На правах рукописи

Моисеев Михаил Юрьевич

АВТОМАТИЧЕСКОЕ ОБНАРУЖЕНИЕ ДЕФЕКТОВ В МНОГОПОТОЧНЫХ ПРОГРАММАХ МЕТОДАМИ СТАТИЧЕСКОГО АНАЛИЗА

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

А В Т О Р Е Ф Е Р А Т

диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург, 2011

Работа выполнена на кафедре «Компьютерных систем и программных технологий» государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный политехнический университет».

Научный руководитель:	кандидат технических наук, доцент **Ицыксон Владимир Михайлович**
Официальные оппоненты	доктор технических наук, профессор **Лисс Александр Рудольфович** кандидат физико-математических наук, доцент **Иванов Александр Николаевич**
Ведущая организация	ГОУ ВПО «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»

Защита состоится 12 мая 2011 г. в 14 часов на заседании Диссертационного Совета Д 212.229.18 при ГОУ ВПО «Санкт-Петербургский государственный политехнический университет» по адресу: 195251, Санкт-Петербург, ул. Политехническая, д. 29, 9-й учебный корпус, ауд. 325.

С диссертацией можно ознакомиться в фундаментальной библиотеке государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный политехнический университет».

Автореферат разослан «____» апреля 2011 г.

Ученый секретарь
диссертационного совета
кандидат технических наук, доцент Васильев А.Е.

1. ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность работы

В настоящее время как никогда остро стоит проблема повышения качества программного обеспечения. Это связано с тем, что все большее число технических систем используют программные компоненты. Программные системы могут решать сложные и ответственные задачи, при этом практически все они содержат ошибки. Одним из видов ошибок в ПО являются **нефункциональные ошибки** – ошибки, связанные с нарушениями правил языка программирования, некорректным использованием библиотечных функций и системных вызовов и т.п.

Большая часть системного ПО, а также ПО для мобильных и встраиваемых систем, систем управления и других критически важных приложений создается с использованием языка С. По данным компании Coverity, в программах на языке С, в среднем, содержится 0.25 нефункциональных ошибок на 1000 строк исходного кода. Нефункциональные ошибки, характерные для последовательных программ, принято называть **программными дефектами**. Наиболее распространенными типами программных дефектов в программах на языке С являются: разыменование некорректного указателя, переполнение буфера и выход за границы объекта, использование неинициализированной переменной, утечки и повторное освобождение ресурсов. Проявление программных дефектов может приводить к выдаче неверных результатов, зависанию или аварийному завершению программы, нарушению конфиденциальности хранимой информации.

Обнаружение программных дефектов является одной из самых сложных и трудоемких задач в процессе разработке ПО, что приводит к необходимости автоматизации решения этой задачи. Современные методы автоматизированного обнаружения программных дефектов, можно разделить на два класса:

- **динамические методы** – используют результаты выполнения программы,
- **статические методы** – используют исходный код программы, модели программы, спецификации и другие артефакты, создаваемые в процессе проектирования.

К динамическим методам относятся: мониторинг, анализ трасс выполнения, различные виды тестирования. Использование динамических методов обеспечивает получение точных результатов. Основными недостатками динамических методов являются: неполнота получаемых результатов и невозможность полной автоматизации процесса обнаружения дефектов. К статическим методам относятся: верификация на основе проверки моделей, дедуктивная верификация и статический анализ. Наиболее перспективным методом обнаружения программных дефектов представляется **статический анализ**. Применение статического анализа позволяет обнаруживать все основные типы программных дефектов, при этом обеспечивается получение полных результатов. Еще одним преимуществом статического анализа является возможность полной автоматизации процесса обнаружения дефектов. Основным недостатком данного метода является неточность получаемых результатов.

В настоящее время существуют эффективные алгоритмы статического анализа, обеспечивающие обнаружение дефектов в последовательных программах, однако многие программные системы используют те или иные механизмы распараллеливания. Одним из широко распространенных классов параллельных программ являются многопоточные программы. В многопоточных программах кроме программных дефектов, могут встречаться **ошибки синхронизации** – нефункциональные ошибки, связанные с неправильной организацией параллельного выполнения программы. К ошибкам синхронизации относятся: конкурентная модификация и использование разделяемых объектов из разных потоков программы (Race Condition), взаимные блокировки потоков (Deadlock), некорректное использование функций синхронизации (Blocking Call Misuse) и др. Применение алгоритмов статического анализа, не учитывающих специфику многопоточных программ, приводит к снижению полноты и точности получаемых результатов, а также не позволяет обнаруживать ошибки синхронизации.

Исследования, посвященные обнаружению программных дефектов в параллельных (многопоточных) программах на основе статического анализа, ведутся в ряде отечественных (ИСП РАН, СПбГУ, СПбГУИТМО, МФТИ и др.) и зарубежных университетов (MIT, Stanford University, University of California и др.), а также в научно-исследовательских центрах (Microsoft Research, Intel Research, HP Laboratories и др.). Существующие методы статического анализа многопоточных программ решают только часть задач, необходимых для обнаружения программных дефектов, и обладают рядом недостатков: анализируется ограниченное подмножество конструкций языка C, имеются ограничения на способы синхронизации потоков программы, используются аппроксимации, приводящие к недостаточной полноте и точности получаемых результатов.

Большое количество проводимых исследований, а также недостаточная степень проработки и ограничения существующих методов, свидетельствуют о том, что задача создания методов автоматизации обнаружения дефектов в многопоточных программах, в том числе в программах на языке C, является актуальной.

Цель работы

Целью данной работы является исследование и разработка методов автоматизации обнаружения программных дефектов в многопоточных программах на основе статического анализа.

Задачи исследования

1. Анализ существующих подходов к обнаружению дефектов в многопоточных программах на основе статического анализа.
2. Разработка модели многопоточной программы.
3. Выбор алгоритмов статического анализа, обеспечивающих извлечение необходимой информации для обнаружения программных дефектов.
4. Разработка алгоритмов анализа параллельного выполнения многопоточной программы.

5. Организация взаимодействия отдельных алгоритмов анализа.
6. Выбор и уточнение алгоритмов обнаружения программных дефектов на основе результатов статического анализа.
7. Реализация прототипа средства автоматического обнаружения дефектов в многопоточных программах на языке С.

Методы исследования

При проведении теоретических исследований применяются методы теории множеств, теории графов, теории отношений, теории решеток, абстрактной интерпретации, статического анализа. При разработке прототипа средства автоматического обнаружения дефектов используются методы объектно-ориентированного анализа и проектирования.

Научная новизна работы

1. Разработан **метод получения информации о параллельном выполнении** многопоточной программы, определяющий параллельно выполняющиеся блоки программы, взаимодействующие конструкции синхронизации, а также состояния объектов синхронизации. Построение допустимых комбинаций блоков программы при анализе конструкций синхронизации обеспечивает повышение точности результатов.
2. Разработан **метод совместного анализа потоков программы**, обеспечивающий корректный анализ многопоточных программ, использующих разделяемые объекты, а также программ с несколькими возможными порядками выполнения конструкций. Данный метод управляет последовательностью анализа конструкций и распространяет изменения значений разделяемых объектов между потоками программы. Определение актуальных значений разделяемых объектов обеспечивает повышение точности результатов.
3. Предложен **способ организации совместного выполнения алгоритмов анализа**, позволяющий расширить алгоритмы анализа последовательных программ на класс многопоточных программ. Данный способ обеспечивает учет всех взаимодействий между используемыми алгоритмами, что позволяет сохранить полноту получаемых результатов. Обмен промежуточными данными между алгоритмами в процессе анализа обеспечивает повышение точности получаемых результатов.

Достоверность результатов

Достоверность теоретических результатов подтверждается доказательствами утверждений, лежащих в основе разработанных методов, а также результатами экспериментальных исследований разработанного прототипа средства обнаружения программных дефектов.

Практическая значимость работы

Разработанные методы предназначены для автоматизации обнаружения программных дефектов в многопоточных программах на языке C. Применение этих методов позволит повысить качество как вновь разрабатываемых, так и уже существующих программных систем, а также снизить трудоемкость и сократить сроки отладки ПО.

Реализованный в рамках данной работы прототип средства обнаружения дефектов позволяет обнаруживать основные типы программных дефектов и некоторые виды ошибок синхронизации в многопоточных программах на языке C, использующих потоки и объекты синхронизации Pthreads.

Теоретические и практические результаты диссертационной работы могут являться основой для разработки промышленного средства обнаружения программных дефектов.

Реализация результатов работы

Реализацией результатов работы является прототип средства обнаружения дефектов, который может применяться для повышения качества многопоточных программ на языке C.

Результаты диссертационной работы используются в системе обнаружения ошибок синхронизации в программах на языке SystemC, разработанной в рамках НИР по заказу ЗАО «Интел А/О» в 2010 году, имеется акт о внедрении.

Апробация работы

Основные идеи и результаты работы докладывались на конференциях «Software Engineering Conference in Russia 2010», «Approaches and tools for efficient design of reconfigurable/programmable SOC. Intel Workshop in Saint-Petersburg 2010», «Технологии Microsoft в теории и практике программирования», «Software Engineering Conference in Russia 2009», а также обсуждались на семинарах кафедры КСПТ, СПбГПУ и кафедры системного программирования, СПбГУ.

Публикации

По результатам диссертационной работы опубликовано 9 печатных работ, в том числе в журналах «Научно-технические ведомости СПбГПУ» и «Информационно-управляющие системы» (входят в «Перечень ведущих рецензируемых научных журналов и изданий, выпускаемых в Российской Федерации»). Всего по теме работы опубликовано 5 журнальных статей и 4 тезиса конференций.

Структура и объем работы

Диссертационная работа состоит из введения, пяти глав, заключения, списка используемых источников. Общий объем работы составляет 173 печатные страницы, работа включает 73 рисунка, список источников из 65 наименований, 4 приложения.

2. СОДЕРЖАНИЕ РАБОТЫ

Во **введении** обосновывается актуальность темы, определяются цель и задачи работы, формулируется положения, выносимые на защиту.

В **первой главе** проводится сравнительный анализ существующих методов и средств обнаружения ошибок в параллельных программах. Для проведения сравнительного анализа формулируются следующие критерии оценки: поддержка всего множества конструкций языка C, анализ интервальных переменных и переменных-указателей, извлечение необходимой информации для обнаружения программных дефектов, анализ произвольного числа потоков программы, анализ произвольного числа объектов синхронизации, совместное выполнение алгоритмов анализа, получение результатов с полнотой равной или близкой к 100%. Для средств обнаружения ошибок дополнительным критерием оценки является наличие открытой информации об используемых методах и алгоритмах, а также об эффективности работы этих средств на тестовых наборах или программах с открытым исходным кодом.

Среди работ в области статического анализа параллельных программ можно выделить несколько основных направлений: подходы на основе извлечения частичных порядков выполнения (работы Д. Падуа, Д. Каллагана, К. Кеннеди, Дж. Саблока, Е. Штольца, Дж. Ли и др.), подходы на основе анализа наборов конструкций синхронизации (работы Д. Энглера, К. Ашкрафта, Ш. Кводера, Р. Чанга, С. Лернера и др.), подходы на основе извлечения инвариантов (работы Дж. Фостера, М. Хикса, Т. Тераучи, А. Готсмана и др.), подходы на основе преобразования программы к последовательному виду (работы Ш. Кводера, Д. Ву, А. Лала, Т. Репса и др.). Рассмотренные подходы позволяют извлекать информацию о параллельном выполнении программы и, в основном, ориентированы на обнаружение ошибок синхронизации. Применение подобных подходов для обнаружения программных дефектов в многопоточных программах на языке C требует существенных доработок.

Вопросы обнаружения программных дефектов в многопоточных программах рассматриваются в работах М. Ринарда, М. Найка, С. Парка, В. Кахлона, и др. В основе используемых подходов лежит совместное применение обычных алгоритмов статического анализа и алгоритмов, извлекающих информацию о параллельном выполнении программы. Выполнение этих алгоритмов производится последовательно, один или несколько раз, с обменом информацией между ними после каждой итерации. Основными недостатками подобных подходов является отсутствие учета всех взаимных влияний между алгоритмами анализа, а также потеря точности результатов из-за раздельного выполнения этих алгоритмов. В результате проведенного сравнительного анализа выяснилось, что ни один из рассмотренных методов полностью не удовлетворяет всем сформулированным критериям.

В настоящее время существуют средства автоматизации обнаружения дефектов в многопоточных программах на языке C, наиболее распространенными являются: Coverity SA, Klocwork Insight, Parasoft C/C++ test, IBM SA, Mathworks

PolySpace, Frama-C. Большинство из существующих средств обнаружения дефектов являются закрытыми. Средства, для которых удалось получить экспериментальные результаты, показали достаточно низкие полноту и точность обнаружения дефектов.

В завершении главы делается вывод о том, что существующие методы и средства обнаружения программных дефектов в многопоточных программах на языке С являются не достаточно эффективными и могут иметь лишь ограниченное применение для повышения качества промышленного ПО.

Вторая глава посвящена разработке модели многопоточной программы и представлению динамических свойств программы, извлекаемых при проведении статического анализа.

В начале главы рассматривается предлагаемый подход, основная идея которого заключается в расширении алгоритмов анализа последовательных программ на многопоточные программы. Предлагаемый подход состоит из трех основных стадий: построения модели программы, анализа построенной модели для извлечения необходимой информации и обнаружения программных дефектов.

В данной работе рассматриваются многопоточные программы, использующие организацию многопоточного выполнения на основе **Pthreads** (реализация стандарта IEEE 1003.1c). В Pthreads определяются потоки программы и объекты синхронизации: мьютексы, семафоры, переменные состояния, барьеры и др. Подход, предлагаемый в данной работе, позволяет выполнять анализ программ, использующих все множество объектов синхронизации Pthreads. Для компактности представления правил разработанных алгоритмов, в работе рассматривается только два типа объектов синхронизации – **мьютекс** и **семафор**.

Для представления функций над потоками и выбранными объектами синхронизации разработан базис конструкций управления параллельным выполнением программы (см. табл. 1). В работе приводятся представления основных функций Pthreads в разработанном базисе.

Таблица 1. Базис конструкций управления параллельным выполнением

Конструкция	**Краткое описание**
$create(t, pf)$	Создание потока и запуск в нем функции $* pf$
$join(t)$	Ожидание завершения потока программы
$init(m, val)$	Инициализация объекта m значением val
$destroy(m)$	Уничтожение объекта синхронизации
$lock(m, act)$	Блокировка мьютекса
$unlock(m, act)$	Освобождение мьютекса
$wait(m, act)$	Блокировка семафора
$post(m, act)$	Освобождение семафора
$state(m)$	Получение состояния объекта синхронизации

Параметр act определяет изменение состояния объекта в данной конструкции.

Модель программы представляет собой модифицированный граф потока управления, содержащий конструкции определения области видимости

переменных, конструкций ветвления (if) и слияния путей выполнения программы (ϕ-функции), конструкции управления параллельным выполнением программы (см. табл. 1) и другие конструкции.

Для привязки конструкций программы к потокам, в которых они выполняются, конструкции объединяются в блоки программы. **Блок программы** – множество последовательных конструкций, среди которых отсутствуют межблоковые конструкции. Межблоковыми конструкциями являются: if, ϕ-функции, $init$, $create$, $join$, $lock$, $unlock$, $wait$, $post$ и $state$. Будем обозначать блок программы – S_i^j, где j – уникальный номер потока программы, i – уникальный номер блока в потоке. После объединения конструкций в блоки, модель программы можно рассматривать как двудольный граф, где первым типом вершин являются блоки программы, а вторым типом – межблоковые конструкции.

В процессе статического анализа из модели программы извлекается информация времени выполнения – **динамические свойства программы**. Динамические свойства программы включают состояние программы и информацию о параллельном выполнении программы. **Состояние программы** – значения всех объектов в некоторой точке программы. Для программ на языке С можно выделить три вида объектов: переменные простых типов, переменные-указатели на объекты и функции, дескрипторы ресурсов.

Конструкции синхронизации взаимодействуют друг с другом с помощью блокировки и освобождения объекта синхронизации – будем представлять взаимодействие между парами конструкций синхронизации с помощью отношений синхронизации между блоками программы. Для блока $S_{i_1}^{j_1}$, непосредственно перед которым имеется конструкция синхронизации s_1, и блока $S_{i_2}^{j_2}$, непосредственно после которого имеется конструкция синхронизации s_2, имеет место **отношение синхронизации**, если существует набор входных данных, на котором выполнение s_1 невозможно без выполнения s_2. Будем обозначать отношение синхронизации между такими блоками – $S_{i_2}^{j_2} \to S_{i_1}^{j_1}$.

Два блока из разных потоков программы являются совместными, если существует набор входных данных программы, на котором могут выполняться оба этих блока. Между блоками $S_{i_1}^{j_1}$ и $S_{i_2}^{j_2}$ существует строгая последовательность выполнения, если на всех наборах входных данных сначала выполняется блок $S_{i_1}^{j_1}$, затем блок $S_{i_2}^{j_2}$ или наоборот. Блоки, из разных потоков программы, могут выполняться параллельно, если эти блоки являются совместными и между ними отсутствует строгая последовательность выполнения – между такими блоками имеется **отношение параллельности**. Будем обозначать отношение параллельности между блоками $S_{i_1}^{j_1}, S_{i_2}^{j_2} - S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}$.

Будем обозначать множество возможных **состояний объекта синхронизации** m в блоке $S_i^j - m\left(S_i^j\right)$. Множество состояний мьютекса может включать значения 0 и 1, при этом значение 0 соответствует заблокированному

состоянию мьютекса, 1 – незаблокированному. Множество состояний семафора может включать значения 0,1,…,+∞, при этом значение 0 соответствует заблокированному состоянию семафора, положительные значения – незаблокированному. Будем называть изменение состояния объекта синхронизации действием над этим объектом. Действие над мьютексом устанавливает его состояние в значение 0 или 1, действие над семафором увеличивает или уменьшает его состояние на 1. **Множество действий над объектом синхронизации** m в блоке S_i^j – $A\left(m, S_i^j\right)$, учитывает возможные изменения состояния объекта m с момента создания потока до начала блока S_i^j на всех путях выполнения программы в данном потоке.

Отношения параллельности, отношения синхронизации и состояния объектов синхронизации вместе представляют **информацию о параллельном выполнении программы**.

В **третьей главе** рассматривается способ организации совместного выполнения алгоритмов анализа, а также методы анализа параллельного выполнения программы: метод получения информации о параллельном выполнении программы и метод совместного анализа потоков программы.

Способ организации совместного выполнения алгоритмов анализа обеспечивает совместную работу алгоритмов в составе методов анализа параллельного выполнения программы и основных алгоритмов статического анализа (алгоритмов анализа последовательных программ).

Алгоритмы в составе методов анализа параллельного выполнения программы обеспечивают анализ создаваемых потоков и запускаемых в них функций, получение информации о параллельном выполнении программы, совместный анализ потоков программы. Основные алгоритмы статического анализа обеспечивают идентификацию объектов синхронизации и объектов-потоков, определение функций, запускаемых в потоках программы через указатели, извлечение информации необходимой для обнаружения программных дефектов.

Предложенный способ организации совместного выполнения алгоритмов анализа обеспечивает учет всех взаимных влияний и обмен промежуточными результатами между алгоритмами. Учет всех взаимных влияний между алгоритмами позволяет сохранять полноту результатов анализа. Обмен промежуточными данными обеспечивает получение результатов с большей точностью по сравнению с подходами, использующими раздельное или итеративное выполнение алгоритмов.

Метод получения информации о параллельном выполнении многопоточной программы включает алгоритмы определения действий и состояний объектов синхронизации, алгоритмы построения отношений параллельности и отношений синхронизации.

Алгоритм определения действий над объектами синхронизации состоит из правил для различных типов конструкций программы. На рис. 1 приведены обозначения блоков программы, используемые в правилах данного алгоритма.

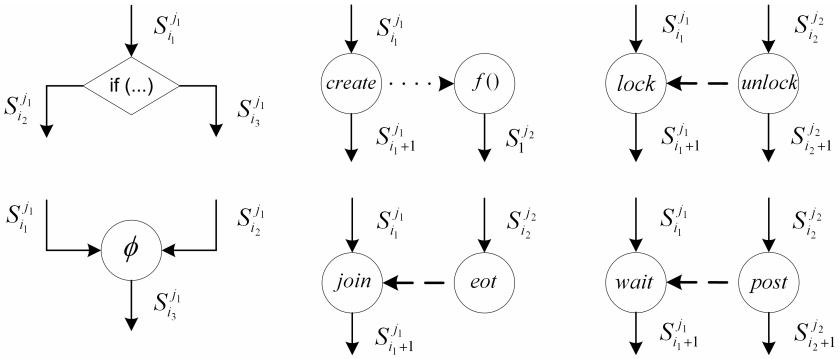

Рис. 1. Обозначения блоков в правилах алгоритмов анализа

Правила для определения множества действий $A(m, S_i^j)$ выглядят следующим образом (см. рис. 1):

$$[S_{i_1}^{j_1}; if()\, S_{i_2}^{j_1} else\, S_{i_3}^{j_1}]: A(m, S_{i_2}^{j_1}) = A(m, S_{i_1}^{j_1}),\ A(m, S_{i_3}^{j_1}) = A(m, S_{i_1}^{j_1}),$$

$$[S_{i_3}^{j_1} = \phi(S_{i_1}^{j_1}, S_{i_2}^{j_1})]: A(m, S_{i_3}^{j_1}) = A(m, S_{i_1}^{j_1}) \cup A(m, S_{i_2}^{j_1}),$$

$$[create(t, pf)]: A(m, S_{i_1+1}^{j_1}) = A(m, S_{i_1}^{j_1}),\ A(m, S_1^{j_2}) = \{1\}\,(мьютекс),$$

$$[create(t, pf)]: A(m, S_{i_1+1}^{j_1}) = A(m, S_{i_1}^{j_1}),\ A(m, S_1^{j_2}) = \{0\}\,(семафор),$$

$$[join(t)]: A(m, S_{i_1+1}^{j_1}) = A(m, S_{i_1}^{j_1}) \oplus A(m, S_{i_2}^{j_2})\ (семафор),$$

$$[lock(m, act)]: A(m, S_{i_1+1}^{j_1}) = \{0\}, [unlock(m, act)]: A(m, S_{i_2+1}^{j_2}) = \{1\},$$

$$[wait(m, act)]: A(m, S_{i_1+1}^{j_1}) = A(m, S_{i_1}^{j_1}) \oplus \{-1\}, [post(m, act)]: A(m, S_{i_2+1}^{j_2}) = A(m, S_{i_2}^{j_2}) \oplus \{1\},$$

где $\oplus$ – операция сложения множеств действий, результат которой все возможные суммы из действий каждого множества. В правиле для конструкции *join* необходимо учитывать действия только для семафоров – на момент завершения потока мьютексы, заблокированные в данном потоке, должны быть освобождены.

Алгоритм определения состояния объектов синхронизации использует действия над объектами синхронизации в потоках программы. Для определения возможных значений m в блоке $S_{i_1}^{j_1}$ ($m(S_{i_1}^{j_1})$) строятся комбинации блоков других потоков, параллельно с которыми может выполняться блок $S_{i_1}^{j_1}$ – **множество допустимых комбинаций**, $C(S_{i_1}^{j_1})$. Множество $C(S_{i_1}^{j_1})$ формируется с помощью следующего правила:

$$\forall C: C \in C(S_{i_1}^{j_1}) \Rightarrow$$
$$(\forall S_i^j: S_i^j \in C \Rightarrow (S_i^j \,||\, S_{i_1}^{j_1}, \forall i_2: i_2 \neq i \Rightarrow S_{i_2}^j \notin C, \forall S_{i_3}^{j_3}: S_{i_3}^{j_3} \in C \Rightarrow S_i^j \,||\, S_{i_3}^{j_3})).$$

В множество $C\left(S_{i_1}^{j_1}\right)$ входят комбинации из блоков программы, которые параллельны блоку $S_{i_1}^{j_1}$, в каждую комбинацию входит не более одного блока из каждого потока, все блоки комбинации попарно параллельны между собой.

Правила определения $m\left(S_{i_1}^{j_1}\right)$ выглядят следующим образом:

$$m_C\left(S_{i_1}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right) \otimes \bigotimes_{\forall S_i^j \in C} A\left(m, S_i^j\right) (\textit{мьютекс}),$$

$$m_C\left(S_{i_1}^{j_1}\right) = A\left(m, S_{i_1}^{j_1}\right) \oplus \bigoplus_{\forall S_i^j \in C} A\left(m, S_i^j\right) (\textit{семафор}),$$

$$m\left(S_{i_1}^{j_1}\right) = \bigcup_{\forall C \in C\left(S_{i_1}^{j_1}\right)} m_C\left(S_{i_1}^{j_1}\right),$$

где $\otimes$ – операция умножения множеств действий, результат которой – все возможные произведения из действий каждого множества, $m_C\left(S_{i_1}^{j_1}\right)$ – состояние объекта синхронизации в комбинации блоков C.

Алгоритм построения отношений параллельности использует следующие правила (см. рис. 1):

$$\frac{\forall S_i^j : S_{i_1}^{j_1} \,||\, S_i^j}{\left[S_{i_1}^{j_1}; \textit{if}\,() S_{i_2}^{j_1} \textit{else} S_{i_3}^{j_1}\right] : S_{i_2}^{j_1} \,||\, S_i^j, S_{i_3}^{j_1} \,||\, S_i^j}, \frac{\forall S_i^j : S_{i_1}^{j_1} \,||\, S_i^j \vee S_{i_2}^{j_1} \,||\, S_i^j}{\left[S_{i_3}^{j_1} = \phi\left(S_{i_1}^{j_1}, S_{i_2}^{j_1}\right)\right] : S_{i_3}^{j_1} \,||\, S_i^j},$$

$$\frac{-}{\left[\textit{create}(t, pf)\right] : S_{i_1+1}^{j_1} \,||\, S_1^{j_2}}, \frac{\forall S_i^j : S_{i_1}^{j_1} \,||\, S_i^j}{\left[\textit{create}(t, pf)\right] : S_{i_1+1}^{j_1} \,||\, S_i^j, S_1^{j_2} \,||\, S_i^j},$$

$$\frac{\forall S_i^j \in C : C \in C\left(S_{i_1}^{j_1}\right), S_{i_2}^{j_2} \in C}{\left[\textit{join}(t)\right] : S_{i_1+1}^{j_1} \,||\, S_i^j}, \frac{\forall S_i^j \in C : C \in C\left(S_{i_1}^{j_1}\right), \exists a \in m_C\left(S_{i_1}^{j_1}\right) : a > 0}{\left[\textit{lock}(m) \,|\, \textit{wait}(m)\right] : S_{i_1+1}^{j_1} \,||\, S_i^j}.$$

Блок после конструкции *join* параллелен блокам, входящим в комбинации для блока перед этой конструкцией, в которые также входит последний блок ожидаемого потока. Блок, следующий после конструкции *lock* или *wait*, параллелен всем блокам, входящим в комбинацию для блока перед этой конструкцией, при условии, что объект синхронизации в данной комбинации может быть не заблокирован. Множество отношений параллельности в конструкциях *unlock* и *post* не меняется.

Правила **алгоритма построения отношений синхронизации** выглядят следующим образом (см. рис. 1):

$$\frac{-}{\left[\textit{join}(t)\right] : S_{i_2}^{j_2} \to S_{i_1+1}^{j_1}}, \frac{S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}}{\left[\textit{lock}(m)\right] : S_{i_2}^{j_2} \to S_{i_1+1}^{j_1}},$$

$$\frac{\exists C : C \in C\left(S_{i_1}^{j_1}\right), \exists a \in m_C\left(S_{i_1}^{j_1}\right) : a = 0, S_{i_2}^{j_2} \in C}{\left[\textit{wait}(m)\right] : S_{i_2}^{j_2} \to S_{i_1+1}^{j_1}}.$$

Отношение синхронизации имеет место для пары конструкций *lock* / *unlock*, оперирующих одним и тем же объектом синхронизации, при условии что блоки

перед этими конструкциями могут выполняться параллельно. При построении отношения синхронизации для пары конструкций $wait/post$ дополнительно проверяется возможность нахождения семафора в заблокированном состоянии. Доказательства корректности рассмотренных алгоритмов приводятся в работе.

Метод совместного анализа потоков программы включает алгоритм учета взаимного влияния потоков и итеративный алгоритм анализа конструкций.

Взаимное влияние потоков в многопоточной программе заключается в изменении значений разделяемых объектов. **Алгоритм учета взаимного влияния потоков** выполняет объединение значений разделяемых объектов с помощью специальных конструкций – **ψ-функций**. Добавление ψ-функций осуществляется для конструкций $join$, $lock$ и $wait$ – каждая ψ-функция имеет один вход из текущего потока и один или несколько входов из других потоков. Входы в ψ-функцию из других потоков строятся на основе отношений синхронизации.

При отсутствии ошибок синхронизации типа Race Condition изменение состояния разделяемого объекта возможно только в блоках одного из параллельно выполняющихся потоков. Поэтому при объединении состояний программы в ψ-функции значения разделяемого объекта либо одинаковы, либо актуальным является значение на одном из входов ψ-функции.

Для определения актуальных значений разделяемых объектов в ψ-функции с двумя входами используются следующие правила (см. рис. 2):

$$\frac{\forall e \in E_{j_1}}{e\left(S_{i_1+1}^{j_1}\right) = e\left(S_{i_1}^{j_1}\right)}, \frac{\forall e \in E_{j_2}}{e\left(S_{i_1+1}^{j_1}\right) = e\left(S_{i_2}^{j_2}\right)}, \frac{\forall e \in E_{j_1,j_2}}{e\left(S_{i_1+1}^{j_1}\right) = e\left(S_{i_1}^{j_1}\right) \cup e\left(S_{i_2}^{j_2}\right)},$$

где $e\left(S_i^j\right)$ – значение разделяемого объекта e в блоке S_i^j, E_{j_1} (E_{j_2}) – множество разделяемых объектов, значение которых менялось только в блоках параллельных $S_{i_2}^{j_2}$ ($S_{i_1}^{j_1}$) и не параллельных $S_{i_1}^{j_1}$ ($S_{i_2}^{j_2}$), E_{j_1,j_2} – разделяемые объекты, не входящие в множества E_{j_1} и E_{j_2} (при определении E_{j_1}, E_{j_2} и E_{j_1,j_2} учитываются блоки, которые параллельны хотя бы одному из входных блоков ψ-функции). Корректность разработанного алгоритма доказана в работе.

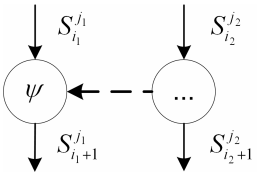

Рис. 2. Обозначения блоков в правилах для ψ-функции

Разработанные алгоритмы допускают произвольную последовательность анализа конструкций разных потоков программы. Однако, для получения полных результатов, при анализе конструкций некоторого блока $S_{i_1}^{j_1}$ необходимо учитывать информацию, полученную при анализе конструкций блоков других потоков, которые могут выполняться строго до блока $S_{i_1}^{j_1}$. При анализе программ,

для которых возможны различные порядки выполнения блоков, в общем случае не существует единственной последовательности анализа конструкций, обеспечивающей получения полных результатов. Для решения этой проблемы разработан итеративный алгоритм. **Итеративный алгоритм** выполняет повторный анализ конструкций, исходные данные для которых изменились в результате анализа конструкций других потоков программы. Повторный анализ может начинаться с конструкций $join$, $lock$, $wait$ и $state$, при этом порождаются цепочки повторно анализируемых конструкций, которые распространяются между потоками программы. При порождении цепочек повторно анализируемых конструкций выбираются только те начальные конструкции, результат анализа для которых увеличился. В работе доказана сходимость итеративного алгоритма и сохранение полноты получаемых результатов при анализе программ с несколькими возможными порядками выполнения.

В **четвертой главе** рассматриваются алгоритмы обнаружения программных дефектов и ошибок синхронизации.

Обнаружение программных дефектов выполняется с помощью обычных правил, используемых для обнаружения подобных дефектов в последовательных программах. В работе приводятся правила для обнаружения следующих групп программных дефектов: ошибки управления ресурсами и динамической памятью (RES), утечки ресурсов и памяти (LEAK), ошибки работы с буферами/массивами (BUF), ошибки отсутствия инициализации (INI). Рассмотрим несколько примеров подобных правил.

Обнаружение дефекта множественного освобождения памяти выполняется с помощью следующего правила:

$$\frac{\exists(p, o_{invalid}) \in \Im_l}{[free(p)]^l : DEFECT_{RES-01}},$$

где $\Im_l$ – состояние программы перед l-й конструкцией, $o_{invalid}$ – специальный объект, представляющий множество некорректных значений указателя. Для обнаружения разыменования указателя выведенного за границу объекта используется правило:

$$\frac{\exists(p,(o_j,k)) \in \Im_l : (k<0) \vee (k \geq size(o_j))}{[*p]^l : DEFECT_{BUF-02}},$$

где предусловие $(p,(o_j,k))$ означает, что указатель p указывает на объект o_j со смещением k, оператор $size(o_j)$ определяет размер объекта o_j. Правило разыменования неинициализированного (освобожденного) или нулевого указателя выглядит следующим образом:

$$\frac{\exists(p,(o_{invalid})) \in \Im_l \vee \exists(p,(o_{null})) \in \Im_l}{[*p]^l : DEFECT_{INI-03}},$$

где $o_{invalid}$ – специальный объект, представляющий значение $NULL$.

В данной работе рассматривается обнаружение ошибок синхронизации, наличие которых может влиять на полноту и точность алгоритмов анализа. К такому классу ошибок, в частности, относятся ошибки типа Race Condition. Обнаружение этих ошибок выполняется с помощью следующих правил:

$$\frac{\exists S_{i_1}^{j_1}, S_{i_2}^{j_2}, e : S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}, e \in Use\left(S_{i_1}^{j_1}\right), e \in Def\left(S_{i_2}^{j_2}\right)}{DEFECT_{RACE}},$$

$$\frac{\exists S_{i_1}^{j_1}, S_{i_2}^{j_2}, e : S_{i_1}^{j_1} \,||\, S_{i_2}^{j_2}, e \in Def\left(S_{i_1}^{j_1}\right), e \in Def\left(S_{i_2}^{j_2}\right)}{DEFECT_{RACE}},$$

где $e \in Def\left(S_i^j\right)$ и $e \in Use\left(S_i^j\right)$ факты изменения и использования разделяемого объекта e в блоке S_i^j соответственно.

В **пятой главе** рассматривается практическая реализация разработанных методов, и приводятся оценки их эффективности.

В качестве основы для реализации использовалась система обнаружения программных дефектов **Aegis**, разработанная на кафедре КСПТ СПбГПУ, при участии автора. Система Aegis позволяет обнаруживать программные дефекты в последовательных программах на языках C/C++. В данной работы была создана новая версия системы – **Aegis MT**, которая выполняет обнаружение дефектов в многопоточных программах на языке C, использующих Pthreads. Структурная схема системы Aegis MT приведена на рис. 3 (блоки, добавленные или модифицированные в версии Aegis MT, выделены серым цветом).

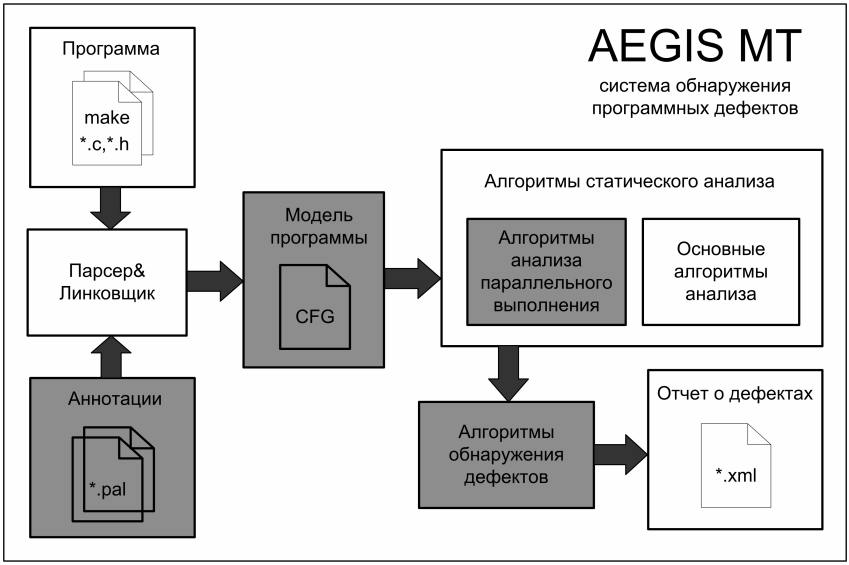

Рис. 3. Структурная схема системы обнаружения дефектов Aegis MT

Разработанная система выполняет анализ и обнаружение дефектов полностью автоматически. В работе доказано, что разработанные методы

обеспечивают сохранение полноты результатов основных алгоритмов анализа. Точность получаемых результатов зависит от конкретной программы и не может быть оценена аналитически. Для проверки полноты и получения оценки точности результатов проведены экспериментальные исследования.

При проведении экспериментальных исследований использовался набор специальных тестовых программ (140 программ), а также реальный программный проект – многопоточная звуковая библиотека, обеспечивающая запись и воспроизведение звука одновременно для нескольких клиентов. Для анализа звуковой библиотеки было разработано четыре тестовых проекта, использующих все основные функции этой библиотеки в многоклиентском режиме. В рамках проведенных исследований сравнивались три средства обнаружения дефектов: **Aegis**, **Aegis MT** и **Parasoft C++ test**.

Экспериментальные оценки полноты и точности результатов, полученные на наборе тестовых программ, представлены на рис. 4 (средство Parasoft C++ test обозначено PS). Результаты, полученные при анализе тестовых проектов звуковой библиотеки, представлены на рис. 5 – на левой части рисунка приведено число истинных дефектов, обнаруженных средствами, на правой части рисунка приведено распределение дефектов, обнаруженных Aegis MT, по типам дефектов.

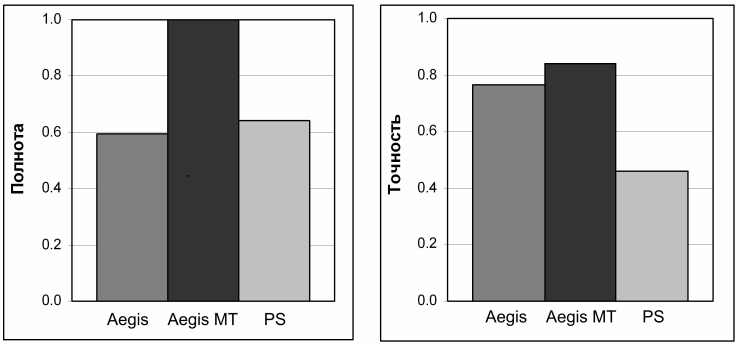

Рис. 4. Оценки полноты и точности результатов на тестовых программах

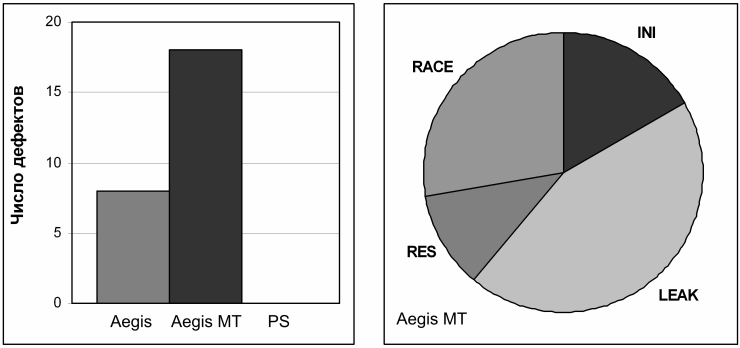

Рис. 5. Результаты анализа звуковой библиотеки

Проведенные экспериментальные исследования подтверждают, что разработанные методы обеспечивают сохранение полноты результатов основных алгоритмов статического анализа: в тестовых программах системой Aegis MT были обнаружены все искусственно внесенные дефекты; при анализе тестовых проектов звуковой библиотеки множество истинных дефектов обнаруженных Aegis MT, включает все дефекты обнаруженные другими средствами. Точность результатов, полученных системой Aegis MT на наборе тестовых программ, выше, чем у других средств.

В работе приводится аналитическая оценка вычислительной сложности разработанных методов. Вычислительная сложность определяется конструкциями синхронизации, для которых строятся множества допустимых комбинаций (конструкции типа *join*, *lock*, *wait* и *state*). Сложность анализа одной такой конструкции оценивается как $O\left(n^k \cdot k^2 \cdot \log(k \cdot n)\right)$, где k – число параллельно выполняющихся потоков, n – число блоков параллельных блоку перед этой конструкцией в каждом потоке программы. Оценка вычислительной сложности анализа всей программы (без учета основных алгоритмов анализа) выглядит следующим образом:

$$O\left(n_S \cdot \left(n^k \cdot k^2 \cdot \log(k \cdot n)\right) \cdot n_I\right),$$

где n_S – число анализируемых конструкций синхронизации, использующих построение допустимых комбинаций, n_I – среднее число выполнений повторного анализа конструкций за счет итеративного алгоритма. В работе приводятся экспериментальные оценки значений k, n и n_I, а также предлагается ряд подходов, направленных на снижение вычислительной сложности.

В **заключении** приводятся основные результаты работы, рассматриваются ограничения разработанных методов и прототипа средства обнаружения дефектов, делается вывод о целесообразности их применения в процессе проектирования и эксплуатации программного обеспечения, предлагаются направления дальнейших исследований.

3. ОСНОВНЫЕ РЕЗУЛЬТАТЫ

1. Разработана модель многопоточной программы на языке C, представляющая исходный код программы в удобном для анализа виде.
2. Разработан формализм для представления динамических свойств многопоточной программы, извлекаемых при статическом анализе.
3. Разработан метод получения информации о параллельном выполнении многопоточной программы.
4. Разработан метод совместного анализа потоков программы.
5. Предложен способ организации совместного выполнения алгоритмов статического анализа многопоточной программы.
6. Разработанные методы реализованы в прототипе средства автоматического обнаружения дефектов в многопоточных программах на языке C.

4. ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. **Моисеев М.Ю. Автоматическое обнаружение дефектов в многопоточных программах методами статического анализа // Научно-технические ведомости СПбГПУ. – СПб.: СПбГПУ, 2010. №3. – С. 77-86.**
2. Моисеев М.Ю. Автоматическое обнаружение дефектов в многопоточных программах методами статического анализа // Материалы межвузовского конкурса-конференции «Технологии Microsoft в теории и практике программирования» – СПб.: СПбГПУ, 2010. – С. 157-158.
3. Моисеев М.Ю., Ицыксон В.М.и др. Автоматическое обнаружение дефектов в программных системах на языке С на основе статического анализа. // Материалы межвузовского конкурса-конференции «Технологии Microsoft в теории и практике программирования» – СПб.: СПбГПУ, 2010. – С. 70-71.
4. Моисеев М.Ю. Итеративный алгоритм статического анализа для обнаружения дефектов в исходном коде программ // Информационные и управляющие системы. – Политехника, 2009. №3. – С. 33-39.
5. Moiseev M.Ju., Itsykson V.M. et al. Automatic Defects Detection in Industrial C/C++ Software // Proceedings of the 5th Central and Eastern European Software Engineering Conference in Russia – IEEE, 2009. – P. 50-55.
6. Моисеев М.Ю., Ицыксон В.М. и др. Алгоритмы анализа указателей для обнаружения дефектов в исходном коде программ // Системное программирование. – СПб.: СПбГУ, 2009. – С. 5-30.
7. Моисеев М.Ю., Ицыксон В.М. и др. Алгоритм интервального анализа для обнаружения дефектов в исходном коде программ // Информационные и управляющие системы. – СПб.: Политехника, 2009. №2. – С. 34-41.
8. Моисеев М.Ю., Ицыксон В.М. Исследование и разработка системы автоматического обнаружения дефектов в исходном коде ПО // Материалы конференции по результатам выполнения мероприятий ФЦП "Исследования и разработки по приоритетным направлениям развития научно-технического комплекса России на 2007-2012 годы" – ОАО «ИИЦ», 2009. – С. 14-15.
9. Моисеев М.Ю., Ицыксон В.М., и др. Исследование средств автоматизации обнаружения дефектов в исходном коде программ // Научно-технические ведомости СПбГПУ – СПб.: СПбГПУ, 2008. №5. – С. 119-127.