На правах рукописи

ЗУБОВ Максим Валерьевич

МОДЕЛИ И АЛГОРИТМЫ УНИВЕРСАЛЬНЫХ ПРОМЕЖУТОЧНЫХ ПРЕДСТАВЛЕНИЙ ДЛЯ СТАТИЧЕСКОГО АНАЛИЗА ПОТОКА УПРАВЛЕНИЯ ПРОГРАММ ПО ИХ ИСХОДНОМУ КОДУ

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени кандидата технических наук

Уфа – 2016

Работа выполнена на кафедре компьютерной безопасности и прикладной алгебры ФГОУ ВПО «ЧелГУ»

Научный руководитель: кандидат технических наук, доцент
Пустыгин Алексей Николаевич
ФГБОУ ВПО «ЧелГУ»

Официальные оппоненты: доктор технических наук, доцент
Иванов Александр Иванович
АО «Пензенский научно-исследовательский электротехнический институт», начальник лаборатории биометрии и нейросетевых технологий

кандидат технических наук, доцент
Макеев Григорий Анатольевич
ФБГОУ ВПО «Уфимский государственный авиационный технический университет», доцент кафедры вычислительной математики и кибернетики

Ведущая организация: ФГБУН Институт математики и механики им. Н.Н. Красовского Уральского отделения РАН,
г. Екатеринбург

Защита диссертации состоится 17 июня 2016 г. в 10^{00} часов на заседании диссертационного совета Д-212.288.07 при ФГБОУ ВПО «Уфимский государственный авиационный технический университет» по адресу: 450000, г. Уфа, ул. К. Маркса, 12.

С диссертацией можно ознакомиться в библиотеке ФГБОУ ВПО «Уфимский государственный авиационный технический университет» и на сайте *www.ugatu.su*.

Автореферат разослан « » апреля 2016 г.

Ученый секретарь
диссертационного совета,
доктор технических наук, доцент И.Л. Виноградова

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы. Разработка программного обеспечения в современных условиях — это задача, требующая быстроты и эффективности. Если программа выпускается не вовремя, не содержит важных функций или имеет ошибки, то она проигрывает в конкурентной борьбе. Для решения этих проблем некоторые задачи выполняются машинно и в автоматическом режиме. Это позволяет экономить время программистов, а также повысить эффективность выполнения таких задач. К таким задачам можно отнести: интегрированную отладку, автоматизированное тестирование, непрерывную интеграцию и статический анализ. Статический анализ позволяет находить множество ошибок в программах на ранней стадии разработки, помогает лучше понимать структуру программ. Он берет свое начало с разработок Bell Labs в 70-х годах XX века и работ П. Кусо (P. Cousot) и Ф. Аллена (F. Allen).

Большинство анализаторов делают упор на поиск ошибок, специфичных для конкретного языка программирования. Например, популярные анализаторы FindBugs и CheckStyle для Java или PVS Studio для C/C++ узко специализированы именно на ошибках, что показано в работах Н. Айваха (N. Ayewah) и А.Н. Карпова. Такие анализаторы не могут быть расширены или применены где-то кроме их целевого языка. Из такой ситуации следует, что область построения модели для анализа развита недостаточно. Многие просто используют дерево разбора парсера, как наиболее изученную модель в области компиляции, в качестве промежуточного представления — набора машинных данных, семантически эквивалентных исходному коду, над которыми выполняется анализ.

На сегодняшний день существуют научные работы, описывающие использование различных промежуточных представлений. В основном, это труды зарубежных авторов: М. Линтона (M. Linton), Р. Крю (R. Crew), Е. Хаджиева (E. Hajiyev), Д. Бэйера (D. Beyer), С. Джарзабэка (S. Jarzabek). Например, в Omega предлагается использовать реляционные базы данных, в ASTLOG – базу знаний на языке Prolog, а в BLAST – конечный автомат. На практике, в основном, распространенные анализаторы используют AST, что может осложнять некоторые цели анализа, или делать другие неэффективными.

При статическом анализе важным является анализ различных возможных путей исполнения программы. Такой анализ можно разделить на анализ потока управления программы и анализ графа вызовов. Первый позволяет исследовать путь исполнения конкретного функционального модуля, а второй позволяет анализировать взаимодействие функциональных модулей между собой.

Кроме того, в настоящее время огромное распространение получает программное обеспечение с открытым исходным кодом. Такое программное обеспечение содержит в себе много информации об архитектурных решениях и алгоритмах. Однако, в случае больших программных комплексов технических систем, таких как сервера приложений, исследование исходного кода достаточно затруднительно ввиду больших его объемов. Таким примером может выступить веб-сервер Apache, под управлением которого работает большинство веб-серверов. Исследование такого программного кода представляет очень большой ин-

терес. В качестве средства исследования и извлечения знаний из такого программного обеспечения может выступать статический анализ. Извлечение информации и реверс-инжиниринг исследуются в работах Р. Коцке (R. Koschke), В.М. Ицыксона, Ю.К. Язова.

Степень разработанности темы исследования. Развитию статического анализа в области поиска ошибок, а также предложения подсказок по рефакторингу посвящено большое количество работ российских и зарубежных исследователей: Д. Дамнса (D. Damns), А.Ю. Аветисяна, В.М. Ицыксона.

Идеи развития модели промежуточных представлений предлагались Дж. Церански (J. Czeranski), Р. Коцке (R. Koschke), Дж. Веббом (J. Webb) в их анализаторах Bauhaus и MALPAS. Основные идеи — это использование некоторого псевдокода для описания нескольких входных языков или описания программы для разных ее уровней в виде отдельных промежуточных представлений.

Среди исследований, посвященных анализу потока управления программы, следует выделить работы О. Шиверса (O. Shivers), В.А. Битнера и А.В. Чернова. Эти работы в основном направлены на использование графа потока управления при оптимизации или анализе алгоритмов. Кроме того, стоит выделить исследователей, работы которых посвящены анализу графа вызовов программ, – К. Али (K. Ali), Д. Гроува (D. Grove), О. Костакиса (O. Kostakis).

Объектом исследования являются методы и алгоритмы выполнения статического анализа исходного кода и модели используемых промежуточных представлений.

Предмет исследования — модели, методы и алгоритмы построения универсальных промежуточных представлений для исследования потока управления программ при статическом анализе исходного кода.

Целью работы является повышение эффективности выполнения статического анализа программ, написанных на разных языках, и извлечения информации из них с помощью использования универсальных промежуточных представлений.

Для достижения цели в работе решаются следующие задачи:

1. Разработка модели и формата промежуточного представления, позволяющего эффективно выполнять анализ потока управления в исходных текстах на нескольких языках программирования.
2. Разработка методов и алгоритмов получения предложенного промежуточного представления в разработанном формате из исходного кода.
3. Разработка методов и алгоритмов проверки эффективности использования полученного универсального промежуточного представления при выполнении анализа, программная реализация, и проведение вычислительного эксперимента.
4. Разработка методов и алгоритмов анализа предложенного промежуточного представления в задачах рефакторинга, анализа потока управления и извлечения информации.
5. Реализация предложенных методов и алгоритмов получения и анализа промежуточного представления в виде программного комплекса. Выпол-

нение вычислительных экспериментов для выработки оптимальных входных значений разработанных инструментов.

Методы исследования

Основные результаты диссертационной работы получены с использованием математического аппарата теории множеств, теории графов, теории автоматов; методов вычислительных экспериментов, методов объектно-ориентированного анализа, проектирования и программирования.

Положения, выносимые на защиту

1. Модель универсального промежуточного представления потока управления для анализа программ, отличительной особенностью которой является возможность описания нескольких входных языков программирования, что позволяет создавать инструменты анализа, которые можно будет одинаково использовать для всех языков, для которых будет реализовано представление.
2. Метод и алгоритм получения предложенного промежуточного представления для статического анализа, основанный на использовании абстрактного цифрового автомата с памятью, который позволяет формализовать процесс получения представления, что ускоряет разработку новых генераторов по готовой модели. Генератор представления основан на открытом исходном коде компилятора, что, в отличие от типового решения — генератора парсера, позволяет избежать дополнительной верификации результатов работы, так как такую реализацию можно считать эталонной.
3. Метод и алгоритм практической проверки эффективности выполнения анализа программ с использованием предложенных универсальных промежуточных представлений в экспериментальных задачах рефакторинга позволяет оценить, насколько можно увеличить быстродействие утилит, выполняющих анализ.
4. Методы и алгоритмы выполнения анализа потока управления с целью оценки качества исходного кода, использующие разработанные универсальные промежуточные представления, выполняют анализ исходного кода единообразно для нескольких входных языков без их модификации.
5. Программный комплекс, выполняющий статический анализ исходных текстов на нескольких языках программирования, реализующий предложенные модели и алгоритмы получения и обработки универсального промежуточного представления в 9 анализаторах.

Научная новизна результатов работы заключается в следующем:

1. Модель универсального промежуточного представления графа потока управления для статического анализа программ, в отличии от известных, позволяет без потери и избыточности данных анализировать граф потока управления, отличительной особенностью которого являются свойства узлов, позволяющие построить граф вызовов. Такая модель описывает в виде дерева ориентированный граф произвольной структуры, а также позволяет взаимодействовать универсальным утилитам с генераторами для различных языков программирования.

2. Метод получения промежуточного представления использует синтаксический анализатор компилятора с открытым исходным кодом, что отличает его от типового решения — генератора парсера по входной грамматике языка программирования, что обеспечивает точность получаемых данных. Кроме того, алгоритм преобразования дерева разбора в целевое промежуточное представление с его обходом в глубину отличается использованием абстрактного цифрового автомата, что позволяет описать его получение в виде модели.
3. Методы и алгоритмы статического анализа с использованием предложенных промежуточных представлений основываются на численном подходе для оценки качества (необходимости рефакторинга) участка исходного кода, используя входные сравнительные величины, рекомендуемые значения которых были получены с помощью вычислительного эксперимента.
4. Методы визуализации для анализа исходного кода программ, в отличие от известных, используют предложенные модели универсальных представлений, что позволяет получать графические изображения потока управления и графа вызовов из единого формата данных для нескольких языков программирования.

Практическую значимость представляют следующие результаты:

1. Разработанный формат универсального промежуточного представления потока управления позволяет единообразно анализировать исходные тексты на различных языках, поддерживающих императивную парадигму программирования. Такой формат позволяет взаимодействовать универсальным анализаторам с генераторами представлений для различных языков.
2. Предложенные алгоритмы выполнения статического анализа, основанные на универсальном формате, позволяют получать количественные оценки для улучшения существующего кода, а так же визуализировать исходный код для извлечения информации о его функционировании.
3. Разработанный комплекс программ позволяет выполнять статический анализ исходного кода на нескольких языках программирования и поддерживает повторное использование единожды созданных универсальных инструментов.

Разработанное программное обеспечение может использоваться в процессе разработки нового программного обеспечения или улучшения существующего. Результаты работы внедрены в рабочий процесс разработки продуктов и сервисов направления решений в образовании ЗАО «Нау-сервис» (ГК NAUMEN). В результате внедрения упростился процесс доработки, модернизации и развития существующих программных модулей систем путем проведения исследования кода и выполнения предложений по рефакторингу.

Степень достоверности и апробация результатов

Полученные в диссертационной работе результаты не противоречат известным теоретическим положениям. Достоверность результатов продемонстрирована на серии примеров и подтверждена путем разработки программ-

ного обеспечения, базирующегося на предложенных моделях и алгоритмах и использованием математического аппарата, а также результатами апробации и внедрения в процесс разработки в компании ЗАО «Нау-Сервис».

Основные научные и практические результаты диссертационной работы докладывались и обсуждались на следующих конференциях и семинарах:

- 7, 8 и 9-я конференции «Свободное программное обеспечение в высшей школе» – г. Переславль-Залесский, ИПС РАН, 2012, 2013, 2014 гг.;
- Семинар Научно-Практического Центра СКВ и ОПО – г. Челябинск, ЧелГУ, 2012 г.;
- Международная научно-практическая конференция «FOSS Lviv-2012» – Украина, г. Львов, Львовский национальный университет имени Ивана Франко, 2012 г.;
- 4-я корпоративная конференция «Naumen Devel Camp #4» – г. Екатеринбург, 2012 г.;
- Восьмая международная конференция «Linux Vacation / Eastern Europe» – Республика Беларусь, г. Гродно, 2012 г.;
- Научно-практическая конференция «Тверские интернет технологии» – г. Тверь, 2013 г.;
- Семинары Сектора визуализации ОСО ИММ УрО РАН – г. Екатеринбург, 2014 г.;
- Семинары кафедры компьютерной безопасности и прикладной алгебры ЧелГУ — г. Челябинск, 2014, 2016 гг.;
- Научный семинар под руководством профессора В.Е. Федорова на кафедре математического анализа ЧелГУ — г. Челябинск, 2015 г.;
- Корпоративная конференция «Naumen Devel Camp 15#2» – г. Екатеринбург, 2015 г.;
- III Международная конференция «Интеллектуальные технологии обработки информации и управления» (ITIPM’2015) — г. Уфа, 2015 г.

Публикации по теме диссертации

Основные материалы диссертационной работы опубликованы в 16 работах, среди них 8 статей – в журналах из списка периодических изданий, рекомендованных ВАК РФ; 8 работ – в трудах ведущих тематических изданий и трудах российских и международных конференций. Имеется свидетельство о регистрации программы для ЭВМ.

Личный вклад автора

Все результаты получены лично автором. Из совместных публикаций в работу включены только результаты, которые получены соискателем. В совместных работах с А.Н. Пустыгиным научному руководителю принадлежат постановка задачи, общее руководство и частично выводы. Результаты совместных статей с Е.В. Старцевым использовались в части, относящейся к диссертации автора.

Структура и объем диссертации

Диссертация состоит из введения, четырех глав, заключения и списка используемой литературы. Текст диссертации изложен на 134 листах, включая 46 рисунков, 15 таблиц, библиографический список из 125 наименований, список

иллюстративного материала, список сокращений и условных обозначений, одно приложение.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Во введении обоснована актуальность темы диссертационной работы, сформулирована цель и основные задачи исследования, показана научная новизна и практическая значимость работы, дано краткое описание работы.

Первая глава посвящена детальному обзору области статического анализа, а также различным используемым промежуточным представлениям.

Статический анализ исходного кода — актуальная задача, возникающая при разработке программного обеспечения. Для эффективного статического анализа требуется *промежуточное представление* - набор машинных данных, над которыми выполняется анализ. Самыми распространенными промежуточными представлениями являются:

1) Абстрактное синтаксическое дерево (AST). Оно получило большое распространение, как наиболее изученное в области компиляции. . При построении дерева для большого проекта целиком возникает проблема большого размера этого дерева. Однако, анализ на основе AST позволяет обнаруживать специфичные для данного языка дефекты за счет использования всей информации об исследуемом исходном коде.
2) Реляционное представление предполагает следующий подход: каждый элемент грамматики представляется в виде реляционного отношения. Эти отношения хранятся в реляционной базе данных. Стоит отметить, что реляционное представление получается на основе AST, а уже потом по нему заполняется БД. Кроме того, предлагается использовать аппарат логических языков программирования: базы знаний (Prolog) или дедуктивными базами (Datalog). При использовании логических языков возрастает повторная используемость однажды написанных предикатов, потому что их поведение легко понять из описания.
3) Абстрактный автомат в качестве представления применяется для верификации. Примером может служить инструмент BLAST. В нем для каждой из функций программы с состояниями связываются управляющие точки программы, а с переходами – операции. Для верификации из автомата строятся абстрактные деревья достижимости (ART), объединяющие переходы между состояниями различных автоматов для функций. Такое представление является похожим на Static Single Assignment (SSA), распространенное в области компиляции для оптимизации генерируемого машинного кода.

На основе данных о разработках в области промежуточных представлений можно выделить перспективные направления развития: универсальные и высокоуровневые представления. Анализ потока управления потребует соответствующее представление, описывающее поток управления в виде блоков кода и переходов между ними. Оно должно содержать описание всех методов и функций проекта и учитывать все возможные ветвления в потоке управления.

Назовем *универсальными* промежуточные представления, пригодные для использования с несколькими входными языками. В представлении выделяются общие для них особенности. Универсальные решения позволяют производить типовые процедуры анализа для различных языков. *Частные представления* — промежуточные представления, позволяющие описать исходный код только на одном входном языке.

Многоуровневые представления исходного кода используют только интересующие данные, для разных задач использовать представления разной степени абстракции. Идея такого подхода была предложена С. Джарзабэком (S. Jarzabek).

На более высоких уровнях абстракции лучше использовать универсальные представления, так как в них имеется больше общих элементов для разных языков. Это позволит найти компромисс между затратами на хранение и обработку данных. Каждое представление хранит свой уровень абстракции, что позволит избежать избыточности. При необходимости получения данных другого уровня могут быть использованы соответсвующие представления в их *совместном анализе*.

Во второй главе разработаны математические модели универсальных многоуровневых промежуточных представлений и модели их генераторов.

Эффективной, изученной и неоднократно используемой формой представления исходного кода является дерево. Разрабатываемое промежуточное представление тоже будет являться деревом. В полном дереве можно выделить конечное множество поддеревьев A, $A_i \in A$ - некоторое поддерево, A_0 - корневое поддерево, содержащее самый верхний узел всего дерева.

Любое поддерево, включая корневое, можно описать как $A_i = (o, AS_i, L_i)$, где $o \in O$ - узел из множества узлов представления, $L_i \subset L$ - множество листьев текущего корня, L – множество всех листьев, $AS_i \subset AS$ - множество поддеревьев текущего корня, AS – множество всех вложенных поддеревьев, $AS \cup A_0 = A$. Используя такую модель, можно разрабатывать промежуточные представления, ограничивая множество узлов и листьев.

Для анализа потока управления исходного кода необходимо описать множества узлов и листьев. Для разрабатываемого представления обозначим их за V и I. Основным элементом потока управления является функция (метод класса). Ограничим V следующими значениями: project — исследуемый проект целиком, method — метод класса в проекте, function — функция в проекте (для процедурных языков), block — блок кода в потоке управления, flow — переход между блоками кода, tryExcept — блок обработки исключений, finally — блок, выполняющийся независимо от исключений, for — цикл со счетчиком, if — условный оператор, while — цикл с условием, call — вызов функции или метода. В таблице 1 показано наличие таких элементов в языках Java, Python, C++, PHP. Отсутствующие элементы в некоторых языках влияют только на различные ветвления, потому их отсутствие в представлении не повлияет на анализ.

$$V = \{project, method, function, block, flow, tryExcept, finally, for, if, while, call\}$$

Таблица 1. Наличие узлов представления в популярных языках

Язык	**project**	**method/function**	**block/flow**	**try Except**	**finally**	**for**	**if**	**while**	**call**
Java	описать проект целиком можно для любого языка	методы классов	блоки кода и переходы между ними есть в этих языках	есть try/ catch	есть	есть for и foreach	условный оператор есть в этих языках	есть while /do-while	вызов функций есть в этих языках
Python		функции и методы		есть try/ except	есть	есть foreach		есть while	
C++		функции и методы		есть try/ catch	нет	есть		есть while /do-while	
PHP		функции и методы		есть try/ catch	есть с версии 5.5	Есть for и foreach		есть while /do-while	

В такой модели промежуточного представления у дерева будет ограниченная вложенность в отличие от абстрактного дерева разбора. Для обозначения поддеревьев модели вводится двойная индексация. Каждое поддерево будем обозначать как $U_{i,j} \in U$, где i - номер поддерева на уровне, а j - номер уровня, U – множество всех поддеревьев. $US_{i,j}$ – множество поддеревьев уровня j для узла с номером i . На каждом уровне также будет свое множество допустимых листьев $I_i \subset I$.В предлагаемом представлении потока управления будет возможно использовать только следующие уровни:

1) Корневое поддерево, содержащее проект целиком $U_{1,1} = (p, US_{i,1}, \{\})$;

2) Участок потока управления (метод или функция) $U_{i,2} = (v_{i,2}, US_{i,2}, IF_i)$, где $v_{i,2} \in \{method, function\}$, $lf_i \in IF_i$ - листья с именами функционального блока и класса (если это метод), описываемого узлом $v_{i,2}$, $IF_i \subset I_2$ - множество всех листьев уровня 2, $I_2 = \{name, class\}$;

3) Элементы потока управления (блоки, ветвления и переходы) $U_{i,3} = (v_{i,3}, US_{i,3}, IB_i)$, где $v_{i,3} \in \{block, flow, tryExcept, finally, for, if, while\}$, $IB_i \subset I_3$ - характеристики конкретного узла уровня 3, $I_3 = \{id, toId, fromId\}$;

4) Вызовы внутри блоков кода $U_{i,4} = (call, \{\}, IC_i)$, где $IC_i \subset I_4$ - характеристики конкретного узла (id — для блоков, from и to — для дуг графа).

Для получения разрабатываемого представления потока управления из абстрактного дерева разбора необходимо выполнить обход одного дерева и формирование другого. Такое дерево можно представить в виде потока его узлов и атрибутов в соответствии с алгоритмом обхода в прямом порядке в глубину. Задачи преобразования и обработки потока некоторых входных данных эффективно решаются с помощью абстрактных цифровых автоматов.

Степень ветвления в AST велика, а вложенность может быть произвольной, однако, она соответствует контекстно-свободной грамматике входного язы-

ка. Для разбора таких последовательностей применяют конечные автоматы с магазинной (стековой) памятью (МП-автоматы). В отличие от классической модели будем хранить в стеке не входные сигналы, а состояния. Таким образом, можно формализовать общую модель такого автомата для получения промежуточных представлений.

$M=(S, K, X, Y, z_0, \delta, \lambda, \varphi, s_0)$, где S – конечное множество состояний автомата, K - алфавит стека состояний, являющийся множеством $S \subset K$ (любое состояние можно поместить в стек), z_0 - нулевой (начальный) символ стековой памяти состояний $z_0 \in K$, X – множество входных сигналов, Y – множество выходных сигналов, δ - функция переходов, λ - функция выходов, φ - функция получения следующего элемента для записи в память (функция памяти), а s_0 - начальное состояние.

Каждое состояние автомата будет соответствовать преобразованию определенного узла AST или группы узлов, которые описывают одну сущность высокоуровневого представления. Так как один узел выходного представления может соответствовать нескольким узлам входного, то целесообразно воспользоваться моделью автомата Мили, в котором выходные сигналы зависят не только от состояния, но и входных данных. Таким образом, функция переходов в автомате является отображением $\delta: S \times X \times K \rightarrow S$, а функция выходов для автомата Мили, в свою очередь, отображением $\lambda: S \times X \rightarrow Y$. Функция памяти будет показывать, какой символ необходимо записать в стек в конце такта $\varphi: S \times X \rightarrow K$.

Обозначим за e - некоторый сигнал, обозначающий завершение обработки узла и переход к его родительскому. Этот символ будет показывать, что необходимо перейти к предыдущему состоянию. Он также будет использоваться в выходных данных.

На вход автомата будут поступать элементы абстрактного синтаксического дерева разбора Java, а именно поддеревья вида $A_i=(o, AS_i, L_i)$.

Будем считать, что входной сигнал автомата состоит из узла входного представления, множества листьев и возможного сигнала e для перехода к предыдущему состоянию. Таким образом, $x=(o, L_i, e)$, где $x \in X$ - входной сигнал, $o \in O$ - узел AST, L_i - листья текущего узла, которые можно извлечь в данный момент, e - символ перехода к родительскому узлу.

На выходе автомата будем получать поддеревья универсальных многоуровневых представлений вида $U_{i,j}=(v, US_{i,j}, I_{i,j})$.

По аналогии со входными сигналами, на выходе будем получать сигналы, состоящие из узлов нового представления, его листьев (характеристик) и признака e . Структура выходного сигнала - $y=(v, I_{i,j}, e)$, где $y \in Y$ - выходной сигнал, $v \in V$ - узел получаемого представления, $I_{i,j} \subset I_i$ - некоторое подмножество листьев узла, которые можно выдать в данный момент, e - символ перехода к родительскому узлу.

Для описания поведения автомата используются таблицы переходов и выходов на основе функций переходов и выходов, соответственно. Для описания входных и выходных сигналов и их связей с узлами входных и выходных пред-

ставлений также используются таблицы сигналов. Общая схема разработанного автомата представлена на рисунке 1.

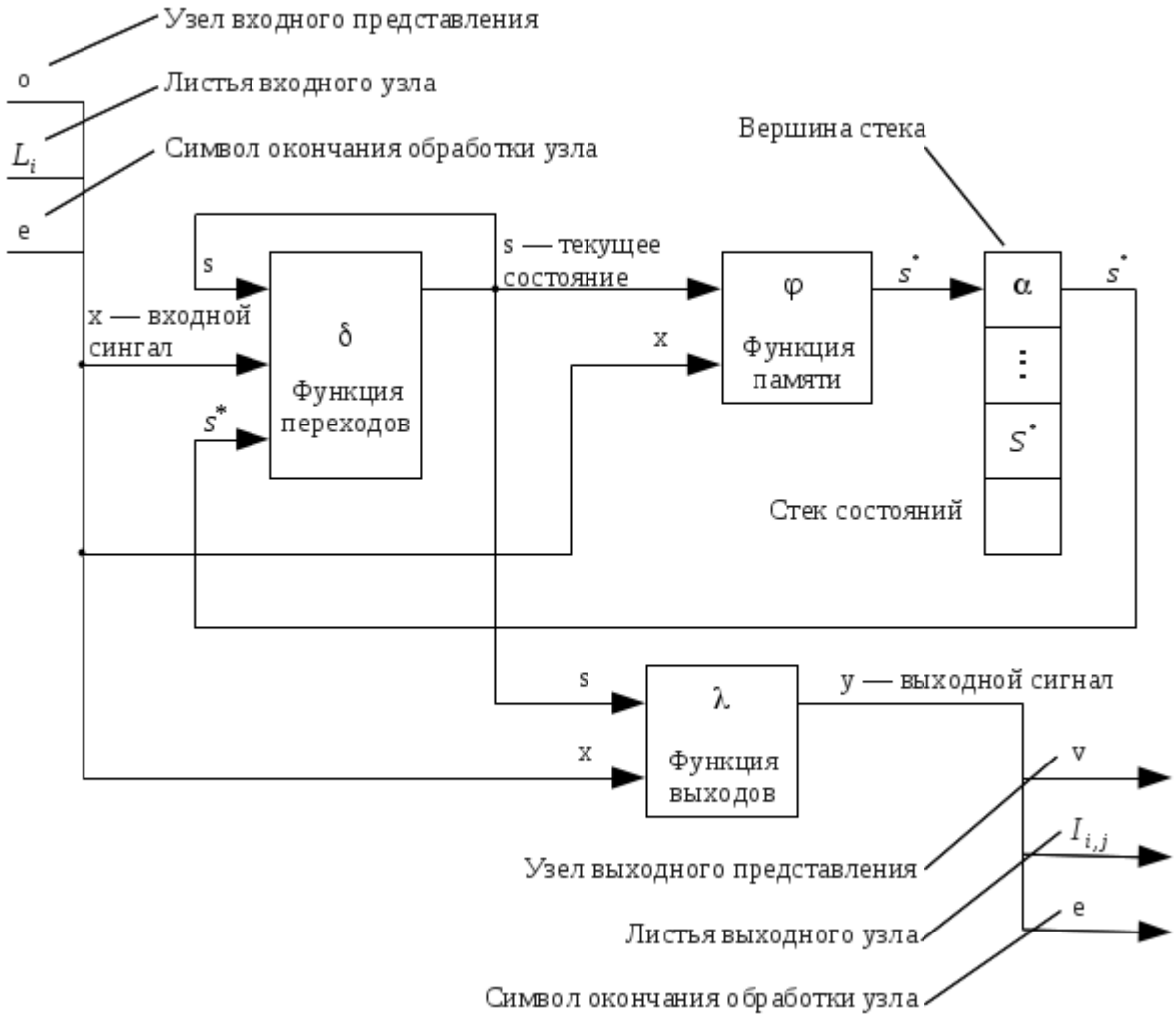

Рисунок 1. Схема абстрактного цифрового автомата для генерации промежуточного представления

Третья глава посвящена разработке и тестированию генераторов представлений для языка Java. А также проверке эффективности использования разработанного представления.

Можно составить список анализаторов, которые можно реализовать с помощью разработанных представлений для решения поставленных задач. Анализаторы, предлагающие улучшение кода: анализ «жадных» функций/методов; контроль положения создаваемых объектов; анализ разделения ответственности между классами. Анализаторы, вычисляющие метрики исходного кода: анализ недостижимого кода; анализ количества независимых функциональных модулей. Анализаторы с помощью визуальных диаграмм: визуализация представления потока управления; визуализация заданной трассы. Анализаторы трасс: поиск создаваемых объектов вдоль трассы; поиск небезопасных вызовов вдоль трассы.

Исходными данными для получения представления является абстрактное синтаксическое дерево разбора. На его основе, используя описанную модель, можно получить дерево необходимого представления. Решено было использо-

вать парсер из компилятора javac, входящего в пакет OpenJDK. Исходный код компилятора распространяется под открытой лицензией GNU GPL v.2.

Такой генератор AST из исходного кода на основе использования внутренних данных компилятора назовем *препаратом компилятора*, так как извлекаются его внутренние скрытые данные. Для дальнейшего преобразования дерево разбора можно сохранить в виде XML. В таком случае AST как промежуточный набор данных будет обладать преимуществами экспортируемых промежуточных представлений.

На рисунке 2 представлена UML-диаграмма классов разработанного препарата компилятора javac, выполняющего получение XML формы AST для языка Java.

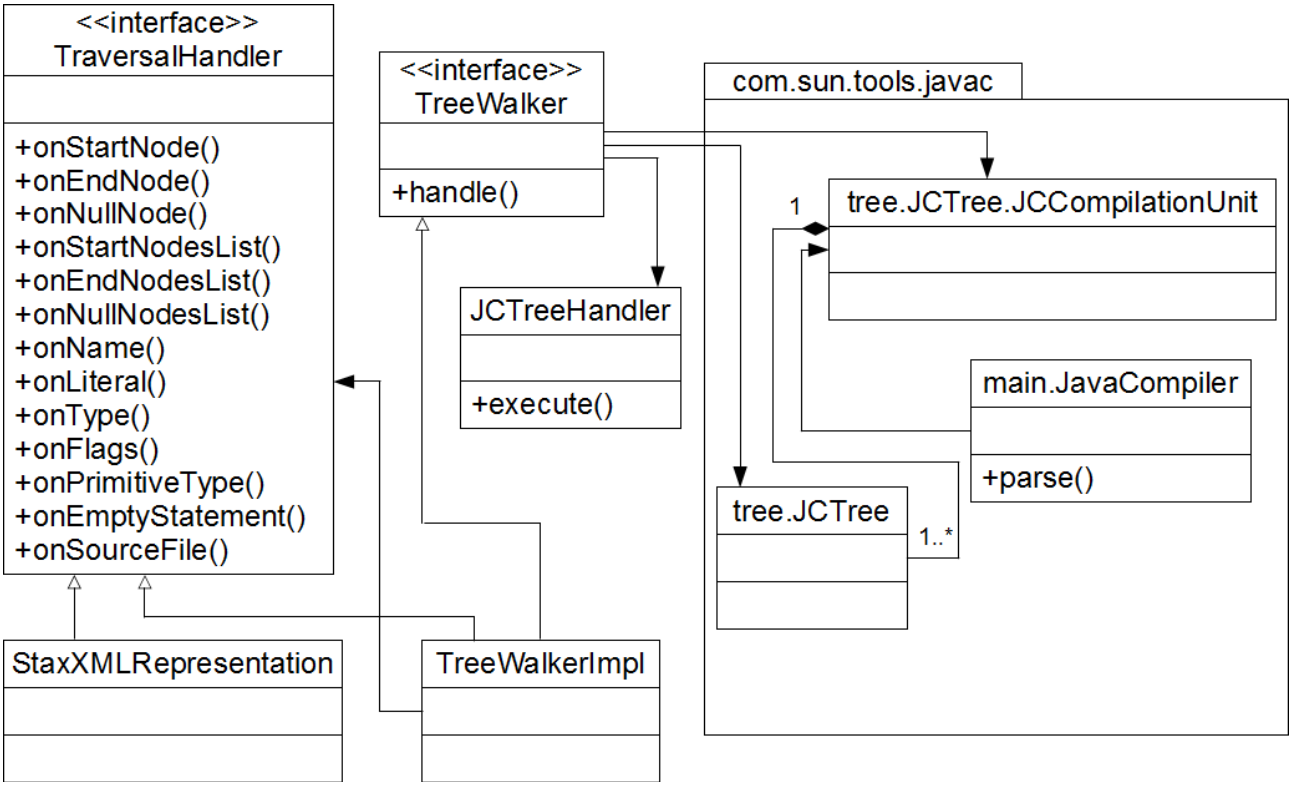

Рисунок 2. UML-диаграмма классов генератора AST-XML для языка Java на основе препарата компилятора javac

Функционирование генератора представлений для Java основывается на обработке входного XML AST и получении XML-представления, описываемого конкретной схемой в соответствии с моделью. Для универсального промежуточного представления потока управления была разработана XML-схема на основе предложенной и описанной ранее модели.

Работа генератора основывается на программной реализации модели абстрактного автомата. В коде имеются объекты, отвечающие за магазинную память и таблицу переходов. На вход автомата будут поступать объекты из исходного XML, а на выходе получаться объекты для результирующего XML. Кроме того, может понадобится получение дополнительной информации для уточнения данных представления.

Для считывания XML используется библиотека потоковой обработки XML для языка Java — StAX. Базовая реализация StAX входит в состав

стандартной поставки JavaSE начиная с 6 версии. Работа с XML в StAX основана на событиях, возникающих на входном потоке.

Для записи выходного представления в XML используется технология связывания данных. Связывание данных заключается в том, что на основе описания данных XML в виде схемы генерируется исходный код, описывающий в удобном объектно-ориентированном виде данные, хранимые в XML. Используя этот код, можно выполнять преобразование данных из XML в объекты в памяти и преобразование данных из памяти в XML. Связывание данных реализуется на основе технологии Java Architecture for XML Binding (JAXB). Эта технология включается в поставку Java EE, начиная с версии 6. Кроме того, интерфейсы для работы с ней стандартизованы, и имеется несколько реализаций, в том числе, входящая в JRE.

Для реализации абстрактного автомата основным классом будет состояние автомата. В состоянии на основе входных данных будут осуществляться переходы, получаться новые состояния и формироваться выходные данные.Так как состояние само по себе является активным объектом, то магазинную память можно реализовать с помощью шаблона проектирования «цепочка ответственности». Входные события, возникающие на XML можно разделить на открывающиеся теги и закрывающиеся. Это вполне соответствует признаку e во входном сигнале автомата.

В качестве оценки эффективности использования разработанных представлений можно выбрать время работы утилиты-анализатора. Для сравнения быстродействия можно воспользоваться абстрактным синтаксическим деревом в качестве второго источника данных. Для этого можно реализовать на нем один из готовых анализаторов высокоуровневых представлений. Был выбран анализатор «Контроль разделения ответственности» (BigClassAnalyzer), выполняющий поиск классов, ответственность которых слишком высока. Этот анализатор предлагает выполнить рефакторинг таких классов.

Для более точного подсчета временных оценок необходимо собирать статистические данные на большом количестве запусков утилит, чтобы исключить случайную составляющую. Эти данные на практике доказывают, что использование высокоуровневых представлений дает прирост в производительности в сравнении с использованием AST. Величины среднего времени работы анализатора отличаются в 10 раз.

В четвертой главе предлагаются математические модели и алгоритмы выполнения анализов над полученными промежуточными представлениями для решения предложенных ранее задач рефакторинга и визуализации.

Для программной реализации техник рефакторинга требуется разработать алгоритмы, которые позволят подсчитать численные величины, на основе которых можно сделать вывод о том, стоит ли производить рефакторинг участка кода или нет. Согласно книге М. Фаулера, от «жадных» функций/методов нужно избавляться путем переноса их в другой класс. Численный алгоритм выполнения такого анализа можно свести к следующим шагам:

1) Для каждого метода проекта $m \in M$ формируется отношение всех вызовов внутри этого метода $GM = \{(m, g) \mid m \in M, g \in G\}$, где G - множество

элементов потока управления, являющихся вызовами типа «Getattr» (вызовы, которые обращаются к переменной, полю или параметру).

2) На основе GM вычисляются пары, обращающиеся к одной цели с помощью функции $s(m,v)=\{(m,g)\,|(g,v_i)\in GV, v_i=v\}$, где $GV=\{(g,v)\,|g\in G, v\in V\}$ - отношение всех getattr-вызовов и их целей, V – множество всех целей вызовов такого типа.
3) Обозначим за $\varphi(m)$ функцию, вычисляющую множество P_m для данного метода m – $\varphi(m)=P_m$. $P_m=\{|s(m,v_i)|\,|\forall\, v_i \textit{ определенных в методе } m\}$. Для всего метода вычисляется функция $\varphi(m)$ и формируется множество P_m.
4) Для метода рассчитывается значение функции $\epsilon(m)$ для элементов множества, получаемого из функции $\varphi(m)$.

$\epsilon(m)=k_m$, $k_m=p_{m,max}$, $p_{m,max}\in P_m$, $\forall\, p\in P_m\; p_{m,max}\geq p$

Полученный результат будет численной величиной, характеризующей жадность метода. Для решения задачи поиска входных значений k для анализатора нужно провести вычислительный эксперимент. В качестве тестовых проектов для анализа можно использовать различные проекты с открытым исходным кодом. В результате было получено, что наиболее подходящие значения k для начала исследования проекта лежат в диапазоне $[5,8]$.

Часто необходимо проконтролировать, что объекты определенных классов не создаются в ненужных местах, что может нарушить инкапсуляцию и привести к ошибкам. Можно предложить математическую модель выполнения такого анализа и алгоритм для реализации в программном коде:

1) Для каждого класса $c\in C$ проверяются все методы проекта $m\in M$
2) Для метода m формируется множество вызываемых в нем конструкторов $DM=\{(m,d)\,|m\in M, d\in D\}$, где D - множество элементов потока управления, являющихся вызовами типа «Direct» (вызовы методов напрямую, конструкторы класса).
3) Для метода m на основе DM формируется множество вызываемых конструкторов выбранного класса с помощью функции $w(c,m)=\{(m,d)\,|(m,d,c_i)\in CONSTRUCT, c_i=c\}$, где $CONSTRUCT=\{(m,d,c)\,|m\in M, d\in D, c\in C\}$ - отношение всех direct-вызовов внутри потока управления метода $m\in M$, являющихся конструкторами класса $c\in C$.
4) По всему проекту для класса рассчитывается функция $r(c)=\sum_{i=1}^{n}|w(c,m_i)|$.

Полученный результат будет численной величиной, показывающий в скольких местах создавался объект класса. Если эта величина не превышает заданный порог r, то можно считать, что класс создается в ограниченном количестве мест.

Для определения оптимальной входной величины порога r для анализатора нужно провести вычислительный эксперимент. В качестве тестовых проектов для анализа можно использовать различные проекты с открытым исходным кодом. В результате вычислительного эксперимента было получено,

что входные значения наиболее подходящие для начала исследования лежат на участке [2, 4].

Для визуализации трасс и графа потока управления были разработаны две утилиты на языке Python. Для обработки XML используется распространенная библиотека lxml. Для построения картинки используется утилита dot пакета Graphviz. Библиотека PyDot позволяет работать с текстовым форматом dot. В результате визуализации получается набор картинок в векторном формате svg. Архитектура таких утилит в формате диаграммы компонентов UML представлена на рисунке 3.

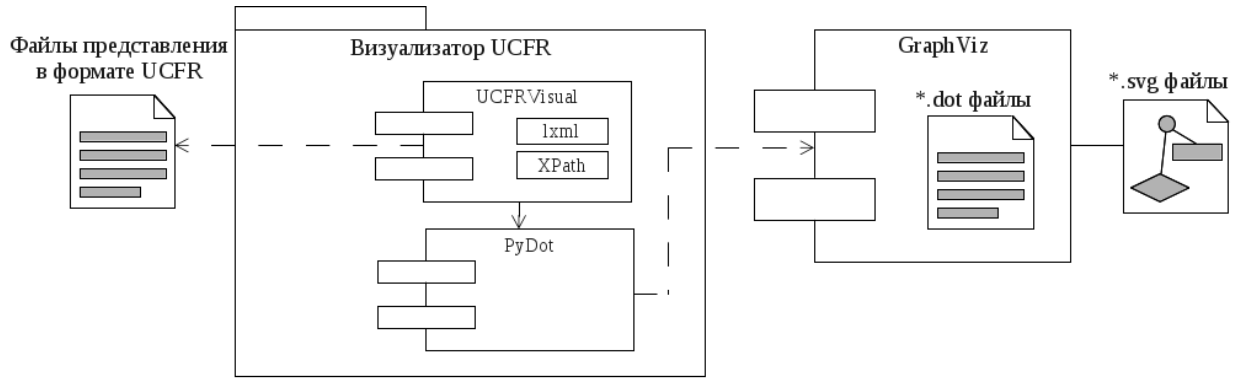

Рисунок 3. Диаграмма компонентов UML утилит визуализации

Особый интерес при анализе потока управления программы представляет анализ трасс. *Трасса* – это последовательность вызываемых функций или методов, которая может существовать при выполнении программы. Определенная трасса соответствует некоторому возможному сценарию исполнения анализируемой программы.

Исследуемый проект представляется в виде графа вызовов CG. Узлами графа выступают функциональные блоки (функции или методы классов), ребрами — вызовы между ними. Множество узлов $ExecNodes$ совпадает с множеством уникальных идентификаторов методов или функций - $UCFR$. Определим отношение $ExecEdges \subseteq UCFR \times UCFR$, в которое входят все пары идентификаторов функций/методов проекта, между которыми происходят вызовы.

$$ExecEdges = \{(f1, f2) | f1 \text{ вызывает } f2, f1 \in UCFR, f2 \in UCFR\}$$

Множество ребер графа вызовов представлено в этом отношении. Тогда весь граф вызовов можно представить в виде $CG = (ExecNodes, ExecEdges)$.

Трасса — это возможный путь на графе CG. Поиск трассы между двумя функциональными блоками f_0 и f_n можно представить в виде поиска пути $(f_0, f_1, \ldots, f_{n-1}, f_n)$ на ориентированном графе. Трасса задается пользователем в виде кортежа, в которой входят узлы графа вызовов в порядке, в котором они расположены вдоль пути в графе, соответствующем трассе $T = (f_0, f_1, \ldots, f_{n-1}, f_n)$, где T – трасса. Трасса является участком графа вызовов программы.

Можно ввести отношение, в котором представлены блоки потока управления функций и методов $Blocks = \{(f, b) | f \in UCFR, b \in \mathbb{N}\}$, где b – идентификатор блока потока управления f. Введем отношение в котором представлены потоки управления функций и методов $Flows = \{(f, b1, b2) | f \in UCFR, b1, b2 \in \mathbb{N}\}$, где тройка $(f, b1, b2)$ представляет поток управления между блоками $b1$ и

$b2$ функции или метода f. Также можно ввести отношение, которое описывает в каких блоках функций происходит вызов других функций, получая переходы между блоками на графе вызовов. $BlockCalls = \{(f1, b, f2) | f1, f2 \in UCFR, b \in \mathbb{N}\}$, где b – идентификатор блока потока управления функции или метода $f1$, в котором происходит вызов функции или метода $f2$. Важно отметить, что для одной и той же трассы на уровне функций и методов может быть несколько путей распространения на уровне блоков потока управления.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ И ВЫВОДЫ

1. Разработана математическая модель и формат универсального промежуточного представления потока управления, отличающаяся тем, что она одинаково описывает исходный текст на нескольких входных языках программирования, и сочетает в себе свойства, позволяющие строить граф вызовов. Использование такого представления позволяет создавать инструменты анализа, общие для всех поддерживаемых языков, что даже при 2 входных языках уменьшает затраты на создание таких инструментов в 2 раза.
2. Предложен метод и алгоритм получения промежуточного представления, основанный на использовании абстрактного цифрового автомата с памятью, отличающегося использованием магазинной памяти для записи состояния автомата, в отличие от канонической модели, а входные и выходные сигналы позволяют считывать абстрактное синтаксическое дерево разбора и формировать дерево универсального представления. Наличие готовой обобщенной модели упрощает создание генераторов представлений для новых входных языков. Разработанный метод анализа исходного текста с целью получения промежуточного представления основан на функционале компилятора с открытым исходным кодом, не требующем дополнительной верификации.
3. Предложен и реализован на практике метод и алгоритм проверки эффективности использования предложенного универсального промежуточного представления в экспериментальных задачах рефакторинга. Вычислительный эксперимент показал, что применение разработанного промежуточного представления ускоряет выполнение анализа, в среднем, в 10 раз.
4. Разработаны методы и алгоритмы выполнения анализа над исходным кодом с использованием разработанных промежуточных представлений. Для двух входных языков (Java и Python) они требуют только одной реализации и работают абсолютно одинаково.
5. Реализован программный комплекс статического анализа, реализующий предложенные модели и алгоритмы получения и обработки универсального промежуточного представления в 9 анализаторах. С помощью вычислительного эксперимента получены входные значения для этих методов анализа. Этот программный комплекс внедрен в разработку продуктов и сервисов направления решений в образовании ЗАО «Нау-сервис» (ГК NAUMEN).

Перспективы дальнейшего развития темы. В ходе дальнейших исследований планируется разработка модели представления, описывающего исходный код на функциональных языках программирования с учетом их ключевых особенностей: функции высших порядков, лямбда-выражения, каррирование, сопоставление с образцом. Также важным направлением является изучение промежуточного представления для больших проектов с помощью методов анализа данных.

ПУБЛИКАЦИИ, ОТРАЖАЮЩИЕ РЕЗУЛЬТАТЫ РАБОТЫ

Основные положения диссертационного исследования опубликованы в следующих работах:

В рецензируемых журналах из списка ВАК

1. Применение универсальных промежуточных представлений для статического анализа исходного программного кода / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Доклады ТУСУР. – 2013. – Т. 27, №1. – С. 64-69.
2. Получение типов данных в языках с динамической типизацией для статического анализа исходного кода с помощью универсального классового представления / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. – 2013. – №2. – С. 66-75.
3. К вопросу об автоматическом комментировании на естественном языке исходных текстов программ / М.В. Зубов, О.А. Машин, А.Н. Пустыгин, Ю.К. Язов // Программная инженерия. – 2013. – №11. – С. 17-21.
4. Численное моделирование анализа исходного кода с использованием промежуточных представлений / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. – 2014. – №4. – С. 55-66.
5. Математическое моделирование универсальных многоуровневых промежуточных представлений для статического анализа исходного кода / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Доклады ТУСУРа. – 2014. – Т. 33, №3. – С. 94-99.
6. Применение пороговых функций для извлечения информации о типах данных при получении промежуточных представлений текстов программ для языков с динамической типизацией / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Инфокоммуникационные технологии. – 2014. – Т. 12, № 4 – С. 51-56.
7. Использование абстрактного цифрового автомата для получения универсального промежуточного представления исходного кода программ / М.В. Зубов, А.Н. Пустыгин // Вестник Астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика. – 2015. – №4 – С. 57-65.
8. Анализ путей исполнения программ по исходному коду с использованием универсальных промежуточных представлений / М.В. Зубов, А.Н. Пусты-

гин // Известия Юго-Западного государственного университета. – 2015. – Т. 61, №4 – С. 12-19.

Свидетельства о регистрации программ для ЭВМ

9. Свидетельство о государственной регистрации программ для ЭВМ № 2014663489. Генератор универсального классового промежуточного представления в формате XML из представления AST исходного текста на языке Java / М.В. Зубов, А.Н. Пустыгин – Заявка № 2014663489; зарегистрирована в Реестре программ для ЭВМ 24.12.2014.

В других изданиях

10. Прототипы построителей промежуточных представлений исходных текстов программ, основанные на компиляторах с открытым исходным кодом / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Седьмая конференция «Свободное программное обеспечение в высшей школе». Тезисы докладов. – М.: Альт-Линукс, 2012. – С. 82-86.
11. Статический анализ ПО с помощью его промежуточных представлений и технологий с открытым исходным кодом / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Материалы второй научно-практической конференции FOSS LVIV 2012. – Украина, Львов: Львів, 2012. – С. 165-168.
12. Подходы к статическому анализу открытого исходного кода / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Сборник материалов Восьмой Международной конференции разработчиков и пользователей свободного программного обеспечения Linux Vacation / Eastern Europe. – Беларусь, Брест: Альтернатива, 2012. – С. 36-40.
13. Выделение типов в универсальном классовом представлении для статическгого анализа исходного кода / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Восьмая конференция «Свободное программное обеспечение в высшей школе». Тезисы докладов. – М.: Альт-Линукс, 2013. – С. 53-58.
14. Получение типов данных в динамических языках при статическом анализе / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Тверские интернет технологии. Научно-практическая конференция: Сборник трудов. – Тверь: Тверской гос. Университет, 2013. – С. 53-57.
15. Сравнительный анализ существующих инструментов исследования программ по исходному коду / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Суперкомпьютерные технологии и открытое программное обеспечение. – 2013. – №1. – С. 37-44.
16. Краткий анализ и исследование помежуточных представлений исходного текста программ / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Суперкомпьютерные технологии и открытое программное обеспечение. – 2013. – №1. – С. 45-53.
17. Построение универсального представления графа потока управления для статического анализа исходного кода / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Девятая конференция «Свободное программное обеспечение в высшей школе». Тезисы докладов. – М.: Альт-Линукс, 2014. – С. 46-51.

Диссертант  М.В. Зубов