**DOI: 10.15514/ISPRAS-2025-37(1)-10**



# Соревнования по формальной верификации VeHa-2024: накопленный в течение двух лет опыт и перспективы

[1] *Д.А. Кондратьев, ORCID: 0000-0002-9387-6735 <apple-66@mail.ru>*
[2] *С.М. Старолетов, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>*
[3] *И.В. Шошмина, ORCID: 0000-0001-5120-0385 <shoshmina_iv@spbstu.ru>*
[4] *А.В. Красненкова, ORCID: 0009-0009-5742-1555 <akrasnenkova@astralinux.ru>*
[5, 6] *К.В. Зиборов, ORCID: 0000-0002-5676-9105 <krl.ziborov@gmail.com>*
[7] *Н.В. Шилов, ORCID: 0000-0001-7515-9647 <shiloviis@mail.ru>*
[1] *Н.О. Гаранина, ORCID: 0000-0001-9734-3808 <garanina@iis.nsk.su>*
[4] *Т.Ю. Черганов, ORCID: 0009-0000-6119-9234 <cherganov@gmail.com>*

[1] *Институт систем информатики им. А.П. Ершова СО РАН,*
*Россия, 630090, г. Новосибирск, пр. Академика Лаврентьева, 6.*
[2] *АлтГТУ им. И.И. Ползунова,*
*Россия, 656038, Алтайский край, г. Барнаул, пр. Ленина, 46.*
[3] *Санкт-Петербургский политехнический университет Петра Великого,*
*Россия, 195251, г. Санкт-Петербург, ул. Политехническая, 29.*
[4] *РусБИТех-Астра,*
*Россия, 117105, г. Москва, ул. Варшавское шоссе, д. 26.*
[5] *Positive Technologies,*
*Россия, 107061, г. Москва, Преображенская площадь, 8.*
[6] *Московский государственный университет имени М.В. Ломоносова,*
*Россия, 119991, г. Москва, Ленинские горы, д. 1.*
[7] *Университет Иннополис, Россия, 420500, г. Иннополис, ул. Университетская, д. 1.*

**Аннотация.** В нашей статье описан опыт организации второго соревнования по формальной верификации программ среди молодых специалистов и студентов российских вузов. Соревнования проводились в связке с семинаром по семантике, спецификации и верификации программ (PSSV) в Иннополисе в октябре 2024 года. Формат соревнования был близок к формату так называемых хакатонов. Участникам было предложено решить на выбор 5 задач по верификации с использованием заранее определенных инструментов проверки моделей и дедуктивной верификации (Frama-C, Coq, C-lightVer, SPIN, TLC). Мы обсуждаем 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:** Кондратьев Д.А., Старолетов С.М., Шошмина И.В., Красненкова А.В., Зиборов К.В., Шилов Н.В., Гаранина Н.О., Черганов Т.Ю. Соревнования по формальной верификации VeHa-2024: накопленный в течение двух лет опыт и перспективы Труды ИСП РАН, том 37, вып. 1, 2025 г., стр. 159-184. DOI: 10.15514/ISPRAS–2025–37(1)–10.

# VeHa-2024 Formal Verification Contest: Two Years of Experience and Prospects

[1] *D.A. Kondratyev, ORCID: 0000-0002-9387-6735 <apple-66@mail.ru>*
[2] *S.M. Staroletov, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>*
[3] *I.V. Shoshmina, ORCID: 0000-0001-5120-0385 <shoshmina_iv@spbstu.ru>*
[4] *A.V. Krasnenkova, ORCID: 0009-0009-5742-1555 <akrasnenkova@astralinux.ru>*
[5,6] *K.V. Ziborov, ORCID: 0000-0002-5676-9105 <krl.ziborov@gmail.com>*
[7] *N.V. Shilov, ORCID: 0000-0001-7515-9647 <shiloviis@mail.ru>*
[1] *N.O. Garanina, ORCID: 0000-0001-9734-3808 <garanina@iis.nsk.su>*
[4] *T.Y. Cherganov, ORCID: 0009-0000-6119-9234 <cherganov@gmail.com>*

[1] *Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, 6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia.*
[2] *Polzunov Altai State Technical University, 46, Lenin Ave., Barnaul, 656038, Russia.*
[3] *Peter the Great St. Petersburg Polytechnic University, 29, Politekhnicheskaya street, St. Petersburg, 195251, Russia.*
[4] *RusBITech-Astra, 26, Varshavskoe Highway, Moscow, 117105, Russia.*
[5] *Positive Technologies, 8, Preobrazhenskaya sq., Moscow, 107061, Russia.*
[6] *Lomonosov Moscow State University, 1, Leninskiye Gory, Moscow, 119991, Russia.*
[7] *Innopolis University, 1, Universitetskaya Str., Innopolis, 420500, Russia.*

**Abstract.** We present our experience of organizing the second contest in formal program verification for young engineers, researchers, and students from Russia. The contest was held in conjunction with the Workshop on Program Semantics, Specification, and Verification (PSSV) in Innopolis in October 2024. The contest format was close to the format of the so-called hackathons. Participants were asked to select and solve any of 5 verification problems using pre-defined model checking and deductive verification tools (Frama-C, Coq, C-lightVer, SPIN, TLC). We discuss the issues of organizing of the contest, the problems set, lessons learned from solutions submitted by contestants, and the overall feedback from the participants.

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

**For citation:** Kondratyev D.A., Staroletov S.M., Shoshmina I.V., Krasnenkova A.V., Ziborov K.V., Shilov N.V., Garanina N.O., Cherganov T.Y. VeHa-2024 Formal Verification Contest: Two Years of Experience and Prospects. Trudy ISP RAN/Proc. ISP RAS, vol 37, issue. 1, 2025., pp. 159-184. DOI: 10.15514/ISPRAS-2025-37(1)-10.

## *1. Введение и мотивация проведения соревнования*

Основная цель соревнования VeHa-2024 – привлечь студентов Российских вузов к решению задач верификации с помощью формальных методов. Это соревнование продолжает соревнование VeHa-2023 [1], которое проводилось нами впервые. После завершения проведения первого такого соревнования была получена положительная обратная связь, поэтому было решено продолжить организацию соревнований, сделав их ежегодным дополнением к семинару по формальной верификации программ PSSV, в настоящее время проходящем в Университете Иннополис. VeHa означает «Verification Hackathon», то есть состязание по верификации, и проводится по правилам такого рода соревнований – за небольшое ограниченное время в одном месте. Для расширения круга участников в нашем соревновании разрешено и онлайн участие. Отличительной особенностью соревнования 2024

года являлось то, что организаторы стремились предложить задачи индустриальной направленности: верификация функций прав доступа, моделирование процесса исполнения параллельных программ, решение всем известной задачи SAT и верификация протокола консенсуса. Часть из этих задач была предложена российскими компаниями, работающими в сфере безопасности программного обеспечения. Таким образом, дополнительной мотивацией текущего соревнования стало показать возможность применения формальных методов для реальных задач.

Дальнейшее содержание статьи представляет собой обзор структуры соревнования (раздел 2), других схожих соревнований по формальной верификации (раздел 3), сравнение с командным соревнованием по программированию (раздел 4), описание предложенных задач (раздел 5), обзор ошибок в решениях (раздел 6), результаты соревнования (раздел 7), а в заключении делаем некоторые выводы (раздел 8).

## *2. Структура соревнования*

Формат мероприятия в 2024 году сложился в результате беседы во время награждения победителей прошлогоднего соревнования из Москвы между сессиями Открытой конференции ИСП РАН 2023. Участниками и победителями того соревнования были представители Группы Астра (известная по продукту Astra Linux), которые после получения наград предложили в следующем году организовать свою секцию по Frama-C и Coq, которые реально используются в их компании в отделе анализа безопасности и формальной верификации. В итоге к двум имеющимся секциям по проверке моделей и дедуктивной верификации на основе инструмента C-lightVer добавились две секции от группы Астра, также сотрудник Positive Technologies и МГУ им. М. В. Ломоносова предложил секцию по проверке модели протокола консенсуса. Кроме этого, планировалась и секция по выводу типов на основе реализуемого в Иннополисе средства Rzk, но данная секция была отменена в последний момент по личным обстоятельствам организатора. Поскольку секций стало больше, чем две, а задачи усложнились (по сравнению с 2023 годом), было решено подводить итоги по каждой из секций отдельно, что отличается от процедуры 2023 года, когда нужно было обязательно решать задачи из обеих секций.

В 2024 году организаторами соревнования являлись:

- Гаранина Наталья (к.ф.-м.н., с.н.с. Института Систем Информатики, Института Автоматики и Электрометрии СО РАН, доцент НГУ) – ответственная за секцию проверки моделей;
- Зиборов Кирилл (аспирант механико-математического факультета МГУ им. М. В. Ломоносова и инженер Positive Technologies) – ответственный за секцию "Построение и проверка модели протокола консенсуса IBFT";
- Кокорин Артём (Старший специалист по анализу безопасности отдела анализа СЗИ и формальной верификации, группа Астра) – Ответственный за секцию Frama-C;
- Кондратьев Дмитрий (к.ф.-м.н., н.с. Института систем информатики им. А.П. Ершова СО РАН и старший преподаватель Новосибирского Государственного Университета) – ответственный за секцию дедуктивной верификации с помощью системы C-lightVer;
- Красненкова Анастасия (Специалист по анализу безопасности отдела анализа СЗИ и формальной верификации, группа Астра) – ответственная за секцию Frama-C;
- Никольский Денис (Университет Иннополис) – координатор задач;
- Сим Виолетта (Университет Иннополис) – координатор задач (Участник соревнования VeHa-2023);
- Старолетов Сергей (к.ф.-м.н., доцент АлтГТУ, с.н.с. Института Автоматики и Электрометрии СО РАН) – ответственный за секцию проверки моделей и за сайт

соревнований;

- Черганов Тимофей (Старший специалист по анализу безопасности отдела анализа СЗИ и формальной верификации, группа Астра) – ответственный за секцию Coq;
- Шилов Николай (к.ф.-м.н, Университет Иннополис) – организатор родительского семинара PSSV, координатор;
- Шошмина Ирина (к.т.н, доцент кафедры распределенных вычислений СПбПУ) – ответственная за секцию проверки моделей.

Были определены координаторы задач, которые проверяли условия задач на непротиворечивость, а материалы к ним на доступность и воспроизводимость примеров.

Задачи, которые вошли в финальный перечень предложенных на соревнование:

1. Верификация функции проверки прав доступа на Frama-C.
2. Верификация функции проверки прав доступа на Coq.
3. Моделирование вычислительного конвейера графического процессора с поиском оптимальных параметров с помощью проверки моделей.
4. Дедуктивная верификация программы, относящейся к задаче проверки выполнимости булевых формул (SAT).
5. Построение и проверка модели протокола консенсуса IBFT.

Через форму предварительной регистрации было подано более 100 заявок, в аффилиациях участников были указаны следующие университеты:

- НГУ;
- МГТУ им Н.Э.Баумана;
- МФТИ;
- РТУ МИРЭА;
- СПБПУ;
- Университет Иннополис;
- МГУ им. М. В. Ломоносова;
- Университет ИТМО.

Как и в 2023 году, был предложен гибридный формат участия: можно было участвовать как онлайн, так и очно в Иннополисе, с проживанием в гостинице и возможностью посетить сессии и экскурсии совместно с участниками семинара PSSV. Распределение участников по планируемой форме участия показано на рис. 1.

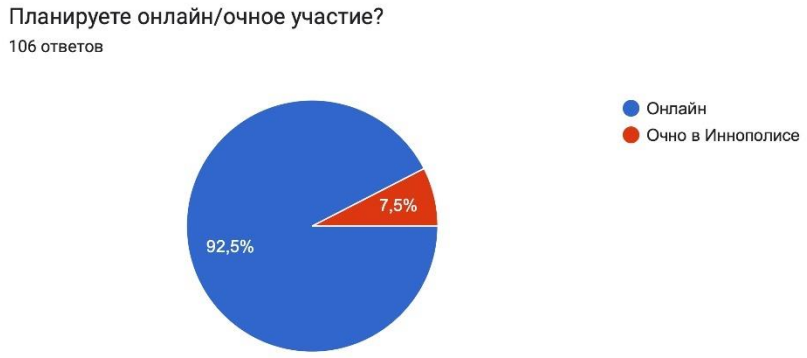


*Рис. 1. Распределение голосования за онлайн и очное участие.*
*Fig. 1. Distribution of votes by online/offline.*

В форме регистрации также предлагалось указать планируемую технологию верификации, с небольшим перевесом тут оказалась проверка моделей, см. рис. 2.

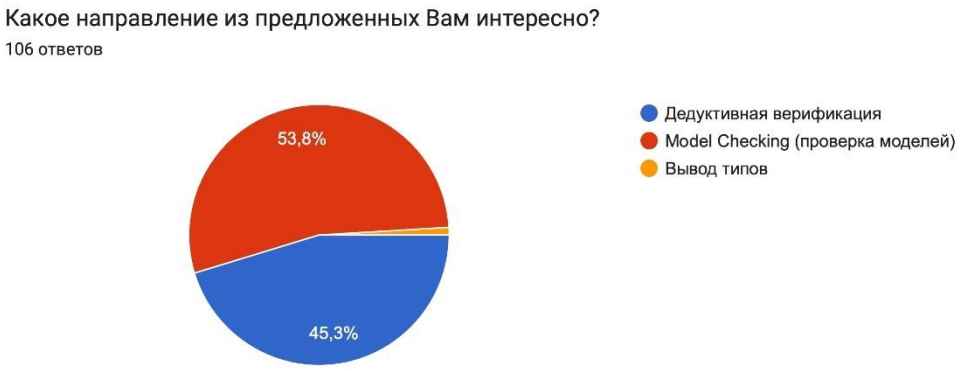


*Рис. 2. Распределение предпочтений по технологиям верификации.*
*Fig. 2. Distribution of preferences by verification technologies.*

По предварительной регистрации на задачи небольшой перевес был отдан задаче проверки моделей для конвейера GPU (рис. 3).

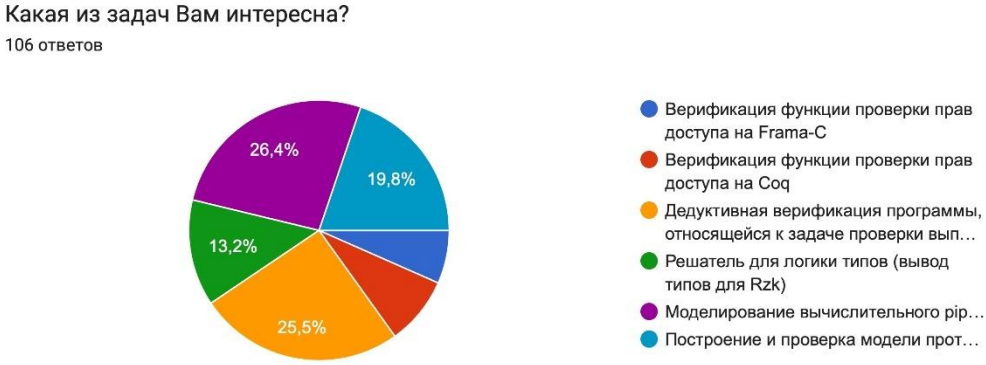


*Рис. 3. Распределение выбора по предложенным задачам.*
*Fig. 3. Distribution of choices by proposed tasks.*

Для агрегации всех материалов, как и в прошлом году, мы разработали сайт соревнований на платформе Google Sites. На сайте были размещены материалы по задачам, включая образ виртуальной машины для задач 1 и 2, предварительные описания и полные описания задач соревнования (открыты при старте соревнований), даны ссылки на Telegram-группу для быстрых обсуждений, Skype-группу для обучающих уроков и консультаций, а также GitHub-репозиторий для последующей загрузки решений с использованием технологии «pull-request» (с созданием там своей папки в формате solution_x_name, где x – номер задачи, а name – название команды).

Основное взаимодействие с участниками (объявления, ответы на вопросы, уведомления о появлении новых материалах на сайте) осуществлялось через мессенджер Telegram.

Нами было установлено следующее расписание соревнований:

- 13 октября 2024 (за неделю до старта соревнования) – проведение онлайн-уроков по двум задачам в Skype.
- 10:00 18 октября – публикация полных текстов заданий на сайте.
- Во время соревнований проводились онлайн и очные консультации.
- 23:59 21 октября – завершение загрузки решений.
- 26 октября – встреча оргкомитета соревнования, подведение результатов и решения по грамотам.
- 27 октября – публикация информации о победителях на сайте соревнования.

## *3. Обзор соревнований, курсов по формальной верификации и сравнение с описанным форматом соревнования*

Относительно подробный обзор соревнований по формальной верификации описан в статье о VeHa-2023 [1]. Но, если тот обзор [1] более ориентирован на историю соревнований по формальной верификации, чем на сравнение этих соревнований с мероприятиями серии VeHa, то в данном обзоре рассмотрим более подробно именно такое сравнение вместо исторических аспектов.

Наиболее родственными с серией соревнований VeHa в целом являются мероприятия серии TOOLympics [2-4]. Пока что соревнования TOOLympics состоялись два раза, в 2019 году [2-3] и в 2023 году [4]. Главная схожесть VeHa и TOOLympics состоит в том, что каждое из этих мероприятий объединяет в себе несколько соревнований, покрывающих оба основных направления формальной верификации программ, и дедуктивную верификацию, и проверку моделей. Например, в годы проведения TOOLympics под эгидой данного мероприятия проходят и соревнования по дедуктивной верификации программ серии VerifyThis [2-7], и соревнования по проверке моделей серии RERS [2-4, 8-9]. И, если соревнование VeHa-2023 отличалось от мероприятий серии TOOLympics тем, что итоги VeHa-2023 подводились для всего соревнования суммарно по всем трекам, то уже VeHa-2024 не отличается в этом плане от мероприятий серии TOOLympics, так как итоги VeHa-2024 подводились для каждого трека по-отдельности. Основное отличие VeHa от TOOLympics состоит в том, что на мероприятиях TOOLympics участники могут выбирать любые программные системы для решения задач по верификации, тогда как на соревнованиях VeHa участникам даются указания, какие именно программные системы использовать для решения задач. Таким образом, на мероприятиях TOOLympics важную роль играют не только компетенции участников, но и возможности выбранных участниками программных систем. Это отражено в названии мероприятий TOOLympics. А на соревнованиях VeHa проверяются именно компетенции участников, что ближе к изначальной идее хакатона, отраженной в названии серии VeHa (Verification Hackaton). Другое важное отличие состоит в том, что в рамках TOOLympics проходят соревнования не только в области дедуктивной верификации и проверки моделей, но и соревнования в некоторых других областях формальных методов в программировании, например, соревнование Termination Competition (termCOMP) [2-4, 10] по проверке выполнения свойства завершаемости исполнения программ и систем переписывания термов.

Также родственным в целом для серии VeHa является серии мероприятий SpecifyThis [11-12]. Мероприятия SpecifyThis пока что состоялись два раза, в 2022 году [11], и в 2024 году [12]. Главное отличие SpecifyThis от соревнований VeHa состоит в том, что SpecifyThis не является соревнованием. Мероприятие SpecifyThis больше похоже на рабочий семинар, участники которого обсуждают проблематику задания функциональных спецификаций, относительно которых проверяется корректность программы в ходе дедуктивной верификации, и спецификаций, написанных на языке временной логики, относительно которых проверяется корректность программных систем в ходе проверки моделей.

Обсуждение на мероприятии конкретных примеров задания спецификаций, а также название мероприятия, созвучное названию соревнования VerifyThis, демонстрируют потенциал трансформации в будущем мероприятия SpecifyThis в соревнование по заданию спецификаций программ. Главная схожесть SpecifyThis и серии соревнований VeHa состоит в том, что на большинстве треков VeHa важную роль играет задание функциональных или темпоральных спецификаций программных систем. Таким образом, проблематика задания спецификаций играет важную роль на серии соревнований VeHa в целом.

Кроме того, родственной по отношению к серии соревнований VeHa деятельностью является составление наборов задач, позволяющих проверять возможности различных подходов к формальной верификации программ. Отметим, что такие наборы составляются как отдельно от соревнований [13-16], так и по итогам соревнований по формальной верификации. Например, такой набор задач составляется по итогам серии соревнований по проверке моделей RERS [9, 17]. Особенно отметим набор задач, который составляется по итогам серии соревнований по дедуктивной верификации VerifyThis [5, 18]. Данный набор задач и их решений структурирован по годам проведения соревнования и доступен на отдельной веб-странице [19]. Условия задач прошлогоднего соревнования VeHa-2023 доступны на сайте соревнования [20], а их решения участниками доступны в репозитории соревнования [21]. Аналогично условия задач соревнования VeHa-2024 доступны на сайте соревнования [22], а их решения участниками доступны в репозитории соревнования [23].

И кроме того, важно отметить связь соревнований VeHa с такой деятельностью, как обучение формальной верификации программ. В первую очередь такая связь состоит в активном участии на соревнованиях серии VeHa студентов, проходящих обучение формальным методам в программировании. Например, многие победители и призеры соревнований VeHa из НГУ обучены/обучаются на профиле "Формальные методы анализа программ и систем", где основные дисциплины состоят в обучении различным методам формальной верификации программ [24]. Также отметим успешное выступление на соревновании VeHa-2023 студентов из Университета Иннополиса и соответствующий курс по формальной верификации программ [25]. Участие на соревнованиях позволяет студентам развить полученные на курсах навыки по формальной верификации и лучше понять практическую применимость формальных методов в программировании. Эти результаты соревнования особенно важны, поэтому, чтобы облегчить достижение таких результатов, появилась актуальная задача по встраиванию соревнований VeHa в существующие курсы по формальной верификации программ. Во вторую очередь связь соревнований VeHa с обучением формальным методам состоит в возможности проверить, могут ли студенты успешно применять разрабатываемые организаторами соревнований системы формальной верификации программ. Отметим, что применение систем формальной верификации программ на практике требует привлечения высококвалифицированных специалистов. Разработчики практически всех систем формальной верификации стремятся снизить квалификационные требования к пользователям таких систем. Для этого в таких системах реализуют методы упрощения и автоматизации формальной верификации. Такие подходы, естественно, не могут обеспечить полную автоматизацию формальной верификации, однако важно проверить, могут ли такие методы настолько снизить требования к компетенциям пользователей систем верификации, чтобы такие системы могли применять обучающиеся формальным методам студенты. Также важно получить обратную связь от студентов, какую именно функциональность рекомендуется реализовать в тестируемых таким образом системах, чтобы упростить применение таких систем. Особенно это актуально для системы дедуктивной верификации C-lightVer [26-29], которая разрабатывается в ИСИ СО РАН, одним из организаторов VeHa. Можно провести аналогию с предложениями студентов Иннополиса об улучшениях системы дедуктивной верификации AutoProof по итогам обучения применению данной системы на курсе по формальной верификации [30].

Далее рассмотрим сравнение отдельных треков соревнования VeHa-2024 с родственными соревнованиями по формальной верификации программ.

Треки соревнования VeHa-24 “Верификация функции проверки прав доступа на Frama-C” и “Дедуктивная верификация программы, относящейся к задаче проверки выполнимости булевых формул (SAT)” схожи с соревнованиями серий VSComp [31-32], VerifyThis [5-7, 33-35] и с долговременными соревнованиями VerifyThis [36-38]. Это известные соревнования по дедуктивной верификации программ, проведение которых было во многом вдохновлено проектом Verified Software Initiative по масштабному применению формальной верификации в индустриальном программировании [39-40]. Главным отличием связанных с дедуктивной верификацией треков VeHa от данных соревнований является возможность выбора участниками этих соревнований любых программных инструментов для дедуктивной верификации, что делает результат этих соревнований зависимым не только от компетенций участников, но и от возможностей выбранных программных средств [18, 41]. Отдельно отметим долговременные соревнования VerifyThis. Если обычные соревнования серии VerifyThis ограничены по времени днями проведения научных конференций, в рамках которых проводятся соревнования, то долговременные соревнования проводятся на значительных промежутках времени между научными конференциями. Такая длительность позволяет использовать масштабные задачи по индустриальной верификации, что является еще одним важным отличием от связанных с дедуктивной верификацией треков соревнования VeHa.

Трек соревнования VeHa-2024 “Верификация функции проверки прав доступа на Coq” схож не только с соревнованиями по дедуктивной верификации программ VSComp [31-32], VerifyThis [5-7, 33-35] и долговременными VerifyThis [36-38], но и с соревнованием по интерактивному доказательству теорем Proof Ground [42]. Данный трек соревнования VeHa-2024 посвящен такому этапу дедуктивной верификации, как проверка истинности в системе доказательства (на данном треке система Coq [43]) условий корректности программ. Умение успешно справляться с данным этапом дедуктивной верификации играет важную роль на соревнованиях серий VSComp, VerifyThis и долговременных соревнованиях VerifyThis. Отличие от Proof Ground состоит в том, что доказываемые на соревновании Proof Ground теоремы могут быть из самых разных предметных областей (не только из области условий корректности программных систем).

Треки соревнования VeHa-24 по моделированию вычислительного конвейера графического процессора с поиском оптимальных параметров с помощью проверки моделей и по построению и проверки модели протокола консенсуса IBFT схожи с соревнованиями по проверке моделей серии RERS [8-9, 17, 44-45]. Отметим, что в некоторых задачах соревнований серии RERS, также как и на треках VeHa-2024, для задания верифицируемых свойств в терминах логики LTL применяется язык Promela [17]. Однако на соревнованиях RERS участникам предоставляются уже заданные свойства, которые необходимо верифицировать, тогда как на треках соревнования VeHa участникам необходимо самим задавать верифицируемые свойства и модель, исходя из описания задач на естественном языке.

Из обзора родственных мероприятий можно сделать следующие выводы в отношении соревнований VeHa:

1. Полезно расширять соревнования VeHa новыми треками. В качестве таких треков может быть рассмотрено аналогичное termCOMP [10, 46-48] соревнование по доказательству завершаемости программ. Также отметим, что на соревновании VeHa-2024 планировался трек по автоматическому определению типа выражения в модельном языке с помощью специальной логики типов, однако, по независящим от организаторов причинам, данный трек в этом году был отменен. Такой трек планируется проводить, начиная со следующего соревнования серии VeHa.

2. Полезно накапливать наборы задач и их решений по итогам соревнований VeHa в более структурированном виде.
3. Актуальной задачей является встраивание соревнований серии VeHa в учебные курсы по формальной верификации программ.
4. Для развития навыков участников VeHa по решению масштабных задач по индустриальной верификации можно проводить долговременные соревнования по формальной верификации программ в промежутках между обычными соревнованиями серии VeHa.

## *4. Сравнение соревнований VeHa и ICPC*

Мы уже писали в разделе 3, что родственными с VeHa являются мероприятия серии TOOLympics [2-4], соревнования по дедуктивной верификации программ серии VerifyThis [5-7, 33-35], соревнования по проверке моделей серии RERS [8-9, 17, 44-45] и мероприятия серии SpecifyThis [11-12].

Но есть еще одни студенческие соревнования по программированию, популярность которых среди *десятков тысяч студентов* по всему миру заставляет сравнить эти соревнования с VeHa: речь идет о Международной студенческой олимпиаде по программированию, называемой также Студенческим командным чемпионатом мира по программированию (The International Collegiate Programming Contest, ICPC, до 2017 – ACM ICPC) [49-50]. Первый финал ACM ICPC был проведён в рамках ежегодной конференции ACM по информатике, 5-ой ACM Computer Science Conference (CSC'77) [51] (то есть, как соревнования VeHa проходят в аффилиации с семинаром PSSV [52]).

ICPC – соревнования между университетскими командами из трёх студентов (допускаются аспиранты первого года обучения, но не старше 24 лет), они проходят в несколько туров/этапов: отдельных университетов, региональных и финала. Каждый этап проходит следующим образом: каждой команде выдаётся один компьютер на пять часов, всем командам – один набор из 8-12 задач (условия которых написаны в свободной форме на английском языке).

Команды пишут решения на языках программирования Pascal, C, C++, Java, Python или Kotlin, и отправляют решения на *тестирующий* сервер. Корректность решений, поступивших на тестирующий сервер, проверяется *на большом количестве различных входных тестов*, подготовленных жюри, но неизвестных участникам. Решение считается корректным, если программа прошла все тесты с правильным результатом за специфицированное для каждого теста время, использовав специфицированный для каждого теста объем памяти.

Нам представляется, что ни тестирование, как метод проверки корректности решений на ICPC, ни экспертный метод разработки самих тестов для проверки корректности решений, ни неверифицированные «образцовые» решения, предлагаемые жюри после каждого тура ICPC, нельзя считать современными методами оценки корректности программ и квалификации участников соревнований. Мы считаем, что интеграция формальной верификации в студенческие соревнования по программированию (включая ICPC) является требованием времени для повышения качества оценки решений и повышения уровня образования участников. Теперь вопрос о проведении такого эксперимента на студенческих соревнованиях по программированию.

## *5. Обзор предложенных задач*

Полные условия всех задач можно найти на сайте соревнования VeHa-2024 [22].

## 5.1 Задача 1. Верификация функции проверки прав доступа на Frama-C

В качестве условий задачи даны спецификации к исходному коду на языке Си из фрагментов кода Linux Kernel в формате ACSL [53], см. листинг 1.

Функция `compute_mask()` вычисляет маску (с битами MAY_WRITE и MAY_READ), которая в дальнейшем используется для принятия решения о разрешении доступа к заданному файлу. Доступы в маске означают наличие прав соответствующего доступа.

Функция `check_permission()` проверяет доступ к заданному файлу с учетом прав пользователя на чтение/запись (см. листинг 2).

```
static int compute_mask(struct file *file, unsigned int cmd)
{
    struct inode *inode = file_inode(file);
    int mask = 0;
    int i = 0;
    // Количество событий в массиве event_numbers
    unsigned long size_array = (sizeof (event_numbers) /
                                sizeof((event_numbers)[0]));
    // Проверяем, является ли файл публичным.
    // Если да, то доступ разрешен сразу
    if (inode->i_flags & SHIFT)
        return 0;

    if (cmd > IMPORTANT) {
        return MAY_WRITE;
    }
    if (cmd < EXOTIC) {
        return MAY_READ;
    }
    // Нужно верифицировать цикл внутри функции
    for (i = 0; i < size_array; ++i)
        if (cmd == event_numbers[i][0]) {
            mask = event_numbers[i][1];
            break;
        }
    if (!mask)
        mask = MAY_READ | MAY_WRITE;
   return mask;
}
```

*Листинг 1. Функция “compute_mask”.*
*Listing 1. “compute_mask” function.*

Функция `check_permission()` возвращает 3 значения:

- 0 – доступ к файлу разрешен;
- -13 – доступ к файлу запрещен (макрос NO_PERM);
- -1 – недостаточно высокий уровень целостности пользователя, ошибка доступа (макрос NO_ILEV).

В код одной из функций организаторами была намеренно внесена ошибка. Участникам было необходимо найти её при помощи инструмента Frama-C, не изменяя уже написанные спецификации к коду; а также исправить код (обосновав своё решение) и полностью

верифицировать получившиеся функции (все цели – как сгенерированные Frama-C, так и созданные верификатором – должны быть доказаны).

Для подготовки к решению задания участникам было предложено небольшое пособие по Frama-C и даны ссылки на известные и хорошо зарекомендовавшие себя учебники по данному инструменту верификации.

```
static int check_permission (struct file *file, unsigned int cmd)
{
    const PDPL_T *sl = getCurrentLabel();
    // Вычисляем уровень целостности пользователя
    unsigned int ilev = slabel_ilev(sl);
    // Если пользователь не максимального уровня целостности,
    // то возвращаем ошибку доступа
    if (ilev != max_ilev){
        return -NO_ILEV;
    }
    // Вычисляем маску
    int mask_final = compute_mask(file, cmd);
    // Текущий процесс -- суперпользователь, и у пользователя
    // есть право на запись в файл -- доступ разрешен
    if (current->process->fsuid == 0)
        if ((mask_final & MAY_WRITE)){
            return 0;
        }
    // Текущий процесс -- не суперпользователь, и у пользователя
    // есть право на чтение из файла -- доступ разрешён
    if (!(current->process->fsuid == 0))
        if ((mask_final & MAY_READ)){
            return 0;
        }

    // В других случаях доступ запрещен
    return -NO_PERM;
}
```

*Листинг 2. Функция “check_permission”.*
*Listing 2. “check_permission” function.*

## 5.2 Задача 2. Верификация функции проверки прав доступа на Coq

Задача заключается в формальной верификации одной из функций модуля безопасности ядра Linux [54]. Поведение системы можно описать переходами между её состояниями. Состояние системы из задачи (ядра Linux):

- множество субъектов (например, процессов);
- множество объектов (например, файлов);
- матрица доступов субъектов к объектам;
- метки целостности субъектов;
- метки целостности объектов.

Переход из одного состояния в другое происходит при наступлении события. В задаче рассматривается событие получения доступа субъекта к объекту. Если в запросе есть доступ на запись, то метка целостности объекта не должна превышать метку целостности субъекта.

Функция vsm_inode_permission из модуля безопасности ядра отвечает за проверку метки целостности процесса при обращении к индексному дескриптору (см. листинг 3). Она принимает два аргумента: указатель на индексный дескриптор, запрашиваемые параметры доступа. Функция возвращает 0, если процессу разрешен доступ к индексному дескриптору.

```
#define MAY_EXEC    0x00000001
#define MAY_WRITE   0x00000002
#define MAY_READ    0x00000004
#define EACCES      13

int vsm_inode_permission(struct inode *inode, int mask)
{
        mask &= MAY_WRITE;

        if (!mask)
                return 0;

        const struct security *isec = inode_security(inode);
        const struct security *tsec = cred_security(current_cred());

        if (tsec->ilev >= isec->ilev)
                return 0;

        return -EACCES;
}
```

*Листинг 3. Функция "vsm_inode_permission".*
*Listing 3. "vsm_inode_permission" function.*

Участникам были предложены следующие задания:

- Написать спецификацию этой функции.
- Метки целостности функциональной спецификации обладают типом Z. Метки целостности формальной модели системы – элементы решетки. Нужно доказать, что тип Z является решеткой.
- Чтобы показать корректность функциональной спецификации vsm_inode_permission, нужно доказать, что функция inode_permission возвращает 0 тогда и только тогда, когда функция формальной модели checkRight разрешает доступ субъекта к объекту.
- Чтобы убедиться в корректности формальной модели, нужно доказать, что любой переход системы из одного состояния в другое сохраняет следующее свойство: у субъекта есть доступ на запись к объекту только в том случае, если метка целостности объекта не превышает метку целостности субъекта.

Для подготовки к выполнению задачи участникам было предложено краткое введение в Coq.

## 5.3 Задача 3. Моделирование вычислительного конвейера графического процессора с поиском оптимальных параметров с помощью проверки моделей

Данная задача является переработкой задачи с прошлогоднего соревнования. В задаче предлагается моделировать процесс исполнения вычислительных инструкций видеокартой с графическим процессором (такой процесс называется конвейером или пайплайном). Для

поиска оптимальных параметров (задачи автонастройки) используется метод проверки модели. Верификатор находит контрпримеры, что позволяет определить оптимальные параметры (согласно статьям авторов [55-56]). В 2023 году к этой задаче приступила лишь одна команда, полученное от нее решение было неполным. В этом году задача коллективом авторов была продумана глубже: выделены различные уровни решения, предложены в качестве начальных авторские модели конвейера.

Для решения задач, требующих интенсивных расчетов, которые хорошо распараллеливаются (вроде операций с матрицами, расчет хэш функций для майнинга, приложений ИИ), применяются графические процессоры (GPU). Все эти задачи используют особенности архитектуры графических процессоров, называемой SIMT (single instruction-multiple thread – одна инструкция – множество потоков). Эта архитектура позволяет эффективно решать задачи с одновременным выполнением однотипных операций над разными данными на большом количестве вычислителей. Для программирования задач под архитектуры GPU используются технологии OpenCL (открытая) или CUDA (закрытая от Nvidia), при этом код, исполняемый на видеокарте, пишется на C-подобном языке и называется “ядром”. Этот код впоследствии транслируется в ассемблерный код (PTX для Nvidia), а потом в его бинарное представление для исполнения на видеокарте конкретной архитектуры.

Работа программы, оптимальной по потреблению GPU-ресурсов (времени, памяти, энергии и т.п.), зависит от множества параметров, подбор которых затруднен. Задача подбора таких параметров называется задачей (авто)настройки параллельной программы. Как было показано в работах авторов [55-56], задачу автонастройки возможно решить с помощью метода проверки модели. Для этого необходимо описать модель исполнения настраиваемой программы на выбранной архитектуре GPU. Модель исполнения должна быть описана на входном языке верификатора с учётом настраиваемых параметров. В работах [55-56] авторами была разработана абстрактная модель OpenCL программы, реализована автонастройка параметров оптимизации для программы (WG – размер рабочей группы в программе на OpenCL, TS – размер массива данных, обрабатываемый потоком), написанной на OpenCL, с помощью метода проверки модели.

В предложенных участникам на ознакомление публикациях также проведено моделирование конвейера абстрактного GPU для параллельной программы поиска минимума в большом массиве. Для этого на языке Promela, входном языке верификатора SPIN, в отдельных процессах моделируются модули, соответствующие хосту, устройствам, узлам (т.е. элементы вычислительного конвейера) и программным элементам (отвечающим за модули программы). Процессы взаимодействуют друг с другом с помощью каналов, реализуя таким образом последовательность работы GPU конвейера при решении задачи. В качестве подбираемых параметров моделируется время для доступа к локальной и глобальной памяти, а также используются размеры рабочей группы и разбиения данных. Решение задачи автонастройки с помощью метода проверки модели состоит в том, что делается запрос верификатору на невозможность достижимости конечного состояния с заданными параметрами. Когда верификатор возвращает контрпример, нарушающий запрос, то в этом контрпримере содержатся оптимальные значения параметров.

Для детального моделирования GPU, более приближенного к реальности (с учетом представления программы в виде PTX инструкций, кэша таких инструкций, пулов потоков исполнения (warp, единица планирования планировщика GPU), сборщика операндов, дивергенции ветвей исполнения и т.д.), участникам предлагалось воспользоваться наработками проекта GPGU Sim [57], в котором на основе анализа открытых патентов Nvidia исследователи воссоздали последовательность работы GPU (исполнительный конвейер), который и предлагалось реализовать участникам.

В итоге, на соревнование были вынесены задачи на разных уровнях, от простой модификации решения авторов для другой задачи до детальной реализации конвейера:

- Первый уровень. Решить задачу поиска оптимальных параметров решения задачи суммы четных элементов в большом массиве данных, путем модификации авторского решения и абстрактной Promela-модели для OpenCL программы.
- Второй уровень. Дополнительно к уровню 1 модифицировать решение задачи настройки для OpenCL, добавив работу с потоками исполнения.
- Третий уровень. Реализовать уровень 2 для приложения на PTX, превращая исходную программу в последовательность инструкций, программа на языке Promela должна моделировать работу по выбору и исполнению этих инструкций.
- Четвертый уровень. Дополнительно к уровню 3 реализовать детально исполнительный конвейер для подготовки данных и потоков исполнения.

Поскольку задача предполагает большое число технологий для ознакомления, по ней была проведена обучающая лекция, в которой были рассмотрены следующие темы.

- Архитектура SIMT.
- Архитектуры современных GPU (Nvidia (Pascal, Ampere, Ada), AMD (Polaris)). Показана структура мультипроцессора внутри, исходя из архитектурных документов [58-59]).
- Языки программирования для GPU (OpenCL и CUDA).
- Массивный параллелизм в OpenCL-программах.
- Программирование с делением потоков на локальные и глобальные группы.
- Задачи автонастройки. Автонастройка OpenCL программ.
- Задачи поиска минимума на OpenCL.
- Проверка моделей. Абстрактная модель OpenCL программы.
- Моделирование сущностей абстрактного GPU при работе OpenCL программы на примере задачи поиска минимума.
- Контрпримеры при проверке моделей. Поиск оптимальных параметров при помощи контрпримеров.
- Проект GPGU Sim и основные сущности из конвейера GPU Nvidia.

## 5.4 Задача 4. Дедуктивная верификация программы, относящейся к задаче проверки выполнимости булевых формул (SAT)

Задача заключается в задании такого инварианта цикла для программы, решающей важную часть задачи 2-SAT, который позволяет успешно верифицировать данную программу в системе C-lightVer. Организатором в качестве задачи было выбрано задание инварианта цикла, так как это одна из главных проблем дедуктивной верификации программ [60-61]. Задача 2-SAT [62-63] была выбрана в качестве примера потому, что в последнее время активно развиваются исследования по формальной верификации программных систем для доказательства теорем [64], в том числе таких специализированных систем, как SAT-решатели и SMT-решатели [65-67]. В качестве системы верификации C-lightVer [26-29] была выбрана потому, что в системе C-lightVer применяется система доказательства теорем ACL2 [68], которая может автоматически применять уже доказанные теоремы в качестве лемм для доказательства других теорем, что упрощает доказательство условий корректности программ.

Участникам соревнования предоставлялись пособие по дедуктивной верификации, содержащее пример задания инварианта цикла, система дедуктивной верификации программ C-lightVer, условие задачи и верифицируемая программа с теорией предметной области.

Для понимания предназначения верифицируемой программы участникам нужно было изучить условие задачи, где относительно подробно излагается проблема выполнимости булевых формул в форме 2-КНФ, то есть задача 2-SAT. В условии задачи описано, как формула в форме 2-КНФ может быть представлена в виде конъюнкции импликаций и как можно представить такую конъюнкцию импликаций в виде графа импликаций. Далее в условии задачи определено важное для задачи 2-SAT свойство, которое вводится для некой переменной формулы x: наличие в графе импликаций пути из вершины x в вершину ¬x и одновременно наличие пути из вершины ¬x в вершину x. Затем в условии задачи описано, почему, если существует переменная формулы, для которой выполнено важное для задачи 2-SAT свойство, то исходная формула невыполнима [63]. Потом в условии задачи объяснено, почему, если для всех переменных формулы не выполнено важное для задачи 2-SAT свойство, то исходная формула выполнима [63]. Участникам соревнования нужно было верифицировать программу, которая по уже построенному транзитивному замыканию графа импликаций проверяет для каждой переменной формулы выполнение важного для задачи 2-SAT свойства. Итак, рассмотрим верифицируемую функцию twosat_solver (см. листинг 4).

```
/* Предусловие */
int twosat_solver(int variable_count,
                  int implication_graph_transitive_closure[]){
    int x = 0;
    int satisfiable = 1;
    /* Инвариант цикла */
    while (x < variable_count && satisfiable == 1){
        if (implication_graph_transitive_closure
                 [x + variable_count + x*2*variable_count] == 1 &&
            implication_graph_transitive_closure
                 [x * 2 * variable_count+x+variable_count
                    * 2 * variable_count]
            == 1) {satisfiable = 0;}
        else {x++;}}
    return satisfiable;}
/* Постусловие */
```

*Листинг 4. Функция “twosat_solver”.*
*Listing 4. “twosat_solver” function.*

Данная программа выполняет в цикле поиск переменной, для которой не выполняется важное для задачи 2-SAT условие. Если такая переменная найдена, то функция twosat_solver возвращает значение 0, означающее, что формула невыполнима. Иначе функция twosat_solver возвращает значение 1, означающее, что формула выполнима.

Отметим, что транзитивное замыкание матрицы смежности графа импликаций хранится не в двумерном, а в одномерном массиве, что довольно естественно для программ на языке C. Если интерпретировать транзитивное замыкание матрицы смежности графа импликаций как двумерный массив, то размерность такого массива будет 2 * variable_count на 2 * variable_count, где variable_count является количеством переменных в формуле. Индексы строк такого двумерного массива в диапазоне [0 ... variable_count-1] соответствуют переменным формулы $x_0$, ..., $x_{variable\_count-1}$ без отрицаний, а индексы в диапазоне [variable_count ... 2*variable_count-1] соответствуют переменным формулы $\neg x_0$, ..., $\neg x_{variable\_count-1}$ с отрицаниями. Аналогично индексы столбцов такого двумерного массива в диапазоне [0 ... variable_count-1] соответствуют переменным формулы $x_0$, ..., $x_{variable\_count-1}$ без отрицаний, а индексы в диапазоне [variable_count ... 2 * variable_count -1] соответствуют

переменным формулы $\neg x_0$, ..., $\neg x_{variable\_count-1}$ с отрицаниями. При переходе к одномерному массиву нужно использовать умножение индекса строки на длину строки и прибавить смещение на индекс столбца. Итого, проверка наличия пути из x в ¬x и проверка наличия пути из ¬x в x осуществляется с помощью проверки равенства единице элементов одномерного массива, индексы которых кодируются с помощью описанного способа.

Участникам соревнования было предоставлено уже заданное предусловие верифицируемой программы. Предусловие программы задано на языке Applicative Common Lisp [68], языке задания спецификаций в системе C-lightVer. Отметим, что предусловие играет свою роль при задании инварианта цикла. Предусловие накладывает следующие ограничения на входные данные программы (листинг 5).

```
(and (integerp variable_count) (< 0 variable_count)
        (integer-listp implication_graph_transitive_closure)
        (= (len implication_graph_transitive_closure)
           (* 4 (* variable_count variable_count))))
```

*Листинг 5. Предусловие функции “twosat_solver”.*
*Listing 5. Precondition of “twosat_solver” function.*

Участникам соревнования было также предоставлено постусловие программы (листинг 6).

```
(implies (= satisfiable 0)
         (not (twosat-solver (boolean-variable-values
                                    variable_count nil)
                variable_count
                implication_graph_transitive_closure)))
```

*Листинг 6. Постусловие функции “twosat_solver”.*
*Listing 6. Postcondition of “twosat_solver” function.*

Данное постусловие описывает свойство, что если переменная satisfiable по итогу выполнения программы равна 0, то не выполнен предикат twosat-solver из теории предметной области задачи. Участникам соревнования предоставлялся также файл twosat-solver.lisp, содержащий теорию предметной области задачи, заданную на языке Applicative Common Lisp. В данном файле определен используемый в постусловии предикат twosat-solver. Данный предикат проверяет, является ли формула выполнимой, с помощью перебора всех возможных наборов значений переменных, аналогично проверке выполнимости формулы на таблице истинности. Сама формула передается предикату twosat-solver в виде транзитивного замыкания матрицы смежности графа импликаций. Также в теории предметной области содержится теорема twosat-unsat о том, что если для какой-либо переменной формулы выполнено важное для задачи 2-SAT условие, то применение предиката twosat-solver к этой формуле ложно, то есть формула невыполнима. Для решения задачи полезно то, что система доказательства ACL2, используемая в системе C-lightVer, может автоматически применять данную терему для автоматизации доказательства условий корректности.

Отметим, что постусловие описывает только случай, когда формула невыполнима, и не описывает случай, когда формула выполнима. То есть, участникам нужно было доказать выполнение “частичного” постусловия. Для этого участникам было достаточно задать “частичный” инвариант цикла, который позволяет доказать выполнение “частичного” постусловия. Еще одна особенность постусловия состоит в том, что данное постусловие позволяет в случае невыполнимости формулы в некотором смысле проверить эквивалентность двух реализаций: полиномиальную [62-63] (но нетривиальную) реализацию задачи 2-SAT с помощью графа импликаций и неполиномиальную (но тривиальную) реализацию поверки выполнимости на таблице истинности. Такую возможность

дедуктивной верификации по доказательству эквивалентности (в определенном случае) было полезно в образовательных целях показать участникам соревнования.

Организатором соревнования ожидалось решение в виде инварианта, представляющего собой конъюнкцию трех частей: ограничения на переменные (фактические, копия предусловия), нужное для доказательства условие выхода из цикла, ограничение на переменную-счетчик цикла, позволяющее доказать и условие входа в цикл, и условие продолжения итерации, и условие выхода из цикла, и ограничение на переменную-результат, важное для доказательства условия выхода из цикла. Листинг 7 демонстрирует самую нетривиальную часть инварианта цикла в виде ограничения на переменную-результат.

```
(implies (= satisfiable 0)
         (and(=(nth(+(*x(*variable_count 2)) (+ x variable_count))
                implication_graph_transitive_closure) 1)
             (=(nth(+(*(+x variable_count) (*variable_count 2)) x)
                implication_graph_transitive_closure) 1)
         (< x variable_count))))
```

*Листинг 7. Часть инварианта цикла, являющаяся ограничением на переменную "satisfiable".*
*Listing 7. Loop invariant part that is constraint on "satisfiable" variable.*

Данное ограничение описывает, что если переменная satisfiable равна нулю, то для i-той переменной формулы не выполняется важное для задачи 2-SAT условие. Данное ограничение позволяет доказать выполнение постусловия при завершении цикла с помощью применения в качестве леммы теоремы twosat-unsat из теории предметной области. Это позволяет провести автоматическую успешную верификацию решения в системе C-lightVer.

## 5.5 Задача 5. Построение и проверка модели протокола консенсуса IBFT

В задаче участникам предлагалось построить формальную модель протокола консенсуса IBFT и проверить её свойства в любом удобном средстве проверки модели. Организаторами было рекомендовано использование языка TLA+ и TLC Model Checker.

Протокол консенсуса – это алгоритм, решающий проблему достижения консенсуса в распределенных системах, когда все корректные процессы должны принять решение о некотором общем предлагаемом значении. Алгоритм IBFT (Istanbul Byzantine Fault Tolerance) [69] вдохновлен алгоритмом PBFT (Practical Byzantine Fault Tolerance) [70] и также решает проблему достижения консенсуса в распределенных системах. Ключевая особенность такого рода протоколов консенсуса заключается в их устойчивости к византийским ошибкам, предполагающим наличие вредоносных узлов, которые могут функционировать произвольно или умышленно вредить сети, передавая противоречивую или ложную информацию. Как и в случае с другими BFT-алгоритмами, необходимым условием достижения консенсуса в распределенной системе с N византийскими узлами, реализующей алгоритм IBFT, является общее число узлов, превышающее 3N [71].

Для алгоритма консенсуса обычно необходимо проверить 3 ключевых свойства:

- Согласованность: все корректно работающие узлы принимают одинаковое значение.
- Корректность: принятое значение было предложено одним из имеющихся узлов.
- Завершаемость: все корректно работающие узлы достигают определённого значения.

Выполнение этих свойств для протокола IBFT и предлагалось проверить участникам соревнования VeHa-2024. Реализовать модель алгоритма участники могли на основе псевдокода или описания алгоритма на естественном языке, предложенном в статье [69].

## *6. Обзор решений и ошибки, допущенные участниками*

Приведем некоторые комментарии экспертов по ошибкам в решениях участников. Все решения можно найти в репозитории соревнования VeHa-2024 [23].

**По задаче 1:**

В решении [72] вместо описания behaviors (описание всех возможных сценариев поведения функции, требующее доказательства полноты) участники использовали описание ensures (простое описание постусловий), и вот почему: они выявили не все возможные случаи как в функции compute_mask, так и в функции check_permission – например, не доказывается постусловие (см. листинг 8) и ряд других случаев. (Всего было пропущено около 5 сценариев в базовой функции и 2 – в функции compute_mask).

```
ensures getSecurityLabel(current) !=\null && isMaxIlev(current) &&
                  currentIsFileSystemRoot && !isPublic(file) &&
                  (cmd == ORDINARY || cmd == RARE)
   ==>  \result == -NO_PERM
```

*Листинг 8. Постусловие программы.*
*Listing 8. Postcondition of the program.*

В решении [73] выявлены следующие ошибки:

1. Один из инвариантов цикла задан неполно – поэтому не доказывается условие Termination (условие окончания цикла);
2. Указаны не все возможные инварианты – в частности, нет инварианта с двумя переменными;
3. Код, исправляющий ошибку и заключающийся в проверке того, что метка безопасности – пустая, вставлен не в то место, где нужно (студенты добавили его после вызова функции, получающей данную метку, а надо было до). Поэтому совершенно справедливо не доказывается корректность вызова одной из подфункций, а именно slabel_ilev.

**По задаче 2:**

Из девяти зарегистрировавшихся участников на проверку поступили решения от пяти. Команды Aleksei_Volkov, Gleb, IvanSmirnov, orenty7 выполнили все задания. Команда 1 выполнила задания 1 и 4, но в задании 2 была допущена досадная ошибка – была определена не та структура решетки для целых чисел, эта оплошность не позволила закончить задание 3.

**По задаче 3:**

Решение [74]:

- Уровень 1. Хорошая организация решения, задача решается, код вычисления суммы похож на оригинальное OpenCL решение, с другой стороны, вставки нового кода немного чужеродные для авторского кода по названию переменных, пробелам и т.д. Модель работает, представленные результаты в отчете подтверждаются, из ошибок – не промоделирован барьер, решено синхронизацией между процессами через каналы.
- Уровень 2. Сделано сильно просто для заданной постановки задачи, не до конца разобрались с потоками исполнения (warp). Модель работает. Планировщик, распределяющий по потокам исполнения, не промоделирован.

Решение [75]:

- Уровень 1. Разобрались с кодом, предложили сделать исправление, но решение задачи так и не увиделось. Модель работает. Не удалось получить результаты, представленные в отчете. Нет моделирования задачи из задания (которая была на OpenCL).
- Уровень 2. Идеи с потоками исполнения достойные, но есть ошибки в реализации. Сделана попытка промоделировать планировщик потоков исполнения. Не доделано, исходная задача не промоделирована.
- Уровень 3. Очень много, конечно, разобрали по ветвлениям и маске, но невозможно согласиться с предложением плодить множество процессов в программах на языке Promela, дорогих для верификации.

**По задаче 4:**

Результаты решения данной задачи, как и решения всех остальных задач, доступны на сайте соревнования [23]. Из 5-ти зарегистрировавшихся индивидуальных участников и 6-ти зарегистрировавшихся команд решения сдали 3 индивидуальных участника и 2 команды. Все сдавшие решения отдельные участники и команды успешно освоили сложную предметную область дедуктивной верификации, а также задачи 2-SAT, и сдали решения, очень близкие к ожидаемому организатором решению. Поэтому все сдавшие решения участники и команды получили за свои решения полные 10 баллов. В комментариях в форме обратной связи по итогам соревнования участники отметили, что этому поспособствовало подготовленное организатором пособие, содержащее пример по заданию инварианта цикла. Кроме того, что участники и команды успешно изучили пособие, необходимо отметить, что один из участников ради обучения дедуктивной верификации еще до старта соревнования пытался верифицировать программы с циклами над массивами. Поэтому данный участник был отмечен почетной грамотой за волю к победе. Еще необходимо отметить принимавшего участие в соревновании известного специалиста по формальным методам в программировании. Данный специалист сдал вместе с решением полезное тестовое описание решения и своего мнения о задаче, за что был отмечен благодарственным письмом организаторов.

## *7. Результаты и обратная связь*

По задаче 1 были объявлены 4 победителя (Дипломы 1 и 2 степени, благодарность, почетная грамота за освоение системы верификации Frama-C).

По задаче 2 были объявлены 5 победителей (4 диплома 1 степени и один диплом 2 степени), а также грамоты (за самое быстрое решение, за внимание к корректности функциональной спецификации, за самое короткое решение и использование автоматики, за выделение вспомогательных утверждений).

По задаче 3 были объявлены 2 победителя (две команды с дипломами за 1 и 2 место) + дополнительная почетная грамота “За найденную ошибку в исходном коде организаторов”.

По задаче 4 были объявлены 5 победителей: 4 победителя с дипломом 1 степени (команды и индивидуальные участники), одно благодарственное письмо за лучшее решение задач соревнования VeHa-2024 и, кроме этого, дополнительная почетная грамота за волю к победе.

По задаче 5 не было подано решений на проверку, соответственно, победители не были определены.

В итоге было объявлено 16 победителей (как индивидуальных участников, так и команд). Один из участников получил дипломы в трех номинациях. На рис. 4 представлен процесс вручения грамот и дипломов победителям и призерам.


*Рис. 4. Очное вручение грамот и дипломов победителям и призерам соревнования VeHa-2024.*
*Fig. 4. In-person presentation of winner's certificates and certificates of honor to winners and prize-winners of VeHa-2024 contest.*

По сравнению с прошлым соревнованием, резко выросло число регистраций, некоторые участники пробовали себя в решении разных задач, и даже меняли предварительные предпочтения, поскольку не могли решить заранее выбранную задачу. С другой стороны, решений было получено не так много, скорее всего, большинство участников просто прицеливалось и осваивало инструменты формальной верификации, поскольку обратной связи, почему они не отправили решение, получено не было. Еще одно важное отличие, что по сравнению с прошлым соревнованием, где в основном задания были достаточно абстрактные, в этот раз они стали более практическими, и решения таких задач с помощью формальных методов участвующие в соревнованиях были в состоянии сделать, а не просто усвоить предложенные инструменты. Также была выявлена проблема с использованием общего репозитория на GitHub для создания предложения исправления (pull-request) с решениями участников, заключающаяся в том, что некоторые участники отслеживали чужие решения и копировали их части. Возможно, в будущем необходимо создание выделенной системы отправки решений.

После проведения соревнования была запущена форма обратной связи, которую заполнили 6 человек. Результаты опроса показаны на рис. 5 – рис. 8.

## *8. Заключение*

Мы считаем, что соревнование VeHa-2024 прошло успешно, с бóльшим вовлечением участников, чем на VeHa-2023. Решение предложить участникам несколько разнородных задач на выбор и привлечение компаний к формированию заданий подняло состязание на следующий уровень.

Индустриальные задачи позволили продемонстрировать применимость формальных методов, разобраться с классом их возможных применений, а также показать участникам

потенциальные перспективные задачи для будущей работы в той области, о которой, возможно, они не предполагали ранее. По результатам состязания победителям была предложена стажировка в одной из компаний, организаторов соревнования.

При подготовке такого рода соревнований образовательная составляющая играет немаловажное значение. В вузах формальные методы совмещаются либо с курсами по тестированию программного обеспечения [76], либо преподаются как спецкурсы или курсы по выбору, что недостаточно большой аудитории программистского сообщества. При проведении соревнований возможно улучшить компетенции разработчиков, тех, кому это действительно необходимо, в форме лекций и обучающих материалов, а также продемонстрировать свои разработки в виде средств формальной верификации заинтересованной аудитории.

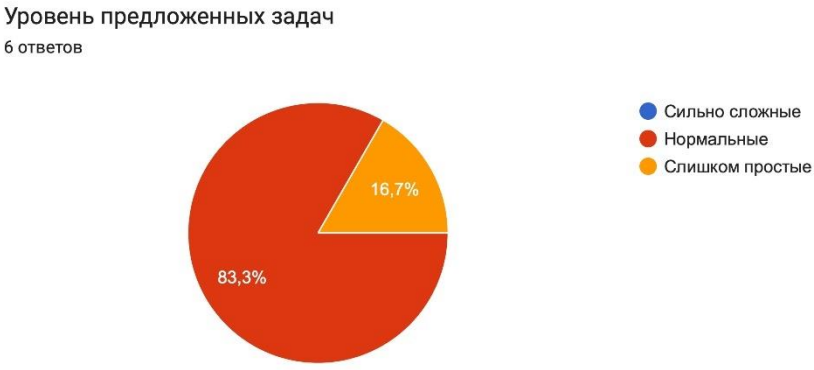


*Рис. 5. Распределение отзывов по уровню предложенных задач.*
*Fig. 5. Distribution of survey responses by proposed tasks levels.*

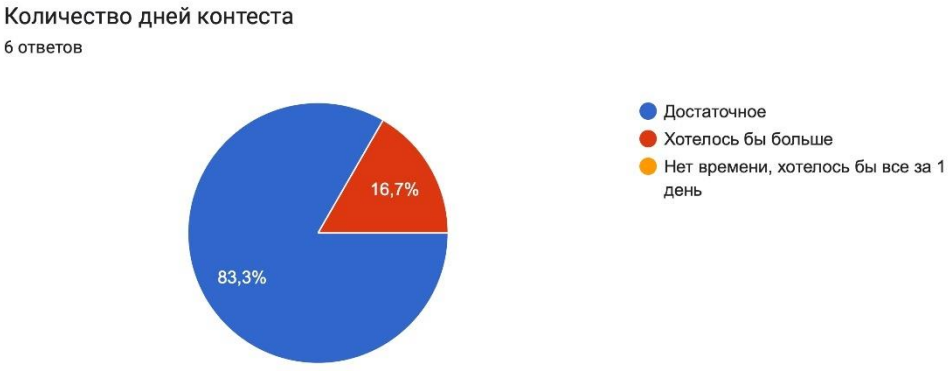


*Рис. 6. Распределение отзывов по количеству дней соревнования.*
*Fig. 6. Distribution of survey responses by quantity of contest days.*

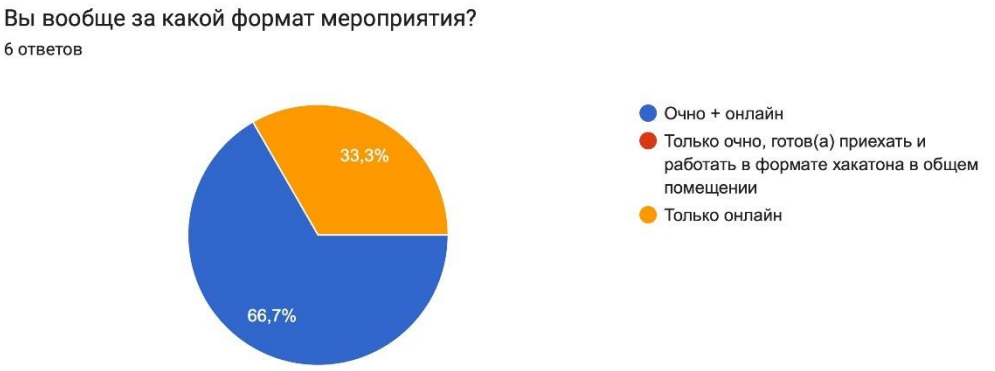


*Рис. 7. Распределение отзывов на формат мероприятия.*
*Fig. 7. Distribution of survey responses by event format.*

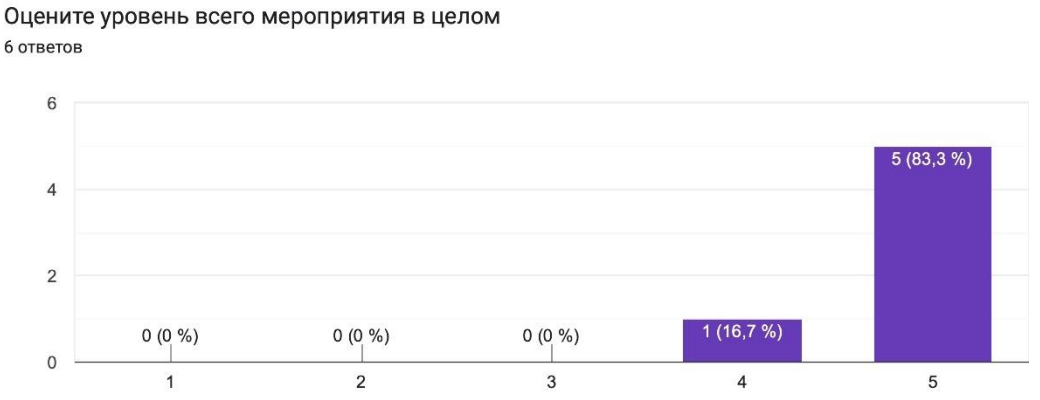


*Рис. 8. Распределение оценок мероприятия.*
*Fig. 8. Distribution of survey responses by event ratings.*

## Список литературы / References

[1]. Старолетов С.М., Кондратьев Д.А., Гаранина Н.О., Шошмина И.В. Соревнования по формальной верификации VeHa-2023: опыт проведения // Труды Института системного программирования РАН. 2024, т. 36, № 2, с. 141-168.

[2]. 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.

[3]. 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.

[4]. TOOLympics Challenge 2023 / Ed. by D. Beyer, A. Hartmanns, F. Kordon. – Cham: Springer, 2025. – 172 p.

[5]. 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.

[6]. 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.
[7]. Denis X., Siegel S.F. VerifyThis 2023: An International Program Verification Competition. In TOOLympics Challenge 2023. TOOLympics 2024. Ser. LNCS. Springer, 2025, vol. 14550, pp. 147-159.
[8]. Jasper M., Mues M., Murtovi A., Schlüter 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.
[9]. 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.
[10]. 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.
[11]. 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.
[12]. Ernst G., Herber P., Huisman M., Ulbrich M. SpecifyThis Bridging Gaps Between Program Specification Paradigms: Track Introduction. In Proc. Leveraging Applications of Formal Methods, Verification and Validation. Specification and Verification. Ser. LNCS. Springer, 2025, vol. 15221, pp. 3-7.
[13]. 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.
[14]. 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.
[15]. 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.
[16]. 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.
[17]. 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.
[18]. 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.
[19]. VerifyThis Archive, 2024. Available at: https://www.pm.inf.ethz.ch/research/verifythis/Archive.html, accessed Nov 06, 2024.
[20]. VeHa contest, 2023. Available at: https://sites.google.com/view/veha23/, accessed Jul 06, 2024.
[21]. VeHa2023 on GitHub. Available at: https://github.com/VeHaContest/VeHa2023, accessed Jul 06, 2024.
[22]. VeHa contest, 2024. Available at: https://sites.google.com/view/veha2024, accessed Nov 06, 2024.
[23]. VeHa2024 on GitHub. Available at: https://github.com/VeHaContest/VeHa2024, accessed Jul 06, 2024
[24]. NSU profile "Formal methods of program and system analysis". Available at https://education.nsu.ru/formal-analysis-methods/, accessed Dec 20, 2024.
[25]. Khazeev M., Aslam H., de Carvalho D., Mazzara M., Bruel JM., Brown J.A. Reflections on Teaching Formal Methods for Software Development in Higher Education. In: Proc. Frontiers in Software Engineering Education, ser. LNCS. Springer, 2020, vol 12271, pp. 28–41.
[26]. 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.
[27]. 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.
[28]. 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.
[29]. 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.

[30]. Khazeev M., Mazzara M., Aslam H., de Carvalho D. Towards a Broader Acceptance of Formal Verification Tools. In: Proc. Impact of the 4th Industrial Revolution on Engineering Education, ser. Advances in Intelligent Systems and Computing. Springer, 2020, vol. 1135, pp. 188–200.

[31]. 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.

[32]. 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.

[33]. 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.

[34]. Huisman M., Klebanov V., Monahan R. VerifyThis 2012. International Journal on Software Tools for Technology Transfer, vol. 17, no. 6, pp. 647–657, 2015.

[35]. 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.

[36]. 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.

[37]. 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.

[38]. Ahrendt W., Ernst G., Herber P., Huisman M., Monti R.E., Ulbrich M., Weigl A. The VerifyThis Collaborative Long-Term Challenge Series. In TOOLympics Challenge 2023. TOOLympics 2024. Ser. LNCS. Springer, 2025, vol. 14550, pp. 160-170.

[39]. 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.

[40]. 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.

[41]. 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.

[42]. Paul M., Haslbeck L., Wimmer S. Competitive Proving for Fun. In Selected Student Contributions and Workshop Papers of LuxLogAI 2018, vol. 10, pp. 9-14.

[43]. Paulin-Mohring C. Introduction to the Coq Proof-Assistant for Practical Software Verification. In Proc. Tools for Practical Software Verification. Ser. LNCS. Springer, 2012, vol. 7682, pp. 45-95.

[44]. 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.

[45]. 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.

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

[47]. 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.

[48]. 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.

[49]. The International Collegiate Programming Contest, ICPC. Available at https://icpc.global/, accessed Dec 21, 2024.

[50]. Chas K. The World's Smartest Programmers Compete: ACM ICPC. Communications of the ACM. Jul 2 2013. Available at https://cacm.acm.org/blogs/blog-cacm/165692-the-worlds-smartest-programmers-compete-acm-icpc/, accessed Dec 21, 2024.

[51]. Slamecka V.(editor) Proceedings of the 5th annual ACM computer science conference, CSC 1977, Atlanta, Georgia, USA, January 31 – February 2, 1977. ACM 1977,

https://doi.org/10.1145/800008.808038.
[52]. Workshop "Program Semantics, Specification and Verification: Theory and Applications". Available at https://persons.iis.nsk.su/ru/PSSV-2024, accessed Dec 20, 2024.
[53]. Baudin P., Filliâtre J. C., Marché C., Monate B., Moy Y., & Prevosto V. (2008). Acsl: Ansi c specification language. CEA-LIST, Saclay, France, Tech. Rep. v1, 2.
[54]. Smalley S., Vance C., & Salamon W. (2001). Implementing SELinux as a Linux security module. NAI Labs Report, 1(43), 139.
[55]. Garanina N., Staroletov S., Gorlatch S. (2022, September). Model Checking Meets Auto-Tuning of High-Performance Programs. In International Symposium on Logic-Based Program Synthesis and Transformation (pp. 63-82). Cham: Springer International Publishing.
[56]. Garanina N., Staroletov S., Gorlatch S. Auto-Tuning High-Performance Programs Using Model Checking in Promela //arXiv preprint arXiv:2305.09130. – 2023.
[57]. GPGU Sim, manual. Revision 1.2 (GPGPU-Sim 3.1.1). Ed.: Tor M. Aamodt, Wilson W.L. Fung, Tayler H.Hetherington, http://gpgpu-sim.org/manual/index.php/Main_Page# Fetch_and_Decode_Software_Model, accessed Nov 06, 2024.
[58]. Lindholm E., Nickolls J., Oberman S., & Montrym J. (2008). NVIDIA Tesla: A unified graphics and computing architecture. IEEE micro, 28(2), 39-55.
[59]. Whitepaper “NVIDIA Tesla P100”. https://www.techpowerup.com/gpu-specs/docs/nvidia-gp100-architecture.pdf, accessed Nov 06, 2024.
[60]. Furia C. A., Meyer B., Velder S. Loop invariants: Analysis, classification, and examples. ACM Computing Surveys, vol. 46, no. 3, pp. 1–51, 2014.
[61]. 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.
[62]. Krom M.R. The Decision Problem for a Class of First-Order Formulas in Which all Disjunctions are Binary. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, vol. 13, no. 1-2, pp. 15-20, 1967.
[63]. Aspvall B., Plass M.F., Tarjan R.E. A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters, vol. 8, no. 3, pp. 121-123, 1979.
[64]. Myreen M.O., Davis J. The Reflective Milawa Theorem Prover Is Sound. In Proc. Interactive Theorem Proving. Ser. LNCS. Springer, 2014, vol. 8558, pp. 421-436.
[65]. Blanchette J.C., Fleury M., Lammich P, Weidenbach C. A Verified SAT Solver Framework with Learn, Forget, Restart, and Incrementality. Journal of Automated Reasoning, vol. 61, no. 1, pp. 333–365, 2018.
[66]. Fleury M., Blanchette J.C., Lammich P. A verified SAT solver with watched literals using imperative HOL. In Proc. of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, 2018, pp. 158–171.
[67]. Ciobâcă Ș., Andrici C.-C. A Verified Implementation of the DPLL Algorithm in Dafny. Mathematics, vol. 10, no. 13, article id: 2264, 2022.
[68]. Moore J. S. Milestones from the pure Lisp theorem prover to ACL2. Formal Aspects of Computing, vol. 31, no. 6, pp. 699–732, 2019.
[69]. Moniz H. (2020). The Istanbul BFT consensus algorithm. arXiv preprint arXiv:2002.03613.
[70]. Castro M., Liskov B. Practical Byzantine Fault Tolerance. Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999.
[71]. Pease M., Shostak R., Lamport L. Reaching Agreement in the Presence of Faults. Association for Computing Machinery, vol. 27, no. 2, 1980.
[72]. https://github.com/VeHaContest/VeHa2024/tree/main/solution_1_re_tofl, accessed Nov 06, 2024.
[73]. https://github.com/VeHaContest/VeHa2024/tree/main/solution_1_DevTools_Itmo, accessed Nov 06, 2024.
[74]. https://github.com/VeHaContest/VeHa2024/tree/main/solution_3_RARe, accessed Nov 06, 2024.
[75]. https://github.com/VeHaContest/VeHa2024/tree/main/solution_3_power_o%60_nine, accessed Nov 06, 2024.
[76]. Staroletov S. Teaching the discipline “Software Testing and Verification” to future programmers. System Informatics. vol. 21, pp. 1-28, 2022.

## *Информация об авторах / Information about authors*

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.

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.

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.

Анастасия Владимировна КРАСНЕНКОВА – специалист по анализу безопасности РусБИТех-Астра. Сфера научных интересов: формальная верификация, дедуктивная верификация, программирование, анализ безопасности, математическая логика, системы распределённого реестра.

Anastasiya Vladimirovna KRASNENKOVA – security analyst at RusBITech-Astra. Research interests: formal verification, deductive verification, programming, security analysis, mathematical logic, distributed ledger systems.

Кирилл Викторович ЗИБОРОВ – инженер Positive Technologies, аспирант МГУ им. М. В. Ломоносова. Сфера научных интересов: формальная верификация, проверка моделей, распределенные системы и алгоритмы, системы распределённого реестра.

Kirill Viktorovich ZIBOROV – formal verification engineer at Positive Technologies, postgraduate student at Lomonosov Moscow State University. Research interests: formal verification, model checking, distributed systems and algorithms, distributed ledger systems.

Николай Вячеславович ШИЛОВ – кандидат физико-математических наук, доцент (ВАК), руководитель Лаборатории программной инженерии Университета Иннополис. Сфера научных интересов: прикладная логика, основания математики и программирования, преподавание фундаментальной математики и теории программирования

Nikolay Vyacheslavovich SHILOV – Cand. Sci. (Phys.-Math.), associate professor, head of the Laboratory of Software and Service Engineering of Innopolis University. Research interests: applied logic, foundations of Mathematics and Programming, Mathematical and Theory of Programming education and teaching

Наталья Олеговна ГАРАНИНА – кандидат физико-математических наук, старший научный сотрудник ИСИ СО РАН, старший научный сотрудник ИАиЭ СО РАН, ведущий научный сотрудник ООО ОКП «АРС», доцент НГУ. Сфера научных интересов: формальная верификация, проверка моделей, системы реального времени, 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.

Тимофей Юрьевич ЧЕРГАНОВ – старший специалист по анализу безопасности РусБИТех-Астра. Сфера научных интересов: формальная верификация, компьютерная безопасность.

Timofey Yuryevich CHERGANOV – senior security analyst at RusBITech-Astra. Research interests: formal verification, computer security.