# Соревнование по формальной верификации программ VeHa-2024: два года большого пути

Кондратьев Дмитрий Александрович

Институт систем информатики им. А. П. Ершова СО РАН
Новосибирский Государственный Университет

# Контест VeHa-2023

Два соревнования:

- Соревнование по дедуктивной верификации.
- Соревнование по model checking.

Тематика данных соревнований покрывает два главных направления формальной верификации программ:

- Дедуктивная верификация.
- Model checking.

Рассмотрим соревнование по дедуктивной верификации программ.

Ближайший аналог – соревнование VerifyThis

# Model checking (проверка на модели)

Проверка на модели позволяет проверять выполнение свойств на модели, описывающей состояния программной системы и переходы между ними.

Спецификации часто представляют собой LTL-формулы, описывающие темпоральные свойства системы.

# Дедуктивная верификация программ

Дедуктивная верификация применяется для проверки корректности программы относительно формальных спецификаций, описывающих результат или процесс ее исполнения.

Виды формальных спецификаций:

- Предусловие программы
- Инварианты циклов
- Постусловие программы

Предусловие, описывает ограничения на входные данные программы.

Постусловие описывает ограничения на выходные данные программы.

Инвариантом цикла является утверждение, которое истинно перед исполнением цикла и для каждой итерации цикла и обеспечивает корректность на выходе из цикла.

# Дедуктивная верификация

Дедуктивная верификация позволяет свести задачу проверки корректности программ к задаче доказательства специальных формул

Входные данные:

Программа, аннотированная спецификациями:

- Предусловие
- Постусловие
- Инварианты циклов

Неформальная постановка задачи:

Доказать, что "программа делает то, что записано в ее спецификациях".

# Теория предметной области

Так как логические формулы задаются в определенной теории, то и спецификации задаются в определенной теории.

Теория, в которой заданы спецификации программ, называется теорией предметной области.

Теория предметной области содержит определения функций, применяемых в спецификациях.

Также теория предметной области может содержать теоремы о функциях, применяемых в спецификациях.

# Соревнование VeHa-2023

## VeHa-2023

- Первое соревнование в России
- 2 – 4 ноября 2023 года
  - в рамках *Семинара по семантике, спецификации и верификации программ* (PSSV-2023)
- Применение формальных методов для проверки корректности программ и систем
  - Две задачи для решения
    - 1 – для решения методом дедуктивной верификации
    - 1 из 3 – для решения методом верификации моделей
  - Команда от 1 до 3 человек
- Результат
  - Популяризация формальных методов
  - Дипломы, грамоты, сертификаты участников, денежные призы и подарки


# Соревнование VeHa-2023

## VeHa-2023

- Ресурсы общие
  - Сайт
  - Телеграм-чат
  - Skype-канал
- Ресурсы организаторов
  - Телеграм-чат
  - Google Meet
  - Google Table
- Организаторы
  - Дедуктивная верификация
    - Дмитрий Кондратьев (ИСИ, Новосибирск)
  - Верификация моделей
    - Ирина Шошмина (Политех, Санкт-Петербург)
    - Сергей Старолетов (АлтГУ, Барнаул)
    - Наталья Гаранина (ИСИ, Новосибирск)




# Соревнование VeHa-2023

## VeHa-2023: участники

- Зарегистрировано
  - 48 участников
    - 13 команд > 1
    - 12 команд = 1
- Дожили до финала (сдали работы)
  - 34 участника
    - 10 команд > 1
    - 4 команды = 1
- География
  - Санкт-Петербург – 23
    - Политех, СПбГУ
  - Иннополис – 10
  - Москва – 3
    - ВШЭ, Астра Линукс
  - Новосибирск
    - НГУ – 3

# Соревнование VeHa-2023

## VeHa-2023: расписание

- 2 ноября – тьюториалы
  - Дедуктивная верификация и инструмент C-LightVer (Skype, онлайн, запись)
  - Язык Promela верификатора SPIN (Skype, онлайн, запись)
  - Практические аспекты верификатора SPIN (Skype, онлайн, *офлайн*, запись)
- 3 ноября
  - 10:00 (Нск) – публикация задач на сайте VeHa
- 3-4 ноября
  - Консультации в телеграм-чате и Skype-канале соревнования
- 5 ноября
  - 10:00 (Нск) – загрузка решений задач на github
- 5 ноября
  - Проверка решений
- 6 ноября
  - 14:00 (Нск) – публикация результатов

# Соревнование VeHa-2023

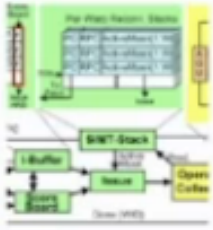
## VeHa-2023: задачи по верификации моделей

Задача о станции "Луна-25"

- Цель: на языке Promela формализовать взаимодействие модулей станции и найти сценарий, приводящий к ошибке, описанной в СМИ
- Задачу предложили: Гаранина, Шошмина

Моделирование pipeline GPGPU

- Цель: Описать pipeline процессов, происходящих в GPU при решении задачи на SIMT-ядрах согласно известным публикациям
- Задачу предложили: Старолетов, Гаранина



Описание протокола телеграмм для европоездов

- Цель: Реализовать алгоритм работы с бинарными сообщениями согласно официальному документу и проверить корректность кодирования/декодирования
- Задачу предложил: Старолетов

# Соревнование VeHa-2023

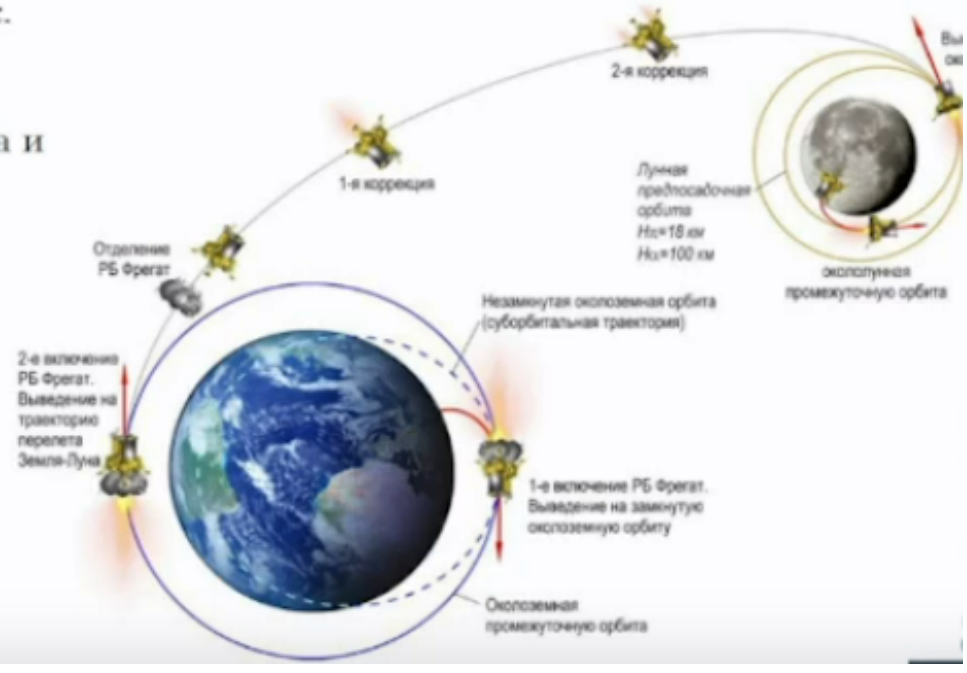
## VeHa-2023: Задача о станции "Луна-25"

- Уровень: начальный
- Цель: на языке Promela формализовать взаимодействие модулей станции и найти сценарий, приводящий к ошибке, описанной в СМИ
- Задачу предложили: Гаранина, Шошмина

# Соревнование VeHa-2023

## VeHa-2023: Задача о станции "Луна-25"

- **Луна 25, Роскосмос** (19.08.2023)
- Модуль не смог выйти на промежуточную окололунную орбиту, промахнулся, т.к. излишне ускорился, т.к
  - двигатель аппарата не отключился штатно во время последнего маневра и проработал 127 секунд вместо 84.
    - Отсутствие сигнала датчика скоростей.



# Соревнование VeHa-2023

## VeHa-2023: Задача о станции "Луна-25"

- **Луна 25, Роскосмос** (19.08.2023)
- Модуль не смог выйти на промежуточную окололунную орбиту, промахнулся, т.к. излишне ускорился, т.к
  - двигатель аппарата не отключился штатно во время последнего маневра и проработал 127 секунд вместо 84
    - Отсутствие сигнала датчика скоростей
      - *Система блока измерения угловых скоростей* БИУСа-Л ждала команду от *бортового комплекса управления* БКУ на включение акселерометров
        - Команда была подана, но анализ ответной реакции не был запрограммирован
          - Отсутствие информации о том, что БИУС-Л измеряет скорость
        - БИУС-Л не получил команды
          - не включился в правильный режим
            - не передавал информацию в БКУ
              - БКУ не передал сигнал двигателю на своевременное отключение.
          - поместил команду в неправильный массив команд

# Соревнование VeHa-2023

## VeHa-2023: Задача о станции "Луна-25"

Требуется:

1. Спроектировать взаимодействие компонент, чтобы оно приводило к катастрофе в результате неправильного расположения в пуле данных БИУС-Л команды рапорта значений скорости аппарата. Доказать, что катастрофа может происходить.
2. Изменить проект так, чтобы избежать катастрофы. Доказать, что катастрофа не может происходить.

Значимые для моделирования компоненты «Луны-25»:

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

# Соревнование VeHa-2023

## VeHa-2023: Задача о станции "Луна-25"

- Бортовой комплекс управления (БКУ)
  - управление всеми компонентами станции
- БИУС-Л
  - Датчики не являются составляющей БИУС-Л
    - БИУС-Л получает информацию от датчиков напрямую
- Двигатель
  - перемещение аппарата
- Другие модули
  - Передают и принимают данные
- Среда коммуникации
  - шина передачи команд
  - шина передачи данных

# Соревнование VeHa-2023

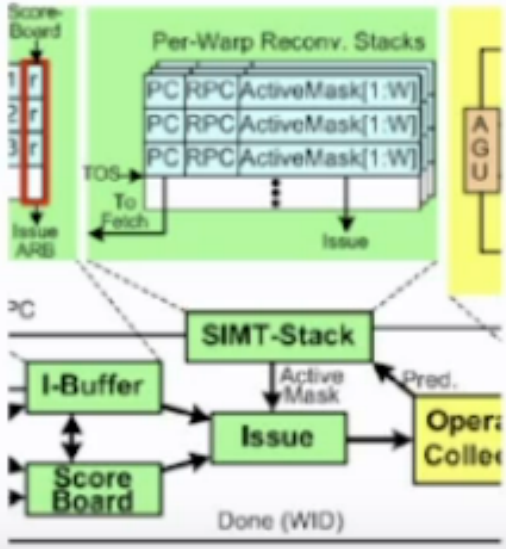
## VeHa-2023: Задача о станции "Луна-25"

- Нужно
  - смоделировать
    - систему на Promela и требования в LTL
  - верифицировать корректность модели относительно требований в инструменте SPIN
- Решения
  - Модель
    - у трети участников модель совсем не получилась (взаимные блокировки)
    - три очень тяжёлых модели (много лишних сущностей)
    - интересные реализации массивов команд, среды коммуникации и вариантов поломок
  - Требования
    - почти все решения описывали требования в терминах реализации модели
    - только два решения сформулировало требование наиболее общим образом
      - *катастрофа не произойдёт (переход на эллиптическую орбиту завершён)*
  - Верификация
    - единственное решение
      - исходная модель некорректна относительно требований безопасности
      - починенная модель корректна относительно требований безопасности

# Соревнование VeHa-2023

## VeHa-2023: Моделирование pipeline GPGPU

- Уровень: сложный
- Цель: Описать pipeline процессов, происходящих в GPU при решении задачи на SIMT-ядрах согласно известным публикациям
- Задачу предложили: Старолетов, Гаранина

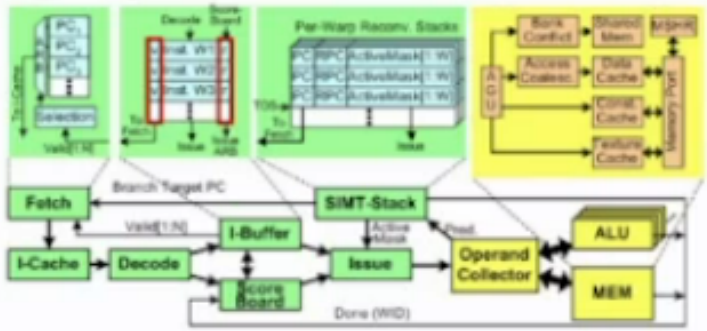


# Соревнование VeHa-2023

## VeHa-2023: задачи по верификации моделей

Моделирование pipeline GPGPU

- Создание реализации решения простой па задачи, заданной в виде последовательности инструкций вроде приведенных PTX для OpenCL программы.
- Пайплайны должен быть примерно соответствовать основным идеям из GPGPU Sim.




- Все сущности должны быть описаны как параллельные процессы Promela, взаимодействующие через каналы (принятие и отправка сообщений).
- Должно быть несколько экземпляров сущностей (работа параллельно) для эмуляции SM и планировщиков внутри них.
- Решение должно быть написано с предоставлением UML диаграмм деятельности по каждому компоненту.

# Соревнование VeHa-2023

## VeHa-2023: Описание протокола телеграмм для европоездов

- Уровень: сложный
- Цель: Реализовать алгоритм работы с бинарными сообщениями согласно официальному документу и проверить корректность кодирования/декодирования
- Задачу предложил: Старолетов

# Соревнование VeHa-2023

## VeHa-2023: Описание протокола телеграмм для европоездов

- Ознакомление с требованиями к телеграммам для бализы (передатчика) из официального стандарта.
- Ознакомление с методами работы с бинарными полиномами для вычисления CRC.
- Реализация на Promela побитовых операций работы с частями телеграмм в соответствии со стандартом с использованием библиотеки в виде макросов для полиномов.
- Реализация скремблинга по методу Galuas LFSR.
- Реализация кодирования и проверки получившихся диаграмм (может быть, сначала на C, потом переписыванием на Promela?).
- Реализация декодирования.
- Реализация верификации кодирования и декодирования путем случайном подмены битов в сообщении.

# Организатор соревнования по дедуктивной верификации в рамках контеста VeHa-2023

Кондратьев Дмитрий Александрович, к.ф.-м.н., научный сотрудник Института систем информатики им. А.П. Ершова СО РАН.

Обратная связь: письма на электронную почту apple-66@mail.ru

# Задача по дедуктивной верификации программы, предназначенной для решения такой проблемы миссии "Луна-25", как проверка равенства всех приоритетов в массиве команд

Цель задачи состоит в том, чтобы задать формальные спецификации программы, предназначенной для решения такой проблемы миссии "Луна-25", как проверка равенства всех приоритетов в массиве команд, и провести автоматическую дедуктивную верификацию данной программы в системе C-lightVer.

# Задача по дедуктивной верификации программы, предназначенной для решения такой проблемы миссии "Луна-25", как проверка равенства всех приоритетов в массиве команд

Рассматриваемая задача является первым приближением к решению проблемы миссии "Луна-25" и состоит в верификации программы, проверяющей, все ли элементы массива равны друг другу.

То есть, это абстракция к проблеме миссии "Луна-25", которая состоит в том, что в массив команд с одинаковым приоритетом попала команда, приоритет которой отличается от всех остальных.

Проверка, есть ли в массиве хотя бы один элемент, отличающийся от остальных, могла бы позволить избежать данной проблемы.

# Задача по дедуктивной верификации программы, предназначенной для решения такой проблемы миссии "Луна-25", как проверка равенства всех приоритетов в массиве команд

Задача состоит в том, чтобы для программы, проверяющей, все ли элементы массива равны, задать постусловие и теорию предметной области с определением применяемой в постусловии функции.

Необходимо, чтобы заданное постусловие и теория предметной области с определением функции, применяемой в постусловии, позволили дедуктивно верифицировать данную программу в системе C-lightVer.

# Система дедуктивной верификации программ

Входные данные системы дедуктивной верификации:

- Программа
- Формальные спецификации программы
- Теория предметной области с определениями функций, применяемых в спецификациях программы, и с теоремами о таких функциях.

Система дедуктивной верификации порождает условия корректности программ. Это логические формулы, которые описывают, что программа корректна относительно своих спецификаций.

Истинность условий корректности в теории предметной области проверяют программные системы для доказательства теорем.

Если все условия корректности истинны, то программа корректна относительно своих спецификаций.

# Система дедуктивной верификации C-lightVer

Настоятельно рекомендуется до соревнования VeHa разместить на компьютере систему C-lightVer.

Архив с системой C-lightVer можно скачать по ссылке:

`https://cloud.mail.ru/public/Quw1/UuE9tV2Wo`

Также данная ссылка на скачивание архива с системой C-lightVer приведена в пособии.

Кроме того, данная ссылка на скачивание архива с системой C-lightVer будет приведена в условии задачи.

Текущая версия системы C-lightVer является результатом портирования на ОС Windows. Весь процесс установки состоит в распаковке архива с папкой C-lightVer.

# Система дедуктивной верификации C-lightVer

Система C-lightVer предназначена для дедуктивной верификации программ на языке C.

Система C-lightVer принимает на вход такие файлы с исходным кодом, где предусловия, постусловия и инварианты циклов заданы внутри комментариев вида /* */.

Задание спецификаций внутри комментариев в файлах с исходным кодом позволяет не нарушать возможность компиляции таких файлов.

Предусловие исходного кода записывается в комментарии до данного кода.

Постусловие программного кода записывается в комментарии после программного кода.

Инварианты циклов записываются в комментариях перед циклами.

# Система дедуктивной верификации C-lightVer

Интерфейсом системы C-lightVer является интерфейс командной строки. Для работы с системой C-lightVer необходимо запустить командную строку Windows и с помощью команды cd перейти в папку C-lightVer. Рекомендуется всю работу осуществлять внутри папки C-lightVer.

Система C-lightVer запускается с помощью следующей команды в командной строке:

```
C-lightVer.exe input report theory
```

где

- `C-lightVer.exe` – имя исполняемого файла
- `input` – имя файла с верифицируемой программой
- `report` – имя создаваемого файла с отчетом
- `theory` – имя файла с теорией предметной области (без расширения .lisp)

# Система дедуктивной верификации C-lightVer

В ходе исполнения системы C-lightVer командную строку Windows будет выведено много неважных сообщений. Верификация завершилась, если командная строка начинает показывать текущую директорию и ожидать ввода.

В папке C-lightVer будут созданы файлы, в их имени префиксом будет имя файла с верифицируемой программой. Названия файлов с условиями корректности будут вида

`input-verification-condition-i.lisp`

где

- `input` – имя файла с верифицируемой программой
- `i` – номер условия корректности

# Система доказательства теорем ACL2

Для доказательства условий корректности в системе C-lightVer применяется система доказательства ACL2.

Если ACL2 доказывает истинность содержимого файла, то тогда система ACL2 порождает файл с таким же названием, но с расширением не .lisp, а с расширением .cert.

Если для каждого файла с i-тым условием корректности был порожден файл с расширением .cert и именем input-verification-condition-i.lisp, то программа корректна относительно спецификаций.

То есть, для каждого файла с условием корректности должен быть порожден файл с таким же названием, но с расширением не .lisp, а с расширением .cert.

# Обучение участников соревнования по дедуктивной верификации в рамках контеста VeHa-2023

До соревнования были подготовлены и размещены на сайте соревнования:

- Пособие для участников соревнования по дедуктивной верификации в рамках контеста VeHa-2023
- Презентация тьюториала для участников соревнования по дедуктивной верификации в рамках контеста VeHa-2023

Также был проведен тьюториал для участников соревнования по дедуктивной верификации в рамках контеста VeHa-2023. Видеозапись тьюториала была выложена на сайт соревнования VeHa-2023.

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

# Исходные данные

```
/* (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;
}
/*  */
```

# Классическое решение: постусловие

```
/* (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) */
```

# Классическое решение: теория предметной области

```
(in-package "ACL2")

(include-book "std/util/defrule" :dir :system)
(include-book "centaur/fty/top" :dir :system)
(include-book "std/util/bstar" :dir :system)
(include-book "std/typed-lists/top" :dir :system)
(include-book "std/lists/top" :dir :system)
(include-book "std/basic/inductions" :dir :system)
```

# Классическое решение: теория предметной области

```
(defun element-equality(i j a)
    (if
        (or
            (not
                (natp
                    i
                )
            )
            (not
                (natp
                    j
                )
            )
        )
        0
```

# Классическое решение: теория предметной области

```
(if
    (>=
        i
        j
    )
    1
```

# Классическое решение: теория предметной области

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

# Результаты

Из 15 команд, участвовавших в соревновании VeHa-2023, решения задачи по дедуктивной верификации прислали 13 команд.

Из них 10 баллов набрали 9 команд.

Из них 9 баллов набрали 3 команды.

И 8 баллов набрала 1 команда.

Основные ошибки: диапазон элементов массива в постусловии функции.

# Интересные решения: Артем Кокорин

```
/* (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 n a) result) */
```

# Интересные решения: Артем Кокорин

```
(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)
)
```

# Интересные решения: команда Cache-Invalidation

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

# Соревнование VeHa-2023

## VeHa-2023: результаты

Дипломы

- 1 место — команда Vericheck (Политех, Санкт-Петербург)
- 2 место — команда FIIT (Политех, Санкт-Петербург)
- 3 место — команда NSU (НГУ, Новосибирск)

Почётные грамоты

- Команда AmNyam (Политех, СПб) — за попытку решения задачи Pipeline GPU
- Команда Team HYPE (Иннополис) — за оригинальное решение задачи Луна-25 (mc)
- Команда Cache-Invalidation (Иннополис) — за оригинальное решение задачи Луна-25 (dv)
- Артём Кокорин (Астра Линукс, Москва) — за оригинальное решение задач контеста
- Анастасия Красненкова (Астра Линукс, Москва) — за волю к победе

# Контест VeHa-2024

Несколько соревнований по формальной верификации программ.

Тематика соревнований в рамках контеста VeHa-2024 покрывает два главных направления формальной верификации программ:

- Дедуктивная верификация.
- Model checking.

Научная статья о предыдущем соревновании VeHa-2023:

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. DOI: `https://doi.org/10.15514/ISPRAS-2024-36(2)-11`

# Соревнование VeHa-2024

## Цель соревнования

- Основная цель соревнования VeHa-2024 – привлечь студентов Российских вузов к решению задач верификации с помощью формальных методов в формате хакатона

**Хакатон**

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”

- 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

# Соревнование VeHa-2024

## Первое соревнование VeHa – 2023 год

- с 03.11.2023 по 04.11.2023
- 35 регистраций
- Две обязательных к выполнению задачи по дедуктивной верификации и проверке моделей

# Соревнование VeHa-2024

## Родительский семинар для хакатона

PSSV-2024

**Семинар (workshop)**

***Семантика, спецификация и верификация программ – теория и приложения***

**18-21 октября 2024 г. в гибридном формате: онлайн и в Университете Иннополис и IT-парке Иннополис в Республике Татарстан (Россия)**

**Приглашаем познакомиться с выполненной Программой семинара (содержащей ссылки на записи и презентации всех докладов, а также на предварительные публикации отобранных работ)!**

Семинар проходит ежегодно начиная с 2010 года
(за единственным исключением в 2022 г.)

PSSV-2021: https://pcmsm.ru.rnk.su/en/pssv21

PSSV-2023: https://pcmsm.ru.rnk.su/en/PSSVYears2022towards2023

- Страницы семинара в Интернете:
  - Краткая информационная страница на EasyChair: https://easychair.org/cfp/PSSV-2024
  - Представление материалов через EasyChair: https://easychair.org/conferences/?conf=pssv2024
  - Seminar webpage in English
- Важные даты:
  - ~~Представление заявлений (необязательно): воскресенье 8 сентября 2024~~
  - ~~Представление расширенных тезисов для рецензирования: воскресенье 15 сентября 2024~~
  - ~~Решение о принятии представленных работ: воскресенье 29 сентября 2024~~
  - ~~Соревнования по верификации Verification Hackathon Velia-2024: активны 18 сентября – понедельник 21 сентября 2024~~
  - ~~Семинар PSSV-2024: суббота 19 сентября – понедельник 28 сентября 2024~~
  - ~~Объявление итогов конкурса им. В.А. Непомнящего молодых ученых: понедельник 21 сентября 2024~~
  - ~~Приглашение отобранных работ для публикации: среда 25 сентября 2024~~
  - ~~Объявление итогов соревнований по верификации Velia-2024: воскресенье 27 сентября 2024~~
  - ~~Представление переработанных и дополненных приглашенных работ: воскресенье 3 ноября 2024~~
  - ~~Решение о принятии представленных переработанных и дополненных приглашенных работ: воскресенье 17 ноября 2024~~
  - Публикация принятых переработанных и дополненных приглашенных работ в журнале *Моделирование и анализ информационных систем*: декабрь 2024

- PSSV (Program Semantics, Specification and Verification: Theory and Applications) – Семантика, спецификация и верификация программ – теория и приложения
- Проводится начиная с 2010 года
- Публикации в журналах “Системная информатика” (РИНЦ), Моделирование и анализ информационных систем (ВАК, ядро РИНЦ), перевод – Automatic Control and Computer Sciences (Springer)

# Соревнование VeHa-2024

## Ключевые факты о соревновании 2024 года

- Сайт соревнований : https://sites.google.com/view/veha2024
- Время проведения: с 18.10.2024 по 21.10.2024
- Формат проведения: очно (Иннополис) + онлайн
- Мероприятие было организовано группой энтузиастов по формальным методам имеющих корни в Новосибирском Академгородке в сотрудничестве с Astra group и Positive Technologies
- Секции: (1) Дедуктивная верификация: FramaC, Coq, C-lightVer; (2) Model checking (проверка моделей): SPIN, TLA+
- Использовалась минимальная инфраструктура: сайт на конструкторе, GitHub, гугл-формы

# Соревнование VeHa-2024

## Состав организаторов (1)

- Гаранина Наталья (к.ф.-м.н., с.н.с. Института Систем Информатики, Института Автоматики и Электрометрии СО РАН, доцент НГУ) — ответственная за секцию Model Checking;
- 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 Deductive Verification C-lightVer;
- Красненкова Анастасия (Специалист по анализу безопасности отдела анализа СЗИ и формальной верификации, группа Астра) — ответственная за секцию Frama-C;

# Соревнование VeHa-2024

## Состав организаторов (2)

- Никольский Денис (Университет Иннополис) — координатор задач;
- Сим Виолетта (Университет Иннополис) — координатор задач (Участник контеста VeHa-2023);
- Старолетов Сергей (к.ф.-м.н., доцент АлтГТУ, с.н.с. Института Автоматики и Электрометрии СО РАН) — ответственный за секцию Model Checking и за сайт соревнований / GitHub репозиторий;
- 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 Model Checking.

# Соревнование VeHa-2024

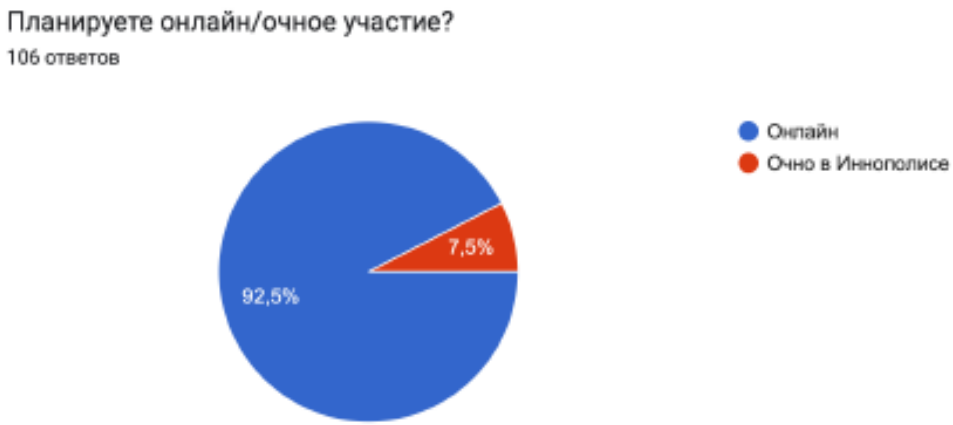
## Состав участников

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.

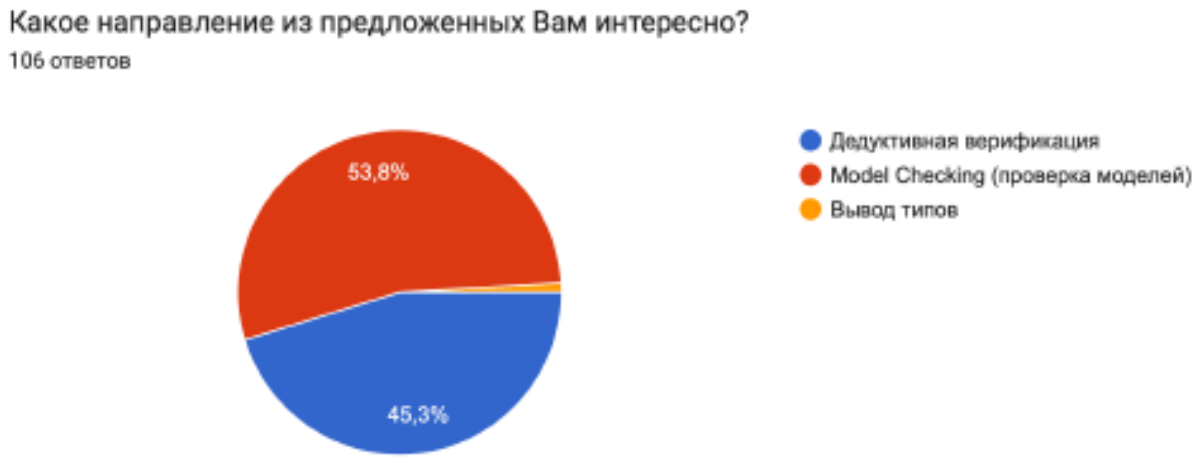
## Распределение регистраций за онлайн и очное участие

Как и в прошлом году, был предложен гибридный формат участия: можно было участвовать как онлайн, так и очно в Иннополисе, с проживанием в хостеле и возможностью посетить экскурсии совместно с участниками семинара PSSV. Распределение участников по планируемой форме участия:



## Распределение регистраций за онлайн и очное участие

В форме регистрации также предлагалось указать планируемую технологию верификации, с небольшим перевесом тут оказалась проверка моделей (Model Checking):



# Соревнование VeHa-2024

## Задачи (треки) соревнования

- Задача 1 — Верификация функции проверки прав доступа на Frama-C
- Задача 2 — Верификация функции проверки прав доступа на Coq
- Задача 3 — Моделирование вычислительного pipeline GPU с поиском оптимальных параметров с помощью Model Checking
- Задача 4 — Дедуктивная верификация программы, относящейся к задаче проверки выполнимости булевых формул (SAT)
- Задача 5 — Построение и проверка модели протокола консенсуса IBFT

Тексты задач доступны на sites.google.com/view/veha2024/задачи

# Соревнование VeHa-2024

## 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 ядра Linux в формате ACSL.

```c
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;
}
```

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

Функция check_permission проверяет доступ к заданному файлу с учетом прав пользователя на чтение/запись. Функция 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).

```
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;
}
```

14

# Соревнование VeHa-2024

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

- 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 проверифицировать 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

# Соревнование VeHa-2024

## Задача 2 — Верификация функции проверки прав доступа на Coq

- Задача заключается в формальной верификации одной из функций модуля безопасности ядра Linux
- 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!

# Соревнование VeHa-2024

## Задача 2 — Верификация функции проверки прав доступа на Coq

Функция vsm_inode_permission из модуля безопасности ядра отвечает за проверку метки целостности процесса при обращении к inode. Она принимает два аргумента: указатель на inode, запрашиваемые параметры доступа к этому inode. Функция возвращает 0, если процессу разрешен доступ к inode.

```c
#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;
}
```

# Соревнование VeHa-2024

## Задача 2 — Верификация функции проверки прав доступа на Coq

Подход верификации такого рода функций в компании Astra:

- В рамках проекта по созданию формально верифицированного компилятора CompCert[2] был разработан инструмент clightgen, который отвечает за генерацию AST на Coq по исходному коду на C (с некоторыми ограничениями вроде no casting between integers and pointers[3])
- Затем это AST используется инструментом VST[4] для доказательства соответствия кода его функциональной спецификации
- В заданиях для VeHa этот этап был пропущен: в одном из заданий мы просили участников самостоятельно написать функциональную спецификацию упрощенной реализации функции inode_permission, проверяющей права доступа

[2] https://compcert.org
[3] https://github.com/PrincetonUniversity/VST/raw/master/doc/VC.pdf
[4] https://vst.cs.princeton.edu/

# Соревнование VeHa-2024

## Задача 2 — Верификация функции проверки прав доступа на Coq

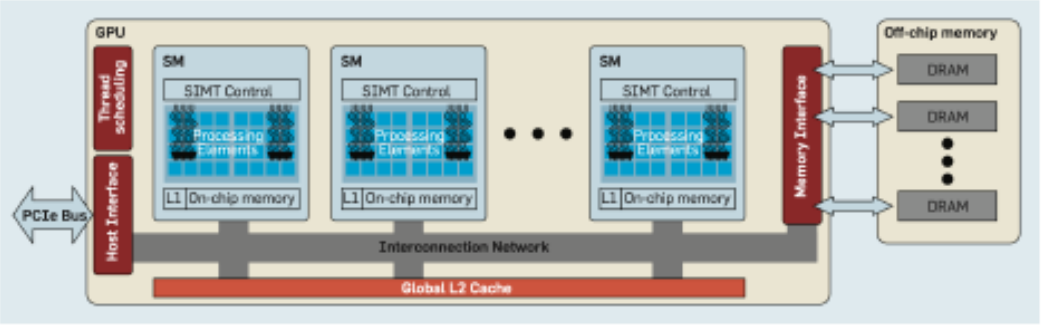
Участникам были предложены следующие задания:

- 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.

# Соревнование VeHa-2024

## Задача 3 — Моделирование вычислительного pipeline GPU с поиском оптимальных параметров с помощью Model Checking

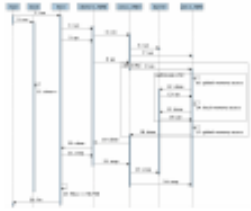
- Для решения задач, требующих интенсивных расчетов, которые хорошо распараллеливаются (вроде операций с матрицами, расчет хэш функций для майнинга, приложений ИИ), в настоящее время используются графические процессоры (GPU)
- Все эти задачи используют изначальную ориентированность графических процессоров видеокарт на одновременное выполнение однотипных операций с разными данными на большом количестве исполнителей. Архитектура графических процессоров называется SIMT (single instruction-multiple thread — одна инструкция – множество потоков)



# Соревнование VeHa-2024

## Задача 3 — Моделирование вычислительного pipeline GPU с поиском оптимальных параметров с помощью Model Checking

- Для программирования под архитектуры GPU используются технологии вроде OpenCL, при этом код, исполняемый на видеокарте, пишется на C-подобном языке и называется “ядром”. В дальнейшем этот код транслируется в ассемблерный код (PTX для Nvidia) и его бинарное представление для дальнейшего исполнения на видеокарте конкретной архитектуры
- Нами[5] была разработана абстрактная модель OpenCL программы, которая представлена участникам на ознакомление. Оптимальная работа программы по потреблению ресурсов, исполняемых на GPU, зависит от множества параметров, подбор которых затруднен.



[5]Garanina, N., Staroletov, S., Gorlatch, S. Auto-Tuning High-Performance Programs Using Model Checking in Promela. https://arxiv.org/pdf/2305.09130

# Соревнование VeHa-2024

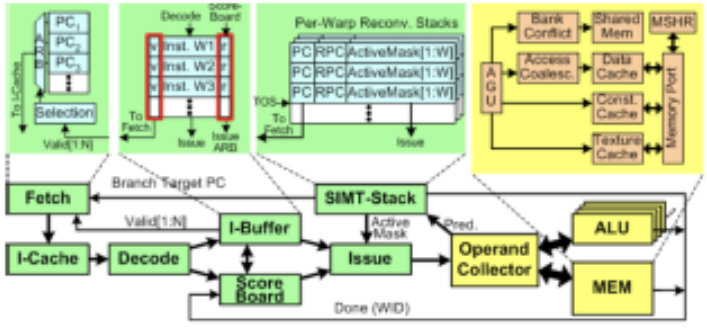
## Задача 3 — Моделирование вычислительного pipeline GPU с поиском оптимальных параметров с помощью Model Checking

- В задаче контеста предлагалось реализовать настройку некоторых параметров оптимизации для программы (WG — размер рабочей группы в программе на OpenCL, TS — размер массива данных, обрабатываемый потоком), написанной на OpenCL, с помощью Model Checking
- Для решения задачи настройки с помощью метода проверки моделей необходимо описать модель исполнения настраиваемой программы на выбранной программной архитектуре. Модель исполнения должна быть описана на входном языке верификатора (в виде взаимодействующих процессов Promela).

# Соревнование VeHa-2024

## Задача 3 — Моделирование вычислительного pipeline GPU с поиском оптимальных параметров с помощью Model Checking

- Для моделирования GPU в более приближенном к реальности виде (с учетом представления программы в виде PTX инструкций, кэша таких инструкций, ворпов (warp, единица планирования одновременного количества потоков), сборщика операндов, дивергенции ветвей исполнения и т.д.) предлагалось воспользоваться наработками проекта GPGU Sim[6], в котором на основе анализа открытых патентов Nvidia исследователи воссоздали последовательность работы GPU (исполнительный пайплайн), который и предлагалось реализовать участникам в некотором виде




[6] http://gpgpu-sim.org/manual/index.php

# Соревнование VeHa-2024

## Задача 3 — Моделирование вычислительного pipeline GPU с поиском оптимальных параметров с помощью Model Checking

В итоге, на контест было принято решение предложить решение задачи на разных уровнях, от простой модификации решения авторов для другой задачи, до детальной реализации пайплайна:

- 1 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
- 2 уровень. Дополнительно к уровню 1 модифицировать решение задачи настройки для OpenCL, добавив работу с ворпами (warp)
- 3 уровень. Реализовать уровень 2 для приложения на PTX, превращая исходную программу в последовательность инструкций и Promela код должен моделировать работу по выбору и исполнению этих инструкций
- 4 уровень. Дополнительно к уровню 3 реализовать детально пайплайн для подготовки данных и ворпов на выполнение

# Соревнование VeHa-2024

## Задача 5 — Построение и проверка модели протокола консенсуса IBFT

**Протокол консенсуса**

Это алгоритм, решающий проблему достижения консенсуса в распределенных системах, когда все корректные процессы должны принять решение о некотором общем предлагаемом значении

- 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

Алгоритм IBFT (Istanbul Byzantine Fault Tolerance)[10] вдохновлен алгоритмом PBFT (Practical Byzantine Fault Tolerance)[11].

[10] Moniz, H. (2020). The Istanbul BFT consensus algorithm. arXiv preprint arXiv:2002.03613.

[11] Castro M., Liskov B. Practical Byzantine Fault Tolerance. Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999.

# Соревнование VeHa-2024

## Задача 5 — Построение и проверка модели протокола консенсуса IBFT

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 хакатона VeHa-2024. Реализовать модель алгоритма участники могли на основе псевдокода или описания алгоритма на естественном языке, предложенном в статье

# Организатор соревнования в рамках контеста VeHa-2024 по дедуктивной верификации программы, относящейся к задаче проверки выполнимости булевых формул (SAT)

Кондратьев Дмитрий Александрович, к.ф.-м.н., научный сотрудник Института систем информатики им. А.П. Ершова СО РАН и старший преподаватель Новосибирского Государственного Университета
Обратная связь: письма на электронную почту apple-66@mail.ru

В рамках предыдущего соревнования VeHa-2023 также организовывал задачу по дедуктивной верификации, что описано в следующей статье:

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. DOI: https://doi.org/10.15514/ISPRAS-2024-36(2)-11

# Пособие для участников соревнования в рамках контеста VeHa-2024 по дедуктивной верификации программы, относящейся к задаче проверки выполнимости булевых формул (SAT)

Настоятельно рекомендуется ознакомиться с данным пособием до соревнования VeHa-2024.

Данное пособие можно скачать по следующей ссылке:

`https://cloud.mail.ru/public/avhh/oLgFb1zz8`

Кроме того, данное пособие можно найти на сайте контеста VeHa-2024 на вкладке "Материалы".

Также данное пособие будет включено в условие задачи по дедуктивной верификации программы, относящейся к задаче проверки выполнимости булевых формул (SAT).

# Контест VeHa-2024

Соревнование по дедуктивной верификации программы, относящейся к задаче проверки выполнимости булевых формул в 2-КНФ (2-SAT)

Организатор данного соревнования: Кондратьев Дмитрий Александрович

Участники данного соревнования применяли систему дедуктивной верификации программ C-lightVer, разработанную в ИСИ СО РАН.

# Соревнование VeHa-2024

## 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
- Задача 2-SAT была выбрана в качестве примера потому, что в последнее время активно развиваются исследования по формальной верификации программных систем доказательства теорем в том числе таким специализированным системам, как SAT-решатели и SMT-решатели

# Соревнование VeHa-2024

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

- В качестве системы верификации C-lightVer была выбрана потому, что в системе C-lightVer[7] [8] применяется система доказательства теорем ACL2[9], которая может автоматически применять уже доказанные теоремы в качестве лемм для доказательства других теорем, что упрощает доказательство условий корректности программ

---

[7] 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

[8] 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

[9] Moore J. S. Milestones from the pure Lisp theorem prover to ACL2. Formal Aspects of Computing, vol. 31, no. 6, pp. 699–732, 2019

# Задача 2-SAT

2-КНФ

Пример формулы в 2-КНФ:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_4)$$

# Переписывание дизъюнктов

Элементарные дизъюнкты в 2-КНФ переписываются с помощью следующих правил эквивалентности:

$$(x \vee y) \equiv ((\neg x \to y) \wedge (\neg y \to x))$$

$$(x \vee \neg y) \equiv ((\neg x \to \neg y) \wedge (y \to x))$$

$$(\neg x \vee y) \equiv ((x \to y) \wedge (\neg y \to \neg x))$$

$$(\neg x \vee \neg y) \equiv ((x \to \neg y) \wedge (y \to \neg x))$$

В результате применения таких правил переписывания получается формула с конъюнкциями импликаций без явных дизъюнктов

# Граф импликаций

Граф импликаций содержит в качестве вершин все переменные и их отрицания, встречающиеся в 2-КНФ после переписывания дизъюнкций

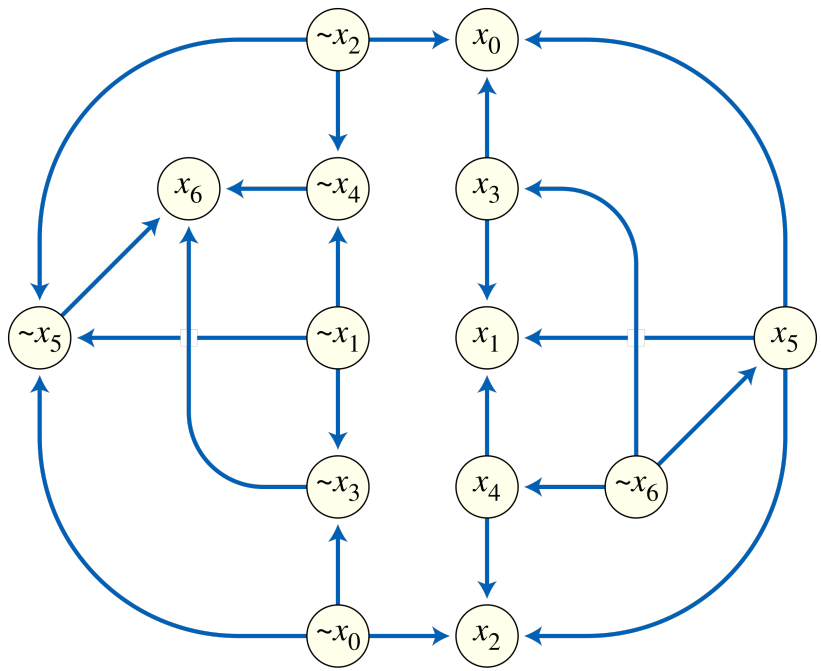
Вершины $V_1$ и $V_2$ в графе импликаций соединены ребром, направленным из $V_1$ в $V_2$, тогда и только тогда, когда в формуле есть импликация из $V_1$ в $V_2$.

# Пример графа импликаций

Формула:

$$(x_0 \vee x_2) \wedge (x_0 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_4) \wedge$$
$$(x_0 \vee \neg x_5) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee \neg x_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$$

Граф импликаций для данной формулы:



# Анализ графа импликаций

Для каждой входящей в формулу переменной $x$ проверяется следующее условие: существует ли в графе импликаций путь из $x$ в $\neg x$ и существует ли в графе импликаций путь из $\neg x$ в $x$. Если для хотя бы одной переменной такое условие выполнено, то формула не выполнима. Если для всех входящих в формулу переменных такое условие не выполнено, то формула выполнима.

# Алгоритм решения задачи 2-SAT

1. Построим граф импликаций
2. Построим матрицу смежности транзитивного замыкание графа импликаций (например, с помощью алгоритма Флойда-Уоршелла).
3. С помощью полученной матрицы смежности для каждой переменной $x$ проверяем условие: существует путь из $x$ в $\neg x$ и существует путь из $\neg x$ в $x$. Если хотя бы для одной переменной данное условие истинно, то формула невыполнима. Если данное условие ложно для всех переменных, то формула выполнима.

Итого, полиномиальное решение.

# Соревнование VeHa-2024

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

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:

```
/* 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 */
```

# Верифицируемая функция `twosat_solver`

Функция `twosat_solver` возвращает значение 1, если формула выполнима. Если формула невыполнима, то функция `twosat_solver` возвращает значение 0.

Результатом исполнения функции `twosat_solver` является значение переменной `satisfiable`.

# Верифицируемая функция twosat_solver

```
/* 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 */
```

# Предусловие функции `twosat_solver`

Предусловие функции `twosat_solver` описывает ограничение на аргументы данной функции. Приведем отдельно предусловие функции `twosat_solver`.

# Предусловие (на языке Applicative Common Lisp)

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

# Предусловие функции `twosat_solver`

Данное предусловие представляет собой конъюнкцию ограничений на аргументы функции `twosat_solver`. Отметим, что данные ограничения будут играть важную роль при задании используемого в функции `twosat_solver` инварианта цикла.

# Постусловие функции `twosat_solver`

Постусловие функции `twosat_solver` описывает свойство, что если формула невыполнима, то принимает ложное значение применение функции `twosat-solver`. Отметим, что в постусловии применяется функция `twosat-solver` из теории предметной области. Приведем отдельно данное постусловие.

# Постусловие (на языке Applicative Common Lisp)

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

# Постусловие функции `twosat_solver`

Данное постусловие является импликацией, описывающей, что если переменная `satisfiable` по итогам исполнения функции равна нулю, то принимает ложное значение применение функции `twosat-solver`. Отметим, что данное постусловие не описывает случай, когда формула выполнима.

# Теория предметной области в файле `twosat-solver.lisp`: определение `twosat-solver` на языке Applicative Common Lisp

```
(defun twosat-solver(variable-values variable-count implication-graph-transitive-closure)
    (if
        (or
            (not (integerp variable-count))
            (not (> variable-count 0))
            (not (integer-listp implication-graph-transitive-closure))
            (not (boolean-variable-values-boolean-listp variable-values))
            (not (= (len implication-graph-transitive-closure)
                    (* 4 (* variable-count variable-count)))))
        nil
        (if
            (endp variable-values)
            nil
            (or
                (formula-sat
                    (- variable-count 1)
                    (car variable-values)
                    variable-count
                    implication-graph-transitive-closure)
                (twosat-solver
                    (cdr variable-values)
                    variable-count
                    implication-graph-transitive-closure)))))
```

# Цикл, для которого задается инвариант

Цикл внутри функции `twosat_solver` выполняет поиск такой переменной формулы, для которой не выполнено важное для задачи 2SAT условие. Приведем отдельно цикл, для которого задается инвариант.

# Цикл, для которого задается инвариант

```
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++;
    }
}
```

# Цикл, для которого задается инвариант

Если в цикле найдена переменная формулы, для который не выполнено важное для задачи 2SAT условие, то значение переменной `satisfiable` становится равным нулю.

# Инвариант цикла

Задача на соревновании состояла в задании инварианта цикла. Приведем отдельно инвариант цикла.

# Соревнование VeHa-2024

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

- Участникам соревнования было предоставлено уже заданное предусловие верифицируемой программы: (and (integerp variable_count) (< 0 variable_count) (integer-listp implication_graph_transitive_closure) (= (len implication_graph_transitive_closure) (* 4 (* variable_count variable_count))))
- Участникам соревнования было также предоставлено постусловие программы: (implies (= satisfiable 0) (not (twosat-solver (boolean-variable-values variable_count nil) variable_count implication_graph_transitive_closure)))
- Ожидалось решение в виде инварианта, представляющего собой конъюнкцию трех частей: ограничения на переменные, нужные для доказательства условия 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.

# Инвариант: ограничения на переменные-аргументы

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

# Инвариант: ограничения на переменную-счетчик

```
...
(integerp
            x
        )
(<=
            0
            x
        )
...
```

# Инвариант: ограничения на переменную-результат

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

# Инвариант: ограничения на переменную-результат

```
...
                (=
                    (nth
                        (+
                            (*
                                (+
                                    x
                                    variable_count
                                )
                                (*
                                    variable_count
                                    2
                                )
                            )
                            x
                        )
                        implication_graph_transitive_closure
                    )
                    1
                )
                (<
                    x
                    variable_count
                )
            )
        )
    )
```

# Инвариант цикла

Инвариант цикла представляет собой конъюнкцию. Входящие в данный инвариант конъюнкты можно условно разделить на три части: ограничения на типы переменных-аргументов, ограничения на переменную-счетчик и ограничение на переменную-результат.

# Три части инварианта цикла

1. Составляющие первую часть инварианта ограничения на типы переменных удобно взять из предусловия. Данные ограничения позволяют обеспечить завершение цикла. Условие завершение цикла напрямую не зависит от предусловия, поэтому в инварианте нужно сохранить информацию об описанных в предусловии ограничениях на переменные-аргументы.
2. Составляющее вторую часть инварианта ограничение на переменную-счетчик описывает ограничение на тип переменной-счетчика. Вторая часть инварианта нужна и для обеспечения условия входа в цикл, и для обеспечения условия продолжения итераций цикла, и для обеспечения условия завершения цикла.
3. Составляющее третью часть инварианта ограничение на переменную-результат описывает, что если переменная `satisfiable` равна нулю, то для $i$-той переменной формулы не выполняется важное для задачи 2SAT условие. Данное ограничение позволяет доказать выполнение постусловия при завершении цикла с помощью применения в качестве леммы теоремы `twosat-unsat` из теории предметной области.

# Дедуктивная верификация примера twosat_solver в системе C-lightVer

Для верификации данного примера нужно сначала разместить в папке с системой C-lightVer файлы twosat_solver.c и twosat-solver.lis а затем в командной строке ввести следующую инструкцию:

```
C-lightVer.exe twosat_solver.c twosat_solver_report.txt twosat-s
```

где

- C-lightVer.exe – имя исполняемого файла
- twosat_solver.c – имя файла с программой
- twosat_solver_report.txt – имя создаваемого отчета
- twosat-solver – имя файла с теорией предметной области (без расширения .lisp)

# Дедуктивная верификация примера twosat_solver в системе C-lightVer

Порождаемые условия корректности:

- `twosat_solver-verification-condition-1.lisp` – файл с условием корректности 1.
- `twosat_solver-verification-condition-2.lisp` – файл с условием корректности 2.
- `twosat_solver-verification-condition-3.lisp` – файл с условием корректности 3.
- `twosat_solver-verification-condition-4.lisp` – файл с условием корректности 4.

# Условие корректности 1 (вход в цикл)

```
(defrule verification-condition-1
    (implies
        (and
            (integerp
                variable_count
            )
            (<
                0
                variable_count
            )
            (integer-listp
                implication_graph_transitive_closure
            )
            (=
                (len
                    implication_graph_transitive_closure
                )
                (*
                    4
                    (*
                        variable_count
                        variable_count
                    )
                )
            )
        )
...
```

# Условие корректности 1 (вход в цикл)

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

# Условие корректности 1 (вход в цикл)

```
...
            (implies
                (=
                    1
                    0
                )
                (and
                    (=
                        (nth
                            (+
                                (*
                                    0
                                    (*
                                        variable_count
                                        2
                                    )
                                )
                                (+
                                    0
                                    variable_count
                                )
                            )
                            implication_graph_transitive_closure
                        )
                        1
                    )
...
```

# Условие корректности 1 (вход в цикл)

```
...
                    (=
                        (nth
                            (+
                                (*
                                    (+
                                        0
                                        variable_count
                                    )
                                    (*
                                        variable_count
                                        2
                                    )
                                )
                                0
                            )
                            implication_graph_transitive_closure
                        )
                        1
                    )
                    (<
                        0
                        variable_count
                    )
                )
            )
        )
    )
    :rule-classes nil
)
```

# Условие корректности 2 (выход из цикла)

```
(defrule verification-condition-2
    (implies
        (and
            (and

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

# Условие корректности 2 (выход из цикла)

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

# Условие корректности 2 (выход из цикла)

```
...
                        (=
                            (nth
                                (+
                                    (*
                                        (+
                                            x
                                            variable_count
                                        )
                                        (*
                                            variable_count
                                            2
                                        )
                                    )
                                    x
                                )
                                implication_graph_transitive_closure
                            )
                            1
                        )
                        (<
                            x
                            variable_count
                        )
                    )
                )
            )
...
```

# Условие корректности 2 (выход из цикла)

```
...
            (not
                (and
                    (<
                        x
                        variable_count
                    )
                    (equal
                        satisfiable
                        1
                    )
                )
            )
        )
...
```

# Условие корректности 2 (выход из цикла)

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

# Условие 3 (продолжение итерации, позитивный if)

```
(defrule verification-condition-3
    (implies
        (and
            (and
                (integerp
                    variable_count
                )
                (<
                    0
                    variable_count
                )
                (integer-listp
                    implication_graph_transitive_closure
                )
                (=
                    (len
                        implication_graph_transitive_closure
                    )
                    (*
                        4
                        (*
                            variable_count
                            variable_count
                        )
                    )
                )
...
```

# Условие 3 (продолжение итерации, позитивный `if`)

```
...
                (integerp
                    x
                )

                (<=
                    0
                    x
                )
...
```

# Условие 3 (продолжение итерации, позитивный `if`)

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

# Условие 3 (продолжение итерации, позитивный if)

```
...
                    (=
                        (nth
                            (+
                                (*
                                    (+
                                        x
                                        variable_count
                                    )
                                    (*
                                        variable_count
                                        2
                                    )
                                )
                                x
                            )
                            implication_graph_transitive_closure
                        )
                        1
                    )
                    (<
                        x
                        variable_count
                    )
                )
            )
        )
...
```

# Условие 3 (продолжение итерации, позитивный if)

```
...
            (and
                (<
                    x
                    variable_count
                )
                (equal
                    satisfiable
                    1
                )
            )
        )
...
```

# Условие 3 (продолжение итерации, позитивный if)

```
...
        (implies
            (and
                (equal
                    (nth
                        (+
                            (+
                                x
                                variable_count
                            )
                            (*
                                (*
                                    x
                                    2
                                )
                                variable_count
                            )
                        )
                        implication_graph_transitive_closure
                    )
                    1
                )
...
```

# Условие 3 (продолжение итерации, позитивный if)

```
...
                (equal
                    (nth
                        (+
                            (+
                                (*
                                    (*
                                        x
                                        2
                                    )
                                    variable_count
                                )
                                x
                            )
                            (*
                                (*
                                    variable_count
                                    2
                                )
                                variable_count
                            )
                        )
                        implication_graph_transitive_closure
                    )
                    1
                )
            )
...
```

# Условие 3 (продолжение итерации, позитивный if)

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

# Условие 3 (продолжение итерации, позитивный `if`)

```
...
                (integerp
                    x
                )
                (<=
                    0
                    x
                )
...
```

# Условие 3 (продолжение итерации, позитивный if)

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

# Условие 3 (продолжение итерации, позитивный if)

```
...
                        (=
                            (nth
                                (+
                                    (*
                                        (+
                                            x
                                            variable_count
                                        )
                                        (*
                                            variable_count
                                            2
                                        )
                                    )
                                    x
                                )
                                implication_graph_transitive_closure
                            )
                            1
                        )
                        (<
                            x
                            variable_count
                        )
                    )
                )
            )
        )
    )
    :rule-classes nil
)
```

# Условие 4 (продолжение итерации, негативный if)

```
(defrule verification-condition-4
    (implies
        (and
            (and
                (integerp
                    variable_count
                )
                (<
                    0
                    variable_count
                )
                (integer-listp
                    implication_graph_transitive_closure
                )
                (=
                    (len
                        implication_graph_transitive_closure
                    )
                    (*
                        4
                        (*
                            variable_count
                            variable_count
                        )
                    )
                )
...
```

# Условие 4 (продолжение итерации, негативный `if`)

```
...
                (integerp
                    x
                )
                (<=
                    0
                    x
                )
...
```

# Условие 4 (продолжение итерации, негативный `if`)

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

# Условие 4 (продолжение итерации, негативный `if`)

```
...
                    (=
                        (nth
                            (+
                                (*
                                    (+
                                        x
                                        variable_count
                                    )
                                    (*
                                        variable_count
                                        2
                                    )
                                )
                                x
                            )
                            implication_graph_transitive_closure
                        )
                        1
                    )
                    (<
                        x
                        variable_count
                    )
                )
            )
        )
...
```

# Условие 4 (продолжение итерации, негативный `if`)

```
...
            (and
                (<
                    x
                    variable_count
                )
                (equal
                    satisfiable
                    1
                )
            )
        )
...
```

# Условие 4 (продолжение итерации, негативный if)

```
...
        (implies
            (not
                (and
                    (equal
                        (nth
                            (+
                                (+
                                    x
                                    variable_count
                                )
                                (*
                                    (*
                                        x
                                        2
                                    )
                                    variable_count
                                )
                            )
                            implication_graph_transitive_closure
                        )
                        1
                    )
...
```

# Условие 4 (продолжение итерации, негативный `if`)

```
...
                    (equal
                        (nth
                            (+
                                (+
                                    (*
                                        (*
                                            x
                                            2
                                        )
                                        variable_count
                                    )
                                    x
                                )
                                (*
                                    (*
                                        variable_count
                                        2
                                    )
                                    variable_count
                                )
                            )
                            implication_graph_transitive_closure
                        )
                        1
                    )
                )
            )
...
```

# Условие 4 (продолжение итерации, негативный `if`)

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

# Условие 4 (продолжение итерации, негативный if)

```
...
                (integerp
                    (+
                        x
                        1
                    )
                )
                (<=
                    0
                    (+
                        x
                        1
                    )
                )
...
```

# Условие 4 (продолжение итерации, негативный if)

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

# Условие 4 (продолжение итерации, негативный if)

```
...
                    (=
                        (nth
                            (+
                                (*
                                    (+
                                        (+
                                            x
                                            1
                                        )
                                        variable_count
                                    )
                                    (*
                                        variable_count
                                        2
                                    )
                                )
                                (+
                                    x
                                    1
                                )
                            )
                            implication_graph_transitive_closure
                        )
                        1
                    )
                    (<
                        (+
                            x
                            1
                        )
                        variable_count
                    )
                )
            )
```

# Дедуктивная верификация примера `twosat_solver` в системе C-lightVer

Об успешности доказательства условий корректности свидетельствует порождение файлов:

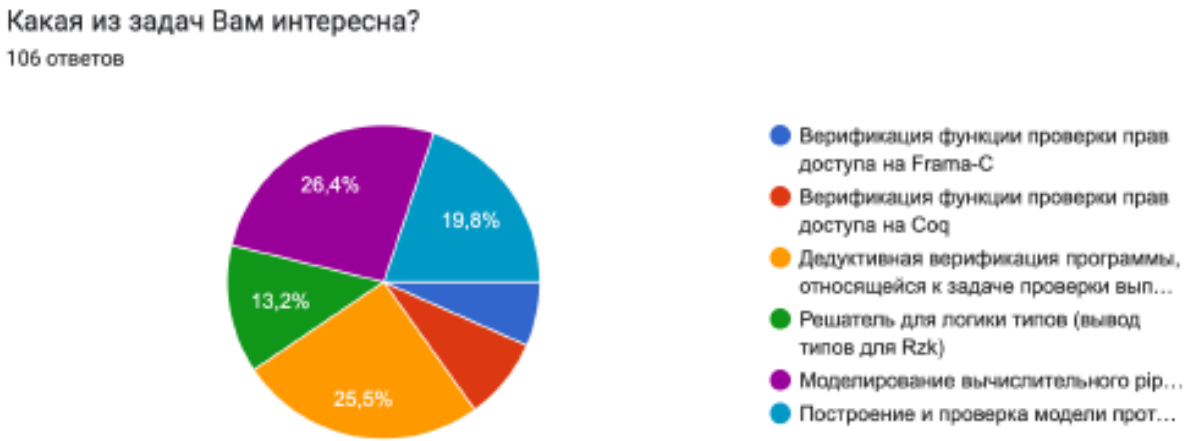
- `twosat_solver-verification-condition-1.cert`
- `twosat_solver-verification-condition-2.cert`
- `twosat_solver-verification-condition-3.cert`
- `twosat_solver-verification-condition-4.cert`

# Соревнование по дедуктивной верификации программы, относящейся к задаче проверки выполнимости булевых формул в 2-КНФ (2-SAT)

Из 5 зарегистрировавшихся отдельных участников и 6 зарегистрировавшихся команд решения сдали 3 отдельных 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, содержащий 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.

## Распределение регистраций по предложенным задачам

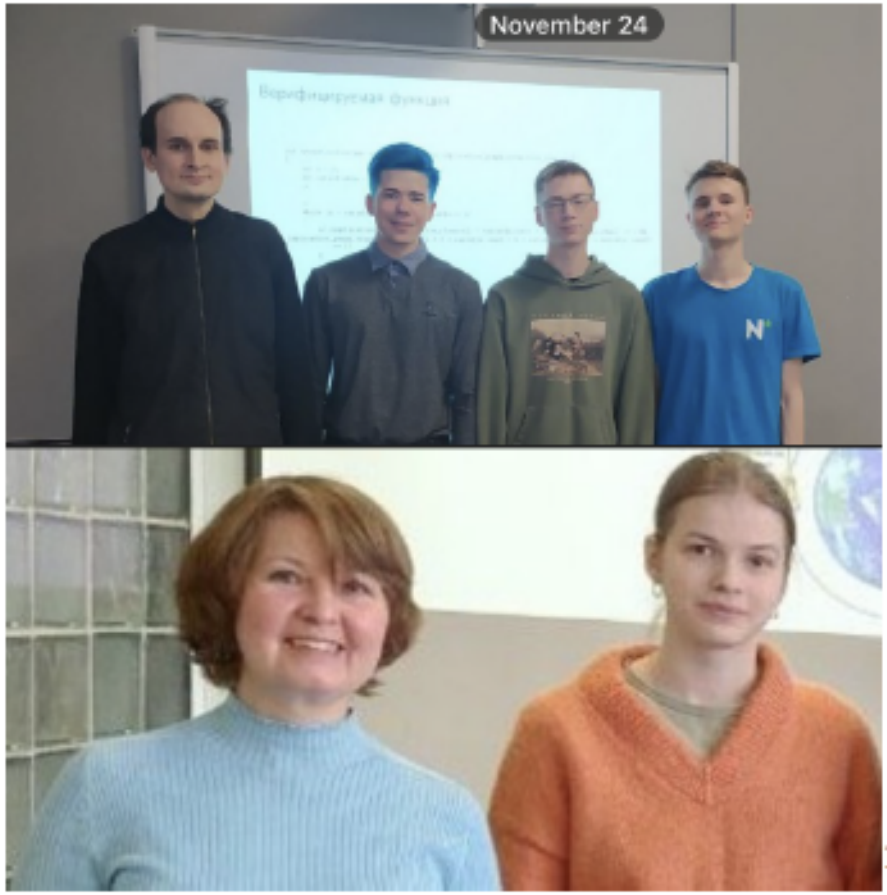
По предварительной регистрации на задачи небольшой перевес был отдан задаче Model Checking для GPU:



## Репозиторий с решениями

https://github.com/VeHaContest/VeHa2024

# Встречи с победителями соревнования VeHa-2024 из НГУ




[12]

[12] https://www.nsu.ru/n/media/news/obrazovanie/komandy-ngu-stali-pobeditelyami-vtorogo-vserossiyskogo-sorevnovaniya-po-formalnoy-verifikatsii-progr/

# Соревнование VeHa-2024

## Результаты

1. Верификация функции проверки прав доступа на Frama-C

   - Диплом I степени: магистранты (в составе команды «Buffer_Overflow») 1-го курса Института компьютерных наук и кибербезопасности СПбПУ им. Петра Великого
     - Жоел Андрес Бонилья Тельес, Роберто Юниор Домингес Молина, Данило Анатоли Меркадо Оудалова
   - Диплом II степени: студенты (в составе команды «re-tofl») 3-го курса кафедры Теоретическая информатика и компьютерные технологии (ИУ9) МГТУ им. Н.Э. Баумана
     - Александр Игоревич Старовойтов, Егор Станиславович Домаскин, Матвей Алексеевич Аринин
   - Почетные грамоты за успешное освоение инструмента Frama-C: магистранты участие (в составе команды «DevTools Itmo») 2-го курса ИТМО
     - Даниил Евгеньевич Бакин, Егор Алексеевич Белехов, Николай Александрович Рулев, Сергей Александрович Федоров

# Соревнование VeHa-2024

## Результаты

2. Верификация функции проверки прав доступа на Coq

- Диплом I степени (и почетная грамота за самое быстрое решение): студент 6 курса ФПМИ МФТИ Алексей Александрович Волков
- Диплом I степени (и почетная грамота за самое короткое решение и использование автоматики): аспирант ФПМИ МФТИ Иван Николаевич Смирнов
- Диплом I степени (и почетная грамота выделение вспомогательных утверждений): Прохор Николаевич Шляхтун
- Диплом I степени (и почетная грамота за внимание к корректности функциональной спецификации): Глеб Павлович Щепа

# Соревнование VeHa-2024

## Результаты

3. Моделирование вычислительного pipeline GPU с поиском оптимальных параметров с помощью Model Checking

- Диплом I степени: студенты (в составе команды «RaRe») магистранты 2-го курса направления «Программная инженерия» СПбПУ им. Петра Великого
  - Руслан Азадович Золотарев
  - Руслан Рестемович Попов
  - Анатолий Сергеевич Лазовый
- Диплом II степени (и почетная грамота за найденную ошибку в исходном коде): студенты (в составе команды «Power O‘ Nine») 3-го курса факультета «Информатика и системы управления» МГТУ им. Н.Э. Баумана
  - Артем Антонович Черников
  - Владимир Владимирович Пирко
  - Екатерина Владимировна Зайковская

# Соревнование VeHa-2024

## Результаты

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

- Диплом I степени (и почетная грамота за волю к победе): студент 6-го курса факультета «Программная инженерия» СПбПУ им. Петра Великого Мария Андреевна Шутова
- Диплом I степени: магистрант 2-го курса ММФ НГУ Наталья Андреевна Панченко
- Диплом I степени: студенты (в составе команды «VeriFLOWers») 3-го курса ФИТ НГУ
  - Тимур Витальевич Гунько, Антон Максимович Чумак, Денис Владимирович Малиновский
- Диплом I степени: магистранты (в составе команды «Point») 1-го курса Института компьютерных наук и кибербезопасности СПбПУ им. Петра Великого
  - Ксения Игоревна Еремина
  - Анатолий Романович Егоров

## Результаты

- Благодарственное письмо за активное участие в номинациях «Верификация функции проверки прав доступа на Frama-C», «Верификация функции проверки прав доступа на Coq» и «Дедуктивная верификация программы, относящейся к задаче проверки выполнимости булевых формул (SAT)» Александру Валентиновичу Когтенкову.

# Соревнование VeHa-2024

## Выводы

- Мы считаем, что соревнование VeHa-2024 прошло успешно, с большим вовлечением участников, чем в прошлый раз
- Решение сделать несколько разнородных задач 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 студентам-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
- На такого рода соревнованиях целесообразно также демонстрировать участником свои подходы и программные средства для верификации, проводить обучение по ним

# Соревнование по формальной верификации программ VeHa-2024: два года большого пути

Кондратьев Дмитрий Александрович

Институт систем информатики им. А. П. Ершова СО РАН
Новосибирский Государственный Университет