ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Факультет Санкт-Петербургская школа физико-математических и компьютерных наук**

Департамент информатики

Вавилов Марк Владимирович

# Реализация операций с плавающей точкой в терминах бит-векторных операций

Выпускная квалификационная работа - БАКАЛАВРСКАЯ РАБОТА по направлению подготовки 01.03.02 Прикладная математика и информатика образовательная программа «Прикладная математика и информатика»

Научный руководитель:
Кандидат физико-математических наук,
доцент, департамент информатики
Д. Н. Москвин,

Рецензент:
Кандидат физико-математических наук, доцент
Д. А. Мордвинов

Санкт-Петербург 2023

# Оглавление

## Аннотация

SMT решатели – важный инструмент в анализе программ и генерации тестов. Для проверки программ важна поддержка теории чисел с плавающей точкой, однако не все решатели её имеют, из-за чего некоторые эффективные решатели не могут быть использованы для программ с числами с плавающей точкой.

Данная работа добавляет поддержку чисел с плавающей точкой в KSMT – платформу для использования SMT-решателей на языке Kotlin. В работе реализованы преобразования операций с числами с плавающей точкой в выражения теории битовых векторов, включая распаковку, нормализацию, операции сравнения, арифметические операции и работу с массивами и функциям, а также обратные преобразования выражений из модели решения.

Преобразования протестированы на корректность и эффективность. Результаты экспериментов подтвердили корректность преобразования операций и показали хорошую эффективность работы решателей. Решатель Yices2, в котором нет поддержки теории чисел с плавающей точкой, теперь на многих тестах является одним из самых эффективных решателей в KSMT.

Ключевые слова: число с плавающей точкой, ГОСТ IEEE 754, числа с плавающей точкой, SMT-решатели, символьное выполнение

SMT solvers are an important tool in program analysis and test generation. Support for floating point theory is important for checking programs, but not all solvers have it, which is why some efficient solvers cannot be used for floating point programs.

This work adds support for floating point numbers to KSMT, a platform for using SMT solvers in the Kotlin language. The paper implements transformations of operations with floating point numbers into expressions of the theory of bit vectors, including unpacking, normalization, comparison operations, arithmetic operations and work with arrays and functions, as well as inverse transformations of expressions from the decision model.

The transformations have been tested for correctness and efficiency. The results of the experiments confirmed the correctness of the transformation of operations and showed a good efficiency of the solvers. The Yices2 solver, which does not support floating point theory, is now one of the most efficient solvers in KSMT in many tests.

Keywords: IEEE-754, Floating-point, Satisfiability modulo theories, SMT, SMT solvers, symbolic execution

# Введение

**Задача выполнимости формул в теориях (**англ. *satisfiability modulo theories*, SMT**) –** проблема определения того, является ли формула первого порядка выполнимой по отношению к некоторой логической теории [1]. Примерами таких теорий для SMT-формул являются: теории целых и вещественных чисел, теории списков, массивов, битовых векторов и т. п.

SMT решатели применяются для множества различных задач, таких как проверка программного обеспечения и автоматизированное тестирование.

В SMT существует **теория чисел с плавающей точкой** (Floating-Point, FP), которую поддерживают не все SMT-решатели [4]. Однако существует подход, позволяющий свести задачу к теории битовых векторов (Bit-vector, BV). В контексте задач анализа кода, где широко используются SMT-решатели, теория FP играет важную роль, так как обеспечивает возможность моделирования операций над числами с плавающей точкой.

Проверка программ, включающих арифметику с плавающей точкой, является актуальной и важной задачей, для решения которой используются различные подходы. CoverMe [7] и Pex [13] используют математическую оптимизацию и задачи поиска. Эти подходы направлены на поиск решений ограничений путем изучения пространства поиска.

Средства ограниченной проверки моделей, такие как CBMC [5], преобразуют операции теории чисел с плавающей точкой в битовые векторы и используют SAT-решатели для проверки на разрешимость. Этот подход использует эффективность решателей SAT в быстром поиске выполнимости решений для сложных формул.

Фреймворк CPBPV [6] использует комбинацию ограниченной проверки модели и интервального решателя для создания тестов, нарушающих

выходные ограничения в программе. Этот подход направлен на поиск входных данных, которые заставляют программу выдавать неправильные выходные данные, обеспечивая ценную обратную связь для отладки и улучшения программы.

То, что не все SMT решатели изначально поддерживают операции с плавающей точкой, может ограничить их применимость в областях, где арифметика чисел с плавающей точкой имеет важное значение. Несмотря на существование множества решателей, поддерживающих теорию чисел с плавающей точкой, поддержка этой теории в других эффективных решателях может принести практическую ценность. Разные решатели работают с разной эффективностью на разных задачах и могут использоваться параллельно. Обеспечивая поддержку этой теории, этот проект имеет потенциал для повышения производительности и возможностей решателей SMT. Это может иметь далеко идущие последствия в различных областях, где используются SMT решатели.

**KSMT**[1] – платформа для унифицированного использования SMT-решателей на языке Kotlin. Позволяет удобно использовать разные SMT-решатели, запускать сразу несколько решателей одновременно, ставить таймеры на их работу. Позволяет писать формулы, которые сразу будут работать для разных решателей.

В этом проекте предлагается реализовать преобразование операций теории чисел с плавающей точкой в терминах операций битовых векторов для KSMT. Предложенное преобразование сохраняет выполнимость формулы. Это позволит обрабатывать операции теории чисел с плавающей точкой решателям, которые изначально их не поддерживают.

[1] https://github.com/UnitTestBot/ksmt – (дата обращения: 16.05.2023).

**Цели и задачи исследования.**

Основной целью данной работы является добавление в KSMT реализации операций теории чисел с плавающей точкой в терминах бит-векторных операций.

Для достижения цели можно сформулировать следующие задачи:

- Реализовать преобразования из операций над выражениями теории чисел с плавающей точкой в теорию битовых векторов.
- Реализовать обратное преобразование модели, возвращаемой решателем, в теорию чисел с плавающей точкой.
- Оценить эффективность реализованного подхода.

**Основные результаты**

Преобразование из теории чисел с плавающей точкой в теорию битовых векторов было реализовано. Эффективность работы решателей с преобразованиями из этой работы сопоставима, а в некоторых случаях лучше, чем эффективность решателей, поддерживающих теорию чисел с плавающей точкой.

**Структура работы**

Обзор литературы описывает существующие SMT решатели и особенности работы с числами с плавающей точкой. Также рассматриваются различные подходы к трансформации операций с плавающей точкой.

Во главе 1 описано преобразование чисел с плавающей точкой: процесс распаковки и нормализации чисел с плавающей точкой, а также операции сравнения, арифметические операции и преобразования типов. Также рассматриваются работа с массивами, функциями и кванторами.

Глава 2 посвящена описыванию обратному преобразованию выражений модели SMT-решателей.

Последняя глава 3 описывает подход к проверке корректности и эффективности реализованного преобразования операций и представляет результаты этих проверок.

# Обзор литературы

## SMT решатели

SMT решатели позволяют проверять, существует ли модель (набор значений переменных), которая удовлетворяет заданной формуле в соответствии с определенными теориями [10]. Другими словами, существует набор значений переменных, при котором заданная формула является истинной, выполняется. SMT-решатели могут вернуть три основных результата проверки:

- SAT (удовлетворимость): формула имеет модель и решатель может предоставить конкретные значения переменных, удовлетворяющие формуле.
- UNSAT (неудовлетворимость): формула невыполнима в рамках заданных теорий.
- UNKNOWN (неизвестно): решатель не в состоянии дать определенный ответ на проверку разрешимости. Это может произойти, например, когда решатель не может найти решение за указанное время.

## Числа с плавающей точкой

Одна из широко используемых теорий SMT – теория чисел с плавающей точкой. Существуют также теории и операции, которые могут обращаться к решателю теории чисел с плавающей точкой. Например, теория массивов или теория неинтерпретированных функций. Также, выражения теории FP могут использоваться в кванторах всеобщности и существования.

**Число с плавающей точкой** — экспоненциальная форма представления вещественных чисел, где число хранится в виде знака, мантиссы и экспоненты (порядка, показателя степени). Это широко используемый метод представления действительных чисел в компьютерных системах. Он позволяет эффективно вычислять числовые значения с различными порядками величины и точности. Однако он также создает проблемы и ограничения, такие как ошибки округления, переполнение, недополнение и исключительные значения. Более того, разные спецификации могут иметь различное поведение и свойства, что затрудняет обеспечение переносимости и надежности программ с плавающей точкой на разных платформах и средах.

**IEEE-754** [9] — это техническая спецификация, в которой изложены правила выполнения арифметики с плавающей точкой. Он определяет формат и поведение для арифметики с плавающей точкой. Он определяет условия исключений и их обработку по умолчанию. Стандарт решил многие проблемы, обнаруженные в различных реализациях с плавающей точкой, что затрудняло их надежное и совместимое использование.

Это наиболее важных стандартов для выполнения арифметики с плавающей точкой в средах компьютерного программирования является IEEE 754, в нем описываются методы, форматы и исключения для выполнения арифметических операций с представленными числами с плавающей точкой. IEEE 754 направлен на обеспечение согласованных и предсказуемых результатов вычислений с плавающей точкой в различных реализациях.

В этой работе под числом с плавающей точкой имеется в виду число стандарта IEEE-754.

Мантисса представляет собой значимые цифры числа, а показатель степени представляет положение десятичной точкой относительно начала числа.

Хотя арифметика выполняется так же, как и с обычными числами, результат может содержать больше цифр, чем заранее определенное число, и должен быть округлен, чтобы поместиться в выделенное пространство. Это создает серьезную проблему для анализа чисел с плавающей точкой, поскольку необходимость округления после каждой операции подразумевает, что сложение и умножение больше не сохраняют своих ассоциативных или дистрибутивных свойств.

Не все числа могут быть точно представлены в выбранном формате, поэтому может потребоваться округление для приближенного представления таких чисел. Существует несколько основных режимов округления:

- Округление к ближайшему, при одинаковом расстоянии к четному
- Округление к ближайшему числу, при одинаковом расстоянии в сторону от нуля
- Округление вниз, в сторону нуля
- Округление вверх, в сторону от нуля
- Округление к нулю, дробная часть числа отбрасывается

Режим округления чисел с плавающей точкой может быть выбран в соответствии с требованиями конкретной задачи и контекстом, в котором используются числа с плавающей точкой.

Процесс округления гарантирует, что мантисса укладывается в заранее определенное количество битов, но он не решает проблему того, что показатель степени становится слишком высоким или слишком низким. В более ранних системах с плавающей точкой обнаружение и обработка этих экстремальных сценариев было серьезным препятствием. Чтобы преодолеть эти проблемы, IEEE-754 ввел серию чисел с плавающей точкой, которые представляют положительные и отрицательные бесконечности, нули и набор

чисел с фиксированной точкой, известных как ненормальные или субнормальные числа. Некоторые операции не определены, например, деление нуля на ноль. Стандарт решает эти проблемы с помощью NaN (англ. *Not-a-Number*, нечисло).

Теория чисел с плавающей точкой включают в себя 30 выражений:

- Арифметические операции:
  - Abs – абсолютная величина
  - Negation – изменение знака
  - Add – сложение
  - Sub – вычитание
  - Mul – умножение
  - Div – деление
  - FusedMulAdd – умножение-сложение с однократным округлением
  - Sqrt – квадратный корень
  - Rem – остаток от деления
  - RoundToIntegral – округление до целого
- Числа
  - Значения
  - Переменные
- Сравнения
  - Min – минимум
  - Max – максимум
  - LessOrEqual – меньше или равно
  - Less – меньше
  - GreaterOrEqual – больше или равно
  - Greater – больше

    - FpEqual – равенство теории чисел с плавающей точкой
- Предикаты
    - IsNormal – число нормальное
    - IsSubnormal – субнормальное
    - IsZero – является нулем (может быть положительным или отрицательным)
    - IsInfinite – является бесконечностью
    - IsNaN – является NaN
    - IsNegative – отрицательно
    - IsPositive – положительно
- Преобразования
    - FpToBv – преобразование в целое число типа битового вектора
    - FpToIEEEBv – преобразование в побитово равный бит-вектор по правилам стандарта IEEE
    - FpFromBv – преобразование из бит-вектора в побитово равное число с плавающей точкой
    - FpToFp – преобразование в число с плавающей точкой другого размера
    - BvToFp – преобразование из целого числа типа бит-вектора в число с плавающей точкой

Кроме того, нужно поддержать выражения, которые могут обращаться к теории чисел с плавающей точкой:

- Теория массивов
    - ArrayConst – константный массив
    - ArraySelect – операция обращения к массиву по индексу
    - ArrayStore – запись в массив

    - ArrayLambda – функция с типом массива с аргументами и телом. При обращении к массиву индексы – это аргументы функции, а значение массива по индексам – это результат функции с индексами в качестве аргументов. ArrayLambda состоит из домена массива (множество индексов, аргументы функции), а также выражения, которое определяет значения массива для каждого индекса.
    - Переменные
- Неинтерпретированные функции
- Условное выражение Ite (*If Then Else*)
- Eq – равенство
- Кванторы
    - ExistentialQuantifier – квантор существования
    - UniversalQuantifier – квантор всеобщности

## Подходы к трансформации операций с плавающей точкой

Традиционный подход к операциям с плавающей точкой представляет собой четырехступенчатый алгоритм: распаковка, арифметической операция, округление и упаковка [11].

1. **Этап распаковки**: число разделяется на три части: знак, экспонента и мантисса. Скрытый бит добавляется к мантиссе, и из экспоненты убирается смещение. Также могут быть нормализованы показатели и значения субнормальных чисел, если это необходимо.

2. **Этап арифметической операции**: преобразование и арифметические операции выполняются над тремя битовыми векторами. Битовые векторы, возможно, придется расширить.

3. **Этап округления**: этот этап округляет расширенные битовые векторы до ближайшего числа с плавающей точкой нужного формата. Последние биты в значении помогают решить, насколько близко битовый вектор находится к ближайшему допустимому значению, а режим округления помогает решить, в каком направлении округлять. Битовый вектор экспоненты также проверяется на недо- или переполнение при округлении, чтобы получить правильную плавающую точку, например бесконечность может быть получена, если показатель степени слишком велик для желаемого формата.

4. **Стадия упаковки**: на последнем этапе финальное выражение с плавающей точкой формируется путем соединения трех бит-векторов.

Брейн и др. ввели технику под названием **SymFPU**, которая преобразует выражения теории чисел с плавающей точкой в выражения теории битовых векторов [3]. Они создали набор преобразований, которые могут быть использованы с решателями SMT, чтобы они могли обрабатывать теорию чисел с плавающей точкой. Исследователи интегрировали SymFPU с SMT-решателями Bitwuzla [12] и CVC4 [2] и протестировали его на широком спектре тестов и обнаружили, что он смог превзойти все предыдущие системы с точки зрения производительности.

Распаковка выражений с плавающей точкой производится только один раз, упаковка обратно в число с плавающей точкой происходит только после выполнения всей цепочки выражений. Это сделано для того, чтобы избежать лишнего шага упаковки и распаковки между каждой операцией. Это делает весь процесс более эффективным и позволяет использовать неупакованные форматы, которые в противном случае были бы слишком медленными для упаковки и распаковки. Выражения принимают и возвращают числа в распакованном и нормализованном формате.

Некоторые значения, такие как бесконечность, NaN и ноль, являются специальными и имеют флаги. Это упрощает операцию, потому что мы можем сначала просто посмотреть на флаги и узнать, установлен ли какой-либо из них. Если это так, нам не нужно делать арифметическую операцию, и мы можем просто использовать значение флага в качестве результата. Предикаты реализованы, используя флаги из распаковки.

Экспонента — это число, которое может быть положительным или отрицательным и не имеет никакого смещения. Это означает, что мы можем уменьшить объем работы, которую должен выполнять множитель, и использовать некоторые существующие методы, которые хорошо работают для чисел, которые могут быть положительными или отрицательными. Мантисса является частью значения, которое имеет наибольшее количество цифр, мы нормализуем его, поэтому он всегда имеет единицу в качестве первой цифры. Если значение очень мало (в случае с субнормальными числами), мы корректируем его так, чтобы первая цифра по-прежнему была единицей. Это усложняет упаковку и распаковку, но также облегчает округление значения, потому что нам не нужно искать первую цифру.

**FP bit-blasting API** [8], используемый в средстве ограниченной проверки модели для C++, ESBMC, может использовать различные решатели SMT в качестве бэкенда. Авторы оценивают, насколько хорошо функционирует API, тестируя его на группе тестов, полученных из реальных приложений, таких как цифровая обработка сигналов, обработка изображений и системы управления. Полученные результаты показывают, что API способен точно и эффективно обрабатывать программы FP, и что выбор решателя SMT оказывает заметное влияние на время, необходимое для проверки программы, и на объем используемой памяти. В реализации преобразования выражений используется метод традиционной трансформации, описанный выше, т. е. в

начале трансформации каждой операции производится распаковка аргументов, а в конце производится упаковка результата.

## Результаты главы

Среди представленных решений нет такого, которое может сразу использоваться для KSMT, так как существующие библиотеки написаны на языке C++, в них не реализованы операции на стыке с теориями массивов и неинтерпретированных функциями, а также обратное преобразование выражений модели.

Кроме того, возникла гипотеза, что можно повысить эффективность, не делая нормализацию простых выражений аналогично FP bit-blasting API, но если делать, то только один раз, как в SymFPU.

Добавление поддержки теории чисел с плавающей точкой включает в себя не только преобразование выражений с плавающей точкой, но и преобразование моделей из битовых векторов в числа с плавающей точкой.

При этом пользователь не должен делать эти преобразования сам. То есть нужна следующая обертка над SMT решателем:

- Обертка может принимать выражения с типом FP
- Обертка позволяет получать модели, содержащие выражения с типом FP
- Внутри обертка производит следующие операции:
    - Преобразует выражения с типом FP в выражения с битовыми векторами
    - Передаёт полученные выражения с битовыми векторами в SMT решатель

- Получает от SMT решателя модель, содержащую выражения с битовыми векторами
- Производит обратное преобразование выражений с битовыми векторами, содержащихся в модели, в выражения теории FP

# 1 Преобразование выражений

SMT-формула – это ориентированный ациклический граф выражений. В KSMT реализован обход этого графа, который позволяет для реализации трансформации выражения переопределить только трансформацию выражений, которые нужно изменить. Трансформер обойдет все выражение и все аргументы каждого выражения и вызовет переопределенные трансформации выражений, где это нужно.

В этой главе рассматривается реализация распаковки, преобразований операций сравнения, арифметических операций, а также операций теорий массивов, теории неинтерпретированных функций и кванторов.

## 1.1 Распаковка и нормализация

В рамках данной работы для представления чисел используется подход, аналогичный SymFPU. В отличие от SymFPU, добавлена оптимизация нормализации для выражений сравнения и преобразования в выражение теории битовых векторов. При распаковке сохраняется как нормализованные битовые векторы для мантиссы и экспоненты, так и исходные. В результате сложных операций исходные бит-векторы не сохраняются и дальше используются только нормализованные. Это позволяет избежать нормализаций в итоговом выражении для простых операций. При этом при необходимости нормализации все еще получается избежать лишнего шага упаковки и распаковки между каждой операцией.

Переменные типа с плавающей точкой трансформируются следующим образом:

- Создается новая переменная типа бит-вектора размера, соответствующего типу с плавающей точкой
- Сохраняется соответствие исходной и новой переменных для восстановления результатов
- Производится распаковка, создаются флаги для специальных значений, нормализованные мантисса и экспонента
- Сохраняется и нормализованное число, и исходные битовые вектора
- После сложных операций, где требуется нормализация, используются только нормализованные битовые вектора

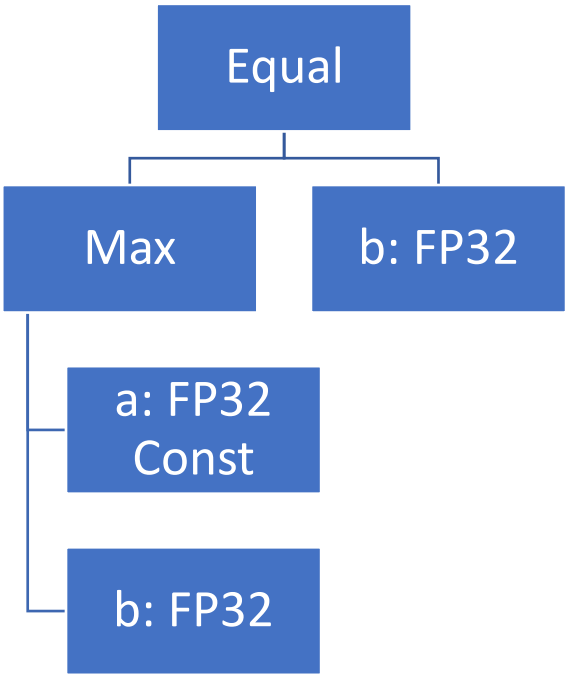


*Рисунок 1. max (a, b) = b*

На рисунке 1 пример выражения, в котором нормализация не требуется, так как используются только простые выражения. Это значительно ускоряет работу SMT-решателей.

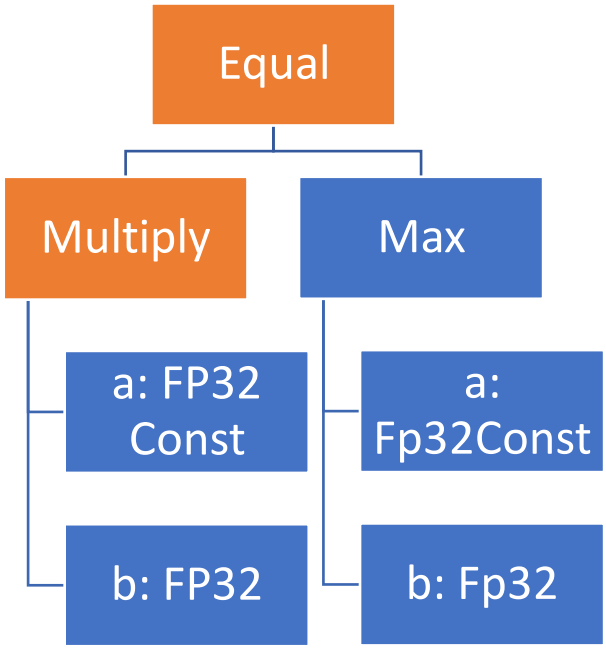


*Рисунок 2. a×b =max (a, b)*

На рисунке 2 пример выражения, в котором нормализация требуется, но не во всех выражениях. Операция умножения использует нормализованные распакованные выражения и возвращает распакованный нормализованный результат. Для операции сравнения уже нет исходных ненормализованных битовых векторов, поэтому выражение максимума из двух чисел тоже приходится нормализовать для сравнения с результатом умножения.

## 1.2 Операции сравнения

Трансформации операций сравнения реализованы для двух случаев:

- Если у обоих аргументов существуют исходные ненормализованные битовые вектора, используются только они, что упрощает итоговое выражение
- Иначе операции используют нормализованное представление

В обоих случаях операции сравнения сводятся к сравнению знака, экспоненты и мантиссы.

## 1.3 Арифметические операции и преобразования типов


*Рисунок 3. Общий алгоритм трансформации арифметических операций*

Арифметические операции и преобразования типов реализованы в 4 этапа (см. рисунок 3). На первом этапе арифметическая операция вычисляется в расширенных бит-векторах без потери точности.

Процесс нормализации упрощается из-за использования нормализованных аргументов. Так, например, после арифметического умножения в мантиссе единица есть либо в первом, либо во втором бите, поэтому не нужно добавлять в итоговое выражение проверку каждого бита.

**Округление** реализовано следующим образом:

- Проверяется, есть ли переполнение или потеря точности (число слишком мало для представления)
- Строится выражение *guard bit* (следующий бит после нужной точности в мантиссе). Для субнормальных чисел при поиске позиции этого бита учитывается значение экспоненты
- Строится выражение *sticky bit* (все ли нули после *guard bit*)
- Создается скорректированная мантисса и экспонента
- Вычисляется выражение для округления в зависимости от стратегии округления (есть 5 разных) и от четности, знака, *sticky* и *guard* битов
- Вычисляется округление для особых случаев. Например, для разных стратегий округления переполнение округляется либо в бесконечность, либо в максимальное возможное значение

## 1.4 Операции преобразования

Преобразование операции FpToIEEEBv – это операция упаковки в бит-вектор. В случае, если исходный бит-вектор выражения сохранился, он и возвращается. Если нет, то производится процесс, обратный распаковке и нормализации: в экспоненту добавляется смещение, убирается нормализация.

FpFromBv реализовано с помощью распаковки, так же как при обработке констант теории чисел с плавающей точкой. В том числе сохраняется исходный битовый вектор.

FpToFp преобразование в число с плавающей точкой другого размера в случае, если размер увеличен, просто расширяет экспоненту и мантиссу, иначе реализуется через округление в нужный размер.

BvToFp преобразование из целого числа теории битовых векторов в число с плавающей точкой делает мантиссу равной этому числу, а экспоненту равной сдвигу, при котором число становится целым. Полученное число преобразуется в число нужного размера с помощью FpToFp.

FpToBv преобразование в целое число теории битовых векторов сдвигает мантиссу на значение экспоненты, делает округление и возвращает мантиссу.

## 1.5 Массивы и функции

Преобразование константного массива (массив, где по всем индексам одно значение) реализовано следующим образом:

- Создается тип массива, где все типы с плавающей точкой заменяются на бит-векторные типы
    - Для этого в типах индексов и значения рекурсивно заменяются типы без чисел с плавающей точкой
    - Индексы и значения тоже могут типами массивов
- Создается новый константный массив, где значение упаковывается в бит-вектор, если оно было числом с плавающей точкой

Преобразование переменных теории массивов, содержащих числа с плавающей точкой:

- Аналогично создается новый тип без чисел с плавающей точкой
- Создается переменная новая типа
- Запоминается соответствие новой и старой переменных

Преобразование записи в массив:

- Массив, индексы и значение трансформируются, в типе трансформированного массива нет чисел с плавающей точкой

- Трансформированные индексы и значение упаковываются в бит-векторы, если они являются выражением теории чисел с плавающей точкой
- Возвращается запись в массив по упакованным индексам упакованного значения

Преобразование операции обращения к массиву по индексу:

- Создается обращение к массиву по упакованным индексам
- В случае, если результат – выражение теории чисел с плавающей точкой, оно распаковывается

Преобразование ArrayLambda:

- Преобразуются определения аргументов
    - Если в них нет чисел с плавающей точкой, ничего не изменяется
    - Если это переменная, то создается новая с преобразованным типом
    - Если это функция, то возвращается новая функция без чисел с плавающей точкой
    - Запоминается соответствие нового и старого выражений

В массивах после преобразования вместо чисел с плавающей точкой в индексах, значениях и теле массивов-лямбд хранятся выражения с битовыми векторами, являющимися упакованными числами с плавающей точкой.

В операциях получения значения из массива результат распаковывается, если до преобразований он является числом с плавающей точкой.

Преобразования функций реализованы аналогично, то есть создаются новые типы и декларации во всех аргументах, далее создается новое итоговое выражение. Преобразованные выражения в результате остаются эквивалентными.

## 1.6 Кванторы

При преобразовании кванторов создаются новые типы и декларации в аргументах, где были числа с плавающей точкой, тело квантора преобразуется, и создается новое итоговое выражение. Преобразованный квантор в результате эквивалентен исходному.

## 1.7 Результаты главы

В этой главе описано, какие операции нужно было трансформировать для преобразования из теории чисел с плавающей точкой. Все нужные операции были реализованы.

Затем описана распаковка и нормализация. В том числе приведена оптимизация нормализаций для простых выражений. Далее описывается, как выглядит трансформация различных операций.

# 2 Обратное преобразование выражений из модели

## 2.1 Реализация

В платформе KSMT есть ModelEvaluator, благодаря которому достаточно определить интерпретацию только базовым выражениям:

- Переменным типа FP
- Массивам
  - Константным (где во всех индексах одно значение)
  - Записям в массив по индексу
  - Массивам-лямбдам
- Функциям
  - Неинтерпретированным функциям
  - Преобразованию типа из функции в массив FunctionAsArray (Многие SMT-решатели поддерживают представление функций в виде массивов, где ключи массива соответствуют аргументам функции, а значения массива - результатам функции для соответствующих аргументов)

Интерпретация поддерживает функции и массивы, поэтому содержит переменные, значения и может содержать значение по умолчанию. Если в типе выражения нет чисел с плавающей точкой, то просто возвращается интерпретация, выданная используемым решателем.

Иначе

- Достается связанное с этим выражением бит-векторное выражение

- Для него из используемого решателя достается интерпретация
- Все части этой интерпретации трансформируются
    - Числа упаковываются в тип с плавающей точкой
    - В массивах трансформируется все:
        - Переменные
        - Тело лямбды
        - Значения
        - В случае преобразования функции в массив достается интерпретация для этой функции

## 2.2 Результаты главы

Эта глава была целиком посвящена обратному преобразованию решений из битовых векторов в числа с плавающей точкой. Была описана модель SMT-решателей и то, что нужно от обертки. Далее было описано, что было необходимо реализовать для корректной работы модели и были показаны шаги для необходимых преобразований. Обертка была реализована, поэтому теперь поддержка теории с плавающей точкой легко добавляется в любые решатели в KSMT.

# 3 Тестирование корректности и эффективности

## 3.1 Подход к проверке корректности преобразования

Для тестирования корректности в этой работе используются SMT-решатели, поддерживающие числа с плавающей точкой: решатель проверяет на эквивалентность трансформированное выражение и исходное. Если что-то было реализовано неверно, решатель найдет пример, когда выражения действительно неравны. Иначе решатель выдаст результат, что таких примеров нет и подтвердит корректность реализации.

Кроме проверки отдельных выражений, проверялись также результаты выражений из тестов SMT-LIB Benchmarks [1] Результаты решений решателей с применением реализованного преобразования сравнивались с результатами решателя Z3. Кроме того, аналогично протестирована преобразованная модель.

## 3.2 Результат проверки корректности

Проверка эквивалентности проводилась для нескольких форматов представления чисел с плавающей точкой: 16, 32 и 64-битных форматах.

Операции умножения, деления, остатка от деления, умножения-сложения (fused multiply add) были исключены из тестирования, так как проверка их эквивалентности долго работает.

Проверка показала корректность реализованного преобразования.

При проверке на наборе бенчмарков SMT-LIB с лимитом на время выполнения в 1 секунду на тест ошибок в преобразованиях и в моделях не выявлено.

## 3.3 Подход к проверке эффективности

Эффективность решателя SMT определяется его способностью быстро и правильно решать SMT-формулы. Эффективность так же проверялась на тестах SMT-LIB Benchmarks для трех логик без кванторов:

- QF_FP – только логика теории чисел с плавающей точкой
- QF_BVFP – к предыдущей логике добавляется теория битовых векторов
- QF_ABVFP – к предыдущей логике добавляется теория массивов

Тестировались решатели Z3, Bitwuzla и Yices2. Yices2 сам не поддерживает теорию чисел с плавающей точкой, Bitwuzla для ее поддержки использует SymFPU. Каждый пример был запущен 5 раз, сравнивается среднее время работы по всем запускам.

Сравниваются 3 результата по каждому решателю:

- Результат самих решателей (в Yices2 его нет)
- Решателей в обертке, созданной в данной работе с оптимизацией нормализации. В результатах название решателя в обертке начинается с *fp2bv-*
- Решатель в обертке без этой оптимизации. В результатах название начинается с *fp2bv-*, а в конце указано отсутствие оптимизации *(no opt)*.

Были получены данные по времени работы преобразования и решения по каждому тесту, дальше они были разбиты по логикам и решателям. По каждому решателю данные отсортированы от самого быстрого до самого медленного времени работы.

Если результат unknown, на графике он учитывается как 5 секунд. В графу unknown попадают тесты, в которых решатели не поддерживают какое-

то выражение, завершаются с ошибкой или не успели решить формулу за отведенное время.

## 3.4 Результат проверки эффективности

Каждый решатель выделен одним цветом, отличить результаты можно по типу линии, все это указано на легенде около графиков.

На графиках по горизонтальной оси отмечен номер теста в отсортированном по времени порядке. По вертикальной оси изменяется время в секундах. В таблицах указано количество каждого вида результатов по набору тестов.

*Таблица 1. Результаты по всем примерам*

| | **SAT** | **UNSAT** | **UNKNOWN** |
|---|---|---|---|
| **Z3** | 44015 | 23732 | 753 |
| **fp2bv-Z3** | 40421 | 20030 | 8049 |
| **fp2bv-Z3 (no opt)** | 42575 | 22245 | 3680 |
| **fp2bv-Yices** | 42934 | 22394 | 3172 |
| **fp2bv-Yices (no opt)** | 43577 | 23062 | 1861 |
| **Bitwuzla** | 44334 | 23991 | 175 |
| **fp2bv-Bitwuzla** | 44338 | 24003 | 159 |
| **fp2bv-Bitwuzla (no opt)** | 44161 | 23993 | 346 |

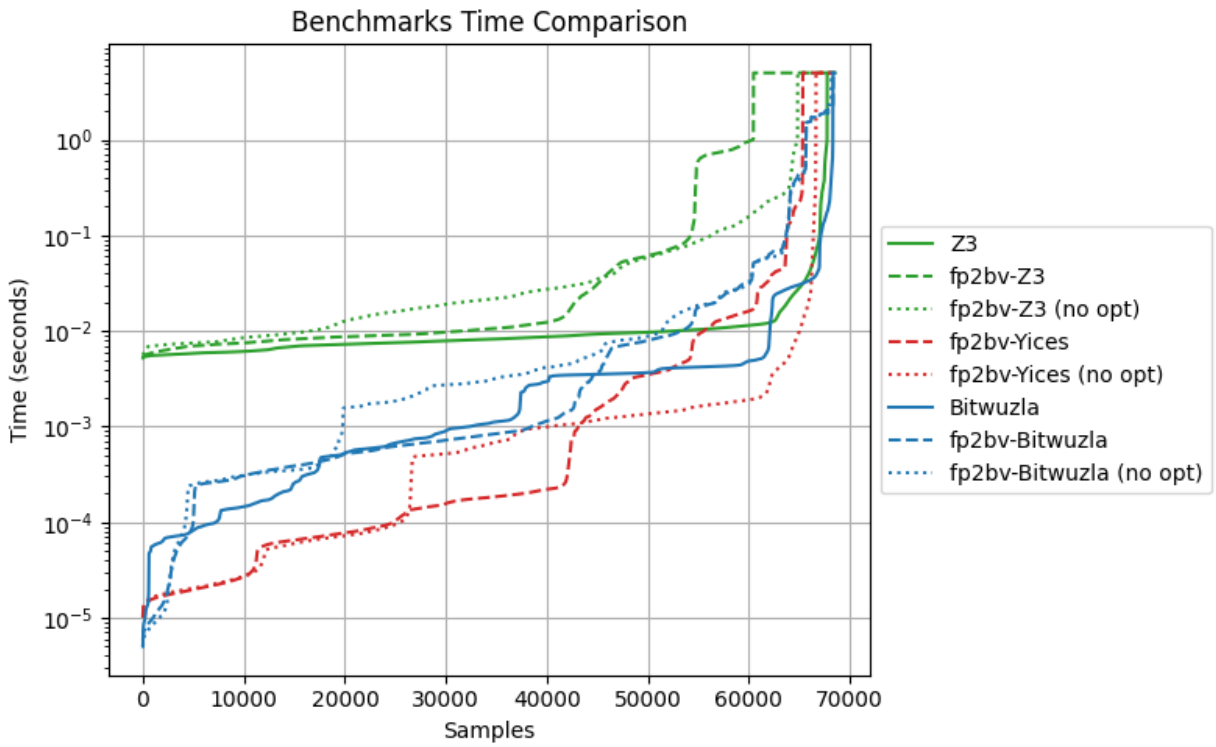


*Рисунок 4. Результаты по всем примерам*

По графику и таблице результатов измерения по всем логикам (см. рис. 4) видно несколько вещей:

- Эффективность значительно зависит от решателя: в целом результаты каждого решателя находятся рядом с результатами в обертках
- Влияние обертки и ее оптимизации на разных решателях проявляется по-разному: например, Z3 преобразования замедлили, а в Bitwuzla в некоторых случаях преобразования улучшили результат
- Время работы с преобразованием, реализованным в данной работе, сопоставимо с временем работы без нее
- Решатель Yices2, который не поддерживал теорию чисел с плавающей точкой, теперь может использоваться как один из

самых эффективных решателей в KSMT для этой теории, что было одной из главных целей этого проекта

- В таблице с результатами видно, что сама обертка и оптимизация работают с разным успехом на разных решателях. Так, например, Bitwuzla в обертке с оптимизацией решила больше формул на UNSAT, чем все остальные решатели, когда в Z3 и Yices2 в обертке с оптимизацией больше всего UNKNOWN результатов.

*Таблица 2. Результаты по логике QF_ABVFP*

| | **SAT** | **UNSAT** | **UNKNOWN** |
|---|---|---|---|
| **Z3** | 12069 | 1748 | 515 |
| **fp2bv-Z3** | 12141 | 1805 | 386 |
| **fp2bv-Z3 (no opt)** | 12156 | 1812 | 364 |
| **fp2bv-Yices** | 12330 | 1991 | 11 |
| **fp2bv-Yices (no opt)** | 12330 | 1964 | 38 |
| **Bitwuzla** | 12327 | 1978 | 27 |
| **fp2bv-Bitwuzla** | 12327 | 1989 | 16 |
| **fp2bv-Bitwuzla (no opt)** | 12327 | 1982 | 23 |

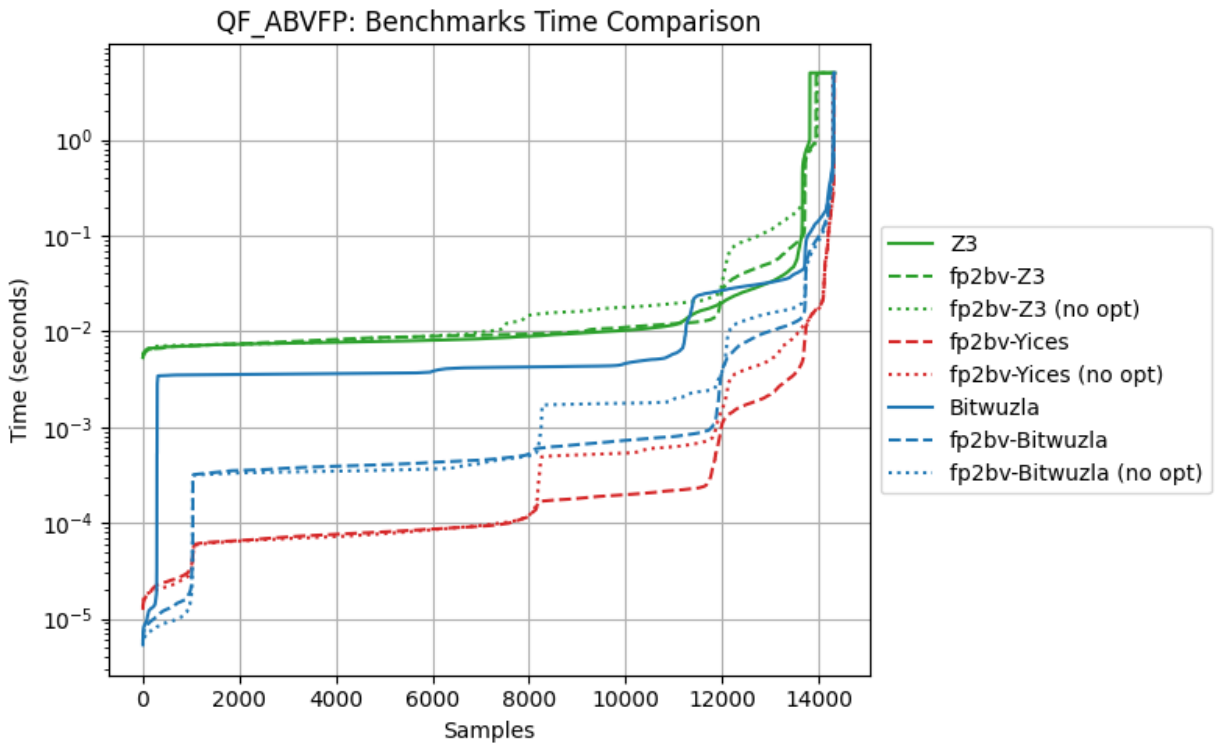


*Рисунок 5. Результаты логики QF_ABVFP*

*Таблица 3. Результаты по логике QF_BVFP*

| | SAT | UNSAT | UNKNOWN |
|---|---|---|---|
| **Z3** | 12301 | 1955 | 25 |
| **fp2bv-Z3** | 12100 | 1794 | 387 |
| **fp2bv-Z3 (no opt)** | 12179 | 1807 | 295 |
| **fp2bv-Yices** | 12316 | 1953 | 12 |
| **fp2bv-Yices (no opt)** | 12316 | 1957 | 8 |
| **Bitwuzla** | 12315 | 1961 | 5 |
| **fp2bv-Bitwuzla** | 12315 | 1962 | 4 |
| **fp2bv-Bitwuzla (no opt)** | 12315 | 1962 | 4 |

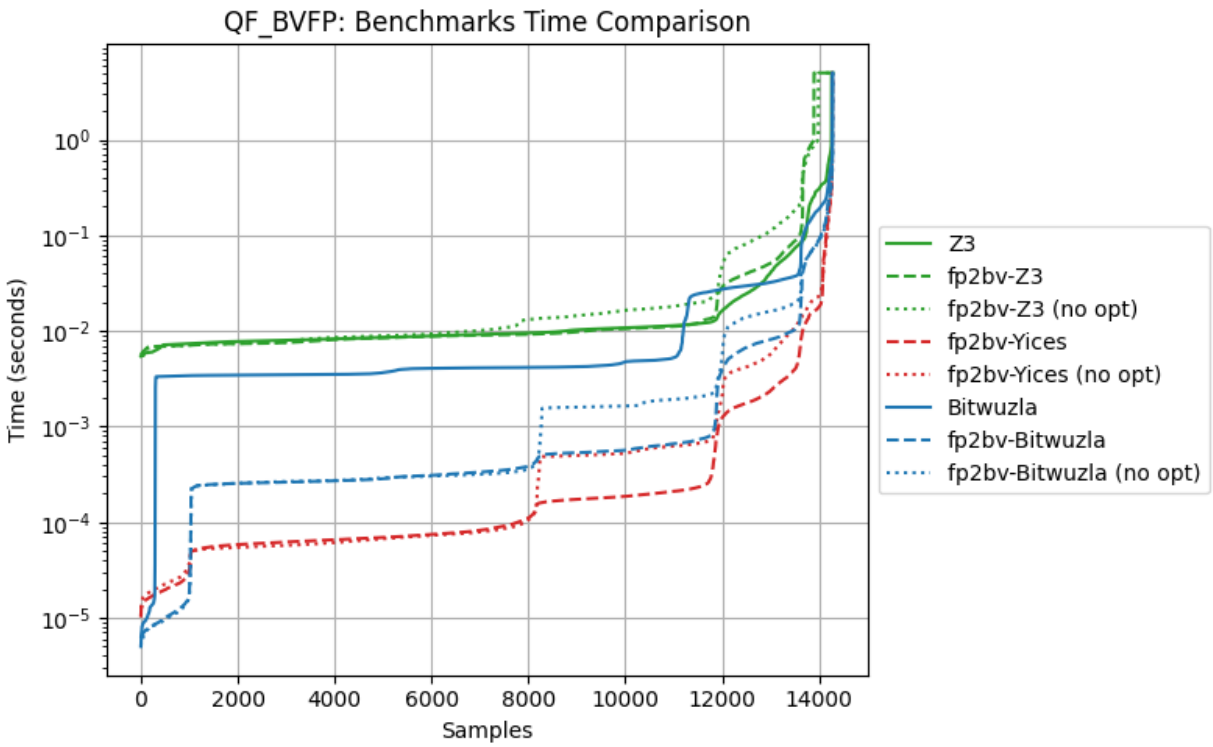


*Рисунок 6. Результаты логики QF_BVFP*

По результатам измерения по логикам QF_BVFP, QF_ABVFP (см. рис. 5, 6) можно отметить:

- На этих логиках преобразование эффективнее, чем в среднем по всем логикам
- Решатель Bitwuzla стал быстрее с нашим преобразованием на этих логиках
- Решатель Yices2 на этих логиках быстрее других решателей
- В логике с массивами и преобразование, и оптимизация работают лучше всего, чем в других

*Таблица 4. Результаты по логике QF_FP*

| | SAT | UNSAT | UNKNOWN |
|---|---|---|---|
| **Z3** | 19645 | 20029 | 213 |
| **fp2bv-Z3** | 16180 | 16431 | 7276 |
| **fp2bv-Z3 (no opt)** | 18240 | 18626 | 3021 |
| **fp2bv-Yices** | 18288 | 18450 | 3149 |
| **fp2bv-Yices (no opt)** | 18931 | 19141 | 1815 |
| **Bitwuzla** | 19692 | 20052 | 143 |
| **fp2bv-Bitwuzla** | 19696 | 20052 | 139 |
| **fp2bv-Bitwuzla (no opt)** | 19519 | 20049 | 319 |

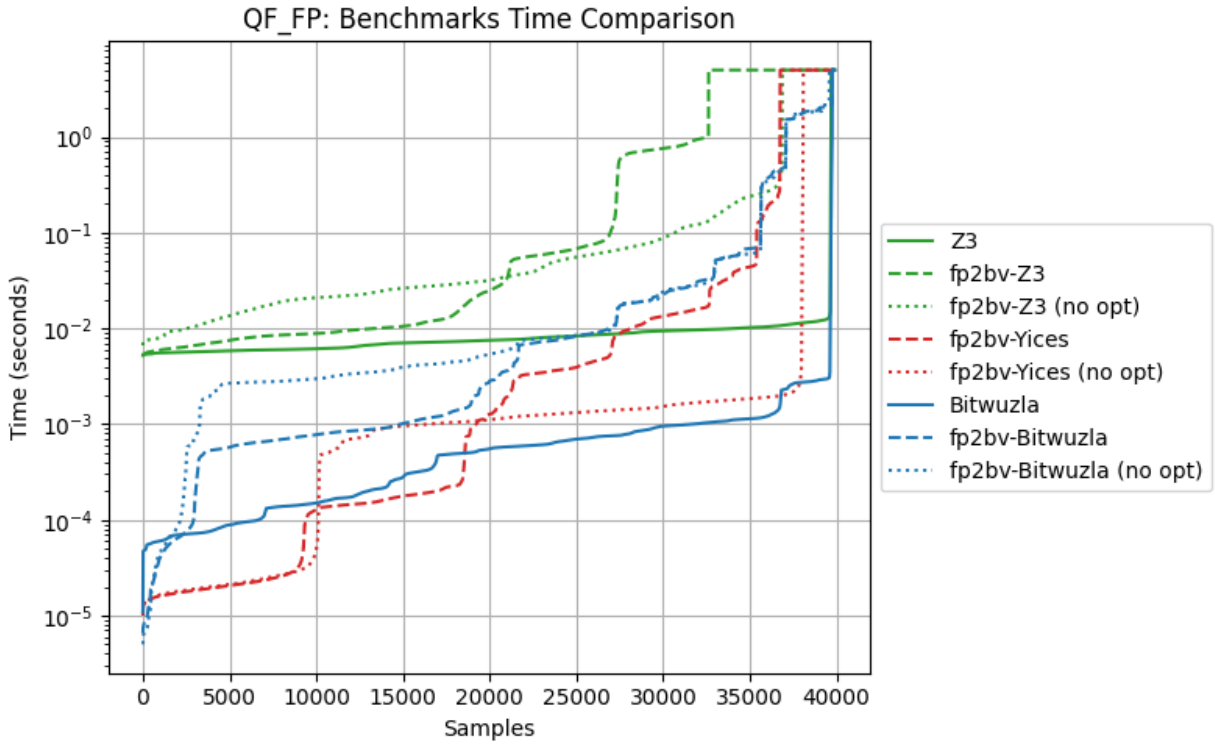


*Рисунок 7. Результаты QF_FP*

По результатам логики QF_FP (см. рис. 7) видно, что:

- На тестах этой логики ни у какого решателя нет явного преимущества
- Оптимизация нормализации работает дольше на сложных тестах, где время работы дольше у всех решателей
- Результаты значительно зависят не только от решателя, но и от логики

## 3.5 Результаты главы

Для проверки результатов использованы решатели SMT для сравнения преобразованного выражения с исходным. Проверка показала корректность реализованного преобразования. По результатам тестирования можно сделать вывод о применимости реализованного подхода.

## Заключение

В рамках работы были реализованы преобразования операций над выражениями теории чисел с плавающей точкой в теорию битовых векторов. Это включало методы распаковки и нормализации, преобразование операций сравнения, арифметических операций, а также операций на стыке с теориями с массивов, функций, неинтерпретированных функций и кванторов.

Было реализовано обратное преобразование модели, возвращаемой решателем, в теорию чисел с плавающей точкой. Это обратное преобразование позволяет получить результаты в формате, привычном для работы с числами с плавающей точкой.

Проведена оценка реализованного подхода. Результаты тестирования показали, что реализованные преобразования обеспечивают корректность и высокую эффективность работы решателей.

Таким образом, данная работа вносит важный вклад в работу с числами с плавающей точкой в KSMT, позволяя большему числу эффективных решателей использоваться в работе с числами с плавающей точкой.

# Список литературы


[1] Barrett Clark and Tinelli Cesare. Satisfiability Modulo Theories // Handbook of Model Checking / ed. by Clarke Edmund M., Henzinger Thomas A., Veith Helmut, and Bloem Roderick. — Cham : Springer International Publishing, 2018. — P. 305–343. — ISBN: 978-3-319-10575-8. — Access mode: https://doi.org/10.1007/978-3-319-10575-8_11.

[2] Brain Martin, Schanda Florian, and Sun Youcheng. Building better bit-blasting for floating-point problems // Tools and Algorithms for the Construction and Analysis of Systems: 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part I 25 / Springer. — 2019. — P. 79–98. — Access mode: https://doi.org/10.1007/978-3-030-17462-0_5.

[3] Barrett Clark, Barbosa Haniel, Brain Martin, Ibeling Duligur, King Tim, Meng Paul, Niemetz Aina, Nötzli Andres, Preiner Mathias, Reynolds Andrew, et al. CVC4 at the SMT competition 2018 // arXiv preprint arXiv:1806.08775. — 2018. — Access mode: https://doi.org/10.48550/arXiv.1806.08775.

[4] Brain Martin, Tinelli Cesare, Rümmer Philipp, and Wahl Thomas. An automatable formal semantics for IEEE-754 floating-point arith- metic // 2015 IEEE 22nd Symposium on Computer Arithmetic / IEEE. — 2015. — P. 160–167. — Access mode: https://doi.org/10.1109/ARITH.2015.26.

[5] Clarke Edmund, Kroening Daniel, and Lerda Flavio. A tool for check- ing ANSI-C programs // Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Held as Part of the Joint European

Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29-April 2, 2004. Proceedings 10 / Springer. — 2004. — P. 168–176. — Access mode: https://doi.org/10.1007/978-3-540-24730-2_15.

[6] Collavizza Hélène, Michel Claude, Ponsini Olivier, and Rueher Michel. Generating Test Cases inside Suspicious Intervals for Floating-Point Number Programs // Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis. — New York, NY, USA : Association for Computing Machin- ery. — 2014. — CSTVA 2014. — P. 7–11. — Access mode: https://doi.org/10.1145/2593735.2593737.

[7] Fu Zhoulai and Su Zhendong. Achieving high coverage for floating- point code via unconstrained programming // ACM SIGPLAN Notices. — 2017. — Vol. 52, no. 6. — P. 306–319. — Access mode: https://doi.org/10.1145/3062341.3062383.

[8] Gadelha Mikhail R, Cordeiro Lucas C, and Nicole Denis A. An efficient floating-point bit-blasting API for verifying C programs // Software Verification: 12th International Conference, VSTTE 2020, and 13th International Workshop, NSV 2020, Los Angeles, CA, USA, July 20–21, 2020, Revised Selected Papers 13 / Springer. — 2020. — P. 178–195. — Access mode: https://doi.org/10.48550/arXiv.2004.12699.

[9] Kahan William. IEEE standard 754 for binary floating-point arithmetic // Lecture Notes on the Status of IEEE. — 1996. — Vol. 754, no. 94720-1776. — P. 11. — Access mode: https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF.

[10] de Moura Leonardo and Bjørner Nikolaj. Z3: An Efficient SMT Solver // Tools

and Algorithms for the Construction and Analysis of Systems / ed. by Ramakrishnan C. R. and Rehof Jakob. — Berlin, Heidelberg : Springer Berlin Heidelberg. — 2008. — P. 337–340. — Access mode: https://doi.org/10.1007/978-3-540-78800-3_24.

[11] Muller Jean-Michel, Brisebarre Nicolas, de Dinechin Florent, Jeannerod Claude-Pierre, Lefèvre Vincent, Melquiond Guillaume, Revol Nathalie, Stehlé Damien, Torres Serge, Muller Jean-Michel, et al. 3 Floating-point formats and environment // Handbook of floating-point arithmetic. — 2010. — P. 55–116. — Access mode: https://doi.org/10.1007/978-3-319-76526-6_3.

[12] Niemetz Aina and Preiner Mathias. Bitwuzla at the SMT-COMP 2020 // arXiv preprint arXiv:2006.01621. — 2020. — Access mode: https://doi.org/10.48550/arXiv.2006.01621.

[13] Tillmann Nikolai and De Halleux Jonathan. Pex–white box test gen- eration for .NET // Tests and Proofs: Second International Conference, TAP 2008, Prato, Italy, April 9-11, 2008. Proceedings 2 / Springer. — 2008. — P. 134–153. — Access mode: https://doi.org/10.1007/978-3-540-79124-9_10.