На правах рукописи

Кондратьев Дмитрий Александрович

МЕТОДЫ КОМПЛЕКСНОГО ПОДХОДА К АВТОМАТИЗАЦИИ ДЕДУКТИВНОЙ ВЕРИФИКАЦИИ ПРОГРАММ С ФИНИТНЫМИ ИТЕРАЦИЯМИ

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

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата физико-математических наук

Новосибирск – 2022

Работа выполнена в *Федеральном государственном бюджетном учреждении науки Институте систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук.*

Научный руководитель: **Промский Алексей Владимирович**
кандидат физико-математических наук, заместитель директора по научной работе ФГБУН Института систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук

Официальные оппоненты: **Петренко Александр Константинович**
доктор физико-математических наук, профессор, ФГБУН Институт системного программирования им. В.П. Иванникова Российской академии наук, заведующий отделом Технологий программирования

Скопин Игорь Николаевич
кандидат физико-математических наук, ФГБУН Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук, старший научный сотрудник Лаборатории синтеза параллельных программ

Ведущая организация: Федеральное государственное бюджетное образовательное учреждение высшего образования «Ярославский государственный университет им. П.Г. Демидова»

Защита состоится 14 сентября 2022 г. в 15 часов на заседании диссертационного совета *Д 999.082.03* на базе *Федерального государственного бюджетного учреждения науки Института систем информатики им. А.П. Ершова Сибирского отделения Российской академии наук (ИСИ СО РАН)* по адресу: *630090, г. Новосибирск, проспект академика Лаврентьева, 6, комн. 254.*

С диссертацией можно ознакомиться в библиотеке и на сайте ИСИ СО РАН: https://www.iis.nsk.su/files/Thesis_Kondratyev.pdf

Автореферат разослан «15» июля 2022 г.

Ученый секретарь
диссертационного совета
Д 999.082.03
к.ф.-м.н. *Промский Алексей Владимирович*

Общая характеристика работы

Актуальность темы исследования. Автоматизация формальной верификации программ — актуальная задача современного программирования. Дедуктивная верификация является сопоставлением программы и ее спецификаций, заданных в виде логических формул. Типичными спецификациями являются пред- и постусловие программы, а также инварианты циклов. Классический процесс верификации заключается в выводе условий корректности (УК) с помощью генератора условий корректности (ГУК), который воплощает в себе аксиоматическую семантику целевого языка программирования. Далее УК проверяются с помощью системы поддержки доказательства теорем. В случае истинности всех УК пара программа-спецификации является согласованной.

Примером подобной системы является проект C-lightVer, реализуемый в Лаборатории теоретического программирования ИСИ СО РАН [14, 35]. Входной язык C-light — выразительное подмножество стандарта C99. В языке C-light выделено ядро — язык C-kernel, для которого была разработана аксиоматическая семантика в стиле Хоара. Процесс дедуктивной верификации разбивается на три этапа. На первом этапе аннотированная C-light программа транслируется в эквивалентную аннотированную программу на языке C-kernel. Далее с помощью аксиоматической семантики языка C-kernel выводятся условия корректности (УК). Потом следует этап доказательства полученных УК.

Отметим, что ранее в системе C-lightVer была реализована процедура, позволяющая от конструкций промежуточной C-kernel программы вернуться к конструкциям исходной C-light программы [35], а также метод смешанной аксиоматической семантики [35] и метод метагенерации УК [9]. Процедура сопоставления промежуточной C-kernel программы и исходной C-light программы позволяет упростить нахождение ошибки в исходной программе в случае ее нахождения в промежуточном представлении. Метод смешанной аксиоматической семантики позволяет использовать специальные версии правил вывода для определенных программных конструкций, что может приводить к генерации более простых УК [35]. Метагенерация УК позволяет использовать правила вывода УК как входные данные системы верификации, что может упростить задачу расширения генератора УК новыми правилами вывода в случае расширения входного языка новыми конструкциями.

Исследования и разработки, осуществлявшиеся автором в проекте C-lightVer в период 2014–2022 гг., послужили основным научным и практическим заделом данной диссертации.

Несмотря на более чем 50-летний период развития теории дедуктивной верификации, в ней можно выделить ряд ключевых проблем, из-за которых она по-прежнему остается во многом уделом академической сре-

ды и непопулярна среди обычных программистов. Основной целью диссертации стала разработка комплекса методов, направленных на решение трех проблем: проблемы инвариантов циклов, проблемы локализации и объяснения ошибок и проблемы автоматизации доказательства УК.

Рассмотрим проблему инвариантов циклов. Инвариант цикла — это утверждение, которое должно выполняться перед началом исполнения, на любой итерации и по завершению цикла. В общем случае задача автоматической генерации инвариантов циклов алгоритмически неразрешима, поэтому частичным решением может стать выделение класса алгоритмов, для которых можно вообще обойтись без инвариантов. За основу в диссертации был взят символический метод верификации финитных итераций. Он применяется к циклам специального вида (финитным итерациям). Тело финитной итерации исполняется один раз для каждого элемента структуры данных конечной размерности. Несмотря на кажущуюся вырожденность, эта ситуация в точности соответствует такому фундаментальному разделу информатики, как итеративная обработка структур данных последовательной природы — строки, массивы, списки, очереди, частично деревья. Основой этого метода является символическая замена финитных итераций специальными рекурсивными функциями, называемыми операциями замены. В рамках проекта РФФИ № 17-01-00789 «Платформенно-независимый подход к формальной спецификации и верификации стандартных математических функций» важной задачей являлась разработка алгоритма генерации операций замены [6], позволяющего порождать условия корректности для программ с финитными итерациями без использования инвариантов циклов. Разработанный алгоритм стал одним из результатов диссертации.

Рассмотрим проблему локализации и объяснения ошибок. Исторически дедуктивная верификация была ориентирована не на локализацию ошибок, а на доказательство отсутствия ошибок. Поэтому значительный интерес представляет задача интерпретации недоказанных УК и соотнесение их с местом потенциальной ошибки в программе или в спецификации.

Денни и Фишер предложили добавить в правила вывода УК семантическую разметку для объяснения результата применения правила. Генератор УК в процессе вывода добавляет к различным подформулам соответствующие метки, которые извлекаются из УК и переводятся в текст о соответствии УК и фрагментов программы. Отчет о результатах верификации, основан на информации, хранящейся в семантических метках.

Но Денни и Фишер предложили лишь ограниченный набор типов семантических меток. В рамках проекта РФФИ № 11-01-00028 «Интегрированный мультиязыковый подход к верификации императивных программ» важной задачей являлась разработка языка, позволяющего использовать в правилах вывода предложенные пользователем семантиче-

ские метки [14, 23]. Разработанный язык стал одним из результатов диссертации. Также результатом диссертации стала разработка семантических меток для финитных итераций [2, 8, 12, 20]. Для более точной локализации ошибок оказалось полезным проверять циклы на выполнение определенных ошибочных свойств и генерировать объяснения в случае выполнения таких свойств. На основе таких проверок был разработан ряд автоматизированных стратегий локализации ошибок [8, 12, 32, 33].

Рассмотрим проблему автоматизации доказательства УК. Несмотря на достаточную развитость систем поддержки доказательства теорем, они не всегда справляются с доказательством УК в полностью автоматическом режиме. Причиной тому может быть необходимость доказательства по индукции из-за наличия в УК применений рекурсивных функций, сложная структура УК и т.д. Поэтому, актуальна задача разработки стратегий для различных классов программ, позволяющих автоматизировать доказательство УК этих программ. В рамках проекта РФФИ № 15-01-05974 «Онтологический подход к формальной семантике языков программирования» важной задачей являлась разработка стратегий автоматизации доказательства УК, содержащих применение функций, выражающих результаты финитных итераций [6]. Разработанные стратегии стали одним из результатов диссертации [3, 6–8, 10–12, 18, 21, 29–33]. Важным достоинством набора стратегий стало доказательство их корректности.

Рассмотрим расширение методов на другие языки и парадигмы и общее повышение автоматизируемости процесса верификации. Помимо разработки теоретических методов решения рассмотренных проблем, диссертация нацелена и на решение важных практических задач. Интерес представляют методы, пригодные не только для языка C, но и для других языков и парадигм программирования.

Важным результатом диссертации стало применение разработанных методов и стратегий в проекте по созданию Системы облачного параллельного программирования (CPPS) [15]. Cloud Sisal является входным языком системы CPPS. Cloud Sisal является функциональным языком программирования, основанным на циклических выражениях. Главной особенностью системы CPPS является неявное параллельное исполнение, основанное на автоматическом распараллеливании циклов Cloud Sisal. Кодогенератор системы CPPS позволяет генерировать код на языке C, соответствующий различным этапам оптимизации входной программы. Работа в проекте шла по двум направлениям: применение системы C-lightVer к готовому промежуточному представлению Sisal-программы на языке C [10, 11, 31] и непосредственная разработка аксиоматической семантики Cloud Sisal [13, 27] с расширением системы C-lightVer новыми правилами [1, 4, 13, 22]. В рамках проекта РНФ № 18-11-00118 «Облачные методы и

средства конструирования эффективных и надежных параллельных программ на основе функциональных спецификаций и семантических преобразований» важной задачей являлось создание расширения языка C конструкциями Cloud Sisal [1, 22], семантики такого расширения [1, 22] и реализация такой семантики в системе C-lightVer [1]. Аксиоматическая семантика созданного расширения языка C конструкциями Cloud Sisal стала одним из результатов диссертации.

Итоговым результатом данной диссертации стал комплексный подход к автоматизации дедуктивной верификации без использования инвариантов циклов. Комплекс включает в себя метод, позволяющий генерировать УК для программ с финитными итерациями без инвариантов циклов, стратегии, позволяющие автоматизировать доказательство порождаемых УК и метод, позволяющий автоматизировать локализацию и объяснение ошибок для программ с финитными итерациями. Комплексный подход оказался применимым для языков C и Cloud Sisal, которые являются яркими представителями процедурной и функциональной парадигмы соответственно.

Степень разработанности темы исследования и обзор родственных работ. Рассмотрим работы об автоматизации дедуктивной верификации программ с циклами. Подход, похожий на символический метод верификации финитных итераций, предложили Майрен и Гордон (Myreen, Gordon). Они также предлагают заменять действие цикла на результат применения рекурсивной функции. Но Майрен и Гордон рассматривают модельный язык, более простой, чем язык C-light. Подобный подход также предложен в работе Бланка (Blanc) и др. для Scala-программ. Однако, в работе предлагается использовать инварианты, которые транслируются в аннотации соответствующих циклам рекурсивных функций. В отличие от символического метода верификации финитных итераций, большинство работ в этой области основано на генерации инвариантов циклов. Ковач (Kovács) предложила метод, основанный на базисах Гребнера, для генерации инвариантов P-разрешимых циклов. Но, в отличие от финитных итераций, правые части присваиваний в теле P-разрешимых циклов должны иметь вид полиномов. В работе Чакраборти (Chakraborty) и др. предлагается генерировать инварианты специального вида для циклов над массивами. Но авторы не рассматривают циклы с инструкцией `break`. В работе Галеотти (Galeotti) и др. предложен динамический метод, основанный на известной идее модификации постусловия, но авторы не доказывали свойство перестановочности, которое традицинно является важным для верификации программ сортировки. Отказ от доказательства этого свойства привел к генерации таких инвариантов циклов, которые могут позволить доказать только свойство упорядоченности. В работе Сриваставы (Srivastava) и др. предложено ис-

пользовать шаблоны инвариантов, задаваемые пользователями, однако, для классических программ сортировки доказано более слабое свойство, чем свойство перестановочности.

Рассмотрим работы в области автоматизации доказательства УК. Распространенным подходом является генерация таких лемм, которые могут помочь доказать целевую теорему. Такой подход для системы доказательства ACL2 предложили Херас (Heras) и др. В отличие от предложенных нами стратегий доказательства для системы ACL2, стратегии Хераса и др. основаны на машинном обучении. Но машинное обучение плохо подходит для доказательства УК, так как предметные области могут сильно отличаться для разных программ. В работе Имине и Ранизе (Imine, Ranise) были предложены стратегии доказательства для УК сортировки вставками, однако не все УК были доказаны автоматически. Актуальность разработки стратегий автоматизации доказательства УК сортировки вставками подтверждает работа Жианга и Жоу (Jiang, Zhou). В работе Сафари и Хьюсман (Safari, Huisman) предложены стратегии автоматизации доказательства свойства перестановочности для известных программ сортировки, но в этих экспериментах авторы задавали инварианты для каждого цикла. В работе де Анжелиса (de Angelis) и др. на базе модельного функционального языка предложена стратегия для автоматизации верификации сортировки с помощью дизъюнктов Хорна. В работе Костюкова (Kostyukov) и др. описано, как дизъюнкты Хорна могут упростить доказательство свойств программ над рекурсивными типами данных, но при использовании такого подхода требуются специальные стратегии. Для системы доказательства Lean были предложены тактики для автоматизации доказательства по индукции, основанные на преобразовании индукционных гипотез. Применение этих тактик было продемонстрировано для доказательства свойств программ на модельном языке. Лейно (Leino) предложил способ автоматизации доказательства по индукции для SMT-решателя Z3. Этот способ основан на классической идее доказательства шага индукции и доказательстве индукционного перехода. Однако, для автоматизации доказательства УК такой тактики недостаточно [3]. Рейнольдс и Кунчак (Reynolds, Kuncak) предложили набор стратегий автоматизации доказательства по индукции для SMT-решателя CVC4. Главной из этих стратегий является индукция по рекурсивному определению структуры данных. Подобная стратегия стала основной стратегией автоматизации доказательства по индукции в новой версии системы Vampire. Но индукции по рекурсивному определению структуры данных также недостаточно для автоматизации доказательства УК [3].

Рассмотрим работы со специализированными стратегиями, ориентированными на упрощение формальной верификации. Туэрк (Tuerk) предлагает задавать для циклов предусловия и постусловия специального ви-

да вместо инвариантов. Обобщение данного подхода описано в статье Эрнста (Ernst). В работе Волкова и др. рассматривается метод лемма-функций, реализованный в системе AstraVer. Этот метод основан на использовании спецификаций специального вида вместо инвариантов. В работе Бланшарда (Blanchard) и др. описан похожий метод, реализованный в системе Frama-C. Однако, эти методы основаны на задании спецификаций пользователем.

Рассмотрим работы в области автоматизации локализации ошибок при дедуктивной верификации. Дайлер (Dailler) и др. описали использование контрпримера, сгенерированного SMT-решателем, для локализации ошибок. Но анализ контрпримера может оказаться достаточно сложным, что продемонстрировано в работе Бекера (Becker) и др. Денни и Фишер (Denney, Fischer) предложили способ установления соответствия между УК и исходным кодом. В работе Кенигхофера (Könighofer) и др. описан новый подход в системе Frama-C к автоматизации локализации ошибок при дедуктивной верификации, основанный на изменении выражений программы. Раад (Raad) и др. предложили логику, которую они назвали некорректной логикой разделения. Истинность специальных формул в этой логике означает наличие ошибок в программе. Однако, Раад и др. предложили сложную модель памяти, приводящую к генерации сложных формул в отличие от используемого нами метода смешанной аксиоматической семантики [14]. В работе де Гоува (de Gouw) и др. описана локализация ошибки при верификации реализации сортировки в Java-машине OpenJDK. Ошибка была локализована с помощью доказательства невыполнения инварианта в определенных случаях. Но решение проблемы инвариантов циклов не было автоматизировано в работах. В работе Меллера (Möller) и др. предложен единый подход к доказательству корректности программ и наличия ошибок в программах. Комплексный подход, описанный в данной диссертации, также является единым подходом к этим проблемам. Однако, комплексный подход основан на доказательстве выполнения свойств функций, выражающих результаты финитных итераций, и, в отличие от подхода Моллера и др., содержит методы для решения проблемы инвариантов в случае определенных классов циклов.

Рассмотрим работы про промежуточные языки верификации программ. Наиболее распространенными промежуточными языками верификации являются Boogie и WhyML. К примеру, следующие системы используют Boogie: AutoProof для верификации Eiffel-программ, VCC для верификации C-программ, Spec# для верификации C#-программ и система верификации Dafny-программ. WhyML (предоставляемый платформой Why3) используется в следующих системах: Frama-C для анализа C-программ, AstraVer для верификации C-программ и SPARK 2014 для верификации Ada-программ. Также платформа Why3 (и ее предыдущие вер-

сии) используются системой Krakatoa для верификации Java-программ и утилитой WP для верификации C-программ в системе Frama-C. Но из-за проблем с автоматизацией верификации Boogie-программ и WhyML-программ для систем верификации Dafny-программ и Ada-программ вместо прямой работы с промежуточными языками верификации были разработаны специальные скриптовые языки для высокоуровневого описания доказательства. В системе C-lightVer промежуточным языком верификации является C-kernel, подмножество входного языка C-light. Преимуществом такого подхода является одинаковая операционная семантика входного и промежуточного языка. Это облегчает доказательство сохранения семантики при трансляции с входного языка в промежуточное представление.

Рассмотрим работы про задание формальных семантик языков программирования. Формальная семантика для современного стандарта C11 была разработана Кребберсом и Виедиджком (Krebbers, Wiedijk). Саммлер (Sammler) и др. разработали новую систему дедуктивной верификации C-программ, в которой семантика языка C задана в системе Coq. В проекте VERISOFT по верификации ядра операционной системы используется язык C0. Семантика языка C0 моделируется в системе доказательства теорем Isabelle/HOL. Отметим, что C0 полностью покрывается языком C-light. В проекте по созданию верифицированного компилятора CompCert для языка C в качестве входного и промежуточного языка используются следующие подмножества C: C_{light} и C_{minor}. Семантика конструкций функциональной парадигмы программирования в новейших стандартах C++ и Java описана в работах Кока и Тасирана (Cok, Tasiran), для Java такая семантика реализована в проекте OpenJML. В системе KeY для верификации Java-программ используется семантика, называемая JavaDL. Недостатком всех вышеперечисленных семантик являются правила вывода для циклов и итераций, которые требуют задания инвариантов.

Рассмотрим работы про расширение языков программирования специальными видами циклов и их семантику. Ранее Аттали (Attali) и др. разработали для циклических выражений Sisal натуральную семантику, но она больше подходит для разработки компиляторов, чем для дедуктивной верификации. Библиотека MapReduce расширяет императивные и объектно-ориентированные языки программирования конструкциями, подобными функциям *map* и *reduce* из функциональной парадигмы программирования. Операционная семантика для расширения языка C-like конструкциями OpenMP была предложена в работе Блома (Blom) и др. Новейший стандарт C++20 вводит диапазоны (ranges) и итерации над ними. Более того, диапазоны (ranges) сходны с триплетами языка Cloud Sisal. Такая особенность Sisal, как конструкции, подобные `break`, является преимуществом Sisal-циклов относительно рассмотренных видов ите-

раций. Актуальность автоматизации дедуктивной верификации в случае циклов с инструкциями `break` продемонстрирована примерами из набора задач по верификации, созданного Якобсом (Jacobs) и др.

Цели и задачи диссертационной работы:

Целью научной работы является разработка методов, обеспечивающих автоматизацию, расширяемость и проблемную ориентированность комплексного похода к формальной верификации программ без использования инвариантов циклов. Для достижения данной цели были поставлены следующие задачи:

1. Разработка метода, позволяющего генерировать условия корректности для программ с финитными итерациями без использования инвариантов циклов.

2. Разработка стратегий, позволяющих автоматизировать доказательство условий корректности программ с финитными итерациями. Доказательство корректности данных стратегий.

3. Разработка метода, позволяющего автоматизировать локализацию ошибок при дедуктивной верификации программ с финитными итерациями.

4. Разработка аксиоматической семантики языка Cloud Sisal, позволяющей проводить дедуктивную верификацию программ на данном языке без использования инвариантов циклических выражений. Разработка аксиоматической семантики расширения языка C конструкциями языка Sisal, позволяющей применять в системе CPPS методы комплексного подхода системы C-lightVer.

5. Реализация методов комплексного подхода в системе C-lightVer, позволяющая при дедуктивной верификации программ с финитными итерациями, заданных на языках C, Cloud Sisal и на расширении C конструкциями Cloud Sisal, автоматизировать доказательство условий корректности и автоматизировать локализацию ошибок. Проведение экспериментов по автоматизированной верификации программ на данных языках.

Соответствие диссертации паспорту специальности. Цели и задачи исследования соответствуют следующим пунктам паспорта специальности 05.13.11: модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования (пункт 1); языки программирования и системы программирования, семантика программ (пункт 2); программные системы символьных вычислений (пункт 5).

Методология и методы исследования. Методология исследования базируется на подходах информатики к формальной верификации программ. В работе используется метод дедуктивной верификации программ, метод слабейшего предусловия, символический метод верификации финитных итераций, метод метагенерации условий корректности для упрощения расширения генератора УК, метод смешанной аксиоматической семантики для упрощения УК, стратегии автоматизации доказательства УК, метод семантической разметки для локализации ошибок.

Положения, выносимые на защиту:

1. Разработанный метод генерации операции замены для циклов позволяет генерировать условия корректности для программ с финитными итерациями без использования инвариантов циклов.

2. Разработанные стратегии доказательства условий корректности позволяют автоматизировать проверку на истинность условий корректности программ с финитными итерациями.

3. Разработанный метод локализации ошибок позволяет автоматизировать сопоставление конструкций программы и подформул условий корректности, а также локализацию ошибок в программах с финитными итерациями.

4. Разработанная аксиоматическая семантика языка Cloud-Sisal-kernel позволяет проводить дедуктивную верификацию программ на этом языке без использования инвариантов циклических выражений. Разработанная аксиоматическая семантика расширения (C-Sisal-kernel) языка C конструкциями языка Sisal позволяет применять в системе CPPS методы комплексного подхода системы C-lightVer.

5. Реализация методов комплексного подхода в системе C-lightVer позволила провести эксперименты по автоматизированной дудуктивной верификации программ с финитными итерациями на языках C, Cloud-Sisal-kernel и C-Sisal-kernel.

Научная новизна. Научная новизна работы состоит в

1. Автоматизации доказательства УК программ с финитными итерациями без использования инвариантов циклов.

2. Автоматизации локализации ошибок при дедуктивной верификации программ с финитными итерациями без использования инвариантов циклов.

3. Применимости разработанного комплексного подхода к программам с финитными итерациями на языках императивного программирования (на примере языка C), на языках функционального программирования (на примере языка Cloud Sisal) и на языках, основанных на обеих парадигмах программирования (на примере языка C-Sisal-kernel).

Теоретическая и практическая значимость. Теоретическая значимость работы состоит в разработке методов для комплексного подхода [8], которые позволяют в случае финитных итераций решить проблемы инвариантов циклов, автоматизации доказательства УК и автоматизации локализации ошибок. Данный подход может быть применен к широкому классу языков императивного и функционального программирования, позволяющих задавать финитные итерации над структурами данных. Это продемонстрировано применением комплексного подхода для автоматизации верификации программ на императивном языке C, программ на функциональном языке Cloud Sisal и программ на языке C-Sisal-kernel, основанным на обеих парадигмах программирования.

Практическая значимость работы состоит в реализации прототипа системы верификации C-lightVer [8, 39]. Данная система применима для верификации программ на языках, предствляющих две парадигмы программирования: язык C, язык Cloud Sisal и язык C-Sisal-kernel.

Степень достоверности и апробация результатов. Достоверность и обоснованность результатов исследования обеспечивается формальными доказательствами, а также компьютерными экспериментами. Основные результаты работы докладывались на следующих научных конференциях и семинарах: международные научно-практические конференции PSI-2017 [5], PSI-2019 [7, 31], TOOLS-2019 [11], SIBIRCON-2019 [10], TMPA-2015 [23, 38], международные научно-исследовательские семинары PSSV-2017 [36], PSSV-2018 [30], PSSV-2019 [32], PSSV-2021 [1], всероссийские конференции молодых ученых по математическому моделированию в 2015–2021 гг. [17, 18, 20–22, 24, 26], всероссийский онлайн-семинар ruSTEP (записи доступны на видеоканале ИСИ СО РАН), научно-исследовательские семинары ИСИ СО РАН, коллоквиумы ИСИ СО РАН.

Результаты, полученные в ходе выполнения данной работы, применены на практике в следующих проектах: РФФИ № 11-01-00028-а, РФФИ № 15-01-05974, РФФИ № 17-01-00789 и РНФ № 18-11-00118.

Публикации. Материалы диссертации опубликованы в 38 [1–38] печатных работах, в том числе 14 [1–14] статей в изданиях перечня ВАК, из них 11 публикаций [4–14], входящих в международные базы цитирования Scopus и Web of Science. Было получено свидетельство о государственной регистрации программы для ЭВМ [39].

Положения, выносимые на защиту в данной диссертации, описаны

в следующих публикациях: метод генерации операции замены описан в публикациях [3, 6, 8, 12, 18, 32, 33], стратегии доказательства условий корректности описаны в публикациях [3, 6–8, 10–12, 21, 29–31], метод локализации ошибок описан в публикациях [2, 5, 8, 12, 14, 17, 20, 23, 24, 32, 33, 35], семантика языка Cloud-Sisal-kernel и расширение ей языка C-kernel описаны в публикациях [1, 13, 15, 22, 27], реализация методов комплексного подхода для языков Cloud-Sisal-kernel и C-Sisal-kernel описана в публикациях [1, 4, 10, 11, 13, 15, 22, 31, 39], реализация методов комплексного подхода для языка C-light описана в публикациях [2, 3, 5–12, 16–19, 21, 23–26, 28–34, 36, 38, 39].

Положения, выносимые на защиту, описанные в работах с несколькими соавторами [3, 4, 6–15, 27–37], получены лично автором.

Личный вклад автора. Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в опубликованные работы. Подготовка к публикации полученных результатов проводилась совместно с соавторами, причем вклад диссертанта был определяющим. Все представленные в диссертации результаты получены лично автором.

Структура и объем диссертации. Диссертация состоит из введения, 5 глав, заключения и библиографии. Общий объем диссертации 238 страниц, из них 212 страниц текста, включая 7 рисунков и 2 таблицы. Библиография включает 181 наименование на 18 страницах.

Содержание работы

Во Введении обоснована актуальность диссертационной работы, сформулирована цель и аргументирована научная новизна исследований, показана практическая значимость полученных результатов, представлены выносимые на защиту научные положения.

В первой главе описаны методы, составляющие основу дедуктивной верификации (логика Хоара, метод слабейшего предусловия), и методы, используемые в комплексном подходе для решения проблем, возникающих при дедуктивной верификации программ (метод метагенерации условий корректности, метод семантической разметки, символический метод верификации финитных итераций). Также рассмотрен задел по системе дедуктивной верификации C-lightVer, созданный до работы над данной диссертацией.

Символический метод верификации финитных итераций применяется к циклам специального вида (финитным итерациям). Рассмотрим следующий вид цикла: $for\ x\ in\ S\ do\ v := body(v, x)\ end$, где S – последовательность данных, x – переменная типа “элемент S”, v вектор переменных цикла, который не содержит x, и $body$ представляет тело цикла,

которое не изменяет x и которое завершается для каждого $x \in S$. Тело цикла может содержать только инструкции присваивания, инструкции if (возможно вложенные) и инструкции `break`. Такие циклы for называются финитными итерациями. Пусть v_0 – вектор значений переменных v до исполнения цикла. Чтобы выразить эффект финитной итерации, определим операцию замены $rep(n, v, S, body)$, где $rep(0, v, S, body) = v_0$, $rep(i, v, S, body) = body(rep(i-1, v, S, body), s_i)$ для каждого $i = 1, 2, \ldots, n$. Если оператор выхода из цикла сработал на итерации i $(1 \le i \le n)$, то финитная итерация продолжает свое исполнение, но вектор v не изменяется: $\forall j (i \le j \le n)\ rep(i, v, S, body) = rep(j, v, S, body)$.

Во второй главе описаны алгоритмы генерации функций, выражающих результаты различных классов финитных итераций [3, 6, 8]. Использование таких алгоритмов приводит к генерации УК, содержащих применения функции rep. В данной главе описаны также стратегии доказательства таких УК [3, 6–8].

Алгоритм генерации функций, выражающих результаты итераций над изменяемыми массивами, основан на трансляции конструкций тела цикла в конструкции языка системы ACL2. Рассмотрим конструкцию $(b* \ (\ldots (var\ expr) \ldots)\ result)$, где конструкция вида $(var\ expr)$ означает связывание переменной var со значением выражения $expr$. Выражение $expr$ может зависеть от связанных ранее переменных. Значения переменных вектора изменяемых переменных v соответствуют значениям полей структуры fr типа $frame$. Поэтому, для моделирования изменения значения переменной вектора v мы связываем объект fr с новым объектом, который отличается от старого значением соответствующего поля. Для моделирования выхода из цикла мы используем булевское поле $loop\text{-}break$ объекта fr. Это поле истинно только после срабатывания `break`. Для моделирования `break` мы используем связывание вида $((when\ t)\ fr)$. Так как в этом случае условием $when$ является t, т.е. "истина", то такое связывание прекращает исполнение текущего блока $b*$ и возвращает fr.

Определим генератор операции замены как рекурсивную функцию gen_rep [12]:

- gen_rep(**empty statement**) = $(fr\ fr)$
- gen_rep(**break**;) = $((when\ t)\ fr)$
- gen_rep(**c = b**;) = $(fr\ (change\text{-}frame\ fr\ :c\ b))$
- gen_rep(**a[i] = b**;) = $(fr\ (change\text{-}frame\ fr\ :a\ (update\text{-}nth\ i\ b\ fr.a)))$
- gen_rep(**if (c) b else d**) =
  $(fr\ (if\ c\ (b*\ (gen\_rep(b))\ fr)\ (b*\ (gen\_rep(d))\ fr)))$
  $((when\ fr.loop\text{-}break)\ fr)$
- $gen\_rep(\{\mathbf{a_1}\ \mathbf{a_2}\ \ldots\ \mathbf{a_{k-1}}\ \mathbf{a_k}\}) = (fr\ (b^*(gen\_rep(a_1)\ \ldots\ gen\_rep(a_k))\ fr))$
  $((when\ fr.loop\text{-}break)\ fr)$

- $gen\_rep(\mathbf{for}\,(\mathbf{j}=\mathbf{0};\mathbf{j}<\mathbf{m};\mathbf{j}++)\;\mathbf{u}:=\mathbf{body}(\mathbf{u},\mathbf{j})\;\mathbf{end}) = (fr\;(rep_k\;m\;fr_k))$

Алгоритмы генерации функций, выражающих результат финитных итераций, создают операции замены rep, возвращающие дополнительную информацию о данных итерациях. В качестве примера можно привести информацию о том, исполнилась ли инструкция `break`, или информацию о значении счетчика цикла при окончании исполнения итерации. Такая информация позволяет стратегиям доказательства УК генерировать леммы о структуре финитных итераций и свойствах операций замены rep. В качестве примера рассмотрим стратегию для программ, постусловием которых является разбор случаев [6]. Стратегия применяется для содержащих финитные итерации программ, постусловие которых имеет вид конъюнкции импликаций. Для каждой такой импликации полезно рассмотреть, эквивалентен ли описанный ее посылкой случай исполнению операции `break`. Стратегия генерирует леммы, где в посылку УК добавляется специальный конъюнкт. Такой конъюнкт генерируется для каждой импликации постусловия в двух видах: для значения $rep(\ldots).loop\text{-}break$ и для отрицания значения $rep(\ldots).loop\text{-}break$. Данный конъюнкт является проверкой эквивалентности посылки импликации постусловия и значения/отрицания значения $rep(\ldots).loop\text{-}break$.

Все предложенные стратегии доказательства, кроме стратегии усиления УК, основаны на генерации лемм о свойствах программ и финитных итераций, которые могут помочь доказать УК. Если истинность этих лемм удается доказать, то леммы добавляются в теорию предметной области и их можно использовать для доказательства целевой теоремы. Поэтому, для алгоритмов генерации лемм, на которых основаны такие стратегии, не требуется доказательство корректности. Стратегия усиления УК основана на генерации такой формулы, из истинности которой следует истинность УК. Поэтому, для такой стратегии была сформулирована и доказана теорема о корректности, которая утверждает, что сгенерированная формула является усилением УК [10, 11].

В третьей главе описан метод автоматизации локализации ошибок [2, 5, 8, 12], реализованный в системе C-lightVer. Данный метод включает в себя стратегии локализации ошибок [8, 12] и метод генерации текста о сопоставлении подформул условий корректности и фрагментов программы [2, 5].

Дополнительная информация, которую возвращают функции, выражающие результат финитных итераций, позволяет генерировать леммы о свойствах финитных итераций не только стратегиям доказательства УК, но и стратегиям локализации ошибок. Таким образом, комплексный подход предоставляет единый способ анализировать вопрос корректности и некорректности программ, используя информацию, сохраненную в определениях функций rep и их возвращаемых значениях.

В качестве примера стратегии локализации ошибок можно рассмотреть стратегию поиска циклов с неиспользуемыми присваиваниями элементам массива [8]. Пусть в цикле, реализующем финитную итерацию над массивом, содержатся присваивания элементам этого массива и значения элементов массива после исполнения цикла равны значениям элементов массива до исполнения цикла. Значит, эти присваивания могут быть неиспользуемыми операциями.

Стратегия проверяет каждый цикл над массивом с присваиваниями элементам массива. Стратегия основана на генерации и проверке истинности леммы $P \to (a = rep_i(a, args).a)$, где P — предусловие, a — массив, над которым исполняется финитная итерация, rep_i — операция замены для финитной итерации, $args$ — аргументы rep_i, вычисленные с помощью обратного прослеживания, $rep_i(a, args).a$ — массив a после исполнения цикла. Если такую лемму удалось доказать, то генерируется текст с соответствующим предупреждением.

В комплексном подходе используется не ограниченный, а произвольный набор концепций семантических меток [8]. Эта возможность реализована как задание пользователем новых концепций семантических меток и правил вывода с этими метками [5]. Для каждого типа метки пользователь системы верификации задает шаблон текста [12]. Такие текстовые шаблоны подаются на вход метагенератору [2].

Для поддержки произвольных типов меток в системе C-lightVer язык описания правил вывода был расширен специальной конструкцией *label* [5, 12], используемой для описания меток.

Разработка алгоритмов генерации операции замены позволила расширить систему C-lightVer правилом вывода для финитных итераций. Для упрощения расширения системы C-lightVer данным правилом вывода был применен метод метагенерации УК. Метод метагенерации УК позволяет не переписывать ГУК "вручную" для добавления новых концепций меток [2]. Это позволило удобным образом задать для рассматриваемого правила новый тип метки – rep_iter. Меткой с концепцией rep_iter снабжается подформула, образованная операцией замены.

Денни и Фишер предложили способ снабжать семантическими метками формулы, но они не предложили способа снабжать семантическими метками определения применяемых в формулах функций. В случае применения операции замены возникла необходимость исправить этот недостаток метода семантической разметки. Поэтому, было создано расширение метода семантической разметки для снабжения семантическими метками определения операции замены. Для реализации этого подхода был предложен способ генерации снабженного метками определения функции rep, способ извлечения меток из определения функции rep и способ генерации текста для списка извлеченных меток.

В четвертой главе описано применение комплексного подхода к программам на языке C [6–8, 12]. Рассмотрена созданная в результате работы, описанная в данной диссертации, модифицированная система C-lightVer [2, 5, 8, 39]. Описаны эксперименты по верификации программ с финитными итерациями на языке C-light с помощью модифицированной системы C-lightVer [3, 6–8]. Продемонстрировано использование комплексного подхода для избежания задания инвариантов циклов, автоматизации доказательства УК и автоматизации локализации ошибок [5, 6, 8, 12].

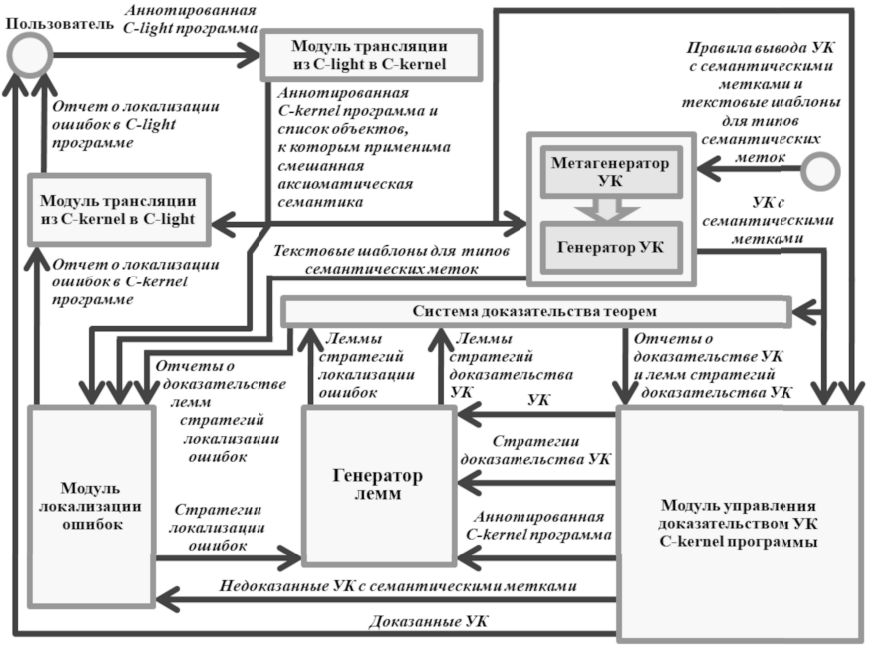

Рис. 1. Модифицированная версия системы C-lightVer

Модифицированная версия системы C-lightVer отличается от исходной реализацией методов комплексного подхода. Схема системы C-lightVer изображена на рис. 1. Реализация комплексного подхода привела к модификации всех модулей системы и созданию модулей управления доказательством УК, генерации лемм и локализации ошибок.

В пятой главе описано применение комплексного подхода не только к программам на императивном языке программирования C, но и к программам на языке Cloud Sisal [4]. Рассмотрены два реализованных в качестве модулей CPPS способа дедуктивной верификации данных программ: дедуктивная верификация промежуточного представления на язы-

ке C [10, 11] или дедуктивная верификация подмножества Cloud Sisal (язык Cloud-Sisal-kernel) [15]. Описана разработанная аксиоматическая семантика языка Cloud-Sisal-kernel [13], позволяющая проводить дедуктивную верификацию без использования инвариантов циклических выражений. Также рассмотрено расширение языка C циклами Cloud Sisal (язык C-Sisal-kernel) [12], позволяющее применить в системе CPPS методы комплексного подхода системы C-lightVer. Продемонстрировано применение комплексного подхода при проведении экспериментов по верификации программ на языках Cloud Sisal, Cloud-Sisal-kernel и C-Sisal-kernel.

Триплетом Cloud-Sisal-kernel является структура вида [lower bound .. upper bound .. step]. Она задает арифметическую прогрессию между заданными границами с шагом step. Рассмотрим цикл Cloud-Sisal-kernel, совершающий итерации над декартовым произведением триплетов:
$for\ var_1\ in\ tripl_1\ cross\ \ldots\ var_n\ in\ tripl_n\ do\ returns\ reduction\ expr\ end\ for$,
где var_i и $tripl_j$ – переменные и триплеты соответственно, $expr$ – редуцируемое выражение, $reduction$ – редукция. Значением цикла после определенной итерации является значение редукции, примененное к значению $expr$ на этой итерации и к значению цикла после предыдущей итерации. Семантика таких циклов основана на символическом методе верификации финитных итераций. Цикл транслируется в применение функции rep. Определение rep на языке системы ACL2 генерируется в следующем виде:

$(defun\ rep\ (range\_tuples)$
$\quad (b*\ (((when\ (endp\ range\_tuples))\ reduction\_init(reduction))$
$\qquad (tuple\ (car\ range\_tuples))$
$\qquad (var_1\ (car\ tuple))\ \ldots\ (var_n\ (car\ (cdr\ (cdr\ \ldots\ (cdr\ tuple)\ \ldots)))))$
$\qquad reduct2acl2(reduction,\ sisal2acl2(expr),\ (rep\_id\ (cdr\ range\_tuples)))))$

Функция $reduct2acl2$ реализует трансляцию редукций.

В Заключении приведены выводы из работы, сформулированы основные результаты и перспективы развития направления исследований.

Основные результаты и выводы

Основные научные и практические результаты, полученные в диссертационной работе и выносимые на защиту, состоят в следующем:

1. Разработан метод генерации операции замены для циклов, позволяющий генерировать условия корректности для программ с финитными итерациями без использования инвариантов циклов. Метод включает специализированные алгоритмы для случаев неизменяемых/изменяемых массивов и вида вложенности инструкций `if`.

2. Разработаны стратегии доказательства условий корректности для программ с финитными итерациями, позволяющие автоматизиро-

вать проверку на истинность. Были предложены следующие стратегии: выбора посылок; для программ, постусловием которых является разбор случаев выхода из цикла; для программ с финитными итерациями над массивами; интерактивного доказательства; усиления условий корректности; для программ, спецификации которых содержат функции со свойством конкатенации; для финитных итераций с инструкцией `break`; для программ с выходом из цикла; для финитных итераций над изменяемыми массивами; для программ с вложенными циклами. Доказана корректность стратегии усиления условий корректности.

3. Предложен метод локализации ошибок, позволяющий автоматизировать сопоставление конструкций программы и подформул условий корректности, а также локализацию ошибок в программах с финитными итерациями. Метод включает: язык представления семантических меток; семантические метки для функций, выражающих результаты финитных итераций; алгоритм генерации функций, выражающих результаты финитных итераций, с семантическими метками для итераций над изменяемыми массивами; алгоритм генерации объяснений недоказанных условий корректности, содержащих операцию замены; стратегию проверки ложности недоказанных условий корректности; стратегию поиска циклов с неиспользуемыми присваиваниями элементам массива; стратегию проверки исполнения инструкции `break` на первой итерации цикла.

4. Разработана аксиоматическая семантика языка Cloud-Sisal-kernel, включающая правила вывода для циклических выражений без инвариантов. Предложена аксиоматическая семантика расширения языка C конструкциями языка Sisal (C-Sisal-kernel). Разработан алгоритм генерации функций, выражающих результат циклических выражений языка Cloud-Sisal-kernel и языка C-Sisal-kernel.

5. Реализованы методы комплексного подхода в системе C-lightVer для автоматизированной дедуктивной верификации программ с финитными итерациями на языках C, Cloud-Sisal-kernel и C-Sisal-kernel. Были проведены эксперименты по автоматизированной верификации представлений Sisal программ на языке C, программ на языке Cloud-Sisal-kernel, программ на языке C-Sisal-kernel. Были проведены эксперименты по автоматизированной верификации C-программ с финитными итерациями над неизменяемыми/изменяемыми массивами в случае отсутствия/наличия инструкции `break` и по автоматизированной локализации ошибок в таких программах. Проведен успешный эксперимент по автоматической верификации программы сортировки вставками без инвариантов циклов.

Перспективы развития направления исследований. В будущем планируется разработать расширение комплексного подхода на программы с более общими видами итераций над различными структурами данных. Примерами таких структур данных являются файлы, списки, деревья. В качестве примера итераций над ними можно рассмотреть различные реализации топологической сортировки ациклического ориентированного графа. Расширение комплексного подхода может включать новые алгоритмы генерации операций замены, новые стратегии автоматизации доказательства УК и новые методы автоматизации локализации ошибок. Представленные в диссертации методы комплексного подхода могут упростить разработку и реализацию такого расширения.

Публикации по теме диссертации

В изданиях из перечня ВАК:

1. Кондратьев Д.А. На пути к автоматической дедуктивной верификации C-программ с Sisal-циклами в системе C-lightVer // Моделирование и анализ информационных систем. — 2021. — Т. 28, № 4. — С. 372–393.
2. Кондратьев Д. А. Расширение системы C-light символическим методом верификации финитных итераций // Вычислительные технологии. — 2017. — Т. 22, Специальный выпуск 1. — С. 44–59.
3. Maryasov I. V., Nepomniaschy V. A., Kondratyev D. A. Invariant Elimination of Definite Iterations over Arrays in C Programs Verification // Modeling and Analysis of Information Systems. — 2017. — Vol. 24, Issue 6. — P. 743–754.

В изданиях из баз цитирования Web of Science и Scopus:

4. Kasyanov V. N., Kasyanova E. V., Kondratyev D. A. Formal verification of Cloud Sisal programs // Proc. Applied Physics, Simulation and Computing, Rome, Italy, May 23–25, 2020. — Journal of Physics: Conference Series. — 2020. — Vol. 1603. — Article ID: 012020.
5. Kondratyev D. Implementing the Symbolic Method of Verification in the C-Light Project // Lect. Notes Comput. Sci. — Cham, 2018. — Vol. 10742. — P. 227–240.
6. Kondratyev D. A., Maryasov I. V., Nepomniaschy V. A. The Automation of C Program Verification by the Symbolic Method of Loop Invariant Elimination // Automatic Control and Computer Sciences. — 2019. — Vol. 53, Issue 7. — P. 653–662.
7. Kondratyev D., Maryasov I., Nepomniaschy V. Towards Automatic Deductive Verification of C Programs over Linear Arrays // Lect. Notes

Comput. Sci. — Cham, 2019. — Vol. 11964. — P. 232–242.
8. Kondratyev D. A., Nepomniaschy V. A. Automation of C-program deductive verification without using loop invariants // Programming and Computer Software. — 2022. — Vol. 47, Issue 5 (To appear).
9. Kondratyev D. A., Promsky A. V. Developing a Self Applicable Verification System. Theory and Practices // Automatic Control and Computer Sciences. — 2015. — Vol. 49, Issue 7. – P. 445–452.
10. Kondratyev D., Promsky A. Correctness of Proof Strategy for the Sisal Program Verification // Proc. 2019 Intern. Multi-Conf. Eng., Comput. and Inf. Sci., Novosibirsk, Russia, October 21-27, 2019. — IEEE, 2019. — P. 641–646.
11. Kondratyev D., Promsky A. Proof Strategy for Automated Sisal Program Verification // Lect. Notes Comput. Sci. — Cham, 2019. — Vol. 11771. — P. 113–120.
12. Kondratyev D. A., Promsky A. V. The Complex Approach of the C-lightVer System to the Automated Error Localization in C-Programs // Automatic Control and Computer Sciences. — 2020. — Vol. 54, Issue 7. — P. 728–739.
13. Kondratyev D. A., Promsky A. V. Towards verification of scientific and engineering programs. The CPPS project // Journal of Computational technologies. — 2020. — Vol. 25, Issue 5. — P. 91–106.
14. Maryasov I. V., Nepomniaschy V. A., Promsky A. V., Kondratyev D.A. Automatic C Program Verification Based on Mixed Axiomatic Semantics // Automatic Control and Computer Sciences. — 2014. — Vol. 48, Issue 7. — P. 407—414.

В других изданиях:

15. Касьянов В. Н., Гордеев Д. С., Золотухин Т. А., Касьянова Е. В., Кондратьев Д. А. Система облачного параллельного программирования CPPS: визуализация и верификация Cloud Sisal программ. Новосибирск: ИПЦ НГУ, 2020. — 256 с.
16. Кондратьев Д. А. Анализ аннотированных Си-программ и их трансляция в промежуточное представление // Материалы 51 Международной научной студенческой конф. «Студент и научно-технический прогресс»: Информационные технологии, Новосибирск, Россия, 12–18 апреля, 2013. —Новосибирск: Изд–во Новосиб. гос. ун–та, 2013. — С. 202.
17. Кондратьев Д. А. Верификация программ инженерной математики в системе C-light // Материалы XVIII Всероссийской конф. молодых ученых по математическому моделированию, Иркутск, Россия, 21 – 25 августа, 2017. — Новосибирск: Институт вычислительных технологий

СО РАН, 2017. — С. 78.

18. Кондратьев Д. А. Доказательство условий корректности Си-программ, осуществляющих финитные итерации над структурами данных // Тез. XIX Всероссийской конф. молодых ученых по математическому моделированию и информационным технологиям, Кемерово, Россия, 29 октября – 2 ноября, 2018. – Новосибирск: Институт вычислительных технологий СО РАН, 2018. — С. 65.
19. Кондратьев Д. А. Метагенератор условий корректности для языка Си // Материалы 52 Международной научной студенческой конф. «Студент и научно-технический прогресс»: Информационные технологии, Новосибирск, Россия, 11–18 апреля, 2014. — Новосибирск: Изд–во Новосиб. гос. ун–та, 2014. — С. 126.
20. Кондратьев Д. А. Модификации метода автоматизации локализации ошибок в С-программах, реализованного в системе C-lightVer // Тез. XXII Всероссийской конф. молодых ученых по математическому моделированию и информационным технологиям, Новосибирск, Россия, 25 – 29 октября, 2021. — Новосибирск: ФИЦ ИВТ, 2021. — С. 54.
21. Кондратьев Д. А. На пути к автоматической верификации С-программ с вложенными циклами в системе C-lightVer // Тез. XX Всероссийской конф. молодых ученых по математическому моделированию и информационным технологиям, Новосибирск, Россия, 28 октября – 1 ноября, 2019. — Новосибирск: Институт вычислительных технологий СО РАН, 2019. — С. 62.
22. Кондратьев Д. А. На пути к дедуктивной верификации С-программ, расширенных циклами языка Cloud Sisal // Тез. XXI Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям, Новосибирск, Россия, 7 – 11 декабря, 2020. —Новосибирск: ФИЦ ИВТ, 2020. — С. 35–36.
23. Кондратьев Д. А. Расширение метагенерации условий корректности концепцией семантической разметки // Материалы третьей международной научно-практической конф. «Инструменты и методы анализа программ», Санкт-Петербург, 12–14 ноября, 2015. — СПб.: Изд–во С.-Петерб. Политехн. ун–та Петра Великого, 2015. — С. 107–118.
24. Кондратьев Д. А. Расширение системы C-light символическим методом верификации финитных итераций // Материалы XVII Всероссийской конф. молодых ученых по математическому моделированию, Новосибирск, Россия, 30 октября – 3 ноября, 2016. — Новосибирск: Институт вычислительных технологий СО РАН, 2016. — С. 91–92.
25. Кондратьев Д. А. Самоприменимый метагенератор условий корректности для языка Си // Материалы 53 Международной научной студенческой конф. «Студент и научно-технический прогресс»: Информационные технологии, Новосибирск, Россия, 11–17 апреля, 2015. —

Новосибирск: Изд–во Новосиб. гос. ун–та, 2015. — С. 136.
26. Кондратьев Д. А. Семантические метки в проекте C-light // Материалы XVI Всероссийской конф. молодых ученых по математическому моделированию, Красноярск, Россия, 28–30 октября 2015. — Новосибирск: Институт вычислительных технологий СО РАН, 2015. — С. 76.
27. Кондратьев Д. А., Промский А. В. Аксиоматическая семантика подмножества языка Cloud Sisal для верификации программ инженерной математики // Тез. Междунар. конф. «Марчуковские научные чтения 2020», посв. 95-летию со дня рождения акад. Г. И. Марчука, Новосибирск, Россия, 19 – 23 октября 2020. — Новосибирск : ИПЦ НГУ, 2020. — С. 106.
28. Кондратьев Д. А., Промский А. В. Комплексный подход к локализации ошибок при верификации Си-программ // Системная информатика. — 2013, № 1. — С. 79–96.
29. Kondratyev D., Maryasov I., Nepomniaschy V. Towards Automatic Deductive Verification of C Programs Over Linear Arrays // Preliminary Proc. Intern. Conf. PSI–2019: A.P. Ershov Informatics Conf. "Perspectives of System Informatics", Novosibirsk, Russia, July 2–5, 2019. — Novosibirsk: Publishing center of Novosibirsk State University, 2019. — P. 162–171.
30. Kondratyev D. A., Maryasov I. V., Nepomniaschy V. A. Towards Loop Invariant Elimination for Definite Iterations over Changeable Data Structures in C Programs Verification. // Proc. 9th Workshop "Program Semantics, Specification and Verification: Theory and Applications", Yaroslavl, Russia, June 21–22, 2018. — Yaroslavl: Yaroslavl Demidov State University, 2018. — P. 51–57.
31. Kondratyev D., Promsky A. Automated Sisal program verification with ACL2 // Preliminary Proc. Intern. Conf. PSI–2019: A.P. Ershov Informatics Conf. "Perspectives of System Informatics", Novosibirsk, Russia, July 2–5, 2019. — Novosibirsk: IPC NSU, 2019. — P. 172–185.
32. Kondratyev D. A., Promsky A. V. Towards automated error localization in C programs with loops // Abstracts of the 10th Workshop "Program Semantics, Specification and Verification: Theory and Applications". — Novosibirsk, Russia, July 1–2, 2019. — Novosibirsk: IPC NSU, 2019. — P. 24.
33. Kondratyev D. A., Promsky A. V. Towards automated error localization in C programs with loops // System Informatics. — 2019. —Issue 14. — P. 31–44.
34. Kondratyev D. A., Promsky A. V. Towards the 'verified verifier'. Theory and practice // Proc. Fifth Workshop "Program Semantics, Specification and Verification: Theory and Applications", Moscow, Russia, June 6,

2014. — P. 68–78.
35. Maryasov I. V., Nepomniaschy V. A., Promsky A. V., Kondratyev D. A. Automatic C Program Verification Based on Mixed Axiomatic Semantics // Proc. Fourth Workshop "Program Semantics, Specification and Verification: Theory and Applications", Yekaterinburg, Russia, June 24, 2013. — Yaroslavl: Yaroslavl Demidov State University, 2013. — P. 50–59.
36. Maryasov I. V., Nepomniaschy V. A., Kondratyev D. A. Verification of Definite Iteration over Arrays with a Loop Exit in C Programs // Abstracts of the 8th Workshop "Program Semantics, Specification and Verification: Theory and Applications". — Moscow, Russia, June 26, 2017. — Moscow: MAKS Press, 2017. — P. 10.
37. Maryasov I. V., Nepomniaschy V. A., Kondratyev D. A. Verification of Definite Iteration over Arrays with a Loop Exit in C Programs. // System Informatics. — 2017. — Issue 10. — P. 57–66.
38. Promsky A. V., Kondratyev D. A. Implementing the MetaVCG approach in the C-light system // Proc. 3rd Internat. workshop-conf. Tools & Methods of Program Analysis, St. Petersburg, Russia, November 12–14, 2015. — St. Petersburg: Peter the Great St. Petersburg Polytechnic University, 2015. — P. 101–106.

Свидетельства о государственной регистрации программ для ЭВМ:

39. Кондратьев Д. А. Программа-транслятор аннотированных С-программ в семантически аксиоматизируемое представление для задач верификации. Свидетельство о государственной регистрации программы для ЭВМ № 2021614956, 01.04.2021.

Подписано в печать 13.07.2022 г. Печать офсетная.
Бумага офсетная. Формат 60x84 1/16. Усл. печ. 1,4 л.
Тираж 100 экз. Заказ № 764

Отпечатано в типографии «АЛЕКСПРЕСС»
ИП Малыгин Алексей Михайлович
630090, Новосибирск, пр-т Академика Лаврентьева, 6/1, оф.104
Тел. (383) 217-43-46