Московский государственный университет
имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

На правах рукописи

Булычёв Пётр Евгеньевич

АЛГОРИТМЫ ВЫЧИСЛЕНИЯ ОТНОШЕНИЙ ПОДОБИЯ В ЗАДАЧАХ ВЕРИФИКАЦИИ И РЕСТРУКТУРИЗАЦИИ ПРОГРАММ

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата физико-математических наук

МОСКВА
2010

Работа выполнена на кафедре автоматизации систем вычислительных комплексов факультета вычислительной математики и кибернетики Московского государственного университета имени М.В. Ломоносова.

Научные руководители:	доктор физико-математических наук, профессор, академик РАЕН Смелянский Руслан Леонидович; кандидат физико-математических наук, доцент Захаров Владимир Анатольевич.
Официальные оппоненты:	доктор физико-математических наук, профессор Ломазова Ирина Александровна; кандидат физико-математических наук, Романенко Сергей Анатольевич.
Ведущая организация:	Вычислительный центр им. А. А. Дородницына РАН.

Защита состоится «10» декабря 2010 г. в 11:00 на заседании диссертационного совета Д 501.001.44 при Московском государственном университете имени М.В. Ломоносова по адресу: 119991, ГСП-1, Москва, Ленинские горы, МГУ, 2-ой учебный корпус, факультет вычислительной математики и кибернетики, аудитория 685.

С диссертацией можно ознакомиться в библиотеке факультета ВМиК МГУ имени М.В. Ломоносова, с текстом автореферата — на официальном сайте ВМиК МГУ имени М.В. Ломоносова: `http://www.cs.msu.su` в разделе «Наука» — «Работа диссертационных советов» — «Д 501.001.44».

Автореферат разослан «_____» ноября 2010 г.

Ученый секретарь
диссертационного совета Д 501.001.44
профессор Н.П. Трифонов

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Введение.

Отношения подобия и различные их проявления играют большую роль как в математике, естествознании, так и в обыденной практике.

В данной работе исследованы отношения подобия, возникающие в задачах реструктуризации и верификации программ. Работа состоит из двух частей.

В первой части рассмотрена проблема обнаружения синтаксически похожих фрагментов (клонов, дубликатов кода) в исходных кодах программ. Наличие большого числа клонов в программе приводит к ряду негативных последствий, в частности, к увеличению стоимости поддержки кода. В данной работе решается задача обнаружения клонов, от которых можно избавиться при помощи существующих методов реструктуризации (рефакторинга) кода.

Во второй части рассмотрен особый вид отношений подобия — симуляции, часто возникающие в задачах верификации программ. Отношение симуляции $\preceq$ — это бинарное отношение, заданное на множестве моделей вычисления (автоматов, сетей Петри, программ). Справедливость $A \preceq B$ означает то, что структура дерева вычислений модели B оказывается сходной структуре дерева вычислений модели A. Как следствие, многие отношения симуляции сохраняют выполнимость спецификаций, заданных формулами темпоральных логик. Благодаря этому качеству отношения симуляции часто используются в методах формальной верификации программ, позволяя сводить проверку корректности программ к анализу их абстрактных моделей, сохраняющих исследуемые свойства программ. В данной работе рассматривается задача проверки отношений симуляции для случая, когда модели программ задаются структурами Крипке и временными автоматами.

Актуальность задачи поиска клонов. Клоном называются фрагменты, имеющие незначительные синтаксические отличия друг от друга. Суще-

ствует ряд факторов, приводящих к появлению клонов, и наиболее распространённой причиной является использование операции “копирование и вставка” (copy&paste). Результаты многочисленных статистических исследований свидетельствуют о том, что суммарный объём клонов в больших проектах обычно составляет 7–20%.

Присутствие большого числа клонов в исходном коде увеличивает стоимость поддержки программы. Присутствие клонов в программе также приводит к увеличению размера исполняемого файла, что может быть нежелательно при выполнении данной программы в системах с малым объёмом памяти (например, во встроенных системах). Поэтому детекторы клонов могут использоваться для нахождения клонов для последующего их устранения при помощи рефакторинга с целью уменьшения стоимости поддержки программы и уменьшения размера исполняемого файла. В данной работе ставится задача обнаружения клонов, от которых можно легко избавиться при помощи широко известных методов рефакторинга, основу которых составляет процедурная абстракция.

Существуют и другие применения детекторов клонов. Например, они могут использоваться для выделения библиотек часто используемых функций. Найденные клоны могут использоваться для нахождения шаблонов использования классов, и таким образом, для добавления новых методов в эти классы. При добавлении новой функции можно проверять с помощью детектора клонов, не существует ли уже функции, которая имеет ту же функциональность. Относительное количество кода, покрытого клонами, может использоваться в качестве одной из метрик качества кода. Ещё одной областью применимости детекторов клонов является выявление случаев нарушения авторских прав.

Актуальность задачи проверки симуляции между моделями программ. Одним из наиболее распространенных методов проверки правильности программ является верификация моделей программ. В этом методе прове-

ряется не сама программа, а её модель, которая, с одной стороны, сохраняет исследуемые свойства программы, а с другой стороны, является достаточно простой для автоматического анализа.

Важным этапом метода является построение моделей программы. Модели могут отражать поведение программы с разной степенью подробности, они могут строиться как вручную, так и автоматически. Чрезмерно упрощённая модель может давать слишком грубое приближение реальной программы, и на ней не будут проявляться те свойства вычислений, которые присущи программе. С другой стороны, при верификации чрезмерно подробных моделей приходится преодолевать эффект "комбинаторного взрыва": анализ модели становится практически неосуществим из-за слишком большого числа возможных состояний.

Поэтому для применения метода верификации моделей удобно иметь средство, позволяющее сравнивать структуру модели со структурой исходной программы, а также сравнивать структуру моделей между собой. Данное средство в своей работе может опираться на отношения симуляции, т. е. бинарные отношения, заданные на множестве моделей вычисления и сохраняющие структуру их деревьев вычислений. Замечательной особенностью отношений симуляции является то, что они сохраняют выполнимость спецификаций, заданных формулами темпоральных логик. Поэтому для того, чтобы проверить, что все формулы некоторой темпоральной логики, выполнимые на модели A, выполняются и на другой модели (или программе) B, достаточно проверить выполнимость подходящего отношения симуляции $\preceq$ между A и B (т. е. проверить выполнимость $A \preceq B$).

Отношения симуляции также используются для вычисления инвариантов бесконечных параметризованных семейств моделей, для нахождения симметрии в моделях, для автоматического уменьшения размеров моделей, для проверки свойств конфиденциальности многопоточных программ.

Отношения симуляции достаточно хорошо исследованы для случая, когда модели задаются структурами Крипке (конечными размеченными системами переходов). Разнообразие таких отношений велико; разные отношения сохраняют выполнимость формул разных темпоральных логик и используются в различных случаях. Кроме того, возможно, в будущем у исследователей в области верификации моделей программ возникнет необходимость в проверке новых отношений симуляции. Поэтому является актуальной задача разработки универсальной системы проверки симуляции, которая могла бы быть использована для проверки различных вариантов отношений симуляции, в зависимости от потребностей пользователя. В данной диссертационной работе ставится задача разработки такой системы.

Помимо структур Крипке при верификации программ на моделях используются и более богатые по своим выразительным возможностям формализмы описания программ, в частности, временные автоматы. Формализм временных автоматов является достаточно выразительным для описания систем, в которых единственным непрерывно изменяемым параметром являются часы. Поэтому временные автоматы часто используются для описания моделей систем реального времени.

Временные автоматы также используются в качестве удобной математической модели при решении задачи синтеза систем автоматического управления исходя из спецификации требуемого поведения устройства управления и возможного поведения среды. Для решения этой задачи используются временные игровые автоматы, которые, фактически, описывают игру двух игроков, контроллера (объекта управления) и среды, в которой задача контроллера — обеспечить корректное функционирование устройства управления независимо от действий среды. Из-за сложности устройства таких автоматов их чрезвычайно трудно разрабатывать. Поэтому особенно важно иметь средство, позволяющее сравнивать устройство временных игровых автоматов

между собой. Имея в наличии такое средство, можно применять инкрементальный подход к построению моделей, т. е. строить модель последовательно и проверять, что каждая следующая версия уточняет предыдущую. В данной работе решается задача разработки отношения симуляции между временными игровыми автоматами и алгоритма его проверки.

Цель работы. Цель диссертационной работы — разработка математических методов и алгоритмов проверки различных отношений подобия, которые возникают в анализе и преобразовании программ. В частности, для решения задачи реструктуризации программ необходимо разработать и реализовать алгоритм обнаружения клонов, совместимых с существующими методами рефакторинга. Для решения задачи верификации программ дискретного времени необходимо разработать платформу, которая могла бы быть использована для проверки целого ряда отношений симуляции между структурами Крипке. Для решения задачи верификации программ реального времени необходимо разработать подходящее отношение симуляции, сохраняющее выполнимость темпоральных спецификаций на временных игровых автоматах, и алгоритм проверки этого отношения.

Методы исследования. При получении основных результатов работы диссертации использовались методы математической логики, теории графов, теории автоматов и теории формальных языков.

Научная новизна. Разработан новый алгоритм поиска клонов на уровне абстрактных синтаксических деревьев, который превосходит по своим показателям существующие аналоги. Впервые предложен формальный язык, позволяющий задавать различные отношения симуляции между структурами Крипке посредством правил антогонистической игры. Ранее существовали только индивидуальные алгоритмы для проверки отдельных отношений. Разработан алгоритм, который проверяет существование определённой на этом языке симуляции между двумя заданными структурами Крипке. Впервые

определено отношение симуляции, сохраняющее выполнимость формул логики ATCTL на множестве временных игровых автоматов; приведён алгоритм проверки выполнимости данного отношения.

Практическая ценность. Разработанное средство обнаружения клонов может использоваться разработчиками программного обеспечения в процессе реструктуризации программ. Разработанное средство проверки симуляций между структурами Крипке может служить дополнением к средству верификации моделей NuSMV. Разработанное средство проверки симуляций между временными автоматами было использовано в задаче синтеза контроллера для системы автоматического управления реального времени.

Апробация работы. Результаты, представленные в работе, докладывались на научном семинаре лаборатории вычислительных комплексов кафедры АСВК факультета ВМиК МГУ имени М.В. Ломоносова под руководством профессора Р.Л. Смелянского; на семинаре кафедры АСВК под руководством заведующего кафедрой член-корр. РАН Л.Н. Королева; на научном семинаре "Теоретические проблемы программирования" под руководством профессора Р.И. Подловченко и доцента В.А. Захарова; на научном семинаре "Дискретная математика и математическая кибернетика" под руководством профессора В.Б. Алексеева, профессора А.А. Сапоженко и профессора С.А. Ложкина; на рабочих совещаниях группы проекта INTAS 05-1000008-8144 «Practical Formal Verification Using Automated Reasoning and Model Checking», а также на следующих конференциях:

- Международная конференция «Дискретные модели в теории управляющих систем» (Москва, март 2006 г.);
- Всероссийская научная конференция студентов, аспирантов и молодых учёных «Технологии Microsoft в теории и практике программирования», секция «Теоретическое программирование» (Москва, апрель 2008 г.);

- Весенний коллоквиум молодых ученых в области программной инженерии (SYRCoSE) (Санкт-Петербург, май 2008 г.);
- Научно-техническая общеевропейская конференция «ЕвроПайтон» (EuroPython) (Литва, Вильнюс, июль 2008 г.);
- Международный научный семинар «Программные клоны» (Workshop on Software Clones) (Германия, Кайзерслаутерн, март 2009 г.);
- Международная конференция памяти академика А. П. Ершова «Перспективы систем информатики» (Новосибирск, июнь 2009 г.);
- Международный научный семинар «Теория игр в приложениях к проектированию и верификации» (Workshop on Games for Design and Verification) (Италия, Удине, сентябрь 2009 г.);
- Международная конференция «Формальное моделирование и анализ временных систем» (Conference on Formal Modelling and Analysis of Timed Systems) (Венгрия, Будапешт, сентябрь 2009 г.);
- Всероссийская конференция «Методы и средства обработки информации» (Москва, октябрь 2009 г.);
- Международная научная конференция студентов, аспирантов и молодых учёных «Ломоносов» (Москва, апрель 2010 г.).

Работа была выполнена при поддержке грантов INTAS и РФФИ.

Публикации. По теме диссертации имеется 10 публикаций (включая 2 в изданиях из перечня ВАК), список которых приводится в конце автореферата.

Структура и объем диссертации. Диссертация состоит из введения, трёх глав, заключения, списка литературы и приложения. Объём работы —

169 страниц (включая 2 страницы приложения). Список литературы содержит 122 наименований.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Работа состоит из введения, трёх глав и заключения.

Во **введении** обоснована актуальность диссертационной работы и сформулирована цель исследований.

В **первой главе** решается задача разработки математической модели программных клонов, совместимой с существующими методами рефакторинга, и разработки алгоритма поиска клонов, удовлетворяющих этой модели.

В *разделе 1.1* рассмотрены операции рефакторинга, при помощи которых можно избавиться от клонов, а именно: “Выделение метода” (процедурная абстракция, Extract Method), “Подъём метода” (Pull Up Method) и “Выделение родительского класса” (Extract Superclass). Для применения этих операций в общем случае необходимо, чтобы фрагменты, составляющие клон, являлись линейными последовательностями операторов (последовательности могут состоять из одного элемента, а операторы могут быть составными). Кроме того, наиболее простыми для рефакторинга являются клоны, один фрагмент которых может быть получен из другого фрагмента путём замены одних подвыражений на другие подвыражения. Для выделения операторов и их подвыражений необходимо провести синтаксический разбор программы, т. е. построить её абстрактное синтаксическое дерево (АСД). Произвольное подвыражение некоторого оператора программы соответствует некоторому поддереву АСД этой программы.

Отсюда мы приходим к следующему определению:

Определение 1.1 (Клон кода)**.** Две последовательности операторов E_1 и E_2 формируют (p_1, p_2)-клон, если абстрактное синтаксическое дерево после-

довательности E_1 можно получить из абстрактного синтаксического дерева последовательности E_2 путём замены некоторых поддеревьев на другие поддеревья, и справедливы неравенства $\min(|E_1|, |E_2|) \geq p_1$ и $\rho(E_1, E_2) \leq p_2$.

В качестве функций размера $|\cdot|$ и расстояния ρ используются следующие две функции. Размером $|T|$ дерева T считается равным количеству всех его листьев, а размером оператора — размер его АСД. Размер оператора, определённый таким образом, инвариантен относительно способа построения АСД, поскольку он равен количеству вхождений всех имён и констант в фрагмент кода. Если АСД оператора E_1 получается из АСД оператора E_2 заменой поддерева $T_{1,1}$ на $T_{2,1}$, $T_{1,2}$ на $T_{2,2}$ ... $T_{1,n}$ на $T_{2,n}$, то будем считать расстояние $\rho(E_1, E_2)$ равным значению $\sum_{i=1..n}(|T_{1,i}| + |T_{2,i}|) - n$. Показано, что функция ρ является семи-метрикой.

В *разделе 1.2* представлен обзор средств автоматического обнаружения клонов. На основании обзора делается вывод о том, что ни одно из существующих средств не позволяет находить клоны, удовлетворяющие определению 1.1. Однако два средства находят клоны, похожие на те, которые удовлетворяют этому определению. Это средство Asta[1], разработанное в Microsoft Research, и средство CloneDR[2], разработанное компанией Semantic Designs. Asta находит клоны, которые могут отличаться в произвольных подвыражениях, но сами фрагменты, составляющие клон, являются отдельными операторами. CloneDR находит клоны, состоящие из последовательностей операторов, однако эти фрагменты могут отличаться только в именах функций и переменных и значениях констант.

Таким образом, определение 1.1 не рассматривалось ранее в литературе

[1] Evans W. S., Fraser C. W., Ma F. Clone detection via structural abstraction // Reverse Engineering, 2007.

[2] Clone detection using abstract syntax trees / I. D. Baxter, A. Yahin, L. Moura et al. // Proceedings of the International Conference on Software Maintenance, 1998.

и, вместе с тем, оно является обобщением существующих определений.

В *разделе 1.3* вводится ряд определений, которые будут использованы при описании разработанного алгоритма обнаружения клонов. Шаблоном называется дерево, в котором часть листьев помечена специальными метками-заполнителями. Дерево T называется примером шаблона U, если T получается из U заменой всех меток-заполнителей на некоторые поддеревья. Наиболее специальный шаблон (НСШ) двух деревьев T_1 и T_2 — это такой шаблон максимального размера, что и T_1, и T_2 являются его примерами.

В *разделе 1.4* приводится алгоритм поиска клонов, удовлетворяющих определению 1.1. Этот алгоритм состоит из трёх процедур.

На первом этапе все операторы разбиваются на группы похожих операторов (кластеры). Используется простой однопроходный алгоритм кластеризации. Каждый новый оператор сравнивается с существующими кластерами; в зависимости от результата сравнения он либо добавляется в подходящий кластер, либо формирует новый. В целях эффективности каждый новый оператор сравнивается не со всеми операторами некоторого кластера, а только с их общим НСШ. По окончании первого этапа все операторы будут помечены номерами кластеров, которым они принадлежат.

На следующем этапе проводится поиск одинаково помеченных последовательностей операторов (пометкой оператора является номер кластера, которому он принадлежит). В алгоритме поиска используются суффиксные деревья. По окончании второго этапа имеется множество пар последовательностей; в каждой последовательности на одинаковых позициях находятся похожие операторы (с точки зрения выбранной метрики ρ).

На последнем этапе все выявленные одинаково помеченные последовательности операторов сравниваются попарно для окончательного выделения клонов.

Разработанный алгоритм является эвристическим, т. е. в некоторых слу-

чаях он может находить не все клоны, удовлетворяющие определению 1.1. Поэтому в *разделе 1.4.4* приведено и доказано достаточное условие полноты алгоритма, т. е. способ получения по заданной программе таких параметров алгоритма, что при их использовании описанный алгоритм найдёт все клоны, удовлетворяющие определению 1.1.

В *разделе 1.5* проводится теоретическое сравнение возможностей разработанного и существующих алгоритмов.

В *разделе 1.6* описывается реализация разработанного алгоритма в программно-инструментальном средстве Clone Digger. Это средство распространяется под лицензией GNU General Public License (GPL) и может быть загружено с сайта `http://clonedigger.sourceforge.net`. Clone Digger написано на языке Python и поэтому является кроссплатформенным. В качестве языков программирования, на которых могут быть написаны анализируемые программы, в настоящее время поддерживаются Python версий 2.X, Java 1.5, Javascript, Lua. Найденные клоны записываются в HTML-отчёт в двухколоночном формате с подсветкой отличий.

В *разделе 1.7* приводятся результаты двух экспериментов, в которых разработанное средство Clone Digger сравнивалось с другими существующими средствами обнаружения клонов.

В первом эксперименте проводилось сравнение с средством CloneDR; как было указано в обзоре, это средство находит клоны, похожие на те, что находит разработаное средство Clone Digger. Было проверено, что все клоны, которые были выданы пробной версией CloneDR, могут быть обнаружены Clone Digger при некоторых значениях параметров-порогов. Кроме того, доказано, что некоторые клоны, найденные Clone Digger, не могут быть найдены CloneDR в принципе (при произвольных значениях порогов).

Во втором эксперименте проводилось сравнение с средствами Dup и CloneDR. Методика данного эксперимента является общепринятой при срав-

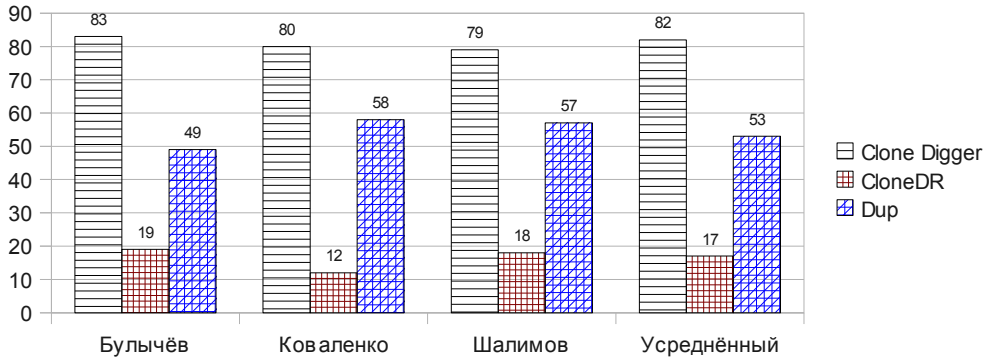

Рис. 1.1. Сравнение полноты обнаружения клонов

нении средств обнаружения клонов[3].

В начале эксперимента был построен объединённый список из всех 31500 клонов, найденных хотя бы одним из трёх средств в проекте Eclipse JDT Core. Затем 2% от этих клонов были классифицированы каждым из трёх экспертов (аспирантов ЛВК ВМиК МГУ), и было сформировано три эталонных множества, в которые попали только те клоны, от которых, по мнению эксперта, можно избавиться при помощи рефакторинга. Четвёртое эталонное множество было автоматически построено методом голосования.

Для каждого эталонного множества E и каждого рассматриваемого средства t были вычислены две характеристики: полнота (recall) и точность (precision) обнаружения клонов. Полнота — это процент от числа клонов из эталонного множества E, которые были обнаружены средством t. Точность — это процент от числа клонов, обнаруженных средством t, которые попали в множество E.

Полученные результаты представлены на диаграммах 1.1 и 1.2. Видно, что разработанное средство Clone Digger превосходит два других средства Dup и CloneDR как по точности, так и по полноте поиска клонов.

Во **второй главе** решается задача создания универсальной среды про-

[3] Bellon S., Koschke R., Antoniol G., Krinke J., Merlo E.. Comparison and Evaluation of Clone Detection Tools // IEEE Transactions on Software Engineering, 2007.

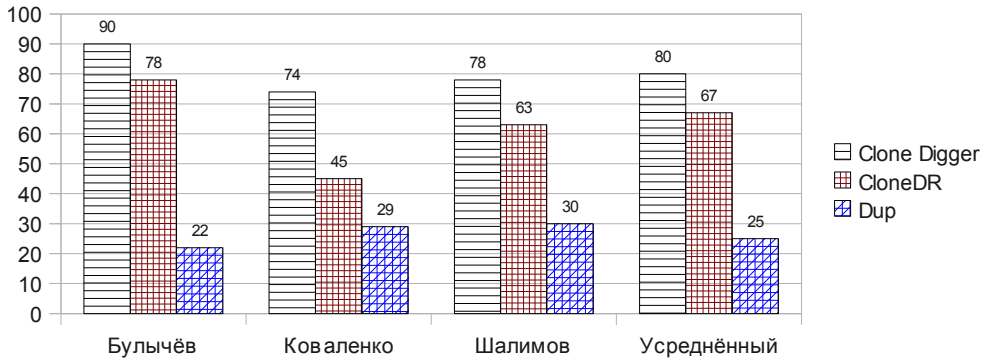

Рис. 1.2. Сравнение точности обнаружения клонов

верки симуляций между моделями программ, заданными конечными структурами Крипке.

Определение 2.1. Структура Крипке задаётся пятёркой $\langle S, s_0, R, \Sigma, L\rangle$, где S — конечное множество состояний, $s_0 \in S$ — начальное состояние, $R \subseteq S \times S$ — отношение переходов, Σ — множество атомарных переменных, $L : S \to 2^{\Sigma}$ — функция разметки состояний.

Для задания спецификаций поведения программ наиболее часто используется темпоральная логика ветвящегося времени CTL^* и её фрагменты LTL, $ACTL^*$, CTL^*_X и $ACTL^*_X$. Эти логики имеют различные выразительные способности и области применения. Так, CTL^* и $ACTL^*$ позволяют описывает свойства *деревьев вычислений*, а LTL — свойства отдельных вычислений. В логиках CTL^*_X, $ACTL^*_X$ и LTL_X отсутствует оператор X. Данный оператор используется для задания свойств, которые должны выполняться в *следующем* состоянии вычисления. Поэтому он не применяется для описания свойств параллельных асинхронных систем, поскольку порядок выполнения процессов в таких системах недетерменирован.

Будем говорить, что бинарное отношение $\preceq$ на множестве структур Крипке сохраняет выполнимость формул логики Λ, если для любой формулы $\varphi \in \Lambda$ и любых структур Крипке M_1 и M_2 таких, что $M_1 \preceq M_2$, из $M_2 \models \varphi$

следует $M_1 \models \varphi$.

Отношения симуляции — это специальная разновидность бинарных отношений на множестве моделей программ; говоря неформально, модель M_2 симулирует модель M_1, если дерево вычислений, порождённое моделью M_1, подобно некоторому фрагменту дерева вычислений, порождённому моделью M_2. Определяя по-разному подобие деревьев вычислений, можно получить большое разнообразие отношений симуляции (строгую симуляцию, прореженную симуляцию, ...). Наибольший интерес представляют отношения симуляции, сохраняющие выполнимость формул различных темпоральных логик.

В *разделе 2.1.3* даны определения строгой и прореженной симуляций (которые в дальнейшем будут использоваться в качестве сквозного примера) и вкратце описаны несколько других симуляций.

Определение 2.2 (Строгая симуляция)**.** Пусть даны две структуры Крипке M_1 и M_2, где $M_i = \langle S_i, s_{i0}, R_i, \Sigma_i, L_i \rangle$, $i = 1, 2$, и $\Sigma_1 = \Sigma_2$. Будем говорить, что между M_1 и M_2 выполняется строгая симуляция (обозначается $M_1 \preceq M_2$), если существует такое $H \subseteq S_1 \times S_2$, что $(s_{10}, s_{20}) \in H$ и для любой пары $(s_1, s_2) \in H$ выполняется:

1. $L_1(s_1) = L_2(s_2)$ и
2. если $(s_1, s_1') \in R_1$, то существует такое состояние $s_2' \in S_2$, что $(s_2, s_2') \in R_2$ и $(s_1', s_2') \in H$.

Будем говорить, что между M_1 и M_2 выполняется строгая бисимуляция, если в определении 2.2 отношение H является симметричным (т. е. $H = H^{-1}$).

Определение прореженной симуляции отличается от определения строгой симуляции тем, что каждому переходу $(s_1, s_1') \in R_1$ может соответствовать *последовательность* переходов $t_0, t_1, \ldots t_n$ $(n \geq 0)$ в R_2, такая, что $t_0 = s_2$, $(s_1', t_n) \in H$ и для всех $i < n$ выполняется $(t_i, t_{i+1}) \in R_2$ и $(s_1, t_i) \in H$.

Если добавить к этому определению требование симметричности, то мы придём к определению прореженной бисимуляции.

Строгая симуляция сохраняет выполнимость формул $ACTL^*$, строгая бисимуляция — выполнимость формул CTL^*, прореженная симуляция — выполнимость формул $ACTL^*_X$, прореженная бисимуляция — выполнимость формул CTL^*_X.

В данной главе преследуется цель разработки универсальной программно-инструментальной среды проверки симуляций. Разрабатываемая среда должна состоять из двух компонент: формального языка, позволяющего описывать различные симуляции, и программно-инструментального средства, которое по двум размеченным системам переходов и описанию симуляции проверяет, выполняется ли заданная симуляция между этими системами переходов.

На основании обзора, проведённого в *разделе 2.2*, был сделан вывод о том, что в основу универсального средства описания и проверки симуляций целесообразно положить теоретико-игровой подход. Этот подход годится для проверки всех известных отношений симуляции, и более того, во многих случаях его использование даёт оптимальные по сложности алгоритмы проверки симуляций.

В теоретико-игровом подходе задача проверки симуляции сводится к задаче нахождения выигрышной стратегии в антагонистической игре двух игроков Spoiler и Duplicator (или для краткости S и D).

Формально такая игра задаётся пятёркой $\langle V_S, V_D, E_S, E_D, v_0\rangle$, где V_S и V_D — множества состояний игроков Spoiler и Duplicator соответственно, $E_S \subseteq V_S \times V_D$ и $E_D \subseteq V_D \times V_S$ — множества допустимых ходов, $v_0 \in V_S$ — начальное состояние. История игры — это такая конечная или бесконечная последовательность игровых состояний $(v_0, v_1, v_2, \dots)$, что $(v_i, v_{i+1}) \in E_S \cup E_D$ для любого $i \geq 0$. Стратегией игрока Duplicator — это функция $W : V_D \to V_S \cup \{halt\}$.

История игры $(v_0, v_1, v_2, \dots)$ удовлетворяет стратегии W игрока Duplicator, если для любого состояния $v_i \in V_D$ из истории игры, за исключением, возможно, последнего, выполняется $v_{i+1} = W(v_i)$, а для последнего состояния (если история игры конечна) выполняется $W(v_i) = \mathit{halt}$. Игрок Duplicator выигрывает в игре, если у него существует стратегия, для которой любая удовлетворяющая ей история игры либо является бесконечной, либо завершается состоянием игрока Spoiler. Игры проверки симуляции проектируются таким образом, что выполнимость отношения симуляции равносильна существованию выигрышной стратегии игрока Duplicator в соответствующей игре проверки симуляции.

Различные варианты игр для проверки многих отношений симуляции имеют много общего. В них всегда участвуют два игрока и условия выигрыша совпадают (один игрок выигрывает, если другой игрок не может сделать ход). Игровые состояния представляются кортежами, компонентами которых являются компоненты моделей. Множества допустимых ходов могут быть описаны булевыми формулами над фиксированной сигнатурой. Это наблюдение было использовано для создания теоретико-игрового языка задания симуляций, который описывается в *разделе 2.3*. Определение некоторого отношения на этом языке представляет собой описание правил игры для проверки этого отношения и включает в себя описание устройства игровых состояний и правил, по которым игроки совершают свои ходы.

Самый простой вид имеет определение правил игры проверки строгой симуляции. Предположим, что необходимо проверить симуляцию между M_1 и M_2, где $M_i = \langle S_i, s_{i0}, R_i, \Sigma_i, L_i \rangle$. Состояния игроков Spoiler и Duplicator в этой игре представляются кортежами, состоящими из одного состояния из S_1 и одного состояния из S_2, т. е. $V_S = V_D = \{(s_1, s_2) | s_1 \in S_1 \wedge s_2 \in S_2)\}$. Если состояния игрока P описывается парой значений типизированных переменных x_P^1 и x_P^2 (x_P^i принимает значения из S_i), то допустимые переходы игроков

описываются формулами

$$\varphi_S(x_S^1, x_S^2, x_D^1, x_D^2) \equiv (x_S^2 = x_D^2) \wedge R_1(x_S^1, x_D^1)$$
$$\varphi_D(x_D^1, x_D^2, x_S^1, x_S^2) \equiv (x_S^1 = x_D^1) \wedge R_2(x_S^2, x_D^2) \wedge (L_1(x_D^1) = L_2(x_D^2))$$

Поэтому множества ходов будут иметь следующий вид: $E_S = \{((s_1, s_2), (s_1', s_2))|(s_1, s_1') \in R_1\}$ и $E_D = \{((s_1, s_2), (s_1, s_2'))|(s_1, s_2') \in R_2 \wedge L(s_1) = L(s_2')\}$. Начальное игровое состояние полагается равным $v_0 = (s_{10}, s_{20})$.

Всего на разработанном языке были записаны правила проверки следующий отношений: строгой симуляции и бисимуляции, прореженной симуляции и бисимуляции, слабой симуляции (сохраняющей выполнимость формул LTL_X), k_2 симуляции (сохраняющей выполнимость формул LTL). В диссертационной работе доказана корректность игр для проверки прореженной симуляции и бисимуляции, корректность игр для проверки остальных симуляций была доказана ранее в литературе.

В *разделе 2.4* приводится алгоритм проверки симуляций, который по двум конечным структурам Крипке и по отношению симуляции, заданном на разработанном языке, проверяет выполнимость этого отношения между этими моделями. В алгоритме последовательно проводится построение множества достижимых состояний игры, построение множества выигрышных для игрока Spoiler состояний и проверка начального состояния на принадлежность этому множеству. Разработанный алгоритм является символьным, т. е. для повышения эффективности в нём проводятся операции над формулами, характеризующими множества, а не над отдельными элементами этих множеств. Доказана корректность приведённого алгоритма.

В *разделе 2.5* описывается разработанное универсальное программно-инструментальное средство проверки симуляций между парами структур Крипке. Данное средство реализовано на языке Python. В качестве языка задания

моделей используется язык широко используемого верификатора NuSMV[4]. Для представления формул в символьном алгоритме были использованы упорядоченные двоичные разрешающие диаграммы (OBDD). В равной мере для символьного представления моделей и игры могут быть использованы и другие математические конструкции: регулярные выражения, прямое формульное представление и др. OBDD были выбраны ввиду широкой распространённости библиотек для манипуляции с ними. В данной диссертационной работе в качестве библиотеки работы с OBDD используется библиотека Cudd.

В *разделе 2.6* приводится пример применения средства проверки симуляции при инкрементальном построении моделей программ. В инкрементальном методе последовательно строятся модели $M_1, \ldots, M_n$ так, что каждая следующая модель является уточнением предыдущей модели и содержит больше подробностей о конструируемой программе. Такой способ построения помогает избежать ошибок при построении моделей. Чтобы убедиться, что M_{i+1} уточняет M_i, достаточно проверить, что между этими моделями выполняется некоторое отношение симуляции (которое выбирается исходя из характера отличий между M_{i+1} и M_i). Если же это отношение симуляции не выполняется, то контрпример, выданный средством проверки симуляций, поможет найти ошибку в модели M_{i+1}.

В диссертационной работе инкрементальный подход был применён для решения задачи справедливого разделения ресурсов в системе асинхронных взаимодействующих процессов (задача "обедающих философов"). Всего были построены 3 модели. В модели M_1 не накладывалось никаких ограничений на действия процессов (философов). В модели M_2 появляется маркер; каждый философ может получить маркер только когда вилки слева и справа от него лежат на столе. После того, как философ получил маркер, он может взять вилки и начать есть; после того как он закончил есть, он освобожда-

[4] NuSMV model checker: `http://nusmv.irst.itc.it`

ет маркер. В окончательной модели M_3 маркер движется строго по кругу, и каждый философ обязан закончить есть прежде, чем он передаст маркер своему соседу.

Исходя из характера отличий между моделями был сделан вывод о том, что между M_2 и M_1 должно выполнятся отношение строгой симуляции, а между M_3 и M_2 — отношение прореженной симуляции. Справедливость этих отношений была проверена при помощи разработанного средства для систем с различным количеством обедающих философов.

В **третьей главе** решается задача разработки алгоритма проверки симуляции между временными игровыми автоматами.

В *разделе 3.1* даются вводные определения. Будем использовать запись $\mathcal{B}(X)$ для обозначения множества формул над множеством X, удовлетворяющих грамматике $\varphi ::= x \sim k \mid \varphi \wedge \varphi$, где $k \in \mathbb{Z}_{\geq 0}$, $x \in X$ и $\sim \in \{<, \leq, =, >, \geq\}$. Будем использовать запись X^Y для обозначения множества всех отображений из множества X в множество Y (это расходится с традиционной записью Y^X). Оценка переменных из X — это элемент $\mathbb{R}^X_{\geq 0}$. Пусть $\delta \in \mathbb{R}_{\geq 0}$, тогда будем обозначать $v + \delta$ такую оценку, что для всех $x \in X$ выполняется $(v+\delta)(x) = v(x)+\delta$. Пусть $Y \subseteq X$, тогда будем использовать запись $v[Y]$ для обозначения оценки, сопоставляющей 0 всем $x \in Y$, и $v(x)$ всем $x \in X \setminus Y$.

Определение 3.1. Временной автомат (timed automata, TA) — это шестёрка $\langle S, s_0, X, \Sigma, R, Inv \rangle$, где S — конечное множество дискретных состояний, $s_0 \in S$ — начальное дискретное состояние, X — конечное множество вещественных переменных (таймеров), Σ — множество пометок переходов (действий), включающее в себя так называемое невидимое действие τ, $R \subseteq S \times \mathcal{B}(X) \times \Sigma \times \wp(X) \times S$ — конечное множество переходов ($\wp(X)$ — множество всех подмножеств X), $Inv : S \to \mathcal{B}(X)$ — функция, сопоставляющая дискретным состояниям формулы (такие формулы называются инвариантами).

Семантика временного автомата определяется размеченной системой переходов $(Q, q_0, \to)$, где $Q = S \times \mathbb{R}^X_{\geq 0}$, $q_0 = (s_0, \vec{0})$. Из состояния q_1 существует дискретный переход по действию a в состояние q_2 (обозначается $q_1 \xrightarrow{a} q_2$), если существует такой переход $e = (s, g, a, Y, s') \in R$, что $v \models g$, $v' = v[Y]$ и $v' \models Inv(s')$. Из состояния $q_1 = (s, v)$ существует задержка по времени δ в состояние $q_2 = (s, v')$ (обозначается $q_1 \xrightarrow{\delta} q_2$), если $v' = v + \delta$ и $v' \models Inv(s)$.

Для простоты будем полагать, что временные автоматы детерменированы по действиям, т. е. для любой пары $q \in Q$ и $a \in \Sigma$ существует не более одного такого q', что $q \xrightarrow{a} q'$. Временной игровой автомат (timed game automata, TGA) — это временной автомат, в котором множество действий Σ разбито на множество контролируемых (Σ^c) и множество неконтролируемых (Σ^u) переходов. Стратегия с нулевой памятью контроллера (соответственно, среды) — это функция f, определённая на множестве состояний Q, и возвращающая некоторый элемент множества $(\mathbb{R}_{\geq 0} \times \Sigma^c)$ (соответственно, $(\mathbb{R}_{\geq 0} \times \Sigma^u)$).

Определение 3.5. Пусть дан TGA $\langle S, l_0, \Sigma, X, R, Inv \rangle$, и пусть (δ^c, a^c) и (δ^u, a^u) — некоторые стратегии контроллера и среды соответственно. Вычисление $q_0 \xrightarrow{\delta_0} q_2 \xrightarrow{a_0} q_3 \xrightarrow{\delta_1} q_4 \xrightarrow{a_1} q_5 \ldots$ будем называть порождённым этими стратегиями, если для любого $i \in \mathbb{N}$ выполняется $\delta_i = \min(\delta^c(q_{2i}), \delta^u(q_{2i}))$ и:

$$a_i = \begin{cases} a^u(q_{2i}) & \text{если } \delta^u(q_{2i}) \leq \delta^c(q_{2i}) \\ a^c(q_{2i}) & \text{иначе} \end{cases}$$

Аналогично определяются понятие стратегии для *недетерменированных* по действиям временных игровых автоматов, в этом случае стратегия будет возвращать не пометки, а переходы.

В *разделе 3.1.4* определяется логика ATCTL, используемая для описания свойств временных игровых автоматов. Эта логика состоит из формул вида $A\sigma_1 U_t \sigma_2$ и $A\sigma_1 W_t \sigma_2$, где $t \in \mathbb{Z}_{\geq 0} \cup \{+\infty\}$, а σ_1 и σ_2 — некоторые множества видимых действий. Будем говорить, что вычисление π временного

автомата M удовлетворяет формуле $\sigma_1 U_t \sigma_2$, если существует такой префикс этого вычисления π', что все видимые действия π' лежат в множестве σ_1, последнее действие π' лежит в σ_2 и длительность π' не превышает t. Вычисление π временного автомата M удовлетворяет формуле $\sigma_1 W_t \sigma_2$, если это вычисление удовлетворяет формуле $\sigma_1 U_t \sigma_2$, или все видимые действия π принадлежат множеству σ_1. Будем говорить, что формула $A\varphi$ логики $ATCTL$ выполняется на TGA M, если существует такая стратегия контроллера f_c, что для любой стратегии среды f_u формула φ выполняется на вычислении, порождённом стратегиями f_c и f_u.

Задача, которая решается в данной главе, заключается в определении отношения $\preceq$, сохраняющего выполнимость формул логики $ATCTL$, и разработке алгоритма проверки этого отношения. В *разделе 3.2* приводится обзор существующих кандидатов на роль такого отношения. На основании обзора сделан вывод о том, что в качестве искомого отношения следует использовать "комбинацию" определений временной, игровой и ослабленной отношений симуляции, описанных ранее в литературе.

В *разделе 3.3* даётся предложенное автором определение симуляции:

Определение 3.7 (twa-симуляция)**.** Пусть даны два TGA M_1 и M_2 с множествами состояний Q_1 и Q_2, где $M_i = \langle S_i, s_{i0}, X_i, \Sigma_i, R_i, Inv_i \rangle$. Будем говорить, что между M_1 и M_2 выполняется отношение слабой альтернирующей временной (timed weak alternating, twa) симуляции, если существует такое отношение $H \subseteq Q_1 \times Q_2$, что пара начальных состояний (q_{10}, q_{20}) находится в этом отношении и для любой пары $(q_1, q_2) \in H$ и для любых $q_1' \in Q_1$, $q_2' \in Q_2$, $\delta \in \mathbb{R}_{\geq 0}$, $a \in \Sigma \setminus \{\tau\}$ выполняются следующие требования:

1. если $(q_2 \xrightarrow{a}_c q_2')$, то существует такое $n \geq 1$ и такая последовательность состояний $q_{1,1}', q_{1,2}', \ldots, q_{1,n}'$ автомата M_1, что $q_{1,1}' = q_1$, $q_{1,n-1}' \xrightarrow{a}_c q_{1,n}'$, $(q_{1,n}', q_2') \in H$ и при этом для всех $i = 2..n-1$ выполняется $(q_{1,i}', q_2) \in H$

и $q'_{1,i-1} \xrightarrow{\tau}_c q'_{1,i}$

2. если $(q_1 \xrightarrow{a}_u q'_1)$, то существует такое $n \geq 1$ и такая последовательность состояний $q'_{2,1}, q'_{2,2}, \ldots, q'_{2,n}$ автомата M_2, что $q'_{2,1} = q_2$, $q'_{1,n-1} \xrightarrow{a}_u q'_{1,n}$, $(q'_1, q'_{2,n}) \in H$ и при этом для всех $i = 2..n-1$ выполняется $q'_{1,i-1} \xrightarrow{\tau}_u q'_{1,i}$

3. если $q_2 \xrightarrow{\tau}_c q'_2$, то $(q_1, q'_2) \in H$

4. если $q_1 \xrightarrow{\tau}_u q'_1$, то $(q'_1, q_2) \in H$

5. если $q_2 \xrightarrow{\delta} q'_2$, то существует такое $q''_1 \in Q_1$, что $q_1 \xrightarrow{\delta} q''_1$ и $(q''_1, q'_2) \in H$

В *теореме 3.1* доказано, что введённое отношение twa-симуляции сохраняет выполнимость формул логики $ATCTL$.

В *разделе 3.4* приведён разработанный теоретико-игровой алгоритм проверки twa-симуляции и доказана его корректность. Поскольку в общем случае множество состояний временных автоматов бесконечно, то бесконечна и игра проверки симуляции. Поэтому в разработанном алгоритме ведётся работа не над отдельными игровыми состояниями, а над формулами, описывающими потенциально бесконечные множества состояний. Для операций над такими формулами используется аппарат матриц ограниченных разностей (difference bounded matrixes).

В *разделе 3.5* описана реализация разработанного алгоритма в рамках среды анализа временных игровых автоматов UPPAAL Tiga.

В *разделе 3.6* описано экспериментальное исследование реализации разработанного алгоритма twa-симуляции для решения задачи синтеза алгоритма алгоритма работы контроллера в системе автоматического управления. Рассмотрим следующий пример. Пусть имеется движущийся конвейер, на который ставится изделие. Затем изделие движется по конвейеру и последовательно проходит N этапов обработки, каждый этап занимает от 1 до 2 единиц времени. Задача контроллера — после завершения последнего этапа

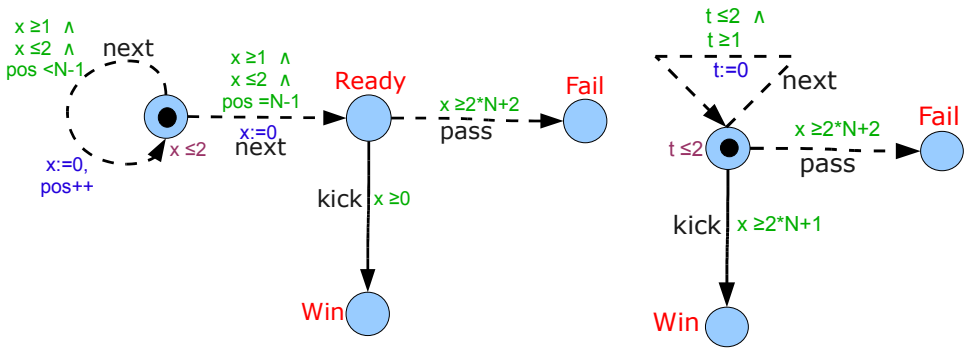

Рис. 3.3. Временные автоматы, на которых проводилось экспериментальное исследование разработанного алгоритма

переместить готовое изделие с конвейера в коробку. Если этого не произойдёт в течение $2N + 2$ единиц времени с начала работы конвейера, то изделие будет потеряно, и, таким образом, задача контроллера не будет выполнена. Данная система описывается TGA S, изображённом на рисунке 3.3 слева. Сплошными дугами обозначены контролируемые переходы, а прерывистыми — неконтролируемые.

Задача контроллера в автомате S — перевести автомат в дискретное состояние Win, т. е. обеспечить выполнимость формулы $A\{next\}U\{kick\}$. Одна из возможных выигрышных стратегий контроллера f^c — подождать, пока автомат перейдёт в дискретное состояние $Ready$, а затем выполнить переход в состояние Win. Предположим теперь, что в реальной ситуации контроллеру неизвестно текущее положение изделие на конвейере. Тогда стратегия f^c не может использоваться в моделируемой (реальной) ситуации.

Поэтому была построена другая модель A, изображённая на рисунке 3.3 справа. Эта модель уже не содержит невидимую контроллеру информацию, и для неё у контроллера существует выигрышная стратегия (её можно автоматически построить при помощи средства UPPAAL Tiga). Для того, чтобы проверить, что та же стратегия будет выигрышной и для исходной модели, достаточно проверить существование twa-симуляции между S и A. Коррект-

ность такого сведения была доказана в *теореме 3.2*. Существование twa-симуляции между S и A было проверено для нескольких значений N.

В **приложении** даются инструкции по использованию разработанного средства обнаружения клонов и разработанного средства проверки симуляций между структурами Крипке.

Основные результаты работы.

1. Предложена математическая модель программных клонов, совместимая с существующими методами рефакторинга. Разработан, реализован и проверен на практике алгоритм поиска клонов, удовлетворяющих этой модели.

2. Предложен формальный язык представления отношений симуляции между структурами Крипке, создан, реализован и испытан на практике универсальный (общий) алгоритм проверки симуляций, записанных на этом языке.

3. Дано определение отношения подобия между временными игровыми автоматами, сохраняющее выполнимость формул логики ATCTL. Разработан и испытан на практике алгоритм проверки этого отношения.

ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. Булычев П.Е., Захаров В.А., Коннов И.В. Применение методов теории игр к поиску некоторых видов симуляции на размеченных системах переходов // Труды седьмой международной конференции "Дискретные модели в теории управляющих систем 2006.

2. Булычев П.Е., Булычёва О.Н., Захаров В.А. Применение методов теории игр к поиску некоторых видов симуляции на размеченных системах переходов с ограничениями справедливости. // Вестник МЭИ, т. 6, 2007. С. 5-9.

3. Bulychev P.E., Konnov I.V., Zakharov V.A. Computing (bi)simulation relations preserving CTL*_X for ordinary and fair Kripke structures. // Сборник «Математические методы и алгоритмы», ИСПРАН. - 2007. - Том 12.

4. Bulychev P., Minea M. Duplicate code detection using anti-unification. Proceedings of the 1st Spring Young Researchers' Colloquium on Software Engineering SYRCoSE 2008, 2008. Pp. 51–54.

5. Bulychev P., Minea M. An evaluation of duplicate code detection using anti-unification. Proceedings of 3rd International Workshop on Software Clones, 2009, IESE-Report; 038.09/E. Pp. 22–27.

6. Bulychev P., Kostylev E., Zakharov V. Anti-unification algorithms and their applications in program analysis. Proceedings of Seventh International Andrei Ershov Memorial Conference, 2009, Lecture Notes in Computer Science vol. 5947, Pp. 413–423.

7. Bulychev P., Chatain Th., David A., Larsen K.G. Efficient on-the-fly algorithm for checking alternating timed simulation. Proceedings of Formal Modeling

and Analysis of Timed Systems, 2009, Lecture Notes in Computer Science vol. 5813. Pp. 73–87.

8. Булычев П.Е., Захаров В.А. О верификации конечных параметризованных моделей распределенных программ // Научные ведомости Белгородского государственного университета. — 2009. — №9. — С. 116–123.

9. Булычёв П.Е., Захаров В.А. Универсальный подход к проверке отношений симуляции моделей программ // Труды научной конференции Методы и Средства Обработки Информации. — 2009. С. 104–109.

10. Булычёв П.Е., Шалимов А.В., Коваленко Д.С. Экспериментальное сравнение средств обнаружения клонов // Труды молодёжной международной научной олимпиады “Ломоносов-2010”. — 2010. С. 11-12.