На правах рукописи

БИТНЕР Вильгельм Александрович

ИССЛЕДОВАНИЕ И РЕАЛИЗАЦИЯ МОДЕЛИ СТАТИЧЕСКОГО АНАЛИЗА НАХОЖДЕНИЯ СОСТОЯНИЯ ГОНКИ В МНОГОПОТОЧНЫХ АЛГОРИТМАХ С ИСПОЛЬЗОВАНИЕМ ЛИНЕАРИЗОВАННОГО ГРАФА ПОТОКА УПРАВЛЕНИЯ

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Москва – 2014

Работа выполнена на кафедре информатики федерального государственного автономного образовательного учреждения высшего профессионального образования «Московский физико-технический институт (государственный университет)»

Научный руководитель: доктор физико-математических наук, профессор
Тормасов Александр Геннадьевич

Официальные оппоненты: доктор технических наук, профессор
Дроздов Александр Юльевич
Институт точной механики и вычислительной техники им. С.А. Лебедева РАН, заместитель руководителя Департамента исследований и разработок, руководитель Лаборатории оптимизирующих компиляторов

кандидат физико-математических наук
Протасов Станислав Игоревич
ООО «Параллелз Рисерч»,
старший инженер-программист

Ведущая организация: Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Казанский национальный исследовательский технологический университет» (ФГБОУ ВПО "КНИТУ")

Защита состоится 18 декабря 2014 г. в 16:00 часов на заседании диссертационного совета Д 002.017.02 при Федеральном государственном бюджетном учреждении науки «Вычислительный центр им. А.А. Дородницына Российской академии наук», расположенном по адресу: 119333, г. Москва, улица Вавилова, 40.

С диссертацией можно ознакомиться в библиотеке и на официальном сайте (http://www.ccas.ru) ВЦ РАН.

Автореферат разослан _____ 2014 г.

Ученый секретарь
диссертационного совета Д 002.017.02
д.ф.-м.н., профессор Рязанов В.В.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы

Среди наиболее сложных для обнаружения ошибок в многопоточных программах являются состояния конкурентного доступа к памяти или состояния гонки (race condition, RC). *Race condition* – ситуация, когда несколько потоков одновременно обращаются к одному и тому же ресурсу, причем хотя бы один из потоков выполняет операцию записи, и порядок этих обращений точно не определен. Наличие состояний гонок приводит к недетерминированному поведению программы.

Методики поиска состояний гонок обычно разделяют на *статический анализ*, *динамический анализ*, *проверку на основе моделей и аналитического доказательства корректности программ*. Динамический анализ посвящен анализу эмпирических запусков программы на определенном наборе тестов и входных данных. Существуют довольно много коммерческих и бесплатных программных утилит, которые позволяют проводить динамический анализ на разных платформах и архитектурах: Intel Thread Checker, CHESS и т.д. Недостатком данного вида анализа является невозможность анализа больших программных систем, т.к. количество необходимых тестов экспоненциально растет в зависимости от количества входных параметров.

Статический анализ – это анализ без исполнения программы. Преимуществом данного вида анализа является возможность проверки и устранения ошибок в коде во время разработки, глубина анализа как правило выше, что позволяет гарантировать более высокую надёжность конечного программного продукта. В качестве недостатка статического анализа можно выделить сложность применения в промышленных масштабах и не самую лучшую надежность методов анализа, которые не давали бы ложные срабатывания: обнаружение состояний гонок, когда их на самом деле нет либо упущение реально присутствующих состояний гонок.

В рамках диссертационной работы за основу взята математическая модель статического анализа многопоточных программ, описанных в различных работах А.Г. Тормасова, М.Ю. Кудрина, А.С. Прокопенко, Н.В. Заборовского. Математическая модель строится на основе анализа графа совместного исполнения потоков, построенного по определённым участкам кода программы. Результаты исследований зарекомендовали себя как перспективный и эффективный метод поиска потенциальных угроз состояния гонки. Однако представленный метод хорошо работает на линейных участках и имеет ряд ограничений при анализе более сложных конструкций кода, таких как циклы, условные конструкции, алиасы. Реализация и исследование математической модели для применения в промышленных комплексах программ остаются актуальной задачей, в рамках которой необходимо разрешить ограничения представленного метода и, как следствие, расширить класс применимости метода среди промышленного ПО.

Цель работы и задачи исследования

В более поздних работах уже была предпринята попытка приблизить математическую модель к промышленному использованию метода анализа. Однако реализация и соответствующая математическая модель, выполненная с использованием высокоуровневого промежуточного представления не поддерживаемой системы и не имеющая широкого использования, не позволяет применять метод в промышленной разработке ПО. С другой стороны, современная промышленная разработка ПО не обходится без использования оптимизирующих компиляторов, которые используют промежуточное представление программы (intermediate representation – IR) для проведения оптимизаций над кодом программы. Таким образом, использование IR промышленного компилятора в качестве основы для проведения анализа позволило бы значительно улучшить применимость представленного метода. Существенную роль в методе может сыграть оптимизации компиляторов, которые могли бы создать наилучший контекст IR перед применением метода анализа многопоточных программ.

Целью диссертационной работы является исследование, адаптация и реализация математической модели статического анализа многопоточных программ.

Для достижения поставленной цели в работе решаются следующие задачи:

- Исследование и анализ современных оптимизирующих компиляторов и их оптимизаций. Выявление наиболее подходящих оптимизаций для создания такого контекста IR, который содержал бы наименьшее количество изменений потока управления, т.е. циклов и условных конструкций.
- Разработка дополнительных оптимизаций над графом потока управления, не влияющих на анализируемый контекст программы в исследуемом методе статического анализа.
- Адаптация математической модели и метода анализа под новый контекст IR.
- Реализация математической модели в контексте промышленного IR в виде автоматизированной системы обнаружения состояний гонок в программе.

Объект исследования

В работе объектом исследования является процедура поиска состояний гонок в многопоточных алгоритмах, а также оптимизация анализируемого контекста на промежуточном представлении с помощью математической модели, основанной на графах совместного исполнения потоков.

Предмет исследования

В работе предметом исследования является математическая модель, основанная на графах совместного исполнения потоков, построение которых осуществляется по коду программ на языках Си и Си++.

Метод исследования

В работе применяются и используются методы теории множеств, теории графов, теории компиляции, методы программирования на языках высокого уровня.

Научная новизна

Научная новизна работы заключается в применении и адаптации математической модели статического анализа на графах совместного исполнения потоков на промежуточное представление программы оптимизирующего компилятора, что позволило упростить анализ и повысить применимость в промышленных комплексах программ.

Разработанный метод линеаризации графа потока управления программы для статического анализа позволил существенно сократить анализируемый контекст программы, улучшив качества статического анализа. Работа имеет принципиальную новизну, как в постановке задачи, так и в выборе методов решения поставленных задач.

Положения, выносимые на защиту

В процессе исследования получены следующие новые научные результаты, выносимые на защиту:

1. Программная реализация системы верификации состояния гонок в многопоточных алгоритмах на основе IR промышленного компилятора, которая позволяет получать IR программ различных языков высокого уровня, разных платформ и архитектур. Использование общепризнанного и популярного компилятора гарантирует применимость метода к широкому классу задач на долгие годы вперед. Возможность получения унифицированного IR программ, ориентированных на такие архитектуры как Intel, ARM, Power, позволяет значительно расширить класс применимости метода в среде коммерческого ПО. Как следствие, появляется весомое средство оценки качества при разработки коммерческого ПО.
2. Адаптированная математическая модель верификации многопоточных алгоритмов.
3. Метод линеаризации графа потока управления для представленного анализа. Особенность представленного метода статического анализа позволила создать дополнительную оптимизацию графа потока управления, с помощью которой удалось в некоторых случаях избавиться от лишних узлов графа потока управления и, как следствие, уменьшить количество ветвления управления на графе.
4. Комплекс программ и результаты вычислительных экспериментов для выявления наличия гонок в некоторых многопоточных алгоритмах, в том числе неблокирующих.

Практическая ценность

Усовершенствована методика нахождения состояний гонок в многопоточных алгоритмах, реализация которых используется в коммерческом ПО. Разработанные программные модули позволили сделать методику нахождения состояний гонок в промышленном ПО более автоматизированной, надежной и универсальной, что было достигнуто благодаря использованию промежуточного представления оптимизирующего компилятора LLVM.

Выявленные и предложенные оптимизации LLVM позволили получать наиболее подходящий контекст для применения анализа с точки зрения производительности поиска состояний гонок в алгоритме.

Предложенная методика линеаризации графа потока управления, которая используется после оптимизаций LLVM и перед основным анализом, позволила расширить класс задач, пригодных для анализа на предмет наличия состояний гонок исследуемого метода.

Таким образом, программная реализация новшеств и улучшений позволила расширить класс задач, для которых применим метод статического анализа на основе графа совместного исполнения потоков. Как следствие, данный метод анализа состояний гонок можно более широко использовать для верификации промышленного ПО.

Публикации и апробация результатов

По теме диссертации опубликовано 9 печатных работ, в том числе 3 в рецензируемых изданиях, рекомендованных ВАК РФ для опубликования основных научных результатов диссертаций.

Публикации в ведущих рецензируемых журналах, входящие в перечень ВАК:

1. Битнер В.А., Тормасов А.Г. Анализ и сокращение промежуточного представления программы в модели статического анализа нахождения состояния гонки в многопоточных алгоритмах // Вестник КГТУ им. А.Н. Туполева, 2014. – №3. – С. 203-212.
2. Битнер В.А., Тимербаев Н.Ф. Контекстно зависимая линеаризация графа потока управления в статическом анализе состояний гонок в многопоточных алгоритмах // Вестник Казанского технологического университета, 2014. – Т. 17, №15. – С. 187-192.
3. Битнер В.А., Заборовский Н.В. Построение универсального линеаризованного графа потока управления для использования в статическом анализе кода алгоритмов // Моделирование и анализ информационных систем, 2013. – Т. 20, №2. – С. 166-177.

Результаты работ докладывались автором на научных конференциях и семинарах:

1. Труды 57-й научной конференции МФТИ «Актуальные проблемы фундаментальных и прикладных наук в современном информационном обществе» (Москва, 2014).
2. Научная конференция «Математическое моделирование и информатика» при ФГБОУ ВПО МГТУ «СТАНКИН» (Москва, 2014).
3. The 8th Congress of the International Society for Analysis, its Applications, and Computation (Москва, 2011).
4. Международной заочной научно-практической конференции «Актуальные проблемы науки» (Тамбов, 2011).
5. Труды 52-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук» (Москва, 2009).
6. XXXVI международная молодежная научная конференция «Гагаринские чтения» (Москва, 2009).

Структура и объем диссертации

Диссертационная работа состоит из введения, трех глав, заключения, одного приложения и списка использованных источников. Список использованных источников включает 37 наименований. Общий объем работы составляет 103 страницы. Объем приложений составляет 12 страниц.

СОДЕРЖАНИЕ РАБОТЫ

Введение

Во введении обоснована актуальность темы, сформулирована цель работы и определен перечень решаемых задач, указана новизна научных изысканий, отмечены особенности подхода, раскрываемого в диссертационной работе, практическая ценность полученных решений и разработок, а также дан краткий обзор содержания по главам.

Первая глава

В первой главе даются необходимые определения и понятия, которыми оперируют в работе. В первой главе приводится аналитический обзор программных разработок и литературы по теме диссертации. Проведен анализ оптимизирующих компиляторов и типов промежуточных представлений, который позволил сформулировать подход к разработке программных модулей для имплементации метода анализа состояний гонок на основе графа совместного исполнения потоков.

Оптимизация – это эквивалентное преобразование над промежуточным представлением компилятора.

Промежуточное представление – структура данных, фиксирующая состояние(я) программы в процессе компиляции от исходной записи на входном

языке до выходного состояния – целевого исходного кода программы, исполняемой на заданной платформе.

Граф потока управления (CFG) – аналитическая структура, которую логически можно представить, как управляющую надстройку над промежуточным представлением. В реализации наиболее распространенной схемой является представление управляющего графа как отдельного объекта, имеющего взаимно однозначное соответствие с операционной семантикой промежуточного представления, в которой выражена вся полнота семантики передачи управления. Представляет собой направленный связный граф, каждый узел (вершина) которого соответствует линейному участку, а дугам – управляющие связи между ними, отображающие передачу управления.

Оптимизирующие компиляторы занимают важную роль в современной промышленной разработке, позволяя максимально эффективно использовать аппаратные особенности платформ и архитектур, на которых предполагается исполнение программ. В первой главе рассматриваются различные оптимизирующие компиляторы: коммерческие таких компаний как Intel, Sun Microsystems, Transmeta, Microsoft, IBM, HP, Elbrus; бесплатные с открытым кодом такие как GCC и CLANG&LLVM. В работе делается обоснованный выбор оптимизирующего компилятора CLANG&LLVM.

В первой главе при анализе оптимизирующих компиляторов делается вывод о возможности использовать и адаптировать некоторые оптимизации компилятора с целью получения наиболее удобного и эффективного представления программ, написанных на языке высокого уровня, что позволит решить задачу линеаризации CFG для повышения эффективности и расширения применимости исследуемого метода статического анализа многопоточных алгоритмов на предмет состояния гонки.

Помимо анализа оптимизирующих компиляторов в главе 1 приводится обзор существующих средств поиска состояний гонок в программах. Уделяется внимание слабым и сильным качествам тех или иных подходов и технологий. Делается вывод о текущем положении статических анализаторов, что на практике они дают неточный ответ о наличии того или иного свойства в программе. Основной принцип статического анализа: либо точность, либо время. Выбор оптимального сочетания времени работы анализатора и точности является практической задачей, которая разрешается в каждом случае отдельно.

В главе 1 приводится описание исследуемой математической модели поиска состояний гонок в многопоточных алгоритмах, а также вводятся необходимые определения и понятия, связанные с исследуемой моделью.

Граф совместного исполнения потоков – ориентированный граф, представляющий всевозможные варианты совместного исполнения потоков в многопоточном алгоритме, где каждая дуга ассоциирована с атомарной операцией

одного из потока, а вершины – с множеством состояний общей памяти после выполнения очередной атомарной операции. В случае двух потоков исполнения граф можно описать следующим образом:

$$G \coloneqq (V, A):\ V = \bigcup_{\substack{i=1,k+1 \\ j=1,n+1}} v_j^i\ ,\quad A = \bigcup_{\substack{i=1,k \\ j=1,n+1}} (v_j^i, v_j^{i+1}) \cup \bigcup_{\substack{i=1,k+1 \\ j=1,n}} (v_j^i, v_{j+1}^i)\ , \qquad (1)$$

где k, n – количество операций первого и второго потока соответственно, i, j – номер атомарной операции первого и второго потока соответственно, V – множество вершин графа, A – множество дуг графа.

Класс эквивалентности – набор полных путей на графе G, для которых состояние ячейки памяти в конечном узле одинаково.

Расчетный граф – конструктивная дискретная модель исходного кода программы, представляющая собой направленный граф, в котором каждому ребру соответствует атомарная операция, а вершинам ставится в соответствие множество значений всех разделяемых переменных.

В более ранних работах по данной тематике была предложена математическая модель обнаружения состояния гонки в многопоточных алгоритмах посредством построения модели взаимного исполнения атомарных инструкций двумя потоками на разделяемой памяти. Общая идея метода сводится к следующим действиям:

1. Для каждого потока выделяются операции, связанные с разделяемыми переменными.
2. Строится граф совместного исполнения потоков на основании операций с разделяемыми переменными в каждом из потоков.
3. Находятся классы эквивалентности полных путей на графе совместного исполнения потоков.
4. Анализируются полные пути на графе совместного исполнения потоков, которые принадлежат разным классам эквивалентности. На основе анализа выбираются пути, для которых возможно состояние гонки.
5. Выделенные пути анализируются, например, с помощью метода неопределенных коэффициентов.
6. По состоянию в финальной вершине делается вывод о наличии гонок.

Гонка (в терминах графа G, описанный в (1)) – существование двух разных полных путей, для которых в финальном состоянии состояния хотя бы одной из ячеек памяти различны. С учётом начальных значений ячеек и того, как их значения меняются, используем метод неопределенных коэффициентов. На каждом ветвлении добавляем коэффициент $\alpha \in \{0, 1\}$ к изменению в левой ветви и $(1-\alpha)$ – правой. Множество неопределенных коэффициентов D полностью описывает путь на графе. В финальной вершине имеем множество значений, где каждая ячейка памяти в общем случае имеет вид:

$$x_i = f_i(\vec{x}_0, D), \quad (2)$$

где $\vec{x}_0$ – состояние всех ячеек памяти в начальной вершине, D – значения неопределенных коэффициентов, f_i – некоторая функция, определенная для каждой из ячеек. Исходя из определения понятия гонки и принципов построения и анализа графа, изложенных выше, получаем, что гонка возможна тогда и только тогда, когда:

$$\exists i, D_1, D_2 : f_i(\vec{x}_0, D_1) \neq f_i(\vec{x}_0, D_2), \quad (3)$$

где D_1 и D_2 – множества бинарных коэффициентов.

Вторая глава

Во второй главе описывается реализация модели статического анализа нахождения состояния гонки в многопоточных алгоритмах с использованием линеаризованного графа потока управления.

Анализ оптимизаций на промежуточном представлении.

В работе был взят за основу компилятор CLANG&LLVM как наиболее активно развивающийся и обладающий рядом преимуществ с точки зрения гибкости реализации и удобства адаптирования под конкретные случаи и задачи. CLANG&LLVM для проведения оптимизаций использует платформонезависимый IR в SSA (Static Single Assignment) форме, причем CLANG выступает front-end'ом, транслируя язык высокого уровня в IR нужного вида (LLVM IR), а LLVM – оптимизирующим компилятором, который проводит необходимые оптимизации, трансформации или программный анализ на LLVM IR и затем транслирует финальный LLVM IR в бинарный код целевой архитектуры.

В данном параграфе второй главы проводится анализ оптимизаций компилятора LLVM. LLVM содержит более 100 уникальных оптимизаций, которые были проанализированы с точки зрения влияния на CFG.

Среди представленных оптимизаций LLVM можно выбрать те, которые могут привести к существенному изменению CFG, а также прочих представлений IR, что в конечном итоге повышает линеаризацию CFG программ и упрощает различные виды статического анализа кода программ на предмет состояния гонок. Стоит заметить, что для успешной линеаризации CFG программ важно не только применить необходимые оптимизации компилятора LLVM, но и правильно подобрать последовательность запусков оптимизаций, так как оптимизации могут быть конфликтующими – применимость одной приводит к неприменимости другой, и наоборот, связанными – неприменимость одной влечёт неприменимость другой.

Таким образом, наибольший интерес с точки зрения влияния на CFG вызывают следующие оптимизации:

- *-lcssa,*
- *-loop-simplify,*
- *-licm,*
- *-loop-reduce,*

- *-loop-unroll,*
- *-loop-unswitch.*

Линеаризация графа потока управления через линейку оптимизаций LLVM

В данном параграфе рассматривается подход использования выявленных оптимизаций на практике. Компилятор позволяет создавать свою собственную линейку оптимизаций либо использовать стандартную. Стандартных линеек оптимизаций у LLVM три: O1, O2, O3. Их также называют уровнями оптимизаций, где O1 – минимальный набор оптимизаций, а O3 – максимальный. Согласно документации LLVM интересуемый набор оптимизаций, выявленный в п.2.1, содержится во втором уровне оптимизации LLVM – O2. Таким образом, чтобы получить минимальный эффект от данных оптимизаций во время компиляции для оценки достаточно использовать второй уровень оптимизации O2.

В качестве примера в данном параграфе рассмотривается CFG-графы взаимоисключающего алгоритма Петерсона для двух потоков, одна из реализации которого на языке Си приводится в работе. Пример линеаризации CFG одного из потока в реализации алгоритма Петерсона приводится на рис. 1.

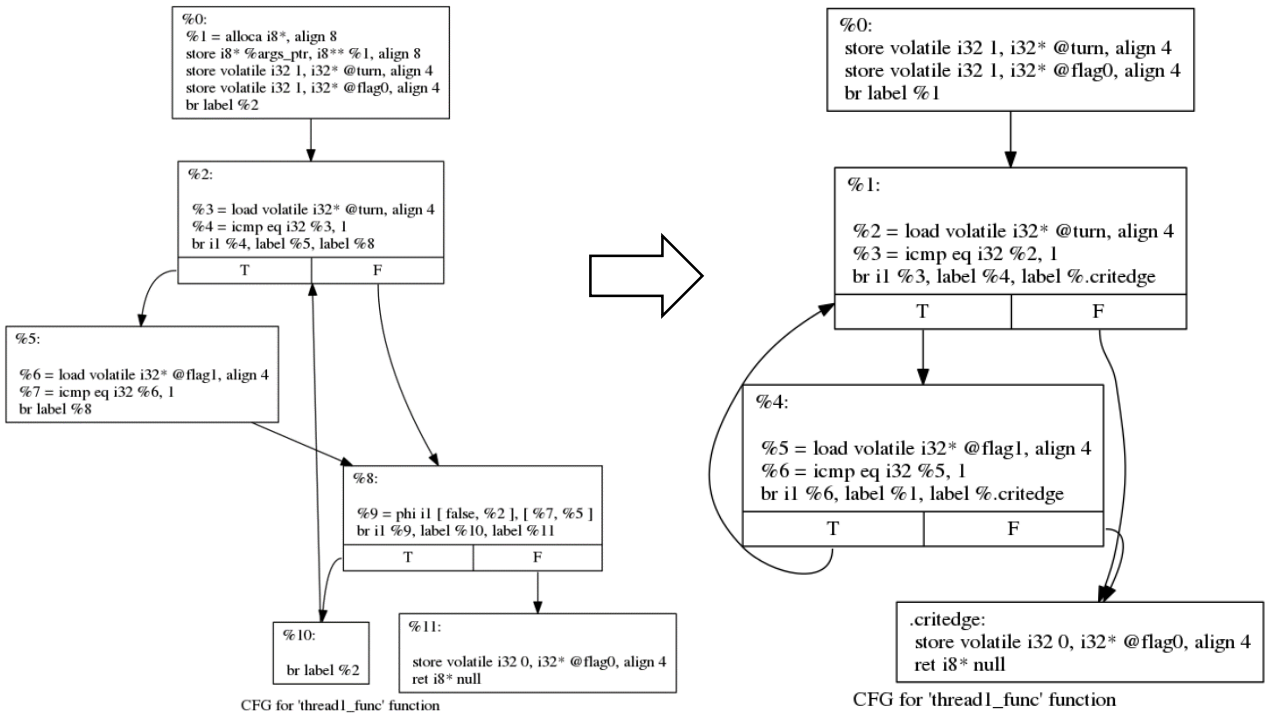

Рис. 1. Линеаризация CFG треда в алгоритме Петерсона

В параграфе также делается акцент на появление в анализируемом контексте φ-узлов в точках схождения потока управления, которые никак ранее в математической модели метода не учитывались. *φ-узел для переменной* – это операция, выбирающая среди множества значений переменной нужное.

Анализ инструкций SSA-формы LLVM IR, не влияющих на математическую модель поиска RC.

Исследуемый метод статического анализа многопоточных алгоритмов основывается на графе совместного исполнения потоков, узлы которого – атомарные операции. Данная особенность метода позволяет иначе взглянуть на промежуточное представление программы. Можно смело утверждать, что некоторые инструкции IR программы могут быть удалены из анализируемого контекста программы в силу их не влияния на процесс анализа в рамках исследуемого метода. В данном параграфе главы 2 анализируются инструкции промежуточного представления LLVM IR, и по результатам анализа делается вывод, какие инструкции важны для статического анализа, а какие нет.

На промежуточном представлении атомарными операциями являются инструкции LLVM IR в SSA-форме. В свою очередь из определения о RC следует, что конкурентный доступ к памяти осуществляется через атомарные операции чтения/записи, которые в LLVM IR представляют собой инструкции load и store. На промежуточном представлении атомарными операциями являются инструкции LLVM IR в SSA-форме. В свою очередь из определения о RC следует, что конкурентный доступ к памяти осуществляется через атомарные операции чтения/записи, которые в LLVM IR представляют собой инструкции load и store. Таким образом, данные инструкции оперируют и с регистрами, и с памятью, что доказывается следующее утверждение.

Утверждение 1. *Инструкции, оперирующие только с регистрами (виртуальными регистрами или временными переменными в SSA-форме), не представляют интереса в методе статического анализа, т.к. не влияют на возникновения RC.*

В данном параграфе уделяется внимание классу инструкций, которые атомарно делают несколько операций, в том числе и операции с памятью, такие как CAS (Compare and Swap/Set). Такие инструкции поддерживаются как на уровне микропроцессоров, так и на уровне ядра операционной системы. В LLVM IR инструкциями, проводящими атомарно несколько операций над памятью, являются следующие: CMPXCHG, ATOMICRMW. Отсюда вытекает следующее утверждение.

Утверждение 2. *Класс атомарных операций типа CAS нельзя удалять из контекста программы, так как они непосредственно могут создать RC в параллельной программе.*

SSA форма промежуточного представления обуславливает наличие φ-узлов. Докажем, что φ-узел не влияет на исследуемый метод статический анализ. В LLVM IR φ-узел называется φ-инструкцией. Особенность SSA-формы LLVM IR включает в себя то, что все φ-инструкции в узлах графа потока управления располагаются в начале, и до них не может быть никаких других инструкций. При этом φ-инструкция имеет следующую форму:

```
phi тип, [значение_1, label метка_1], ..., [значение_N, label метка_N],
```

где значения – это временные переменные SSA-формы, хранящие результат исполнения каких-либо инструкций, а метки – узлы CFG, с которых пришло данное значение. Результат исполнения инструкции в SSA располагается на новую уникальную переменную – виртуальный регистр. Таким образом, входные параметры φ-инструкции всегда являются регистрами, т.е. из памяти не читает. Из определения φ-узла и того факта, что φ-инструкции располагаются в начале узла графа потока управления, вытекает доказательство следующего утверждения.

Утверждение 3. *φ-инструкция оперирует только с регистрами и не влияет на исследуемый метод статического анализа многопоточных алгоритмов*.

В данном параграфе уделяется внимание инструкция передачи управления – branch-инструкциям или инструкции перехода. В LLVM IR каждый линейный участок, что есть CFG-узел в графе потока управления, заканчивается соответствующей branch-инструкцией. Проводя анализ семантики существующих branch-инструкций в LLVM IR в параграфе главы 2 делается заключение, что все инструкции оперируют с метками и адресами на код программы, адреса которых расположены на виртуальных регистрах/переменных SSA-формы. Отсюда делается следующее утверждение.

Утверждение 4. *Класс branch-инструкций не влияет на создание неразрешимого состояния гонки, т.к. не оперирует с памятью*.

Удаление branch-инструкций из анализируемого контекста программы дает основание трансформировать ветвление управления, созданное данными branch-инструкциями, в CFG программы. Здесь возникает отдельная проблема корректного изменения ветвления управления в CFG программы, которая рассматривается в следующем параграфе.

Получения сокращенного оптимизированного промежуточного представления LLVM – Reduced Opt LLVM IR.

В данном параграфе главы 2 приводится описание разработанного программного комплекса по анализу LLVM IR и получения сокращенного промежуточного представления программы. В качестве технического средства для проведения анализа было выбрано представление LLVM IR в виде CFG в DOT-формате. В данном формате генерируется CFG средствами оптимизирующего компилятора, что является удобным, т.к. такой формат легко визуализировать множеством утилит, которые представляют DOT-формат в графическом виде.

Разработанный программный комплекс для анализа LLVM IR – анализатор IR – состоит из различных этапов преобразования LLVM IR, как в оригинальном виде, так и в DOT-формате. Общая архитектура анализатора изображена на рис. 2. На вход анализатору поступает код программы, написанной на языке высокого уровня (ЯВУ), который средствами front-end компилятора CLANG транслируется в LLVM IR. В

качестве front-end системы может выступить любой другой транслятор, который создает корректное плаформонезависимое промежуточное представление LLVM IR.

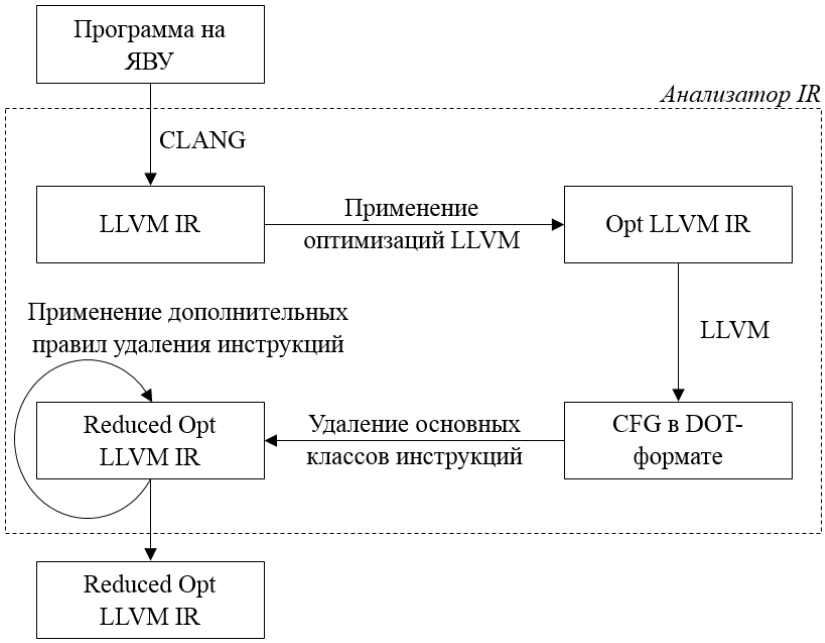

Рис. 2. Архитектура программного анализатора IR.

Согласно первым двум параграфам второй главы выявленные оптимизации LLVM применяются с целью первичной линеаризации CFG и сокращения общего количества инструкций в программе. В конечном итоге получаем новый вид промежуточного представления LLVM IR – Opt LLVM IR. Далее средствами оптимизирующего компилятора LLVM получаем CFG программы, построенное на LLVM IR, в DOT-формате. Выбор формата представления CFG не является принципиальным, и может быть использован любой другой пригодный.

Дополнительные классы инструкций необходимы на тот случай, если основных классов инструкций недостаточно, и обнаружены специфические инструкции в LLVM IR, которые не влияют на статический анализ кода.

Согласно утверждения 1-4 формируется класс удаляемых инструкций. Стоит отметить, что удаление branch-инструкций не влечет изменение в структуре линейных участков CFG, т.е. линейные участки не объединяются и не удаляются. Таким образом структура ветвления управления в CFG сохраняется для последующего этапа анализа графа потока управления.

Этап сокращения промежуточного представления очень эффективен в промышленных комплексах программ, где логика каждого потока более нагружена локальным функционалом, не связанным с разделяемыми переменными.

Для наглядности рассмотрим пример получения Reduced Opt IR в алгоритме Петерсона, полученный с помощью Анализатора IR. Результат получения Reduced Opt IR в графическом виде можно увидеть на рис. 3.

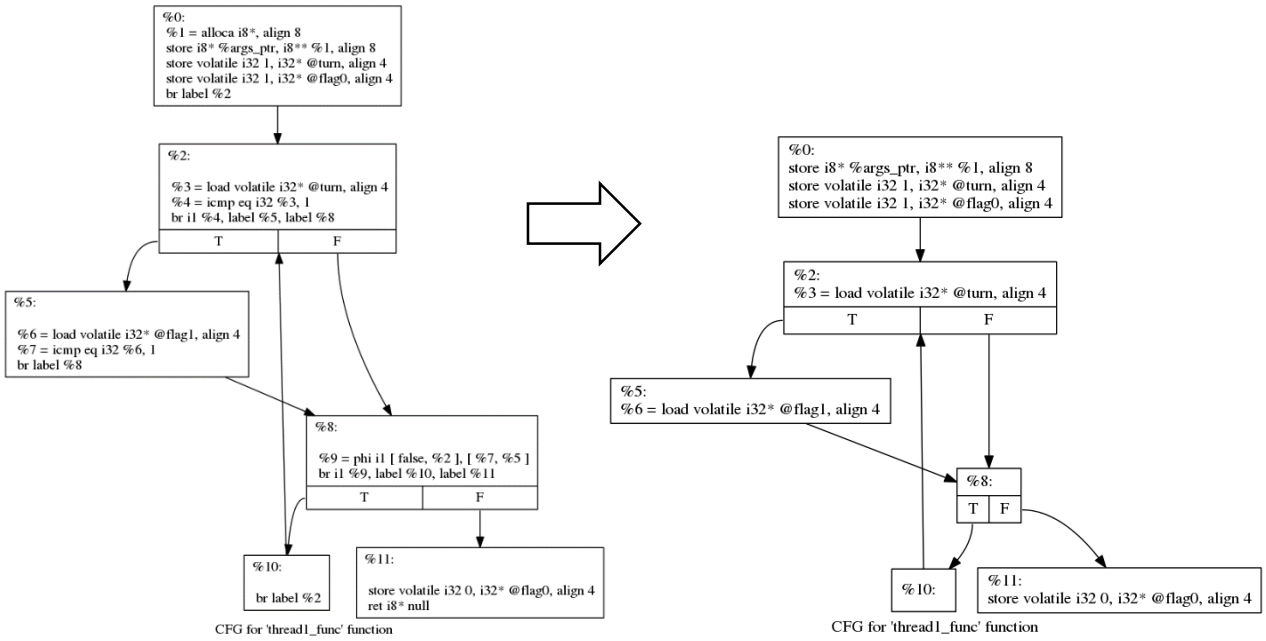

Рис. 3. Пример получения Reduced Opt IR.

Наблюдается не только сокращение количества инструкций, которые надо было бы анализировать в статическом анализе, но и появляются пустые CFG-узлы, которые дают основание для их удаления и, как следствие, проведения более глубокой линеаризации CFG программы.

Контекстно-зависимая линеаризация графа потока управления на сокращенном промежуточном представлении программы.

В данном параграфе доказываются преобразования CFG, которые возможно проводить без ущерба для математической модели статического анализа. Инструкции перехода оперируют только с регистрами, поэтому возможна ситуация, когда некоторые узлы CFG остаются пустыми, т.к. в них изначально не было значимых для статического анализа инструкций, таких как load/store инструкции или атомарные инструкции типа CAS. Отсюда сделаем утверждение.

Утверждение 5. *Узлы CFG, не содержащие значимых инструкций, являются также не значимыми и могут быть удалены из CFG.*

Докажем это утверждение. При удалении CFG-узла возникает нетривиальная задача – как поступить с входящими и выходящими дугами удаляемого узла. Возникают следующие нетривиальные случаи:

1. Выходящая дуга удаляемого CFG-узла является обратной дугой цикла.
2. Из удаляемого CFG-узла выходят несколько дуг в разные непустые CFG-узлы.

Теорема 1. Обратные дуги у пустого CFG могут быть удалены без влияния на математическую модель метода статического анализа.

Доказательство. В работе Н.В. Заборовского была доказана теорема 2: «*для описания цикла в анализирующем графе достаточно одного повторения тела цикла*».

В CFG тело цикла – это все CFG-узлы начиная от головы цикла и до CFG-узла, откуда выходит обратная дуга в голову цикла. Головой цикла называют CFG-узел, куда входит обратная дуга цикла. Отсюда заключаем, что обратную дугу можно удалять без ущерба статическому анализу, что и требовалось доказать.

Теорема 2. При удалении пустого CFG-узла с несколько выходящими дугами нельзя однозначно согласовать изменение потока управления без ущерба математической модели статического анализа.

Доказательство. Проставим каждому ветвлению управления в CFG неопределенный коэффициент β_i^j, где i – номер узла в CFG, j – порядковый номер выходящей дуги, причем $j \in [1, N-1]$, где N – число выходящих дуг. Если из CFG-узла выходит 2 дуги, то неопределённый коэффициент первой дуги – β_i^1, а второй – $(1-\beta_i^1)$. Если из CFG-узла выходит 3 дуги, то неопределённый коэффициент первой дуги – β_i^1, второй – β_i^2, третьей – $(1-\beta_i^1-\beta_i^2)$. Применяя данный принцип присваивания неопределённых коэффициентов к ветвлениям, получаем следующий вид состояния разделяемой ячейки памяти, который изначально был описан в формуле (2):

$$x_i = f_i(\vec{x}_0, D, B),$$

где B – множество коэффициентов β_i^j. К данному описанию ячейки памяти применима, доказанная в работе [4], теорема 1: «*наличие гонки в предлагаемой модели с ветвлениями эквивалентно наличию гонки в задаче*». Отсюда получаем, что состояние гонки на CFG с неопределенными коэффициентами согласно формуле (3) описывается следующим образом:

$$\exists i, D_1, D_2, B: f_i(\vec{x}_0, D_1, B) \neq f_i(\vec{x}_0, D_2, B). \quad (4)$$

Таким образом, в случае наличия гонки в задаче должен существовать фиксированный набор неопределённых коэффициентов β_i^j, который может быть нарушен в случае удалении пустого CFG-узла с несколько выходящими дугами, т.к. любое изменение этих дуг (объединение, удаление и т.д.) влечет изменение соответствующих коэффициентов β_i^j. Отсюда делаем вывод, что удаление данного CFG-узла неоднозначно влияет на математический анализ исследуемого статического анализа, что и требовалось доказать.

На основе теоремы 1 и теоремы 2 был разработан и реализован алгоритм удаления пустого CFG-узла, который описывается следующим образом:

Алгоритм 1. Удаление пустого CFG-узла.

1. Удаляются сначала пустые CFG-узлы, у которых одна выходящая дуга:
   a. Входящие дуги удаляемого CFG-узла переносятся в CFG-узел, следующий за удаляемым CFG-узлом.
   b. Выходящая дуга из удаляемого CFG-узла просто удаляется.
2. Повторяется п.1 до тех пор, пока представление CFG не перестанет меняться, т.е. все пустые CFG-узлы с одной выходящей дугой будут удалены.

3. Пустые узлы, у которых две и более исходящих дуги, остаются в CFG без удаления.

После применения алгоритма 1 мы получаем контекстно-зависимый линеаризованный граф потока управления – Reduced CFG, который является основном контекстном для анализа в исследуемом методе статического анализа. Прежде, чем проводить статический анализ исследуемым методом докажем, что анализируемый контекст является корректным для математической модели метода в следующей теореме.

Теорема 3. Наличие неразрешимого состояния гонки на редуцированном графе потока управления эквивалентно наличию гонки в задаче.

Доказательство. Из описания алгоритма 1 и доказательства теоремы 2 следует, что формулы (4) и (3) эквивалентно описывают состояние гонки в алгоритме. Отсюда следует доказательство теоремы.

Приведем пример контекстно-независимой линеаризации CFG на алгоритме Петерсона.

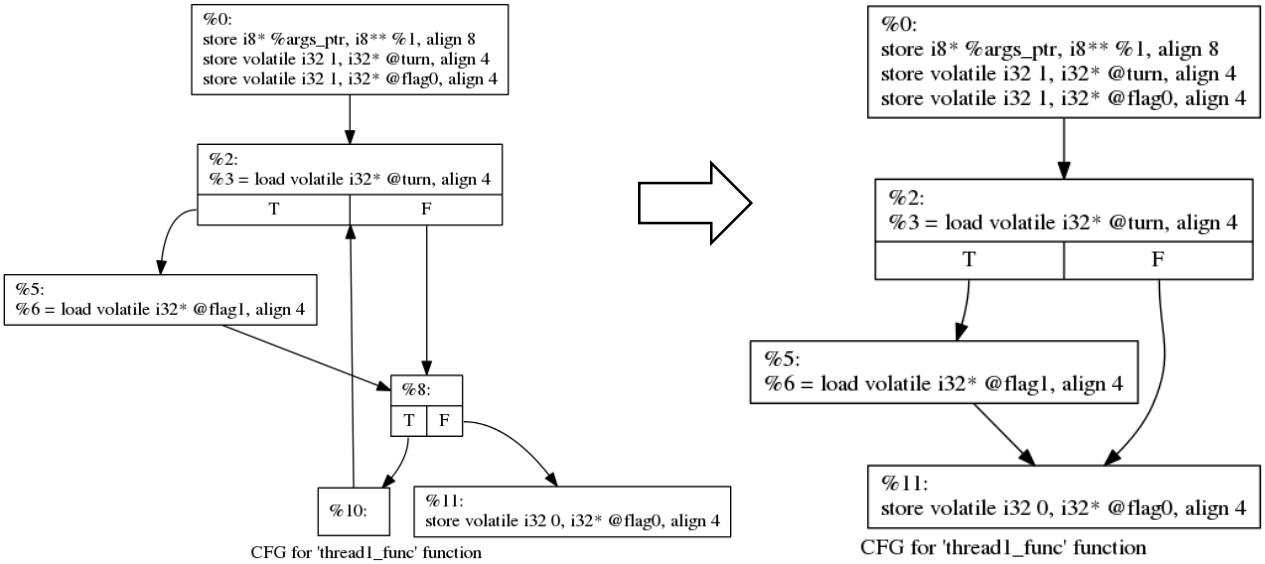

Рис. 4. Контекстно-зависимая линеаризация CFG треда в алгоритме Петерсона.

На рис. 4 осталось ветвление управления, но оно значительно упрощено относительно начального представления. Подобному ветвлению можно снова применить принцип неопределенных коэффициентов для построения расчетного графа, что будет рассмотрено ниже в данной работе.

Поиск состояний гонок в исследуемом методе на Reduced CFG

В данном параграфе приводится описание построения графа совместного исполнения потоков на Reduced CFG согласно алгоритмам, описанным в более ранних работах по данной тематике. Согласно разработанному комплексу программ, имплементирующий алгоритм поиска классов эквивалентности, находятся все

возможные пути исполнения программы на Reduced CFG. Так на алгоритме Петерсона по Reduced CFG (см. рис. 4) было найдено 24 класса эквивалентности.

В дальнейшем процесс поиска состояний гонок связан с анализом расчетного графа, который строится также на Reduced CFG, но учитывает простые ветвления управления. В CFG-узле, в котором более одной исходящей дуги, дугам расставляется неопределённые коэффициенты, также как это проделывается в доказательстве теоремы 2. При анализе путей в графе совместного исполнения потоков дополнительно учитываются данные неопределённые коэффициенты.

При построении расчетного графа вводятся дополнительная функция над вектором состояния системы, атрибуты которого являются значения разделяемых переменных – это функция условного перехода:

$$M: \left(v_j^i, v_j^{i+1}\right) \to P(\vec{x}) = 0, \quad (5)$$

где P – функция-предикат, определенная на множестве значений вектора разделяемых переменных, определяет возможность передвижения системы из текущей вершины по рассматриваемой дуге в расчетном графе. Таким образом, в расчетном графе добавляются дуги-условия, которых в Reduced CFG нет.

Утверждение 6. *В промежуточном представлении LLVM IR инструкциями, которые ассоциируются с дугами-условиями, являются инструкции, подготавливающие аргумент branch-инструкции.*

Доказательство утверждения 1 следует из принципа построения CFG и определения branch-инструкций. Если имеется ветвление управление, то переход осуществляется по условию (branch по условию). В зависимости от условия branch-инструкция осуществляет передачу управления на нужную ветку потока управления.

Утверждение 7. *Для ассоциации дуги-условия достаточно одной инструкции, вычисляющий аргумент для branch-инструкции по условию.*

Доказательство утверждения 2 следует из особенностей промежуточного представления LLVM IR. Поскольку LLVM IR приближен к ассемблерному листингу, то и финальный аргумент передачи управления создается одной инструкцией. В LLVM IR такой инструкцией, например, является fCMP, iCMP.

В дальнейшем в данном параграфе демонстрируется построение расчетного графа и анализ путей согласно алгоритмам, описанным в более ранних работах.

На примере алгоритма Петерсона показывается корректный вывод на основе математической модели статического анализ, адаптированной к анализу контекста на линеаризованном графе потока управления.

Модель автоматизация поиска состояний гонок

В рамках адаптированной математической модели метода статического анализа была предложена и реализована новая модель организации процесса анализа кода программы, который ориентирован на использование LLVM IR. Предложенная модель изображена на рис. 5.

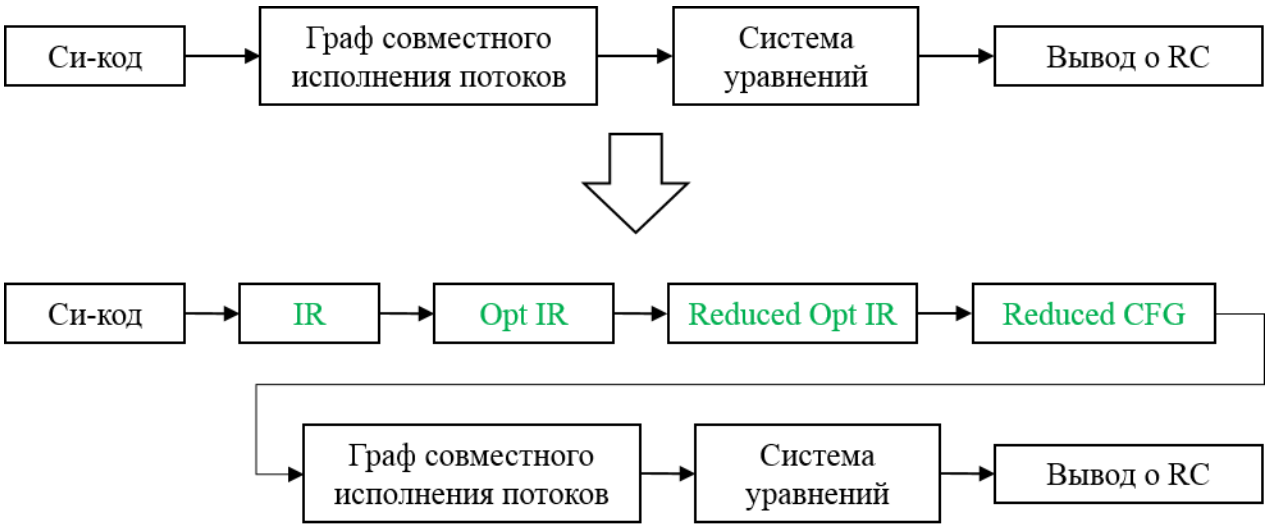

Рис. 5. Модель организации поиска состояний гонок.

Предложенная модель добавляет значительные этапы по подготовке анализируемого контекста. Анализируемый контекст существенно сокращается, но самой сильной стороной является его возможность применить к большому числу промышленных программ. Универсальность LLVM IR позволяет гибко применять метод статического анализа на различных платформах и архитектурах.

В заключительной части второй главы описывается архитектура реализованного комплекса программ, который включает как собственную реализацию утилит на скриптовых языках, так и использование сторонних программ. Предложенная модель позволяет с высокой степень автоматизировать процесс анализа кода программы.

Разработанный комплекс программ по анализу включает в себя реализованный Анализатор IR, который получает на вход код программы, компилируемый компилятором CLANG, после чего анализатор создает контекст программы Reduced CFG, который подается на вход анализатору написанный в системе компьютерной алгебры Wolfram Mathematica. Анализатор в Wolfram Mathematica по алгоритмам, описанным ранее в работах, строит граф совместного исполнения потоков, выделяет классы эквивалентности, представители классов, затем – расчетный граф и анализ путей на нем. На выходе из анализатора на Wolfram Mathematica происходит вывод результатов: отображаются возможные способы исполнения. Если есть пути, в которых все потоки одновременно проходят через критическую секцию, значит автоматически делается вывод о возможности неразрешимого состояния гонки. Архитектура системы поиска состояний гонок изображена на рис. 6.

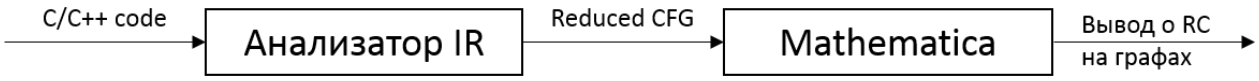

Рис. 6. Архитектура системы поиска состояний гонок.

Третья глава

В третьей главе описывается практическое применения реализованного комплекса программ на различных алгоритмах. Показывается, что на модельных задачах получается верные результаты и демонстрируется корректность подхода, используя промежуточное представление программы и его дальнейшее упрощение.

В третьей главе приводится программный код, подробные результаты процесса получения Reduced CFG, а также результаты анализа Reduced CFG в Wolfram Mathematica. Рассматриваются следующие алгоритмы:

- Спинлок (спин-блокировка);
- Некорректный спинлок;
- Алгоритм Петерсона;
- Стек Трейбера.

По результатам применения прототипа комплекса программ делается заключение, что подход работает корректно на модельных задачах и может быть использован в других реальных задачах с двумя потоками, компилируемых оптимизирующим компилятором CLANG&LLVM.

Приложение

В приложении подробно иллюстрируется работа предложенной модели организации поиска состояния гонок. Приводятся промежуточные анализируемые графы, полученные анализатором на Wolfram Mathematica, такие как классы эквивалентности на графе совместного исполнения потоков, анализ путей на расчетном графе. Приведенные данные в приложении являются дополнением к третьей главе, где приводятся практические результаты анализа различных многопоточных алгоритмов

Заключение

В заключении сформулированы основные результаты диссертационной работы. Результаты и положения, выносимые на защиту, совпадают.

СПИСОК ПУБЛИКАЦИЙ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Битнер В.А., Тормасов А.Г. Анализ и сокращение промежуточного представления программы в модели статического анализа нахождения состояния гонки в многопоточных алгоритмах // Вестник КГТУ им. А.Н. Туполева, 2014. – №3. – С. 203-212.

2. Битнер В.А., Тимербаев Н.Ф. Контекстно зависимая линеаризация графа потока управления в статическом анализе состояний гонок в многопоточных алгоритмах // Вестник Казанского технологического университета, 2014. – Т. 17, №15. – С. 187-192.

3. Битнер В.А. Применение линеаризованного графа потока управления в модели статического анализа состояний гонок в многопоточных программах компиляторе // Труды 57-й научной конференции МФТИ «Актуальные проблемы фундаментальных и прикладных наук в современном информационном обществе». Управление и прикладная математика. Том 1. – М.: МФТИ, 2014. – 182 с.

4. Битнер В.А. Статический анализ кода алгоритмов с использованием линеаризации графа потока управления // Труды XVI научной конференции «Математическое моделирование и информатика». / Под ред. Д.Ю. Рязанова. – М.: ИЦ ФГБОУ ВПО МГТУ «СТАНКИН», 2014. – 308 с.

5. Битнер В.А., Заборовский Н.В. Построение универсального линеаризованного графа потока управления для использования в статическом анализе кода алгоритмов // Моделирование и анализ информационных систем, 2013. – Т. 20, №2. – С. 166-177.

6. Vilgelm Bitner. Comparative analysis of open source hypervisors for ARM-architecture // The 8th Congress of the International Society for Analysis, its Applications, and Computation. - M.: PFUR, 2011. - 517 p. ISBN 978-5-209-04088-0

7. Битнер В.А. Исследование гипервизоров с открытым кодом для ARM-архитектуры // Актуальные проблемы науки: сб.науч.тр. по мат-лам Междунар. науч.-практ. конф. 30 мая 2011 г.: М-во обр. и науки РФ. Тамбов: Изд-во ТРОО "Бизнес-Наука-Общество", 2011. – 160с.

8. Битнер В.А. Система интерпретации промежуточного представления программы в оптимизирующем компиляторе // Труды 52-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук». Часть I. Радиотехника и кибернетика. Том 1. – М.: МФТИ, 2009. – 182 с.

9. Битнер В.А. Интерпретация промежуточного представления исходной программы в оптимизирующем компиляторе на основе трансляции представления в программный код языка Си // Сборник научных трудов XXXV Международной молодежной научной конференции Гагаринские чтения. – М.: МАТИ, 2009. – С. 111-112.

Личный вклад соискателя в работах заключается в следующем:
[1,2,5] – исследование и адаптация метода статического анализа состояний гонок на основе графе совместного исполнения потоков на промежуточном представлении;
[1] – метод контекстно зависимой линеаризации графа потока управления в статическом анализе;
[2] – анализ инструкций, которые можно удалять из анализируемого контекста в статическом анализе;
[5] – анализ оптимизаций компилятора и применение его в статическом анализе;
[6,7,8,9] – исследование влияния использования промежуточного представления в статическом анализе.