DOI: 10.15514/ISPRAS–2024–36(2)–11

Соревнования по формальной верификации VeHa-2023: опыт проведения

[1] *С.М. Старолетов,* ORCID: 0000-0001-5183-9736 <*serg_soft@mail.ru*>
[2,3] *Д.А. Кондратьев*, ORCID: 0000-0002-9387-6735 <*apple-66@mail.ru*>
[2] *Н.О. Гаранина*, ORCID: 0000-0001-9734-3808 <*garanina@iis.nsk.su*>
[4] *И.В. Шошмина,* ORCID: 0000-0001-5120-0385 <*shoshmina_iv@spbstu.ru*>
[1] *АлтГТУ им. И.И. Ползунова*,
Россия, 656038, Алтайский край, г. Барнаул, пр. Ленина, 46.
[2] *Институт систем информатики им. А.П. Ершова СО РАН*,
Россия, 630090, г. Новосибирск, пр. Академика Лаврентьева, 6.
[3] *Новосибирский государственный университет*,
Россия, 630090, г. Новосибирск, ул. Пирогова, 1.
[4] *Санкт-Петербургский политехнический университет Петра Великого*,
Россия, 195251, г. Санкт-Петербург, ул. Политехническая, 29.

Аннотация. Для создания современного конкурентоспособного и доверенного программного обеспечения необходимо использовать знания формальных методов. В настоящее время огромное количество студентов обучается специальностям, связанным с программированием. Однако при обучении в вузе сложно получить навык практического применения теоретических знаний. Короткие соревнования с нестандартными близкими к промышленным задачам могут пробудить интерес студентов к области формальных методов. В нашей статье описан первый опыт организации соревнования по формальной верификации программ среди студентов российских вузов. Соревнования проводились в связке с семинаром по семантике, спецификации и верификации программ (PSSV) в Иннополисе в ноябре 2023 года. Формат соревнования был близок к формату так называемых хакатонов. Участникам было предложено решить задачи по верификации с использованием заранее определенных инструментов проверки моделей и дедуктивной верификации. Мы рассмотрим вопросы организации такого мероприятия, предложенные задачи, результаты решений и обратную связь от участников.

Ключевые слова: формальные методы; соревнования по формальной верификации; проверка моделей; дедуктивная верификация; хакатон.

Для цитирования: Старолетов С.М., Кондратьев Д.А., Гаранина Н.О., Шошмина И.В. Соревнования по формальной верификации VeHa-2023: опыт проведения. Труды ИСП РАН, том 36, вып. 2, 2024 г., стр. 141-168. DOI: 10.15514/ISPRAS–2024–36(2)–11.

VeHa-2023 Formal Verification Contest: The Experience

[1] *S.M. Staroletov,* ORCID: 0000-0001-5183-9736 <*serg_soft@mail.ru*>
[2,3] *D.A. Kondratyev,* ORCID: 0000-0002-9387-6735 <*apple-66@mail.ru*>
[2] *N.O. Garanina,* ORCID: 0000-0001-9734-3808 <*garanina@iis.nsk.su*>
[4] *I.V. Shoshmina,* ORCID: 0000-0001-5120-0385 <*shoshmina_iv@spbstu.ru*>

[1] *Polzunov Altai State Technical University,*
46, Lenin Ave., Barnaul, 656038, Russia.
[2] *Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences,*
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia.
[3] *Novosibirsk State University,*
1, Pirogova street, Novosibirsk 630090, Russia.
[4] *Peter the Great St. Petersburg Polytechnic University,*
29, Politekhnicheskaya street, St. Petersburg, 195251, Russia.

Abstract. To create modern competitive and trusted software, it is necessary to use knowledge of formal methods. Currently, a huge number of students are studying specialties related to programming. However, when studying at a university, it is difficult to gain the skill of practical application of theoretical knowledge. Short competitions with non-standard, industrial-related problems can arouse students' interest in the field of formal methods. The article describes the first experience of organizing a competition in formal verification of programs among students of Russian universities. The competition was held in conjunction with a seminar on program semantics, specification and verification (PSSV) in Innopolis in November 2023. The format of the competition was close to the format of so-called hackathons. Participants were asked to solve verification problems using predefined model checking and deductive verification tools. We consider the issues of organizing such an event, proposed tasks, results of decisions and feedback from participants.

Keywords: formal methods; formal verification competitions; model checking; deductive verification; hackathon.

For citation: Staroletov S.M., Kondratyev D.A., Garanina N.O., Shoshmina I.V. VeHa-2023 Formal Verification Contest: The Experience. Trudy ISP RAN/Proc. ISP RAS, vol 36, issue. 2, 2024., pp. 141-168 (in Russian). DOI: 10.15514/ISPRAS-2024-36(2)-11.

1. Введение и мотивация проведения соревнования

Многие обучающиеся заинтересованы в собственном развитии и готовы изучать перспективные технологии. Однако большинство из будущих программистов при саморазвитии следуют текущим трендам, например, таким как нейронные сети или блокчейны. В подобных направлениях в интернете можно найти большое количество легко доступной информации, включая вводные лекции (тьюториалы) и описания разработки проектов с нуля. В результате студенты, пытаясь развиваться самостоятельно, увеличивают и без того огромный вклад в наиболее популярные направления.

При этом фундаментальные дисциплины не очень популярны, хотя влияние этих дисциплин на развитие области программирования велико и постоянно.

Непопулярность фундаментальных дисциплин легко объясняется: у них высокий порог входа для получения результатов, а также существуют сложности с выбором первых проектов, которые молодой специалист преодолеть не в состоянии.

Можно констатировать, что проектная деятельность в этой области в вузах практически отсутствует и, как правило, работа заканчивается либо на уровне индивидуальных проектов, либо она ограничивается выполнением упражнений в рамках учебных курсов.

С другой стороны, можно отметить рост интереса у студентов как к математике в целом, так и изучению вопросов, связанных с семантикой языков программирования и надежности программного обеспечения.

Задумываясь о проведении очередного семинара по спецификации и верификации программ PSSV [1], в оргкомитете возникло предложение компенсировать недостаточное практическое использование молодыми специалистами фундаментальных методов и провести соревнования по формальным методам верификации программ. Наш оптимизм основывался на личном опыте организаторов при решении практических задач, а также на опыте проведения подобных соревнований в смежных областях. Было предложено использовать для соревнования название VeHa (Verification Hackathon). Организаторами VeHa-2023 стали: Наталья Олеговна Гаранина (ИСИ СО РАН, НГУ), Дмитрий Александрович Кондратьев (ИСИ СО РАН, НГУ), Сергей Михайлович Старолетов (АлтГТУ, ИАиЭ СО РАН), Ирина Владимировна Шошмина (СПбПУ).

В последнее время в сфере программной инженерии среди разработчиков стали популярны так называемые *хакатоны*. Это слово комбинирует “to hack” и “марафон”, причём “to hack” здесь используется в значении “применять нестандартные методы решения сложных задач”, а не в значении “взламывать”. Традиционно, для концентрации на проектной деятельности, для быстрого решения поставленных задач хакатона собираются команды участников, организаторы проводят открытые лекции по используемым технологиям, а далее команды активно работают над проектами. При этом подразумевается, что проекты могут быть реализованы за короткое время и, возможно, продолжены в дальнейшем. Организаторы также фиксируют область соревнования, а если это соревнование решает индустриальную задачу, то задается стек технологий и предлагается вариант технического задания. В конце проводится финальная демонстрация решений и выборы победителей. Хакатоны отличает высокая мотивация и концентрация участников на проектах.

Опыт участия и проведения подобных мероприятий организаторами VeHa-2023 показывает, что знания и контакты, полученные во время такого рода мероприятий, способствуют профессиональному росту и формированию профессиональных социальных связей.

Несмотря на то, что только небольшое количество проектов хакатонов продолжается в дальнейшем, сформировавшиеся связи могут привести к долгосрочным глубоким исследованиям, постановка задачи которых началась в кулуарах соревнований [2].

Сейчас хакатоны проводятся как в рамках городских ИТ-сообществ (например, городской Барнаульский хакатон [3]), так и по решению задач для компаний (например, в нефтегазовом секторе [4]), а также для государственных компаний [5].

В современных университетах для развития у студентов навыков взаимодействия и работы над быстрыми проектами также организовываются хакатоны, например, [6].

Учитывая упомянутые тенденции, организаторы VeHa-2023 решили провести в рамках семинара своего рода хакатон по верификации программ.

Однако, чтобы расширить географию участников, было принято решение изменить пространственный формат соревнования на гибридный – онлайн (удаленно) и офлайн (очно) в Иннополисе. Такое решение не совсем соответствует концепции хакатона, поэтому мероприятие назвали “соревнованием”.

В качестве формальных методов верификации программ рассматривались метод проверки моделей и дедуктивная верификация, а также задачи верификации с использованием лямбда-исчисления и теории типов. Следует отметить, что последние два направления не вошли в описываемое соревнование и оставлены на будущее.

Методы формальной верификации программ осуществляют проверку корректности программы относительно ее формальных спецификаций. В зависимости от формального языка спецификаций отличаются возможности описываемых требований к программам, а также сами методы проверки.

В методах дедуктивной верификации программ спецификации задают ограничения на данные программы в её ключевых точках [7-8]: предусловия описывают свойства входных данных программы; постусловия – свойства выходных данных программы; инварианты

циклов [9] фиксируют свойства в ключевых точках. Эти спецификации, выраженные формулами, называются условиями корректности программы [10]. Таким образом, дедуктивная верификация сводит задачу проверки корректности программы относительно ее спецификаций к задаче проверки истинности формул.

Метод проверки модели позволяет автоматически проверить, выполняется ли заданная логическая формула на данной структуре [11-12]. Структура строится по разрабатываемой программной системе как система переходов с конечным числом состояний, модальная (темпоральная) логическая формула описывает требования к поведению системы. Чтобы проверить, верна ли спецификация на системе переходов, применяется эффективная процедура поиска, которая является полностью автоматической.

Существует большое количество инструментов верификации, реализующих дедуктивную верификацию и метод проверки модели. Для проведения соревнования мы остановили свой выбор на инструментах, которые было бы легко и быстро освоить.

Применение дедуктивной верификации на практике осложняется рядом проблем, требующих привлечения пользователя системы верификации [13]. Такими проблемами, в частности, являются задание инвариантов циклов [14-15] и доказательство условий корректности [16]. Поэтому мы предложили участникам соревнования программную систему дедуктивной верификации, которая может упростить и решение проблемы инвариантов циклов, и решение проблемы доказательства условий корректности программ для выбранных задач верификации, а именно – систему C-lightVer для программ на языке C-light [17-21], представительном подмножестве языка C. В отличие от других известных систем дедуктивной верификации [22-31], система C-lightVer даёт возможность автоматизировать задание инвариантов циклов для программ, циклы которых являются финитными итерациями [32]. Класс финитных итераций над последовательностями данных покрывает такие распространенные виды циклов, как циклы над массивами, циклы над списками, циклы над деревьями и т.д. [33]. Также система C-lightVer позволяет во многих случаях автоматизировать доказательство условий корректности с помощью применения системы доказательств ACL2 [34] и специальных стратегий доказательства [19, 32]. Для задания спецификаций и теории предметной области в системе C-lightVer используется язык Applicative Common Lisp [34], входной язык системы доказательства ACL2.

При использовании инструментов метода проверки модели [35-39] существуют несколько препятствий. Во-первых, существуют сложности с освоением языка моделирования, описывающего структуру переходов в системах. Во-вторых, пользователь должен знать язык темпоральной логики, с помощью которого описываются свойства систем. В-третьих, если верификатор генерирует опровержение свойства (контрпример), необходимо суметь интерпретировать этот вывод.

Нами было выбрано средство верификации SPIN с входным языком Promela [40]. Язык Promela является C-подобным языком с включением конструкций, облегчающих описание взаимодействия между независимыми процессами моделируемой системы. При этом формальную модель по введенному пользователем коду SPIN строит автоматически.

Свойства систем в SPIN задаются с помощью формул линейной темпоральной логики (Linear Temporal Logic – LTL). Эта логика является расширением логики высказываний с добавлением нескольких темпоральных модальностей. Кроме того, свойства, заданные с помощью формул линейной темпоральной логики, легко интерпретируются человеком, так как интуитивно ложатся на представление о линейном течении времени.

Контрпримеры в SPIN выдаются как последовательности шагов с указанием конкретных операторов программы, что позволяет легко анализировать ошибочную трассу.

Для SPIN и Promela существует плагин для популярной среди разработчиков среды Visual Studio Code (около 15000 инсталляций) [41], а подсветка синтаксиса также поддерживается

на GitHub и в Latex, что показывает, что язык Promela является зрелым и используемым достаточно большим количеством разработчиков и исследователей.

Кроме приведенных выше аргументов, для выбранных инструментов верификации авторами уже был разработаны презентации и методические материалы на русском языке.

Для подбора задач мы проводили обсуждение идей в доступных нам сообществах с предложениями задач, которые можно формализовать за два дня. Эти задачи могли поступать как от научного сообщества, так и от компаний, работающих с критическими и доверенными системами.

К сожалению, в этом году наши индустриальные партнеры не смогли предложить интересные задачи для соревнования и выделить менторов для работы с участниками, хотя впоследствии одна из компаний (“БАРС Групп”) обеспечила призы участникам. Таким образом, все задачи соревнования были предложены организаторами – авторами статьи.

Для оценки разных методов верификации мы сформировали подкомиссии из членов жюри.

Критерии для оценки количества баллов, которое получало решение по дедуктивной верификации, включали корректность постусловия и задания теории предметной области, успешность верификации полученной программы в системе.

При оценке решений по методу проверки модели использовались критерии, включающие качество моделирования и формулировки требований к модели, успешность верификации программы в системе.

В зависимости от выполнения данных критериев решение по каждому методу верификации получало оценку от 0 до 10 баллов.

Дальнейшее содержание статьи представляет собой обзор практики соревнований по формальным методам (раздел 2), структуры соревнования и статистические данные участников (раздел 3), подробные описания предложенных задач (раздел 4), способы подачи обучающего материала соревнований (раздел 5), отчет о проведении соревнования (раздел 6) и, наконец, анализ информации, полученной от участников, их решения и основные ошибки, а также результаты обратной связи (раздел 7).

2. Обзор соревнований по формальной верификации

Предшественниками первых соревнований по дедуктивной верификации программ можно назвать наборы задач, составленные в начале XXI века для проверки возможностей известных на тот момент подходов к автоматизации дедуктивной верификации [42-45]. Эти наборы оказали влияние на составление задач на первых соревнованиях в этой области [46].

В 2009–10 годах начался проект Verified Software Initiative по масштабному применению формальной верификации в индустриальном программировании [47], в рамках которого прошло первое соревнование по дедуктивной верификации программ [48].

Это соревнование было организовано совместно с конференцией по формальной верификации VSTTE в 2010 году [49] и проходило среди команд-участников конференции. Оно стало первым мероприятием серии VSComp. Второе и последнее соревнование этой серии, аффилированное с конференцией VSTTE 2011, прошло онлайн [50].

В 2011 году в рамках проекта COST IC0701 состоялось соревнование по дедуктивной верификации в связке с конференцией FoVeOOS 2011 [51], которое также проводилось среди команд. Оно положило начало самой успешной на данный момент серии соревнований по дедуктивной верификации VerifyThis. Опыт проведения этих соревнований описан в статье [52]. Авторы статьи [52] отмечают, что наиболее сложной проблемой является составление задач для соревнования. На первых соревнованиях по дедуктивной верификации организаторы сами формулировали задачи, но позднее была предложена идея перейти к сбору задач в области верификации от всех желающих, что описано в совместной статье организаторов серий VSComp и VerifyThis [53].

Серия соревнований VerifyThis проходит ежегодно с 2011 года (кроме 2020 года) в связке с различными конференциями по применению формальных методов: например, в 2012 году – с конференцией Formal Methods 2012 [46]. Начиная с 2015 года, серия соревнований VerifyThis аффилируется с серией конференций ETAPS [54-55]. Команды-участники соревнований VerifyThis должны присутствовать на месте проведения соревнований, онлайн участие не предусмотрено [56]. Начиная с 2012 года, организаторы соревнований заранее объявляют сбор задач от всех желающих помочь проведению соревнований. Задачи описываются на естественном языке с возможными включениями псевдокода. Команды-участники должны задать формальную спецификацию задачи, реализовать программу, соответствующую данным спецификациям, и доказать, что эта программа корректна относительно своих спецификаций. Жюри оценивает предоставленные решения на предмет корректности, полноты и элегантности.

Отметим, что время соревнований VerifyThis ограничено кратким временем проведения связанной конференции, поэтому для решения трудоёмких задач из области индустриального программирования с недавнего времени организаторы стали объявлять долговременные соревнования под эгидой VerifyThis [57-58]. Для таких задач отводится длительный промежуток времени между конференциями, связанными с VerifyThis. Подобные долговременные соревнования демонстрируют важную идею соревнований по дедуктивной верификации, которая состоит в том, что оценивается не скорость решения задач, а способность решить сложные задачи верификации программ.

В 2019 году соревнование VerifyThis стало частью мероприятия TOOLympics 2019 [59-60], в состав которого вошли несколько соревнований, связанных с применением формальных методов в программировании. Среди соревнований TOOLympics 2019 особо отметим соревнование RERS 2019 [61], которое входит в серию соревнований по методу проверки моделей RERS [62-65]. В соревновании RERS, в отличие от описываемого контеста VeHa-2023, участникам не было необходимости формализовать записанные на естественном языке требования, так как в задачах они заданы на языке логики линейного времени LTL. Поэтому RERS является скорее соревнованием программных инструментов проверки моделей, чем компетенций участников. Серия RERS включает специальное направление по решению задач для моделей параллельных программ, заданных на языке Promela [62], который также использовался в рамках нашего VeHa-2023. В 2019 году впервые в истории проведения RERS индустриальным партнером соревнования были предложены задачи верификации из индустриальных информационных технологий [61]. Итак, как в мероприятии TOOLympics 2019, так и в контесте VeHa-2023 проводились соревнования и по проверке моделей, и по дедуктивной верификации, однако участники VeHa-2023 были обязаны представить решения по обоим направлениям. Также отметим, что в рамках TOOLympics 2019 было проведено соревнование Termination Competition 2019 [66] (входящее в серию соревнований Termination Competition (termCOMP) [66-69]) по проверке выполнения свойства завершимости исполнения программ и систем переписывания термов.

В 2022 году прошло мероприятие SpecifyThis [70]. Названное по аналогии с VerifyThis, событие SpecifyThis не похоже на соревнование. Скорее это рабочий семинар, на котором участники делились, каким образом им приходилось решать проблему задания формальных спецификаций. Однако перспектива соревнований по заданию формальных спецификаций программ выглядит многообещающей в свете нетривиальности формализации свойств корректности индустриальных программ.

Из обзора родственных мероприятий можно сделать следующие выводы:

1. Соревнования по формальной верификации программ проводятся в связке с конференциями по применению формальных методов в программировании. Это позволяет привлекать участников конференции как в качестве организаторов соревнования, так и в качестве соревнующихся.

2. Современным трендом стало проведение таких контестов в рамках конференций по формальным методам в программировании, которые включают в себя несколько соревнований по разным направлениям применения формальных методов в программировании.
3. Все чаще к проведению соревнований по формальным методам привлекаются индустриальные партнеры – компании, работающие в области индустриального программирования. Они заинтересованы в применении формальной верификации в своей отрасли и могут предложить соответствующие задачи.
4. Новым направлением в развитии соревнований по формальной верификации стало проведение долговременных конкурсов между ежегодными проведениями аффилированных конференций. На таких соревнованиях во время проведения конференции объявляются крупные задачи, а решения проверяются через год. Такая схема позволяет участникам решать задачи формальной верификации программ из области индустриального программирования.
5. Традиционно на соревнованиях по формальной верификации за несколько месяцев до самого соревнования объявляют сбор задач. Каждый желающий может прислать организаторам свою задачу, которая может представлять интерес в области формальной верификации программ. Организаторы осуществляют отбор лучших задач.

3. Структура соревнования и участники

Согласно предварительному плану соревнования мы организовали конкурс на первоначальное обсуждение идей с предложениями задач. Как уже упоминалось во введении, промышленные партнеры не смогли предоставить задачи вовремя, поэтому авторы статьи предлагали задачи сами.

При составлении задач, кроме временного ограничения, нам было важно, чтобы задача затрагивала участников на эмоциональном уровне, чтобы задача касалась промышленных объектов.

В качестве задачи для метода проверки модели Н. О. Гараниной была предложена задача о миссии “Луны-25”, падение которой широко обсуждалось в прессе. Ошибка, которую предлагалось промоделировать участникам, касалась нарушения взаимодействия модулей. Окончательная формулировка этой задачи была составлена Н. О. Гараниной и И. В. Шошминой.

Предложение с задачей о миссии “Луны-25” было поддержано Д. А. Кондратьевым и в дедуктивной верификации. Для дедуктивной верификации базовая формулировка была близка к используемой в публичных изданиях, из которой выкристаллизовалась относительно простая задача.

Таким образом, с помощью задачи о миссии “Луны-25” мы пытались сформировать разные взгляды на ошибки, возникающие в программах. Это, с нашей точки зрения, соответствует задаче верификации сложных промышленных программ, где ошибка является следствием множества причин и требует комплексного исследования. Задачи, посвященные миссии “Луны-25”, оценивались нами как относительно простые. С. М. Старолетов предложил задачу верификации протокола телеграмм для европоездов и, совместно с Н. О. Гараниной – задачу верификации архитектурной последовательности работы (pipeline) GPGPU. Эти задачи были выделены авторами из их научно-исследовательских проектов. Сложность этих задач была выше.

Для привлечения большего числа участников мы выбрали гибридный формат проведения соревнования: допускалось и очное, и удаленное участие. От жюри очно на PSSV-2023 работал С. М. Старолетов, остальные члены жюри работали удаленно.

Для поддержки гибридного формата мы разработали сайт [71]. На сайте были размещены материалы по методу проверки моделей и дедуктивной верификации, выложены краткие описания задач соревнования и даны ссылки на Telegram-канал для обсуждений, Skype-митинг для тьюториалов и “живых” консультаций, а также GitHub-репозиторий для последующей загрузки решений.

В процессе создания сайта у нас была возможность быстрого совместного редактирования содержимого на основе подхода WYSIWYG, что позволило сосредоточиться в большей степени на задачах соревнования, чем на оформлении материалов. Для визуального наполнения сайта использовались популярные изображений из сети по теме багов и верификации, в частности изображение “первый баг” [72]. По отзывам участников, сайт соревнования получился удобный и красивый.

Основное взаимодействие с участниками осуществлялось через мессенджер Telegram. Выбор перечисленных технических средств обусловлен прежде всего быстротой разработки, поддержки браузеров как десктоп-систем, так и мобильных устройств.

Нами было установлено следующее расписание соревнований: 2 ноября 2023 – проведение тьюториалов по теоретическим и практическим аспектам формальной верификации онлайн в Skype и на месте проведения соревнований, утром 3 ноября – публикация полных текстов заданий на сайте, утром 5 ноября – завершение загрузка решений. Во время соревнований проводились онлайн- и очные консультации. К установленному сроку участники должны были загрузить решения с помощью pull-request [73] в заданный общий репозиторий на GitHub [74]. Этот репозиторий должен был быть предварительно клонирован участниками. Участники должны были создать в нём свою ветку, совершая по ходу соревнования загрузки (коммиты) в свой локальный репозиторий.

Кроме кода, участники могли снабдить решение текстовыми комментариями и описанием в свободном стиле, например, в виде диаграмм последовательности исполнения. Таким образом, для организации соревнования мы применяли известные программные решения в сфере индустриальной разработки.

По своему опыту наблюдения и участия в такого рода мероприятиях, авторы могут заключить, что участники приходят на соревнования с разными целями: кто-то хорошо программирует и просто ждет интересных задач, чтобы их решить и победить, кому-то нужно присоединиться к команде и чему-то научиться, сделав небольшую часть работы (это могут быть не только разработчики, но и дизайнеры или начинающие менеджеры проектов), а кому-то нравится находиться в ИТ-сообществе единомышленников. На таких мероприятиях можно встретить, в основном, студентов старших курсов, однако там бывают и недавние выпускники, представляющие компании, как в качестве участников, так и в качестве менторов или членов жюри оценки проектов. Рутинная работа разработчиком в компании после динамичной и разнообразной жизни в университете со сдачей множества работ и постоянными дедлайнами рождает тягу к вызовам, которые могут обеспечить соревнования.

На сайте мероприятия мы разместили опросник для предварительной регистрации с использованием готовых средств создания сайтов, форм и таблиц Google Site и Google Docs.

Это позволило нам проанализировать состав и интересы участников еще до начала соревнования.

По результатам предварительного опроса на сайте соревнований (опросник в виде Google-формы заполнили 35 человек), участники соревнования имели следующие аффилиации:

- Университет Иннополис;
- Университет ИТМО;
- Новосибирский государственный университет (НГУ);
- Санкт-Петербургский государственный университет (СПбГУ);
- Санкт-Петербургский политехнический университет Петра Великого (СПбПУ);

- Высшая Школа Экономики (ВШЭ);
- Астра Линукс (Группа Астра), Москва.

Следует отметить, что все без исключения участники были из российских вузов или организаций, при этом один из участников представлял Египет, будучи магистрантом Университета Иннополис.

Распределение участников по предпочтениям физического присутствия показано на рис. 1. Большинство участников были из Сибирских и Санкт-Петербургских вузов, где преподают организаторы соревнования, что объясняет получившееся распределение. Участники на месте были из Университета Иннополис, где проводился связанный семинар PSSV. Кроме того, один участник приехал для очного участия на место проведения соревнований из Москвы.

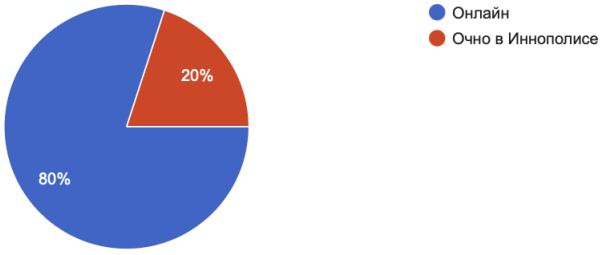

Рис. 1. Распределение участников по физическому присутствию
Fig. 1. Distribution of participants by physical presence

На рис. 2 показано распределение предпочтения участниками методов формальной верификации. Такие результаты отчасти можно снова объяснить тем, что большинство участников пришли на эти соревнования по приглашению от организаторов, которые преподают спецкурсы по проверке моделей, а также, возможно, из-за более интересных предварительных формулировок задач.

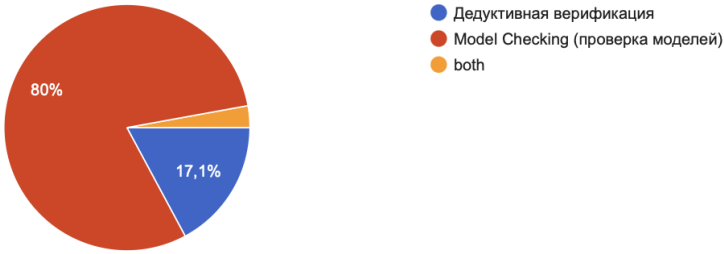

Рис. 2. Распределение участников по предпочтительному методу верификации
Fig. 2. Distribution of participants by a preferred verification method

Распределение по задачам, которые интересны участникам до начала контеста и объявления полных текстов задач, представлены на рис. 3.

В результате анализа этой статистики организаторы приняли решение о необходимости предоставления участниками решений как задачи по проверке моделей, так и задачи по дедуктивной верификации (а не одну из них), чтобы расширить компетенции участников. Соревнования были командными, и в состав команды могло входить от 1 до 3 человек.

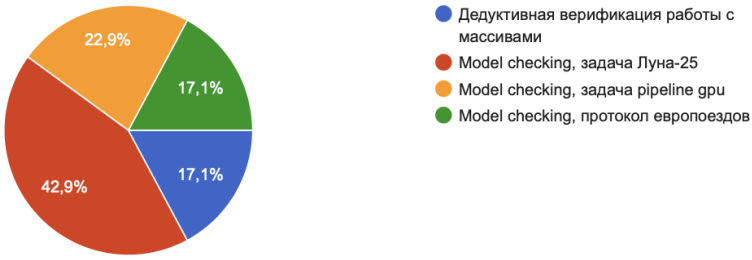

Рис. 3. Распределение участников по предпочтительному выбору задачи
Fig. 3. Distribution of participants by a preferred task

4. Обзор предложенных задач

Первоначальная концепция соревнования подразумевала, что задачи будут предложены как индустриальными партнерами исходя из их потребностей, так и организаторами на основе их научных интересов и публикаций, опыта разработки и преподавания курсов по формальной верификации. К сожалению, в этом году индустриальные партнеры не принимали участия в постановке задач, поэтому все задания были предложены организаторами-авторами статьи.

Поскольку формальная верификация тесно связана с надежными системами, а в отечественной прессе в августе и сентябре 2023 года циркулировало большое количество материалов по поводу недавнего крушения аппарата миссии “Луна-25” (например, [75]), то организаторы нашли интересным предложить в качестве задачи контеста моделирование и верификацию критической программной системы такого рода, несмотря на то, что все публичные материалы расследования причин аварии лишь очень приблизительно освещают детали программного устройства аппарата. Тема моделирования и верификации программных составляющих аппарата “Луна-25” стала основной на контесте. Полные условия всех задач можно найти на сайте контеста VeHa-2023 [71].

4.1 Дедуктивная верификация

Участникам соревнования была предложена задача по дедуктивной верификации программы, предназначенной для решения одной из потенциальных проблем миссии “Луна-25”.

В абстрактной постановке опубликованная версия причины крушения аппарата состоит в том, что в массив команд с одинаковым приоритетом попала команда, приоритет которой отличается от остальных. Программа, проверяющая, есть ли в массиве хотя бы один элемент, отличающийся от остальных, могла бы позволить избежать этой проблемы. Участники должны были верифицировать такую программу в системе C-lightVer.

Вначале рассмотрим общую постановку задачи. Предусловие программы было задано заранее, чтобы участники не проводили трудоёмкую работу с низкоуровневыми деталями. В итоге, задача состояла в том, чтобы участники соревнования:

1. задали постусловие программы в виде равенства результирующей переменной программы и рекурсивной функции, моделирующей цикл программы;
2. задали определение функции, используемой в постусловии и моделирующей цикл программы;
3. верифицировали программу с помощью системы C-lightVer.

Опишем детальную постановку задачи.

В качестве входных данных задачи участникам предоставлялся файл “element_equality.c” с одинаковым содержимым (см. листинг 1)

В этом файле предоставлено определение верифицируемой функции *element_equality* и предусловие данной функции, записанное на языке Applicative Common Lisp в комментарии до определения функции. Функция *element_equality* реализует проверку равенства элементов массива на языке C. Участникам нужно было задать постусловие на языке Applicative Common Lisp внутри пустого комментария после определения верифицируемой функции.

```
/* (and (integer-listp a) (natp n) (< 0 n)
 (<= n (length a))) */
int element_equality(int *a, int n) {
    int result = 1;
    for (int i = 1; i < n; i++)
    {
        if (a[i-1] != a[i])
        {
            result = 0;
            break;
        }
    }
    return result;
}
/*  */
```

Листинг 1. Входные данные (файл “element_equality.c”)
Listing 1. Input data (file “element_equality.c”)

Для решения задачи участникам нужно было выполнить следующую последовательность шагов:

1. Отредактировать файл “element_equality.c”, добавив в него постусловие в виде равенства переменной “result” и применения функции, проверяющей равенство элементов массива от нижней до верхней границы массива, заданных в качестве аргументов.
2. Создать файл с теорией предметной области, содержащий определение функции, примененной в постусловии и возвращающей число “1”, если все элементы массива от нижней до верхней границы равны, и число “0” в иных случаях.
3. Запустить систему C-lightVer, передав ей в качестве параметров отредактированный файл “element_equality.c” и полученный файл с теорией предметной области, и проверить, что информация, полученная в результате сеанса верификации, свидетельствует о корректности программы относительно своих спецификаций.

В качестве решения задачи командам-участникам следовало предоставить отредактированный файл “element_equality.c” с заданным постусловием и полученный файл с теорией предметной области, содержащий определение примененной в постусловии функции.

Критерии для оценки количества баллов, которое получало решение по дедуктивной верификации, были следующие:

1. правильно ли задано постусловие;
2. правильно ли задана теория предметной области с определением примененной в постусловии функции;
3. позволяет ли решение осуществить верификацию в системе.

Отметим, что относительно простая задача по дедуктивной верификации была уравнена в баллах с задачами по проверке на модели (model checking), так как для решения данной задачи участникам пришлось изучать такую сложную научную и практическую область, как дедуктивная верификация программ, а также использование системы C-lightVer. В будущем мы планируем определять количество баллов за задачи более гибкими способами.

4.2 Метод проверки моделей (Model Checking)

Для метода проверки модели были предложены три задачи. Для задачи о станции “Луна-25” было необходимо на языке Promela формализовать взаимодействие модулей станции и найти сценарий, приводящий к ошибке, описанной в СМИ, а затем предложить исправленную модель, в которой нет ошибочных сценариев взаимодействия модулей. Уровень этой задачи оценивался организаторами как простой.

В задаче о моделировании pipeline GPGPU было необходимо, описать pipeline процессов, происходящих в GPU при решении задачи на SIMT ядрах согласно известным публикациям.

В задаче про протокол телеграмм для европоездов требовалось реализовать алгоритм работы с бинарными сообщениями согласно официальному документу и проверить корректность кодирования/декодирования.

Последние две задачи имеют высокий уровень сложности и могут рассматриваться как примеры промышленного моделирования и верификации.

Для участия в соревновании достаточно было решить одну из трех задач. Несмотря на то, что уровень задач разный, для сложных задач было достаточно предложить простую, начальную версию решения. Таким подбором задач организаторы соревнования хотели стимулировать интерес участников к верификации промышленных систем.

При оценке решений по методу проверки модели использовались следующие критерии:

1. качество моделирования;
2. формулирование требований к модели на языке темпоральной логики;
3. наличие корректных вычислений и анализ контрпримера, выдаваемого верификатором;
4. качество алгоритма, исправляющего ошибку.

Две сложные задачи оценивались только по двум первым критериям.

4.2.1 Моделирование проблемы миссии “Луна-25”

В качестве основы для задачи были выбраны сообщения СМИ о крушении станции “Луна-25”, описывающие состав модулей аппарата и предполагаемые причины крушения [76]:

19 августа 2023 года при выдаче корректирующего импульса для перевода космического аппарата с круговой окололунной орбиты на эллиптическую предпосадочную орбиту двигательная установка “Луны-25” проработала 127 секунд вместо запланированных 84 секунд. В итоге станция перешла на нерасчетную незамкнутую орбиту и столкнулась с лунной поверхностью.

Для моделирования в задаче были выделены значимые модули станции “Луна-25”, согласно опубликованной в СМИ открытой информации:

1. бортовой комплекс управления (БКУ);
2. блок измерения угловых скоростей (БИУС-Л – Блок Измерения Угловых Скоростей-Луна) и датчики угловых скоростей;
3. двигатель;
4. другие модули.

“Бортовой комплекс управления” обеспечивает функционирование всего космического аппарата и разных его систем. В его задачи входит управление всеми компонентами.

“БИУС-Л” предназначен для определения ориентации космической станции и определения скорости полета при помощи оптоволоконных гироскопов и акселерометра. При моделировании рассматривались только датчики угловых скоростей.

“Двигатель” отвечает за перемещение аппарата. При посадке он должен обеспечить переход аппарата на эллиптическую орбиту.

Про “Другие модули” известно, что они могут принимать и передавать данные, и этого достаточно для их моделирования.

Дополнительной, хоть и неявной компонентой, требующей моделирования, являлась среда коммуникации, в которой имеется шина передачи команд и шина передачи данных.

В условии задачи было предложено смоделировать следующую версию ошибки, произошедшей на станции “Луна-25” (мы намеренно не рассматриваем версии, связанные с возможными ошибками при планировании процессов в операционных системах реального времени, а основываемся на одной из версий из СМИ). По неизвестным причинам БИУС-Л стал передавать слишком много “нулевых сигналов”. Этим он перегрузил входящий поток данных на БКУ. При такой проблеме БКУ подал БИУС-Л команду о перезагрузке. В массиве входящих команд БИУС-Л присутствовали команды об отслеживании угловых скоростей и о перезагрузке. БИУС-Л недетерминированно выбрал в массиве команду о перезагрузке. При перезагрузке БИУС-Л очистил свой массив входящих команд и таким образом пропустил команду об отслеживании угловых скоростей. Поскольку на БКУ не был запрограммирован анализ ответной реакции БИУС-Л на команду включения акселерометров, то БКУ не смог получить информацию о фактических значениях угловых скоростей. В результате при достижении необходимой орбиты БКУ не передал сигнал двигателю на своевременное отключение.

Для решения задачи от участников требовалось:

1. Спроектировать взаимодействие компонент, чтобы оно приводило к аварии в результате неправильного расположения в пуле данных БИУС-Л команды об отслеживании угловых скоростей аппарата. Доказать, что авария может происходить.
2. Изменить проект так, чтобы избежать аварии. Доказать, что авария не может происходить.

4.2.2 Моделирование архитектурной последовательности работы (pipeline) GPGPU

Эта задача предполагала расширение идей нашего подхода по моделированию на языке Promela поведения процессора видеокарты (GPU) при исполнении кода некоторой параллельной программы. Результирующая Promela-модель использовалась для поиска наименьшего времени исполнения этой параллельной программы, исходя из комбинаций параметров, от которых зависит время завершения исполнения [77].

В цитируемой статье мы использовали только сущности абстрактной архитектуры OpenCL [78], такие как хост, устройство, исполнительный модуль. Однако существуют работы, в которых с хорошей точностью промоделирован процесс работы реального GPU-процессора [79]. Эта модель принимает во внимание прежде всего систему команд (инструкции) и включает такие составляющие как кеш инструкций, буфер инструкций, планировщик исполнения инструкций, модуль сборки операндов и другие.

В процессорах их компоненты организуют pipeline – последовательность взаимодействий в соответствии с принципами архитектуры. Современные GPU реализуют SIMT архитектуру (single instruction, multiple thread), то есть одна и та же инструкция после декодирования исполняется на большом количестве потоков, способных выполнять простые операции, для чего необходимо наличие эффективного кэша и реализации алгоритмов работы с ветвлениями. Поскольку все эти операции представляют собой взаимодействия параллельно исполняющихся компонентов с синхронизацией, то они могут быть промоделированы на языках отвечающих подходам процессных алгебр, например алгебре CSP и языку Promela.

От участников соревнования ожидался результат, удовлетворяющий следующим пунктам:

- создана модель решения простой задачи, заданной в виде последовательности инструкций, сходных с приведенными примерами инструкций для некоторой OpenCL программы;
- взаимодействия компонентов должны соответствовать основным идеям из GPGPU Sim [79];
- все сущности должны быть описаны как параллельные процессы Promela, взаимодействующие через каналы (принятие и отправка сообщений);
- должно быть несколько экземпляров сущностей для симуляции мультипроцессоров GPU и планировщиков внутри них;
- решение должно быть сделано с предоставлением UML диаграмм деятельности по каждому компоненту.

Предлагалось верифицировать требование минимального времени исполнения решения задачи на GPU, согласно нашему подходу [77].

4.2.3 Моделирование протокола телеграмм для европоездов

Эта задача предполагала продолжение исследования по созданию выполнимой модели для побитовых операций с Eurobalise-телеграммами (передаваемыми сообщениями в современных железнодорожных системах) согласно их официальной спецификации с целью проверки кодирования и декодирования таких телеграмм [80]. В частности, ранее автором была реализована библиотека для работы с бинарными полиномами в поле GF(2) на Promela, и теперь предполагалось реализовать модель согласно предложенным в статье подходам. Планировалось, что в процессе соревнования будет проводиться обсуждение задачи с участниками, которое будет включать ознакомление с требованиями к телеграммам для бализы (передатчика) из официального стандарта и с методами работы с бинарными полиномами для вычисления контрольной суммы (CRC). В итоге ожидался какой-либо из следующих результатов:

- реализация на Promela побитовых операций работы с частями телеграмм в соответствии со стандартом с использованием библиотеки в виде макросов для полиномов;
- реализация кодирования и проверки получившихся диаграмм (возможно, сначала на языке C, с последующей формализацией на Promela с проведением абстрагирования);
- реализация декодирования;
- реализация верификации кодирования и декодирования путем случайном подмены битов в сообщении.

5. Обучение участников

Обучение участников проводилось в гибридном формате – дистанционно и на месте. В назначенное время по указанному на сайте контеста адресу были проведены лекции в Skype по дедуктивной верификации и языку Promela с записью, доступной в течение 30 дней. Кроме того, на сайте соревнования были выложены обучающие материалы. Дополнительно один из организаторов прочел лекции по практическим аспектам верификатора Spin в Университете Иннополис с трансляцией и записью в Skype. Также во время решения задач проводились онлайн и офлайн консультации.

5.1 Дедуктивная верификация

До начала соревнования были подготовлены и размещены на сайте следующие материалы [71]:

- мини-пособие для участников соревнования по дедуктивной верификации;
- презентация тьюториала для участников соревнования по дедуктивной верификации.

Кроме того, участникам соревнования было настоятельно рекомендовано установить систему C-lightVer до начала соревнования.

Пособие описывает дедуктивную верификацию, применение системы C-lightVer, и язык Applicative Common Lisp на примере решения демонстрационной задачи по верификации функции, вычисляющей сумму абсолютных значений элементов массива. Демонстрационная задача и ее решение предоставлялись участникам в качестве файлов “abs_sum.c” и “abs-sum.lisp” в дополнение к пособию.

Также отметим, что в тексте пособия были особенно подробно описаны те особенности программирования на языке Applicative Common Lisp и работы с системой C-lightVer, где при решении задачи соревнования можно было допустить ошибку.

Задача по верификации функции, вычисляющей сумму абсолютных значений элементов массива, схожа с задачей соревнования. В этих задачах цикл в программе совершает итерации над всеми элементами массива, а постусловие представляет собой равенство результата программы и применения функции к элементам массива от нижней до верхней границы массива. Кроме того, и там, и там теория предметной области содержит определение примененной в постусловии функции, моделирующей верифицируемую программу. Таким образом, обучение участников включало ознакомление с опытом организатора соревнования.

5.2 Метод проверки моделей

Материалы по теоретическим аспектам метода проверки моделей (структуры Крипке, синтаксис и семантика темпоральных логик, формализация требований) были основаны на курсе лекций летней школы Computer Science Summer in Russia 2019, а также собственных курсов авторов в НГУ. В частности, в опубликованной на сайте презентации рассматриваются известные ошибки в критическом программном обеспечении (в сфере космоса и медицины), основные методы проверки качества программного обеспечения, разница между дедуктивной верификацией и проверкой моделей, и далее приводятся примеры моделей и требований для разных классов программ. В дополнение к теоретическим материалам была выложена инструкция по запуску верификатора SPIN и среды iSpin в среде Windows (обычно это требует установки MinGV + gcc, что не всегда очевидно студентам). Тьюториал к соревнованию по методам проверки моделей включал обзор синтаксиса и семантики языка Promela верификатора Spin и основные способы работы с этим инструментом.

Также на сайт была выложена глава про метод проверки моделей из книги [81] авторов статьи, в которой кратко рассмотрены теоретические сведения о методе, работа с верификатором SPIN и элементы языка Promela.

Этот материал был использован для проведения лекции по практическим аспектам Model Checking, где участникам была показана сборка SPIN с его GitHub-репозитория [82] с помощью make в Unix-среде, написание кода на входном языке Promela в среде Visual Studio Code и запуск моделей некоторых коммуникационных протоколов в среде iSpin, проверка LTL свойств и способ задания параметров верификатора, связанных с используемой памятью, для проверки больших моделей.

5.3 Консультации

Поскольку контест проводился в основном в выходные и онлайн, а исходный GitHub-репозиторий для загрузки решений клонировали не все участники, то организатором сложно было отследить текущее состояние решения задач. Поэтому была организована очная консультация с объявлением в Telegram-канале соревнования, на которую пришли участники двух команд из Иннополиса. Участники консультации сообщили, что, по их мнению, задачи сформулированы нечетко. Они также отметили, что в задаче о станции “Луна-25” причины аварии в разных отчетах в СМИ не всегда соответствуют информации в описании задачи от организаторов. На вопрос о способе недетерминированного появления ошибки в модели системы, участникам было указано на возможности недетерминизма в операторах *if*, *do* и *select* [83]. Относительно задачи GPGPU участники высказали мнение, что задача слишком сложная, так как нужно долго изучать предварительные материалы, поэтому команда не успеет предоставить решение должного качества, так как приходится ещё заниматься задачей по дедуктивной верификации. Тем не менее, участники заверили, что в каком-то виде модели будут построены, так как с синтаксисом и семантикой языка Promela они в целом разобрались. Дополнительно консультант подчеркнул, что в нашем соревновании в решениях участников не ожидается заранее определенного протокола взаимодействия процессов системы, этот протокол они должны придумать и обосновать в своем решении.

Также позже состоялась онлайн-консультация в Skype, на которой желающие получили разъяснения по формулировке задачи о станции “Луна-25” и аспектам взаимодействия процессов через каналы в Promela.

6. Проведение соревнования

Соревнование по дедуктивной верификации оценивал Д. А. Кондратьев, а по методу проверки модели – Н. О. Гаранина, С. М. Старолетов, И. В. Шошмина.

6.1 Соревнование по дедуктивной верификации

Из пятнадцати команд контеста решения задачи по дедуктивной верификации прислали тринадцать команд.

Из них девять команд набрали максимальные десять баллов, три команды – девять баллов и одна команда – восемь баллов. Отдельно итоги соревнования по дедуктивной верификации в рамках VeHa-2023 не подводились, эти баллы были учтены в итоге контеста в целом.

Самым большим недостатком решений задачи было использование единицы вместо нуля в качестве значения нижней границы в применении функции в постусловии. Можно предположить, что к появлению такой ошибки привело использование единицы в качестве значения счетчика цикла в верифицируемой программе. Но итерации цикла в верифицируемой программе покрывают массив, начиная не с элемента с индексом *1*, а с элемента с индексом *0*, так как в качестве индексов в данном цикле используются выражения *i-1* и *i*.

Абсолютное большинство команд прислало решения, которые либо совпадают с точностью до имен идентификаторов с нашим эталонным решением, либо довольно близки к нему, поэтому здесь мы приводим тексты файлов этого решения “element_equality.c” и “element-equality.lisp”. Содержимое файла “element_equality.c” эталонного решения на листинге 2.

Эта часть решения состоит в задании постусловия в виде равенства применения функции *element-equality* и переменной *result* внутри комментария после определения верифицируемой функции. Аргументами применения *element-equality*, соответствующими нижней и верхней границе, являются выражения *0* и *(- n 1)*. Ошибка, о которой упоминалось ранее, состоит в использовании в качестве таких аргументов выражений *1* и *(- n 1)*.

```
/* (and (integer-listp a) (natp n) (< 0 n)
 (<= n (length a))) */
int element_equality(int *a, int n) {
  int result = 1;
  for (int i = 1; i < n; i++) {
    if (a[i-1] != a[i]) {
      result = 0;
      break;
    }
  }
  return result;
}
/* (= (element-equality 0 (- n 1) a) result)
*/
```

Листинг 2. Файл эталонного решения (“element_equality.c”)
Listing 2. Reference solution file (“element_equality.c”)

На листинге 3 приведено определение функции *element-equality* из постусловия в файле “element-equality.lisp”.

```
(defun element-equality(i j a)
 (if (or (not (natp i)) (not (natp j))) 0
   (if (>= i j) 1
     (if (not (equal (nth (- j 1) a)
      (nth j a))) 0
       (element-equality i (- j 1) a)))))
```

Листинг 3. Функция “element-equality”, примененная в постусловии (файл “element-equality.lisp”)
Listing 3. “element-equality” function applied in the postcondition (“element-equality.lisp”)

Функция *element-equality* возвращает число *1*, если все элементы последовательности от нижней до верхней границы равны, и число *0* в иных случаях. Отметим, что функция *element-equality* определена рекурсивно с уменьшением верхней границы на единицу при каждом рекурсивном вызове.

Рассмотрим решения задачи, которые организатор посчитал интересными. Все такие решения содержат особенности при решении следующей подзадачи: каким образом сделать возвращаемым значением функции из постусловия целое число *1* или *0* вместо того, чтобы с помощью конъюнкции равенства элементов массива и рекурсивного применения данной функции сделать возвращаемым значением булево значение. В эталонном решении эта подзадача решалась с помощью использования в определении функции из постусловия вместо конъюнкции условного выражения, показанного на листинге 4.

```
(if (>= i j) 1
  (if (not (equal (nth (- j 1) a)
   (nth j a))) 0
    (element-equality i (- j 1) a)))
```

Листинг 4. Условное выражение из функции “element-equality”, примененной в постусловии
Listing 4. Conditional expression in the “element-equality” function applied in the postcondition

В предварительном решении команды NastyaKrass (Группа Астра, Москва) было предложено использовать вместо этого выражения сравнение с помощью побитовых операций элементов массива и сравнение с помощью побитовых операций полученного результата с рекурсивным применением функции. Такая идея решения основана на том, что элементы массива представляют собой целые числа, сравнение которых можно реализовать с помощью побитовых операций вместо равенства. Но так как NastyaKrass решала задачу соревнования по дедуктивной верификации вручную (без использования программной системы C-lightVer), то получившееся решение содержит недочеты. За попытку решения задачи вручную (без использования C-lightVer) и за попытку решить задачу интересным способом команда NastyaKrass получила почетную грамоту “За волю к победе”.

Команда Cache-Invalidation (Университет Иннополис, Иннополис) предложила использовать иное выражение вместо вышеприведенного условного выражения (см. листинг 5).

```
(if (> i j) 1
(if (= i j) 1
  (min
   (if (= (nth j a) (nth (- j 1) a)) 1 0)
   (element-equality i (- j 1 ) a))))
```

Листинг 5. Альтернативный вариант условного выражения из функции, примененной в постусловии
Listing 5. Alternate version of conditional expression from the function applied in the postcondition

Это решение основано на выборе минимального элемента между результатом рекурсивного применения функции из постусловия и условным выражением со значением в виде числа *1* при равенстве элементов массивов и со значением в виде числа *0* в ином случае. Выбор минимального элемента в данном решении осуществляется с помощью стандартной функции *min* из языка Applicative Common Lisp. Получившееся решение можно развить следующим образом: сначала генерировать по массиву последовательность элементов, в которую будет попадать число *1*, если соседние элементы массива равны, и число *0* в ином случае, а потом искать минимум в полученной последовательности. Особенно отметим, что к функциям, используемым для задания спецификаций, не применяются требования к эффективности реализации, так как в процессе дедуктивной верификации исполнение этих функций не происходит. Поэтому, такое решение имеет полное право на существование, а команда Cache-Invalidation получила почетную грамоту “За оригинальное решение задачи “Луна-25” (дедуктивная верификация)”.

Элегантное решение предложил участник команды Kokorin (Группа Астра, Москва). С одной стороны, функция из постусловия этого решения зависит только от верхней границы. С другой стороны, рассмотрим нетривиальное определение функции *element-equality*, примененной в постусловии, показанное на листинге 6.

```
(defun element-equality-bool(j a)
 (if (not (natp j)) nil
 (if (= 0 j) t
   (and (= (nth (- j 1) a) (nth j a))
     (element-equality-bool (- j 1) a)))))

(defun element-equality(n a)
  (if (element-equality-bool (- n 1) a) 1 0))
```

Листинг 6. Функция “element-equality”, предложенная участником Kokorin (Москва)
Listing 6. “element-equality” function offered by participant Kokorin (Moscow)

Функция *element-equality* определена с помощью применения функции *element-equality-bool*. Таким образом, это единственное решение, где в теории предметной области было определено более одной функции. Использование двух функций позволило разбить решение задачи на две подзадачи: функция *element-equality-bool* возвращает истинное или ложное значение в зависимости от того, все ли элементы последовательности до верхней границы равны, а функция *element-equality* возвращает число *1* или *0* в зависимости от значения применения функции *element-equality*. Функция *element-equality* определена нерекурсивно, а функция *element-equality-bool* определена рекурсивно с помощью конъюнкции своего рекурсивного применения и равенства элементов массива. Также отметим, что кроме декомпозиции задачи, команде Kokorin удалось написать простое и понятное пояснение в дополнение к решению задачи. Так как в дополнение к такому элегантному решению этой команде удалось предоставить оригинальное решение задачи соревнования по проверки моделей, то Kokorin получил почетную грамоту “За оригинальное решение задач контеста”.

6.2 Соревнование по проверке моделей (Model Checking)

Все пятнадцать команд выполняли задание по методу проверки модели.

6.2.1 Ошибки и сложности, возникшие у участников соревнования

Рассмотрим основные трудности, с которыми столкнулись участники в части соревнования, связанного с методом проверки модели, на примере задачи о станции “Луна-25”.

Предложенная некорректная последовательность событий на станции “Луна-25” содержала ошибку синхронизации, что характерно для распределенных алгоритмов. Такой тип ошибок удобно моделируется с помощью языка Promela, поскольку Promela – язык межпроцессного взаимодействия.

При проверке моделирования оценивалось качество моделирования структур данных, качество моделирования механизмов синхронизации, соблюдение алгоритмов работы отдельных процессов/модулей, корректность использования механизмов абстракции – атомарности, чередования. Инструменты моделирования очень мощные, возникает соблазн упростить модель, то есть сформировать простую модель на высоком уровне абстракции. Такая модель может быть верифицируемой, но не будет в достаточной степени отражать деталей реализации.

Во многих моделях были введены лишние параметры, например, несколько значений показаний угловых скоростей. В данной постановке задачи алгоритм не предусматривал разной реакции на разные значения угловой скорости. Для самого тонкого моделирования достаточно было трех значений: 0 – скорость не проверяется, 1 – скорость проверяется, но аппарат не достиг окололунной орбиты, 2 – скорость проверяется, аппарат достиг окололунной орбиты. Более того, можно было обойтись и двумя значениями: 0 – скорость не проверяется, 1 – скорость проверяется, а переключение БКУ по достижению окололунной орбиты моделировать недетерминированно. Правда, такое моделирование потребовало бы более тонкого задания свойств. Ввод ненужных параметров или необоснованных диапазонов переменных приводят к экспоненциальному взрыву пространства состояний модели. Причина состоит в том, что состояние структуры Крипке, которая строится верификатором SPIN, содержит значения переменных и счетчики операторов, поэтому размеры структуры зависят от диапазона значений переменных.

Другим ненужным для верификации параметром были счетчики времени. В формулировке алгоритмов и свойств время не предусматривалось. Здесь рассматривался подход, сформулированный Л. Лэмпортом, что качественные распределенные алгоритмы должны работать правильно всегда, независимо от скорости процессов. Кроме того, линейная темпоральная логика (LTL), для которой реализован метод проверки модели в SPIN, предназначена для задания причинно-следственных зависимостей во временной последовательности событий. Количественные значения времени в этой логике не предусмотрены. Типичным вариантом моделирования событий, которые интуитивно связываются со временем, являются недетерминированные переходы, например, канал недетерминированно отправляет сообщения или теряет их.

Еще одной проблемой стало моделирование среды коммуникации. В условии задачи было сказано, что среда коммуникации проходит по циклу все процессы и в одном цикле либо получает, либо отправляет сообщения. Многие из участников ошибочно увеличили гранулярность модели с помощью операторов *atomic*, включив получение и отправку сообщений в один шаг. Кроме того, многие модели реализовали работу алгоритма, как последовательную работу процессов, то есть до того, как один процесс не выполнит свой шаг, связанный с обработкой сообщений, следующий процесс не приступает к своей работе. Таким образом, участники ошибочно посчитали, что среда коммуникации накладывает ограничение на последовательность работы остальных модулей. Хотя последовательный опрос каналов не означает последовательную работу модулей. Чтобы избежать такой ошибки

при моделировании, можно выделить среду коммуникации в отдельный процесс и аккуратно работать с гранулярностью.

Другим оцениваемым параметром было составление требования на языке LTL. Сложность заключается в переходе от естественного языка к формальному, при этом необходимо не ошибиться, составить требование в соответствии с формальной семантикой.

Некоторые участники не смогли составить требования. Большинство из составленных требований формулировались в терминах реализации модели и в виде “когда-нибудь БИУС-Л будет регистрировать угловую скорость”. При этом только несколько участников смогли отразить, что требование должно выполняться многократно и по условию: “всегда, если команда перехода на эллиптическую орбиту дана, то этот переход будет завершен”. Надо заметить, что на практике такой формулировки было бы недостаточно, поскольку, как правило, в подобных системах предусмотрена штатная обработка ошибок, а, значит, условия только с подачей команды будет недостаточно.

В число оцениваемых параметров входила работоспособность модели – то есть наличие бесконечных вычислений, а также анализ контрпримеров, то есть вычислений, нарушающих свойство.

Приблизительно трети участников не удалось разработать работоспособную модель, а именно, даже в режиме симуляции возникали взаимные блокировки процессов. Многие из этих команд не смогли обнаружить неработоспособность своих моделей самостоятельно. Здесь сказывается отсутствие опыта анализа контрпримеров, а также незнание базовых возможностей верификатора SPIN: в SPIN есть проверка на отсутствие взаимных блокировок процессов. Анализ причин, вызвавших блокировки процессов, показал, что некоторые участники не освоили семантики операторов в Promela: SPIN проверяет все операторы Promela на выполнимость, если оператор по какой-либо причине выполнить нельзя, то выполнение процесса блокируется.

В частности, ошибка состояла в том, что оператор выбора при невыполнимости всех условий блокирует процесс, а не продолжает вычисление, как это происходит в языках высокого уровня.

Всего 4 работы представили контрпримеры, сгенерированные SPIN при опровержении свойства. Ни один из участников не представил анализа контрпримеров. Некоторые из заявленных контрпримеров продемонстрировали непонимание авторами ошибок, найденных верификатором. Заметим, что несмотря на то, что в методе проверки модели верификатор сам генерирует контрпримеры, их анализ не тривиален. Во-первых, контрпример может свидетельствовать о нарушении базовых свойств распределенных систем, например, взаимная блокировка процессов или несоблюдение свойств на несправедливых вычислениях. Во-вторых, контрпример может свидетельствовать как о нарушении моделью свойства, так и о неправильно составленном свойстве при относительно корректной модели.

6.2.2 Интересные идеи и решения

Всего одной команде удалось полностью решить задачу о миссии “Луна-25”. Это команда VeriCheckTeam (СПбПУ).

На диаграмме последовательности сообщений процессов модели, разработанной участниками VeriCheckTeam (рис. 4), показан один из путей, приводящих к проблеме: команда *enable_bius* была отправлена, однако из-за переполнения канала данных от БИУС-Л нулями, отправлена команда *reset*, которая очистила буфер команд вместе с *enable_bius* и БИУС-Л не запустил измерение угловой скорости. Такая ошибка легко находится в результате верификации модели относительно LTL формулы

$$\mathbf{G}\ ((\text{isBiusEnableCommandSend} \,\&\&\, \text{simulationState} == 1) \rightarrow \mathbf{F}\ \text{isBiusEnabled}),$$

где *isBiusEnableCommandSend* – внутренняя переменная модели, устанавливаемая при каждой отправке сообщения *enable_bius, simulationState* – внутреннее состояние модели для

симуляции этого действия, а *isBiusEnabled* – переменная модели, устанавливаемая при актуальном выполнении команды *enable_bius*.

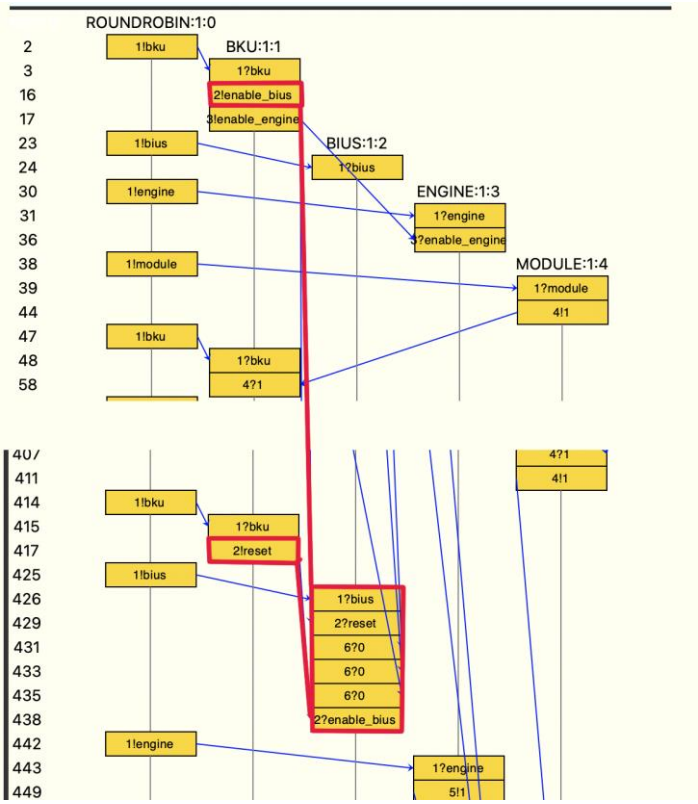

Рис. 4. Моделирование версии аварии станции в решении участников [74]
Fig. 4. Modeling a version of the Luna problem in the participants' solution [74]

Участники команды смогли смоделировать не только исходную задачу, но и исправленное решение. В качестве исправления VeriCheckTeam предложили на стороне БКУ обрабатывать за раунд не по одному сообщению, а все поступившие сообщения. В результате, по мнению команды, у БКУ не будет возникать переполнения, а, значит, не возникнет необходимость сбрасывать состояние БИУС-Л. Предложенное решение не совпадает с решением организаторов соревнования, последние предполагали изменить алгоритм взаимодействия с БИУС-Л после отправки команды на переход эллиптическую орбиту. Однако ценность решения команды VeriCheckTeam состоит в его законченности. Эта команда завоевала первое место.

Хотелось бы отметить несколько решений, представленных другими участниками.

Модель команды NSU (НГУ) отличалась системностью проектирования. Команде удалось выделить состояния каждого процесса и проработать переходы между состояниями в соответствии с описанием задачи, а также смоделировать коммуникационную среду. К сожалению, решение задачи получилось незавершенным. Эта команда завоевала третье место.

Команде Kokorin (Группа Астра) удалось разработать модель, наиболее четко соответствующую основам моделирования распределенных алгоритмов. Решение аккуратно работало с чередованием процессов и гранулярностью шагов модели. От призовых мест эту команду отделило слишком значительное упрощение алгоритма, изложенного в задаче.

Команда Maksina_Vozian_Kiseleva из СПбПУ предложила оригинальный способ очистки каналов посредством каскадной пересылки.

Некоторые из команд описали исправленное решение, согласно последовательности гипотетических взаимодействий, предложенной организаторами, но реализовать его не успели.

Команда Team Hype из Иннополиса снабдила свое, хоть и недостаточно полное решение, презентацией по найденным в сети Интернет материалам по поводу неудачных миссий в истории космонавтики с анализом причин возникших проблем.

Что касается других задач соревнования по Model Checking, по второй задаче (GPGPU) было отправлено только одно решение, предлагающее моделировать несколько планировщиков исполнения инструкций в отдельных процессах, SM модули [77] и кеш инструкции. Решение, однако, не было доделано, прежде всего, из-за того, что отправившая его команда никогда раньше не писала модели на такого рода языках (вручена почетная грамота за попытку решения). Попытки решения третьей задачи (Eurobalise), насколько нам известно, не предпринимались, но, тем не менее, очно приехавший участник заявил, что официальная спецификация этого протокола ему понравилась и он после соревнования планирует разбираться в данной теме самостоятельно.

7. Результаты и обратная связь

В конце соревнований все участники получили сертификаты участия в соревновании, дипломы за 1-3 места, а также почетные грамоты. Решение о выдаче почетных грамот в целом соответствует практикам проведения олимпиад и поощрения участников за достижения по конкретным задачам. На рис. 5 представлен процесс вручения таких грамот и дипломов победителям.

Рис. 5. Очное вручение почетных грамот участникам из Иннополиса и дипломов победителей участникам из СПбПУ и НГУ
Fig. 5. In-person presentation of certificates of honor to participants and winners

После объявления результатов был проведен анонимный опрос с целью получения обратной связи от участников. В результате было получено 9 мнений. В целом, соревнования понравились участникам (рис. 6). Все участники опроса поставили положительные оценки за соревнование, при этом хороших оценок было большинство (рис. 7).

Интересным оказалось мнение участников по поводу формулировок задач: многие отметили их неоднозначность (рис. 8). Однако заметим, что, в целом, формулировки задач по проверке моделей предполагали некоторый уровень креативности в разработке моделей, а при решении двух сложных задач ожидалось переосмысление моделей организаторов, что оказалось сложным для данного формата соревнований.

Уровень сложности задач был оценен по-разному, но в целом он оказался нормальным для большинства участников (рис. 9). Время на выполнение задач оказалось в целом достаточным, хотя были мнения и о необходимости его увеличить (рис. 10).

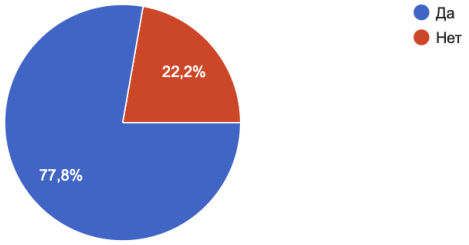

Рис. 6. Распределение участников опроса по общему впечатлению (нравится/не нравится)
Fig. 6. Distribution of survey participants by general impression

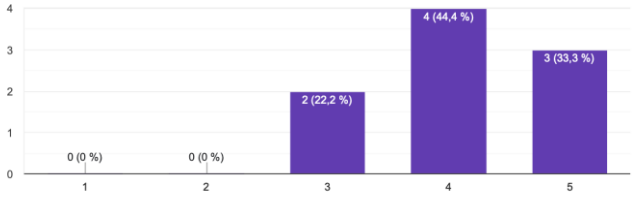

Рис. 7. Распределение участников опроса по оценкам соревнования по пятизначной шкале
Fig. 7. Distribution of survey participants according to competition ratings on a five-digit scale

Рис. 8. Распределение участников опроса по оценкам уровня формулировок задач
Fig. 8. Distribution of survey participants according to assessments of the level of task formulations

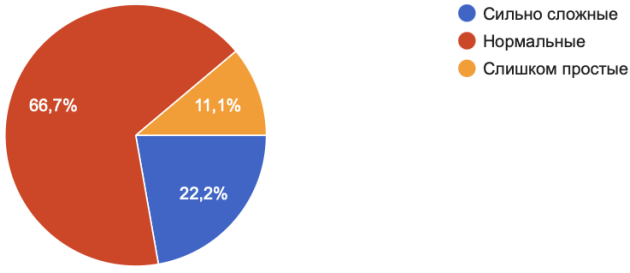

Рис. 9. Распределение участников опроса по оценкам уровня сложности задач
Fig. 9. Distribution of survey participants according to assessments of the level of difficulty of tasks

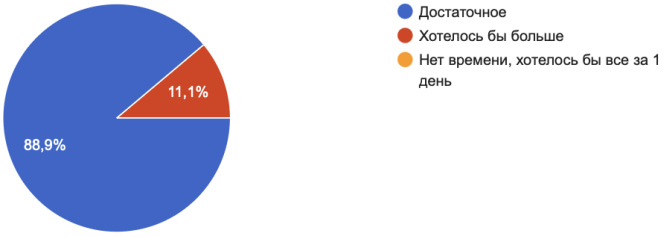

Рис. 10. Распределение участников опроса по оценкам времени на решение задач
Fig. 10. Distribution of survey participants according to estimates of time to solve problems

Среди дополнительных комментариев участники выказали желание проводить больше таких соревнований, поблагодарили организаторов за проведенное время, высказали мнение о необходимости презентации сделанных решений, а также был комментарий о необходимости использовать для дедуктивной верификации современные общеизвестные системы поддержки доказательств.

8. Заключение

Мы считаем, что соревнование VeHa-2023 прошло успешно, особенно в качестве первого опыта проведения такого рода мероприятий.

Абсолютное большинство команд справилось с предложенной на данном соревновании задачей по дедуктивной верификации. Поэтому мы пришли к выводу, что опасения, исходя из которых была выбрана относительно простая задача, оказались преувеличены, и на следующем соревновании по дедуктивной верификации в рамках контеста VeHa можно усложнить задачу и приблизить ее к миру индустриального программирования.

Решения соревнования по проверке моделей продемонстрировали не вполне радужную картину, так как почти все предоставленные модели систем и требований содержали различного рода ошибки. Но при этом участники также предлагали интересные и оригинальные способы моделирования различных составляющих систем. В целом, все участники на приемлемом уровне освоили язык Promela, однако для формирования более качественных решений требуется погружение в его тонкости и ограничения, которого сложно достичь без присутствия менторов рядом с командами.

Кроме того, соревнование показало, что требование решать задачи и по дедуктивной верификации, и по проверке моделей оказалось довольно трудным для исполнения, что, возможно, привело к снижению качества предоставленных решений и к отказу участников от решения сложных задач.

Уровень следующего контеста VeHa-2024 планируется поднять, взяв в качестве задач интересные проблемы верификации программ из области индустриального программирования, а также расширив список используемых инструментов верификации, включая собственные решения. Также мы будем учитывать нынешний опыт проведения, пожелания и предложения участников контеста VeHa-2023.

Список литературы / References

[1]. PSSV 2023. Available at: https://persons.iis.nsk.su/en/PSSVfrom2022towards2023, accessed Jul 06,2024.
[2]. Zavyalov A., Staroletov S. Flovver: A graphical functional language with a compiler focused on recursion optimization. Computing, Telecommunications and Control, vol. 16, no. 1, pp. 46–59, 2023.
[3]. Hackathon Barnaul 2023. Available at: https://innovaltai.ru/sobytiya/detail.php?ID=3161, accessed Jul 06, 2024.
[4]. Participants in the Neftecode hackaton have developed solutions that will help build more durable roads in Russia, 2022. Available at: https://news.itmo.ru/ru/startups_and_business/partnership/news/12855/, accessed Jul 06, 2024.

[5]. Tender Hack, 2023. Available at: https://tenderhack.ru/tenderhack-kazan.pdf?roistat_visit=1154183, accessed Jul 06, 2024.
[6]. Sadovykh A., Beketova M., Khazeev M. Hackathons as a part of software engineering education: Case in tools example. In Proc. Frontiers in Software Engineering Education: First International Workshop, FISEE 2019, Villebrumier, France, 2019, Invited Papers 1. November 11–13, Springer, 2020, pp. 232–245.
[7]. Hoare C. A. R. An axiomatic basis for computer programming. Communications of the ACM, vol. 12, no. 10, pp. 576–580, 1969.
[8]. Apt K. R., Olderog E.-R. Assessing the success and impact of Hoare's logic. In Proc. Theories of Programming: The Life and Works of Tony Hoare, 2021, pp. 41–76.
[9]. Apt K. R., Olderog E.-R. Fifty years of Hoare's logic. Formal Aspects of Computing, vol. 31, no. 6, pp. 751– 807, 2019.
[10]. Filliâtre J.-C. Deductive software verification. International Journal on Software Tools for Technology Transfer, vol. 13, no. 5, pp. 397–403, 2011.
[11]. Clarke J., Emerson E. Design and synthesis of synchronization skeletons using branching time temporal logic. In proc. Logic of Programs: Workshop, Yorktown Heights, ser. LNCS. Springer, 1981, vol. 131, pp. 52–71.
[12]. Quielle J., Sifakis J. Specification and verification of concurrent systems in CESAR. In Proc. of the 5th International Symposium on Programming, ser. LNCS. Springer, 1982, vol. 137, pp. 337–350.
[13]. Hähnle R., Huisman M. Deductive software verification: From pen-and-paper proofs to industrial tools. In Proc. Of Computing and Software Science, ser. LNCS. Springer, 2019, vol. 10000, pp. 345-373.
[14]. Furia C. A., Meyer B., Velder S. Loop invariants: Analysis, classification, and examples. ACM Computing Surveys, vol. 46, no. 3, pp. 1–51, 2014.
[15]. Ernst G. Loop verification with invariants and contracts. In Proc. Verification, Model Checking, and Abstract Interpretation, ser. LNCS. Springer, 2022, vol. 13182, pp. 69–92.
[16]. First E., Brun Y. Diversity-driven automated formal verification. In Proc. of the 44th International Conference on Software Engineering, 2022, pp. 749–761.
[17]. 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, vol. 48, no. 7, pp. 407–414, 2014.
[18]. Kondratyev D. A., Promsky A. V. Developing a self-applicable verification system. Theory and practice. Automatic Control and Computer Sciences, vol. 49, no. 7, pp. 445–452, 2015.
[19]. Kondratyev D. A., Nepomniaschy V. A. Automation of C program deductive verification without using loop invariants. Programming and Computer Software, vol. 48, no. 5, pp. 331–346, 2022.
[20]. Nepomniaschy V. A., Anureev I. S., Mikhailov I. N., Promskii A. V. Towards verification of C programs. C-Light language and its formal semantics. Programming and Computer Software, vol. 28, no. 6, pp. 314–323, 2002.
[21]. Nepomniaschy V. A., Anureev I. S., Promskii A. V. Towards verification of C programs: Axiomatic semantics of the C-kernel language. Programming and Computer Software, vol. 29, no. 6, pp. 338–350, 2003.
[22]. Cuoq P., Monate B., Pacalet A., Prevosto V. Functional dependencies of C functions via weakest pre-conditions. International Journal on Software Tools for Technology Transfer, vol. 13, no. 5, pp. 405–417, 2011.
[23]. Correnson L. Qed. Computing what remains to be proved. In Proc. NASA Formal Methods, ser. LNCS. Springer, 2014, vol. 8430, pp. 215–229.
[24]. Baudin P., Bobot F., Bühler D., Correnson L., Kirchner F., Kosmatov N., Maroneze A., Perrelle V., Prevosto V., Signoles J., Williams N. The dogged pursuit of bug-free C programs: the Frama-C software analysis platform. Communications of the ACM, vol. 64, no. 8, pp. 56–68, 2021.
[25]. Mandrykin M. U., Khoroshilov A. V. High-level memory model with low-level pointer cast support for Jessie intermediate language. Programming and Computer Software, vol. 41, no. 4, pp. 197–207, 2015.
[26]. Mandrykin M. U., Khoroshilov A. V. Region analysis for deductive verification of C programs. Programming and Computer Software, vol. 42, no. 5, pp. 257–278, 2016.
[27]. Mandrykin M. U., Khoroshilov A. V. Towards deductive verification of C programs with shared data. Programming and Computer Software, vol. 42, no. 5, pp. 324–332, 2016.
[28]. Efremov D., Mandrykin M., Khoroshilov A. Deductive verification of unmodified Linux kernel library functions. In Proc. Leveraging Applications of Formal Methods, Verification and Validation. Verification, ser. LNCS. Springer, 2018, vol. 11245, pp. 216–234.

[29]. Volkov G., Mandrykin M. Efremov D. Lemma functions for Frama-C: C programs as proofs. In Proc. 2018 Ivannikov Ispras Open Conference (ISPRAS), 2018, pp. 31–38.
[30]. Sammler M., Lepigre R., Krebbers R., Memarian K., Dreyer D., Garg D. RefinedC: automating the foundational verification of C code with refined ownership types. In Proc. of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2021, pp. 158–174.
[31]. Lepigre R., Sammler M., Memarian K., Krebbers R., Dreyer D., and Sewell P. VIP: verifying real-world C idioms with integer-pointer casts. In Proc. vol. 6 POPL, 2022, pp. 1–32.
[32]. 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, vol. 53, no. 7, pp. 653–662, 2019.
[33]. Nepomniaschy V. A. Symbolic method of verification of definite iterations over altered data structures. Programming and Computer Software, vol. 31, no. 1, pp. 1–9, 2005.
[34]. Moore J. S. Milestones from the pure Lisp theorem prover to ACL2. Formal Aspects of Computing, vol. 31, no. 6, pp. 699–732, 2019.
[35]. Cimatti A., Clarke E., Giunchiglia E., Giunchiglia F., Pistore M., Roveri M., Sebastiani R., Tacchella A. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In Proc. International Conference on Computer-Aided Verification (CAV 2002), ser. LNCS, vol. 2404. Copenhagen, Denmark: Springer, 2002.
[36]. Kwiatkowska M., Norman G., Parker D. PRISM 4.0: Verification of probabilistic real-time systems. In Proc. Computer Aided Verification, ser. LNCS. Springer Berlin Heidelberg, 2011, pp. 585–591.
[37]. Visser W., Păsăreanu C., Khurshid S. Test input generation with java PathFinder. In Proc. of the ACM/SIGSOFT International Symposium on Software Testing and Analysis. ACM Press, 2004, pp. 97– 107.
[38]. Hessel A., Larsen K., Mikucionis M., Nielsen B., Pettersson P., Skou A. Testing real-time systems using Uppaal. In Proc. Formal Methods and Testing, 2008, pp. 77–117.
[39]. Kuppe M. A., Lamport L., Ricketts D. The TLA+ toolbox. In Proc. Fifth Workshop on Formal Integrated Development Environment, F-IDE@FM 2019, Porto, Portugal, 7th October 2019, ser. EPTCS, R. Monahan, V. Prevosto, and J. Proenca, Eds., vol. 310, 2019, pp. 50–62.
[40]. Holzmann G. J. The Spin model checker, primer and reference manual. 2003.
[41]. Victor Duarte da Silva. Syntax highlighting for Promela and Spin Simulation on debugger. Available at: https://marketplace.visualstudio.com/items?itemName=dsvictor94.promela, accessed Jul 06, 2024.
[42]. Jacobs B., Kiniry J. Warnier M. Java program verification challenges. In Proc. Formal Methods for Components and Objects, ser. LNCS. Springer, 2003, vol. 2852, pp. 202–219.
[43]. Leavens G. T., Leino K. R. M., Müller P. Specification and verification challenges for sequential object-oriented programs. Formal Aspects of Computing, vol. 19, no. 2, pp. 159– 189, 2007.
[44]. Leino K. R. M., Moskal M. VACID-0: verification of ample correctness of invariants of data-structures, edition 0. In Proc. of Tools and Experiments Workshop at VSTTE, 2010.
[45]. Weide B. W., Sitaraman M., Harton H. K., Adcock B., Bucci P., Bronish D., Heym W. D., Kirschenbaum J., Frazier D. Incremental benchmarks for software verification tools and techniques. In Proc. Verified Software: Theories, Tools, Experiments, ser. LNCS. Springer, 2008, vol. 5295, pp. 84–98.
[46]. Huisman M., Klebanov V., Monahan R. VerifyThis 2012. International Journal on Software Tools for Technology Transfer, vol. 17, no. 6, pp. 647–657, 2015.
[47]. Hoare C. A. R., Leavens G. T., Shankar N. The verified software initiative: A manifesto. ACM Computing Surveys, vol. 41, no. 4, pp. 1–8, 2009.
[48]. Müller P., Shankar N. The first fifteen years of the verified software project. In Proc. Theories of Programming: The Life and Works of Tony Hoare, 2021, pp. 93–124.
[49]. Klebanov V., Müller P., Shankar N., Leavens G. T., Wüstholz V., Alkassar E., Arthan R., Bronish D., Chapman R., Cohen E., Hillebrand M., Jacobs B., Leino K. R. M., Monahan R., Piessens F., Polikarpova N., Ridge T., Smans J., Tobies S., Tuerk T., Ulbrich M., Weiß B. The 1st verified software competition: Experience report. In Proc. FM 2011: Formal Methods, ser. LNCS. Springer, 2011, vol. 6664, pp. 154 – 168.
[50]. Filliâtre J.-C., Paskevich A., Stump A. The 2nd verified software competition: Experience report. In Proc. of the 1st International Workshop on Comparative Empirical Evaluation of Reasoning Systems, ser. CEUR Workshop Proceedings. RWTH Aachen University, 2012, vol. 873, pp. 36–49.
[51]. Bormer T., Brockschmidt M., Distefano D., Ernst G., Filliâtre J.-C., Grigore R., Huisman M., Klebanov V., Marché C., Monahan R., Mostowski W., Polikarpova N., Scheben C., Schellhorn G., Tofan B.,

Tschannen J., Ulbrich M. The COST IC0701 verification competition 2011. In Proc. Formal Verification of Object-Oriented Software, ser. LNCS. Springer, 2012, vol. 7421, pp. 3–21.

[52]. Huisman M., Klebanov V., Monahan R. On the organisation of program verification competitions. In Proc. of the 1st International Workshop on Comparative Empirical Evaluation of Reasoning Systems, ser. CEUR Workshop Proceedings. RWTH Aachen University, 2012, vol. 873, pp. 50–59.

[53]. Beyer D., Huisman M., Klebanov V., Monahan R. Evaluating software verification systems: Benchmarks and competitions (Dagstuhl reports 14171). Dagstuhl Reports, vol. 4, no. 4, pp. 1–19, 2014.

[54]. Huisman M., Klebanov V., Monahan R., Tautschnig M. VerifyThis 2015. International Journal on Software Tools for Technology Transfer, vol. 19, no. 6, pp. 763–771, 2017.

[55]. Dross C., Furia C. A., Huisman M., Monahan R., Müller P. VerifyThis 2019: a program verification competition. International Journal on Software Tools for Technology Transfer, vol. 23, no. 6, pp. 883–893, 2021.

[56]. Ernst G., Huisman M., Mostowski W., Ulbrich M. VerifyThis – verification competition with a human factor. In Proc. Tools and Algorithms for the Construction and Analysis of Systems. Ser. LNCS. Springer, 2019, vol. 11429, pp. 176-195.

[57]. Huisman M., Monti R., Ulbrich M., Weigl A. The VerifyThis collaborative long term challenge. In Proc. Deductive Software Verification: Future Perspectives, ser. LNCS. Springer, 2020, vol. 12345, pp. 246 – 260.

[58]. Ernst G. Weigl A. Verify This: Memcached–a practical long-term challenge for the integration of formal methods. In Proc. iFM 2023, ser. LNCS. Springer, 2024, vol. 14300, pp. 82–89.

[59]. Bartocci E., Beyer D, Black P. E., Fedyukovich G., Garavel H., Hartmanns A., Huisman M., Kordon F., Nagele J., Sighireanu M., Steffen B., Suda M., Sutcliffe G., Weber T., Yamada A. TOOLympics 2019: An overview of competitions in formal methods. In Proc. Tools and Algorithms for the Construction and Analysis of Systems, ser. LNCS. Springer, 2019, vol. 11429, pp. 3–24.

[60]. Beyer D., Huisman M., Kordon F., Steffen B. TOOLympics II: competitions on formal methods. International Journal on Software Tools for Technology Transfer, vol. 23, no. 6, pp. 879–881, 2021.

[61]. Jasper M., Mues M., Murtovi A., Schluter M., Howar F., Steffen B., Schordan M., Hendriks D., Schiffelers R., Kuppens H., Vaandrager F. W. RERS 2019: Combining synthesis with real-world models. In Tools and Algorithms for the Construction and Analysis of Systems, ser. LNCS. Springer, 2019, vol. 11429, pp. 101–115.

[62]. Geske M., Jasper M., Steffen B., Howar F., Schordan M., van de Pol J. Rers 2016: Parallel and sequential benchmarks with focus on LTL verification. In Proc. Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, ser. LNCS. Springer, 2016, vol. 9953, pp. 787–803.

[63]. Jasper M., Fecke M., Steffen B., Schordan M., Meijer J., van de Pol J., Howar F., Siegel S. F. The Rers 2017 challenge and workshop (invited paper). In Proc. of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, 2017, pp. 11–20.

[64]. Jasper M., Mues M., Schlüter M., Steffen B, Howar F. Rers 2018: CTL, LTL, and reachability. In Proc. Leveraging Applications of Formal Methods, Verification and Validation of Systems, ser. LNCS. Springer, 2018, vol. 11245, pp. 433-447.

[65]. Howar F., Jasper M., Mues M., Schmidt D., Steffen B. The Rers challenge: towards controllable and scalable benchmark synthesis. International Journal on Software Tools for Technology Transfer, vol. 23, no. 6, pp. 917– 930, 2021.

[66]. Giesl J., Rubio A., Sternagel C., Waldmann J., Yamada A. The termination and complexity competition. In Proc. Tools and Algorithms for the Construction and Analysis of Systems, ser. LNCS. Springer, 2019, vol. 11429, pp. 156–166.

[67]. Marché C., Zantema H. The termination competition. In Proc. Term Rewriting and Applications, ser. LNCS. Springer, 2007, vol. 4533, pp. 303–313.

[68]. Giesl J, Mesnard F., Rubio A., Thiemann R., Waldmann J. Termination competition (TermCOMP 2015). In Proc. Automated Deduction - CADE-25, ser. LNCS. Springer, 2015, vol. 9195, pp. 105–108.

[69]. Yamada A. Termination of term rewriting: Foundation, formalization, implementation, and competition. In Proc. 8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023), ser. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum fur Informatik, 2023, vol. 260, pp. 4:1–4:5.

[70]. Ahrendt W., Herber P., Huisman M., Ulbrich M. SpecifyThis – bridging gaps between program specification paradigms. In Proc. Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles, ser. LNCS. Springer, 2022, vol. 13701, pp. 3-6.

[71]. VeHa contest, 2023. Available at: https://sites.google.com/view/veha23/, accessed Jul 06, 2024.
[72]. The First Computer Bug, 1945. Available at: https://commons.wikimedia.org/wiki/File:First_Computer_Bug,_1945.jpg, accessed Jul 06, 2024.
[73]. GitHub, Inc, Creating a pull request, 2023. Available at: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request, accessed Jul 06, 2024.
[74]. VeHaContest2023 on GitHub. Available at: https://github.com/SergeyStaroletov/VeHaContest2023, accessed Jul 06, 2024.
[75]. Luna 25 mission, RBC. Available at: https://www.rbc.ru/technology_and_media/03/10/2023/651bbeab9a794768782f8d0d, accessed Jul 06, 2024.
[76]. Luna-25 mission, 2023. Available at: https://www.roscosmos.ru/39790/, accessed Jul 06, 2024.
[77]. Garanina N., Staroletov S., Gorlatch S. Model checking meets auto-tuning of high-performance programs," in International Symposium on Logic-Based Program Synthesis and Transformation. Springer, 2022, pp. 63-82.
[78]. Khronos. OpenCL Specification by Khronos Working Group, 2021. Available: https://www.khronos.org/registry/OpenCL/specs/3.0-unified/html/OpenCL_API.html, accessed Jul 06, 2024.
[79]. Tor A. GPGPU-Sim. Available at: http://gpgpu-sim.org, accessed Jul 06, 2024.
[80]. Staroletov S. Towards Modeling and Verification of Eurobalise Telegram Encoding Algorithm. Transportation Research Procedia, vol. 61, pp. 447–454, 2022.
[81]. Basics of Software Testing and Verification (in Russian). Lanbook publishing, Saint Petersburg, p. 344, 2020. – EDN SGQVLL. Available at: https://e.lanbook.com/book/319445, accessed Jul 06, 2024.
[82]. An Efficient Logic Model Checker for the Verification of threaded Code, 2023. Available at: https://github.com/nimble-code/Spin, accessed Jul 06, 2024.
[83]. select - non-deterministic value selection, 2021. Available at: http://spinroot.com/spin/Man/select.html, accessed Jul 06, 2024.

Информация об авторах / Information about authors

Сергей Михайлович СТАРОЛЕТОВ – кандидат физико-математических наук, доцент (ВАК). Сфера научных интересов: формальная верификация, проверка моделей, киберфизические системы, операционные системы.

Sergey Mikhailovich STAROLETOV – Cand. Sci. (Phys.-Math.), associate professor. Research interests: formal verification, model checking, cyber-physical systems, operating systems.

Дмитрий Александрович КОНДРАТЬЕВ – кандидат физико-математических наук, научный сотрудник ИСИ СО РАН, старший преподаватель НГУ. Сфера научных интересов: формальная верификация, дедуктивная верификация, логика Хоара и автоматическое доказательство теорем.

Dmitry Alexandrovich KONDRATYEV – Cand. Sci. (Phys.-Math.), researcher at Ershov Institute of Informatics Systems Siberian Branch of the RAS, senior lecturer at Novosibirsk State University. Research interests: formal verification, deductive verification, Hoare logic and automated theorem proving.

Наталья Олеговна ГАРАНИНА – кандидат физико-математических наук, старший научный сотрудник ИСИ СО РАН, доцент НГУ. Сфера научных интересов: формальная верификация, проверка моделей, распределенные системы, инженерия требований.

Natalia Olegovna GARANINA – Cand. Sci. (Phys.-Math.), senior researcher at Ershov Institute of Informatics Systems Siberian Branch of the RAS, associate professor at Novosibirsk State University. Research interests: formal verification, model checking, distributed systems, requirements engineering.

Ирина Владимировна ШОШМИНА – кандидат технических наук, доцент. Сфера научных интересов: формальная верификация, распределенные системы и алгоритмы.

Irina Vladimirovna SHOSHMINA – Cand. Sci. (Tech.), associate professor. Research interests: formal verification, model checking, distributed systems and algorithms.