут быть применены к решению задачи анализа динамически формируемых программ, поскольку предназначены для обработки входных данных, представимых в виде линейной последовательности символов, а такое представление динамически формируемых программ не возможно.

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) и отечественных учёных А.А. Бреслава, М.Д. Шапот. Хорошо изучены вопросы проверки корректности динамически формируемых выражений и поиска фрагментов кода, уязвимых для SQL-инъекций. Однако данные работы исследуют отдельные аспекты проблемы статического анализа динамически формируемых программ, оставляя в стороне создание готовых алгоритмов (в частности, не строят структурное представление анализируемых программ). В связи с этим возникают проблемы масштабируемости данных результатов, например, создание на их основе более сложных видов статического анализа.

Так же важным является предоставление 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 (работы А. Ахо, А. Брукера, С. Джонсона и др.).

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) рассматривается реинжиниринг информационных систем, использующих встроенные SQL-запросы, однако не формулируется общего метода для решения таких задач. Этот вопрос также не затрагивается в классических работах, посвященных реинжинирингу.

Таким образом, актуальной является задача дальнейшего исследования статического анализа динамически формируемых строковых выражений. 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.

Объект исследования

Объектом исследования являются методы, алгоритмы и программные средства обработки динамически формируемых программ, а также задача реинжиниринга информационных систем.

Цель и задачи диссертационной работы

Целью данной работы является создание комплексного подхода к статическому анализу динамически формируемых программ.

Достижение поставленной цели обеспечивается решением следующих **задач**.

1. Разработать алгоритм синтаксического анализа динамически формируемых программ, позволяющий обрабатывать регулярную аппроксимацию множества значений выражения в точке выполнения, и гарантирующий конечность представления леса вывода.

2. Создать архитектуру инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений.

3. Разработать метод анализа и обработки встроенного программного кода в проектах по реинжинирингу информационных систем.

Цели и задачи диссертационной работы соответствуют области исследований паспорта специальности 05.13.11 “Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей” — пункту 1 (модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования), пункту 2 (языки программирования и системы программирования, семантика программ) и пункту 10 (оценка качества, стандартизация и сопровождение программных систем).

Методология и методы исследования

Методология исследования основана на подходе к спецификации и анализу формальных языков, который начал активно развиваться в 50-х годах 20-го века в связи с изучением естественных языков (работы Н. Хомского). В последствии этот подход получил широкое распространение в областях, связанных с обработкой языков программирования. При этом основными элементами данного подхода являются алфавит и грамматика исследуемого языка, разбиение автоматической обработки языка на выполнение таких шагов, как лексический, синтаксический и семантический анализ. Решаемые в связи с этим задачи связаны с поиском эффективных алгоритмов, выполняющих эти шаги.

В работе 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 используется структура данных Shared Packed Parse Forest (SPPF), которую предложил Ян Рекерс (Jan Rekers, University of Amsterdam). Доказательство завершаемости и корректности предложенного алгоритма проводится 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.

Положения, 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 ε-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.

Научная новизна

Научная новизна полученных в ходе исследования результатов заключается в следующем.

1. Алгоритм, предложенный в диссертации, отличается от аналогов (работы Андрея Бреслава, Кюнг-Гу Дох, Ясухико Минамиде) возможностью построения компактной структуры данных, содержащей деревья вывода для всех корректных генерируемых цепочек. Это позволяет как проверять корректность анализируемых выражений, так и проводить более сложные виды анализа. Многие из существующих алгоритмов предназначены только для проверки корректности выражений, основанной на решении задачи о включении одного языка в другой (JSA, PHPSA). Выполнение более сложных видов анализа, трансформаций или построения леса разбора не предполагается.

2. Новизна представленной архитектуры заключается в том, что данная архитектура позволяет создать платформу для разработки инструментов, решающих широкий круг задач анализа динамически формируемого кода. Архитектуры существующих инструментов (JSA, PHPSA, Alvor, Varis) ориентированы на решение конкретных задач для определённых языков программирования. Решение новых задач или поддержка других языков с помощью этих инструментов затруднены ввиду ограничений, накладываемых архитектурой и возможностями используемых алгоритмов.

3. Метод анализа и обработки встроенного программного кода в проектах по реинжинирингу информационных систем предложен впервые. Как отмечают К.В. Ахтырченко и Т.П. Сорокваша в работе “Методы и технологии реинжиниринга ИС”, имеются проблемы в методологической основе реинжиниринга: существующие работы в области реинжиниринга программного обеспечения либо содержат высокоуровневые решения, не касающиеся деталей, важных при решении прикладных задач (например, работы К. Вагнера, Х. Миллера), либо являются набором подходов к решению конкретных задач (например, сборник “Автоматизированный реинжиниринг программ” под редакцией А.Н. Терехова и А.А. Терехова или “Software Reengineering (IEEE Computer Society Press Tutorial)” Р.С. Арнольда). При этом, встроенный программный код часто не учитывается. С другой стороны, работы, например, М.Д. Шапот, Э.В. Попова, С.Л. Трошина, А. Клеви, посвящены решению конкретных задач обработки встроенного программного кода в контексте реинжиниринга ИС, но не предлагают формального метода их решения.

Теоретическая и практическая значимость работы

Теоретическая значимость диссертационного исследования заключается в разработке формального алгоритма синтаксического анализа динамически формируемого кода, решающего задачу построения конечного представления леса вывода, не решаемую ранее, а также в формальном доказательстве завершаемости и корректности разработанного алгоритма.

На основе полученных в работе научных результатов был разработан инструментарий (Software Development Kit, SDK), предназначенный для создания средств статического анализа динамически формируемых выражений. Данный инструментарий позволяет автоматизировать создание лексических и синтаксических анализаторов при разработке программных средств для решения задач реинжиниринга — изучения и инвентаризации систем, автоматизации трансформации выражений на встроенных языках. Также данный инструментарий может использоваться при реализации поддержки встроенных языков в интегрированных средах разработки.

С использованием разработанного инструментария было реализовано расширение к инструменту ReSharper (ООО “ИнтеллиДжей Лабс”, Россия), предоставляющее поддержку встроенного T-SQL в проектах на языке программирования C# в среде разработки Microsoft Visual Studio. 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).

Степень достоверности и апробация результатов

Достоверность и обоснованность результатов исследования опирается на использование формальных методов исследуемой области, выполнение формальных доказательств и инженерные эксперименты.

Основные результаты работы были доложены на ряде международных научных конференций: SECR-2012, SECR-2013, SECR-2014, TMPA-2014, Parsing@SLE-2013, рабочем семинаре “Наукоемкое программное обеспечение” при конференции PSI-2014. Доклад на конференции SECR-2014 был награждён премией Бертрана Мейера за лучшую исследовательскую работу в области программной инженерии. Дополнительной апробацией является то, что разработка инструментальных средств на основе предложенного алгоритма была поддержана Фондом содействия развитию малых форм предприятий в технической сфере (программа УМНИК, проекты № 162ГУ1/2013 и № 5609ГУ1/2014).

Публикации по теме диссертации

Все результаты диссертации изложены в 7 научных работах из которых 3 [1–3] содержат основные результаты работы и опубликованы в журналах из “Перечня российских рецензируемых научных журналов, в которых должны быть опубликованы основные научные результаты диссертаций на соискание ученых степеней доктора и кандидата наук”, рекомендовано ВАК. 1 работа [4] индексируются Scopus. Работы [1–7] написаны в соавторстве. В [1] С. Григорьеву принадлежит реализация ядра платформы YaccConstructor. В [2, 3] и [5] С. Григорьеву принадлежит постановка задачи, формулирование требований к разрабатываемым инструментальным средствам, работа над текстом. В [4] автору принадлежит идея исследования, реализация и описание алгоритма анализа встроенных языков на основе RNGLR-алгоритма. В [6] С. Григорьеву принадлежит реализация инструментальных средств, проведение экспериментов, работа над текстом. В работе [7] автору принадлежит разработка алгоритма синтаксического анализа динамически формируемого кода.

Объем и структура работы

Диссертация состоит из введения, шести глав, заключения и списка литературы. Полный объем диссертации **130** страниц текста с **26** рисунками и **8** таблицами. Список литературы содержит **106** 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 соответствующие инструменты. 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* ε*-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 ω 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 анализируемого 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 задан 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 предложенного выше алгоритма.

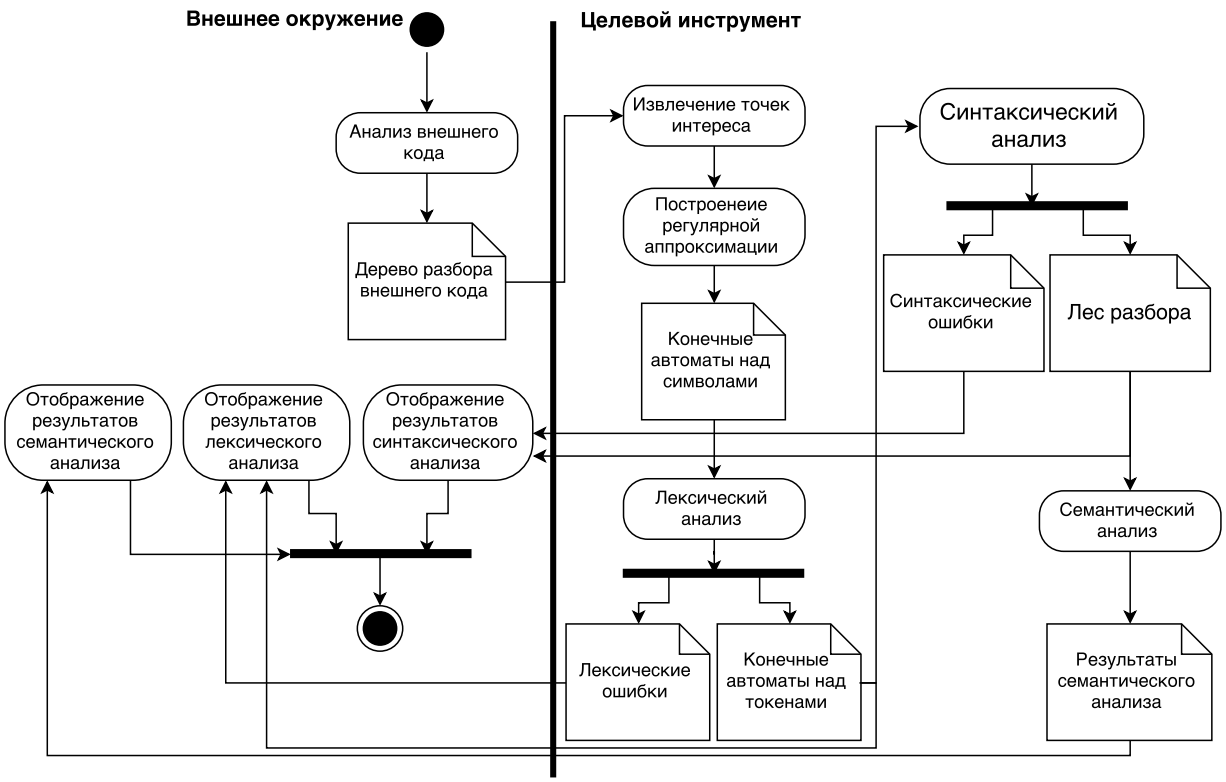

Рис. 1: Пороцесс обработки динамически формируемых выражений

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 за выделение точек интереса, построение регулярной аппроксимации множества значений динамически формируемого выражения, проведение лексического и синтаксического анализа. Также описывается YC.SEL.SDK.ReSharper — “обёртка” для YC.SEL.SDK, позволяющая создавать расширения к ReSharper для поддержки встроенных языков.

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 кода, необходимым для обработки конкретной информационной системы.

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

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

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

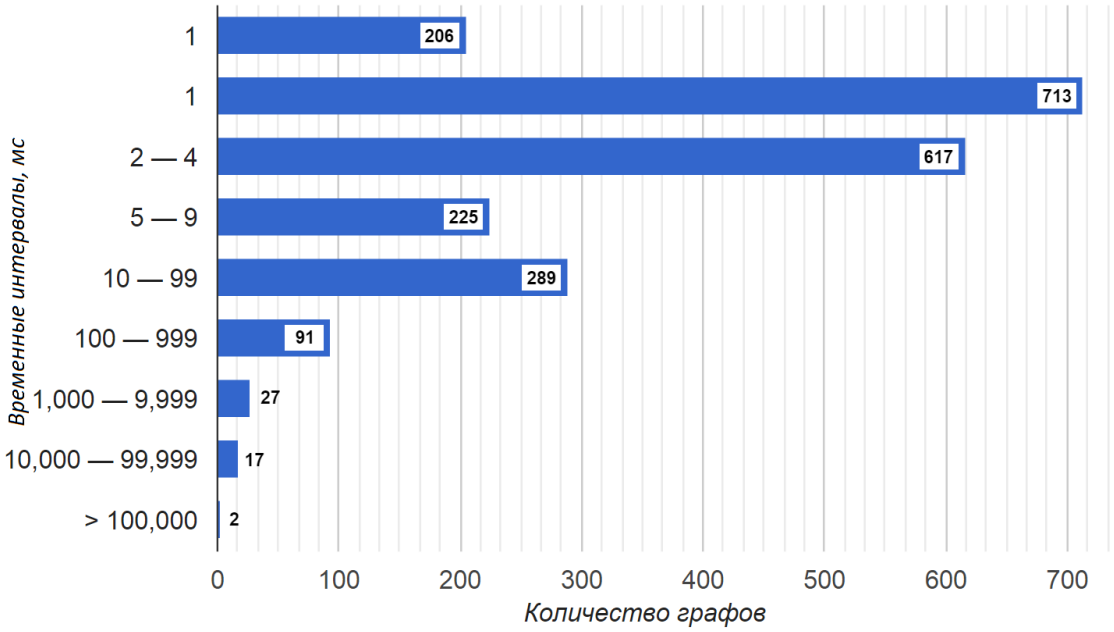

Рис. 3: Распределение запросов по времени анализа

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 интервалам времени, затраченным на анализ, приведено на рисунке 3.

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 деревьев 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.

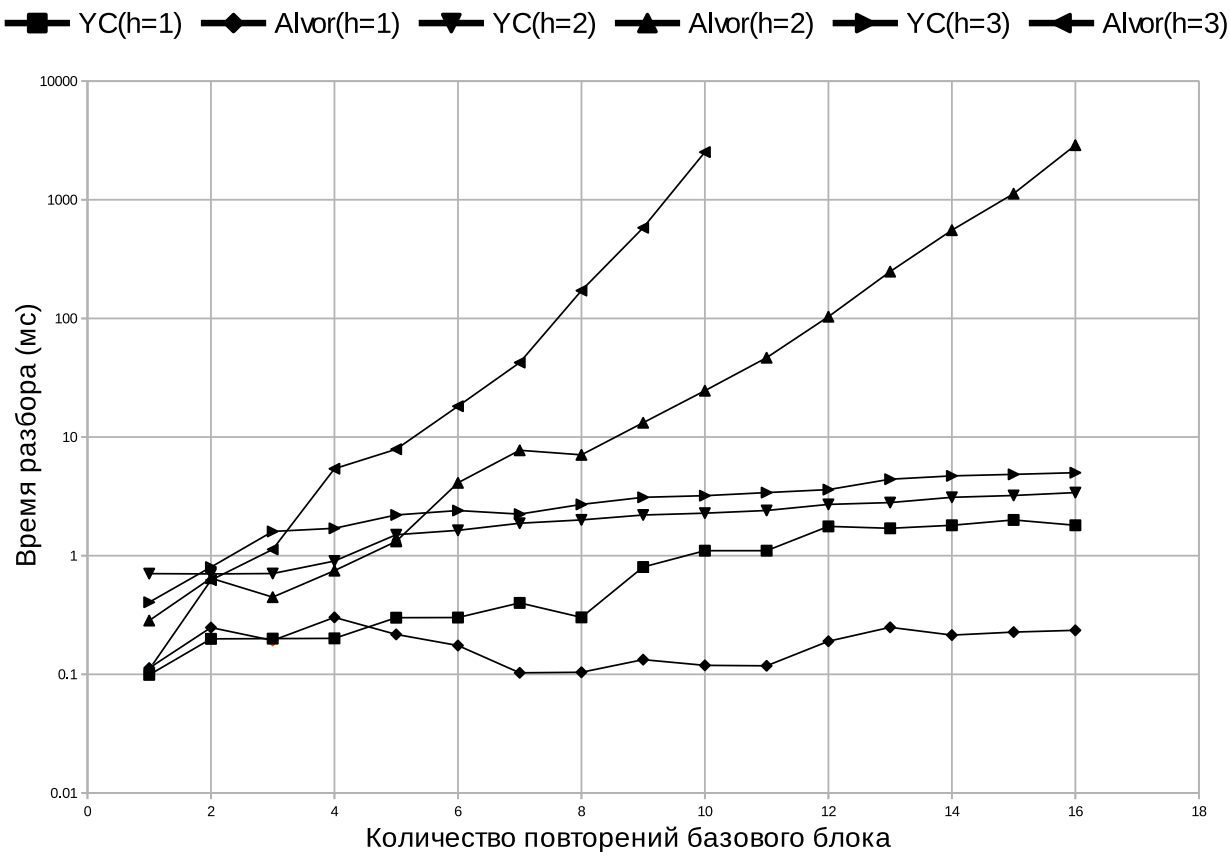

Рис. 4: Сравнение производительности Alvor и синтаксического анализатора на базе YC.SEL.SDK

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. Измерения проводились на синтетических данных, построенных с помощью последовательного соединения базовых блоков, каждый из которых содержит ветвления с h параллельными путями. Результаты экспериментов представлены на графике 4. При более чем шестнадцатикратном повторении блоков с $h = 2$ время работы Alvor превысило 30 минут и измерения были прекращены. Аналогичная ситуация возникает и при более чем десятикратном повторении блоков с $h = 3$. Таким образом, измерения показывают, что время работы анализатора Alvor растёт экспоненциально относительно количества повторений базового блока при $h > 1$. Анализатор созданный на основе YC.SEL.SDK в таких случаях имеет лучшую производительность. При этом на линейном входе Alvor работает быстрее. Однако существуют возможности для оптимизации текущей реализации, благодаря чему производительность на линейном входе может быть улучшена.

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 итоги сравнения и соотнесения полученных результатов с существующими аналогами. Получены следующие выводы.

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** подведены **итоги** диссертационной работы, которые заключаются в достижении следующих результатов.

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 ε-переходов.
2. Создана архитектура инструментария для разработки программных средств статического анализа динамически формируемых строковых выражений. На её основе реализован инструментальный пакет для разработки средств статического анализа динамически формируемых выражений, применённый для реализации расширения для ReSharper.
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 этого были сформулированы **рекомендации по применению результатов работы** в индустрии и научных исследованиях, заключающиеся в необходимости учитывать следующие ограничения.

1. Необходимо, чтобы множество, являющееся аппроксимацией значений динамически формируемого выражения, принимаемое на вход алгоритмом синтаксического анализа, должно быть регулярным.
2. Эталонный язык должен быть описан детерминированной контекстно-свободной грамматикой.

3. Важно учитывать, что платформа разрабатывалась с ориентацией на создание инструментов для реинжиниринга. Поэтому в некоторых компонентах точность анализа является большим приоритетом, чем производительность. Однако архитектура платформы позволяет легко заменять отдельные компоненты, что делает возможным получение желаемого соотношения точности и производительности инструментов.

Также были определены **перспективы дальнейшей разработки тематики**, основными из которых являются исследование возможности выполнения семантических действий непосредственно над SPPF и теоретическая оценка сложности предложенного алгоритма. Кроме этого, с целью обобщения предложенного подхода, а также для получения лучшей производительности и возможностей для более качественной диагностики ошибок, необходимо рассмотреть применение других алгоритмов обобщённого синтаксического анализа (GLL, BRNGLR, RIGLR) для решения рассматриваемой задачи.

Публикации автора по теме диссертации в журналах из перечня российских рецензируемых научных журналов, в которых должны быть опубликованы основные научные результаты диссертаций на соискание ученых степеней доктора и кандидата наук

1. Григорьев, С. В. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем / Я. А. Кириленко, С. В. Григорьев, Д. А. Авдюхин // 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 / С. В. Григорьев, Е. А. Вербицкая, М. И. Полубелова и др. // Моделирование и анализ информационных систем. — 2014. — Т. 21, № 6. — С. 131–143.

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

Публикации автора по теме диссертации в других изданиях

4. Grigorev, S. GLR-based abstract parsing / S. Grigorev, I. Kirilenko // 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 / S. Grigorev, E. Verbitskaia, A. Ivanov et al. // 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 / S. Grigorev, I. Kirilenko // 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 / E. Verbitskaia, S. Grigorev, D. Avdyukhin // Preliminary Proceedings of the PSI 2015: 10th International Andrei Ershov Memorial Conference. — 2015. — P. 1–12.