Национальный исследовательский университет ИТМО
(Университет ИТМО)

На правах рукописи

Чухарев Константин Игоревич

Методы декомпозиции задачи булевой выполнимости для синтеза и верификации моделей автоматных программ

Специальность 2.3.5

Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация
на соискание учёной степени
кандидата технических наук

Научный руководитель:
канд. техн. наук, доцент
Семёнов Александр Анатольевич

Санкт-Петербург, Россия
2024

Оглавление

ВВЕДЕНИЕ

Актуальность темы. В современном информационном обществе автоматизированные системы стали неотъемлемой частью различных областей науки и техники, что подчеркивает важность проблемы эффективной верификации и синтеза автоматных программ. Под *автоматизированными системами* понимается широкий класс объектов, решающих вычислительные задачи путем выполнения конкретных функций. Они находят применение в таких разнообразных областях, как программирование, инженерия, робототехника, управление производственными процессами и многое другое. Для анализа и разработки таких систем используются *абстрактные модели*, которые позволяют формализовать их поведение и свойства.

Одним из ключевых понятий в этой области является понятие *состояния* вычисляющей модели, которое было впервые введено Аланом Тьюрингом [1]. Это понятие является основой для построения абстрактных моделей и играет важную роль в различных прикладных областях, таких как разработка языков программирования, трансляторов, компиляторов, микроконтроллеров и микропроцессоров.

Концепция «*автоматного программирования*», предлагает подход к построению программ, основанный на конечно-автоматной парадигме. Этот подход предполагает разбиение программы на более простые модули, которые могут рассматриваться как отдельные вычислительные единицы, находящиеся в различных состояниях. При этом задачи верификации и синтеза для автоматных программ сводятся к аналогичным задачам для формальных моделей.

В настоящей диссертации рассматриваются два класса таких моделей: конечные автоматы и булевы схемы. Несмотря на различия между этими моделями, между ними существует тесная взаимосвязь: конечные автоматы описывают функции, которые могут принимать входные данные произвольной длины, в то время как булевы схемы оперируют данными конкретной длины. Соответственно, каждой функции, задаваемой конечным автоматом, соответствует счетное число функций, задаваемых булевыми схемами. Для решения конкретных вычислительных задач приходится рассматривать конечные входные данные и, таким образом, переходить к булевым схемам.

Задачи верификации и синтеза для конечных автоматов и булевых схем являются вычислительно сложными и относятся к классу NP-трудных задач. Это означает, что они не могут быть решены известными алгоритмами за полиномиальное время. В таких случаях, как и во многих других ситуациях, касающихся

NP-трудных задач, для решения конкретных экземпляров рассматриваемых проблем используются комбинаторные задачи с хорошо развитой алгоритмической базой. Одной из таких является задача булевой выполнимости (Boolean Satisfiability Problem — SAT), для решения которой за последние 20 лет разработаны весьма эффективные на практике эвристические алгоритмы, применяемые для решения задач символьной верификации [2], компьютерной безопасности и криптографии [3], построению расписаний и планированию [4] и многим другим прикладным областям. В применении к перечисленным задачам программные реализации алгоритмов решения SAT — так называемые SAT-решатели — дают мощные вычислительные инструменты, позволяющие решать частные случаи рассматриваемых задач таких размерностей, перед которыми другие подходы оказываются бессильны.

Таким образом, актуальной является проблема разработки алгоритмов и программных комплексов, основанных на решении задачи SAT, для верификации и синтеза формальных моделей автоматных программ, таких как конечные автоматы и булевые схемы. Одной из ключевых проблем при решении этой задачи является отсутствие априорных оценок времени работы SAT-решателя на сложных формулах, кодирующих рассматриваемые задачи. Грубо говоря, решатель, получив на вход формулу, может работать час, неделю, месяц или даже больше, и нет общего способа определить, сколько времени ему потребуется для завершения работы, притом что формально данный алгоритм является полным и на любой формуле завершает свою работу за конечное время. Это явление известно как *heavy-tailed behavior phenomenon* (HTB) [5], при котором время работы SAT-решателя на некоторых формулах может быть непредсказуемо длинным. В рамках данной диссертации для борьбы с явлением HTB предлагаются специальные декомпозиционные представления булевых формул. Разработанные алгоритмы и методы показали высокую эффективность при решении сложных задач, связанных с синтезом и верификацией конкретных примеров автоматных программ.

Учитывая всё сказанное выше, можно утверждать, что разработка новых методов декомпозиции задачи булевой выполнимости (SAT) для синтеза и верификации моделей автоматных программ является актуальной и важной задачей, имеющей значительные теоретические и практические приложения.

Цель работы. Целью настоящей диссертации является повышение эффективности (снижение времени работы) полных алгоритмов решения задачи булевой выполнимости (SAT) применительно к синтезу и верификации моделей автоматных программ за счет разработки оригинальных методов и техник декомпозиции булевых формул.

Задачи работы. Для достижения поставленной цели были решены следующие научно-технические задачи:

1. Разработка методов кодирования в SAT задач синтеза конечно-автоматных моделей с заданным поведением и свойствами. Новые методы включают в себя кодирование структуры охранных условий в виде деревьев разбора соответствующих формул, что отличает их от существующих решений.
2. Разработка методов кодирования в SAT задач синтеза модульных конечно-автоматных моделей. Эти методы включают автоматизированное модульное разбиение, что улучшает их адаптивность и эффективность.
3. Создание методов кодирования в SAT задач синтеза булевых схем и булевых формул по заданной таблице истинности. В отличие от существующих методов, новые подходы позволяют использовать произвольные элементарные гейты, что расширяет их применение.
4. Разработка методов декомпозиции булевых формул, кодирующих задачи синтеза конечно-автоматных моделей и верификации булевых схем. Новые методы позволяют строить оценки декомпозиционной трудности, что улучшает прогнозируемость времени работы SAT-решателей.
5. Разработка программной библиотеки `kotlin-satlib`, обеспечивающей взаимодействие с SAT-решателями через программный интерфейс. Библиотека предоставляет широкий выбор различных SAT-решателей, контроль за различными этапами построения SAT-кодировок и возможность манипуляции переменными с произвольными конечными доменами.
6. Создание программного комплекса fbSAT для синтеза и верификации конечно-автоматных моделей с использованием SAT-решателей. Этот комплекс интегрирует разработанные методы и алгоритмы, предоставляя удобный инструмент для практического применения.
7. Проведение масштабных вычислительных экспериментов для подтверждения практической эффективности всех разработанных методов.

Основные положения, выносимые на защиту.

1. Методы декомпозиции булевых формул, применяемые к задачам тестирования и верификации моделей автоматных программ и использующие SAT-решатели, отличающиеся от известных методов тем, что с целью получения более точных верхних оценок трудности формул в предлагаемых методах используются специальные конструкции SAT-разбиений.
2. Метод синтеза минимальных представлений булевых функций в виде формул и схем, использующий сведение к задаче выполнимости (SAT), отличающийся от существующих подходов тем, что с целью достижения более высокой эффективности (относительно времени и точности решения) предлагаемый метод использует инкрементальные SAT-решатели.
3. Методы синтеза и верификации монолитных и модульных конечно-автоматных моделей по примерам поведения и формальной спецификации, использующие сведения к задаче выполнимости (SAT) и контрпримеры (Counterexample-Guided Inductive Synthesis — CEGIS), отличающиеся от существующих подходов тем, что с целью повышения эффективности (относительно времени решения) применяется техник явного кодирования структуры охранных условий.
4. Программная библиотека `kotlin-satlib`[1] для взаимодействия с SAT-решателями и обеспечения контроля над всеми этапами построения SAT-кодировок, отличающаяся от известных библиотек тем, что с целью расширения области применимости разработанная библиотека предоставляет широкий выбор различных SAT-решателей и возможность манипулировать переменными с произвольными конечными доменами.
5. Программный комплекс FBSAT[2] для синтеза и верификации конечно-автоматных моделей с помощью SAT-решателей, включающий в себя реализацию всех предложенных методов.

Научная новизна. Новыми являются все основные результаты, полученные в диссертации, в том числе:

1. Методы декомпозиции булевых формул, применяемые к задачам тестирования и верификации моделей автоматных программ с использованием SAT-решателей. Отличие от известных методов заключается в применении

[1] `https://github.com/Lipen/kotlin-satlib`

[2] `https://github.com/ctlab/fbSAT`

специальных конструкций SAT-разбиений, что позволяет получать более точные верхние оценки трудности формул.

2. Метод синтеза минимальных представлений булевых функций в виде формул и схем, основанный на сведении к задаче выполнимости (SAT). В отличие от существующих подходов, предлагаемый метод использует инкрементальные SAT-решатели, что позволяет достичь более высокой эффективности по времени и точности решения.
3. Методы синтеза и верификации монолитных и модульных конечно-автоматных моделей, разработанные на основе сведений к задаче выполнимости (SAT) и использования контрпримеров (Counterexample-Guided Inductive Synthesis — CEGIS). Отличие состоит в применении техники явного кодирования структуры охранных условий, что значительно повышает эффективность по времени решения.
4. Программная библиотека `kotlin-satlib` для взаимодействия с SAT-решателями и обеспечения контроля над всеми этапами построения SAT-кодировок. В отличие от существующих библиотек, разработанная библиотека предоставляет широкий выбор различных SAT-решателей и возможность манипулировать переменными с произвольными конечными доменами.
5. Программный комплекс ғвSAT для синтеза и верификации конечно-автоматных моделей с помощью SAT-решателей, включающий реализацию всех предложенных методов. Этот комплекс позволяет эффективно решать экстремально трудные задачи синтеза моделей автоматных программ.

Соответствие специальности. Содержание диссертации соответствует паспорту специальности 2.3.5 «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей» в пунктах «1. Модели, методы и алгоритмы проектирования, анализа, трансформации, верификации и тестирования программ и программных систем» и «3. Модели, методы, архитектуры, алгоритмы, языки и программные инструменты организации взаимодействия программ и программных систем» в следующих частях:

– в диссертации представлено семейство методов и алгоритмов, применяемых для трансформации автоматных программ в булевы схемы и формулы с целью вычислительного решения задач синтеза и верификации автоматных программ (пункт 1);

- представлены алгоритмы решения задач синтеза, верификации и тестирования моделей автоматных программ при помощи декомпозиционных представлений булевых формул, кодирующих исходные задачи (пункт 1);
- разработанная программная библиотека `kotlin-satlib` обеспечивает взаимодействие между алгоритмами кодирования в SAT задач синтеза и верификации автоматных программ и современными эффективными SAT-решателями (пункт 3).

Теоретическая значимость диссертации заключается в разработке новых методов и алгоритмов для синтеза и верификации моделей автоматных программ. В работе предложены инновационные подходы к декомпозиции булевых формул и построению оценок их декомпозиционной трудности, что расширяет существующие теоретические основы в области применения SAT-решателей и предоставляет более точные инструменты для анализа сложных задач.

Практическая значимость работы проявляется в разработке программной библиотеки `kotlin-satlib` и программного комплекса fbSAT, которые позволяют эффективно применять новые методы и алгоритмы к реальным задачам проектирования и верификации программного обеспечения. Эти инструменты демонстрируют высокую эффективность и могут быть интегрированы в существующие программные системы, что подтверждается лучшими результатами по сравнению с известными подходами и инструментами.

Методы и инструменты исследования. Теоретическая часть работы основана на методологии дискретной математики, математической логики и теории вычислительной сложности. Для синтеза конечно-автоматных моделей применялся программный комплекс fbSAT, разработанный в рамках данной диссертации. Верификация этих моделей осуществлялась с помощью символьного верификатора NuSMV[3]. При построении вычислительных задач из области проверки логической эквивалентности схем использовалась программная система Transalg[4]. Для решения экземпляров задачи SAT применялись различные современные SAT-решатели, такие как MiniSAT[5], Glucose[6], Kissat[7], CaDiCaL[8]. Взаимодействие с SAT-решателями

[3] `https://nusmv.fbk.eu`
[4] `https://gitlab.com/transalg/transalg`
[5] `https://github.com/niklasso/minisat`
[6] `https://github.com/audemard/glucose`
[7] `https://github.com/arminbiere/kissat`
[8] `https://github.com/arminbiere/cadical`

осуществлялось через программную библиотеку `kotlin-satlib`, разработанную в рамках данной диссертации. Вычислительные эксперименты проводились с использованием вычислительного кластера.

Достоверность научных достижений диссертации подтверждается обоснованностью постановок задач, теоретической корректностью предложенных алгоритмов, а также результатами масштабных вычислительных экспериментов, проведенных для проверки и демонстрации эффективности разработанных методов.

Апробация работы. Основные результаты диссертации докладывались на следующих конференциях:

- VIII Конгресс молодых ученых, Университет ИТМО, Санкт-Петербург, 2019.
- Конференция СПИСОК-2019, СПбГУ, Санкт-Петербург, 2019.
- IX Конгресс молодых ученых, Университет ИТМО, Санкт-Петербург, 2020.
- Конференция ППС 2021, Университет ИТМО, Санкт-Петербург, 2021.
- X Конгресс молодых ученых, Университет ИТМО, Санкт-Петербург, 2021.
- Воркшоп SAT/SMT Solvers: Theory and Practice, Санкт-Петербург, 2021.
- XI Конгресс молодых ученых, Университет ИТМО, Санкт-Петербург, 2022.
- Конференция MIPRO 2024, Опатия, Хорватия, 2024.

Диссертационная работа была выполнена при поддержке грантов и проектов:

- Грант РФФИ №19-07-01195 А «Разработка методов машинного обучения на основе SAT-решателей для синтеза модульных логических контроллеров киберфизических систем».
- Грант №19-37-51066 Научное наставничество «Разработка методов синтеза конечно-автоматных алгоритмов управления для программируемых логических контроллеров в распределенных киберфизических системах».
- НИР-ФУНД 77051 «Исследование алгоритма тестирования на основе обучения и улучшения его эффективности», 2020-2021.
- НИР-ПРИКЛ 222004 «Алгоритмы решения SAT для логических схем и анализа программ», 2021-2023.
- НИР-ПРИКЛ 223099 «Алгоритмы решения SAT для логических схем и анализа программ», 2023-2024.

Личный вклад автора. Методы синтеза монолитных и модульных конечно-автоматных моделей по примерам поведения и формальной спецификации, основанные на сведении к задаче SAT, разработаны соискателем в соавторстве с Чивилихиным Д. С. Методы синтеза и верификации модульных конечно-автоматных

моделей по примерам поведения и формальной спецификации, основанные на сведении к задаче SAT и использовании контрпримеров, разработаны соискателем в соавторстве с Чивилихиным Д. С. и Суворовым Д. М. Программный комплекс fbSAT разработан лично соискателем. Реализация всех разработанных методов синтеза и верификации конечно-автоматных моделей в программном комплексе fbSAT выполнена лично соискателем. Метод синтеза булевых формул и схем по заданной таблице истинности, основанный на сведении к задаче SAT, предложен и разработан лично соискателем. Прототип программной библиотеки `kotlin-satlib` для взаимодействия с SAT-решателями через унифицированный программный интерфейс разработан в соавторстве с Гречишкиной Д. С. Дальнейшая разработка и расширение программной библиотеки `kotlin-satlib`, что включает в себя поддержку дополнительных SAT-решателей (например, Kissat) и разработку модуля для манипуляции переменными с конечными доменами, производилась лично соискателем. Общая стратегия оценивания трудности формул, кодирующих проверку эквивалентности (задача верификации) булевых схем, разработана соискателем в соавторстве с Семёновым А. А., Кондратьевым В. С., Кочемазовым С. Е. и Тарасовым Е. А. Описанные конструкции декомпозиций формул, кодирующих эквивалентность булевых формул, предложены лично соискателем. Теоретическое обоснование корректности предложенных конструкций выполнены соискателем в соавторстве с Семёновым А. А.

Публикации по теме диссертации. Основные результаты по теме диссертации изложены в 11 публикациях. Из них четыре изданы в изданиях, индексируемых в базе цитирования Scopus, и одна в журнале, рекомендованном ВАК.

Структура и объём работы. Диссертация состоит из введения, трёх глав и заключения. Полный объём диссертации составляет 135 страниц, включая 12 рисунков и 7 таблиц. Список литературы содержит 125 наименований.

Глава 1. Обзор предметной области

В данной главе представлены основные понятия, методы и результаты, связанные с предметной областью диссертационного исследования.

В разделах 1.1 и 1.2 рассматриваются *конечные автоматы* и *булевые схемы*, которые являются двумя ключевыми формализмами, используемыми для моделирования и анализа поведения автоматных программ. Конечные автоматы описывают системы, переходящие из одного состояния в другое в ответ на входные сигналы, в то время как булевые схемы оперируют булевыми функциями и используются для логического анализа и синтеза цифровых систем.

В разделе 1.3 обсуждается международный стандарт IEC 61499, который определяет модель исполнения для автоматизированных систем и является основой для моделирования поведения программных компонентов. Этот стандарт позволяет описывать поведение системы на уровне *функциональных блоков* и их взаимодействия, что существенно упрощает процесс проектирования и анализа систем.

В разделе 1.5 рассматриваются *сценарии выполнения*, которые описывают поведение автоматных программ в различных условиях и с различными входными данными. Дополнительно, вводится понятие *дерева сценариев*, которое позволяет компактно представлять множество сценариев выполнения и использовать его для анализа и верификации поведения программ.

Один из инструментов формальной верификации — линейная темпоральная логика (раздел 1.6), которая позволяет формулировать и проверять свойства программы относительно временны́х (*temporal*) аспектов ее выполнения. Это позволяет выявлять ошибки и недочеты в программном коде до его применения на практике. В разделе 1.7 рассматривается метод *проверки моделей* (*model checking*), который позволяет автоматически проверять соответствие спецификации и находить ошибки в поведении программ.

Раздел 2.3 посвящен методам синтеза конечно-автоматных моделей, которые позволяют автоматически создавать программы на основе спецификации их поведения. Это полезно при разработке систем, где требуется генерация программного кода из формальных спецификаций, что упрощает процесс разработки и повышает надежность системы.

В разделах 1.9 и 1.10 рассматриваются задачи проверки эквивалентности булевых схем и генерации тестовых шаблонов для верификации таких схем, что является важным этапом в процессе разработки и верификации цифровых систем.

Эти методы позволяют обнаруживать и исправлять ошибки в проектировании до выпуска системы в эксплуатацию.

В контексте формальной верификации особое внимание уделяется задаче булевой выполнимости (SAT) и основным алгоритмам ее решения. Это важная задача, которая возникает при автоматической проверке моделей на выполнимость и поиске контрпримеров. В разделе 1.11 приводится формальное определение задачи SAT. В разделе 1.12 рассматриваются различные алгоритмы решения SAT. В разделе 1.13 рассматриваются ограничения на кардинальность и способы их кодирования.

Наконец, в разделе 1.14 рассматриваются вопросы декомпозиции задачи SAT и различные способы построения SAT-разбиений. Отдельное внимание уделяется декомпозиционной трудности булевых формул и методам ее оценки. В частности, в разделе 1.14.3 приводится вероятностый метод оценки декомпозиционной трудности на основе метода Монте-Карло.

Все эти концепции и методы составляют фундаментальную базу для дальнейшего изучения и разработки новых подходов к синтезу и верификации моделей автоматных программ, что является целью данного исследования.

1.1. Конечные автоматы

Конечный автомат (КА) — это одна из простых моделей вычислений, свойствам которых посвящено огромное число работ. КА описывает систему с конечным числом состояний и переходов между ними. Конечный автомат может быть задан в виде пятерки $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$, где:

- Σ — алфавит входных символов;
- Q — (*конечное*) множество состояний;
- $q_0 \in Q$ — начальное состояние;
- $F \subseteq Q$ — множество терминальных (принимающих) состояний;
- $\delta\colon Q \times \Sigma \to Q$ — функция переходов.

Конечный автомат *принимает* (*accepts*) слово $w = w_1 w_2 \dots w_n \in \Sigma^*$, если после прочтения w автомат оказывается в одном из терминальном состоянии $s_n \in F$, то есть существует последовательность состояний $s_0, s_1, \dots, s_n$ такая, что $s_0 = q_0$, $s_{i+1} = \delta(s_i, w_{i+1})$ для всех $i \in \{0, 1, \dots, n-1\}$ и $s_n \in F$.

В настоящей работе исследуются задачи синтеза КА, обладающих определенными свойствами, а также задачи верификации уже построенных КА на предмет соответствия конкретным спецификациям [6].

1.2. Булевы схемы

Булевы схемы также являются простыми и естественными моделями вычислений. Однако, в отличие от конечных автоматов, булевые схемы задают функции, получающие на вход двоичные слова фиксированной конечной длины. История булевых схем берет свое начало в работах К. Э. Шеннона [7] и В. И. Шестакова [8]. Помимо огромного числа практических применений, булевы схемы тесным образом связаны со многими фундаментальными разделами современной информатики (Computer Science), в частности, с вычислительной сложностью [9].

Булева схема — это направленный ациклический ориентированный граф $G = \langle V, E \rangle$, где V — множество вершин, а $E \subseteq V^2$ — множество рёбер (*дуг*). Вершины такого графа делятся на три типа: входные вершины (*inputs*), выходные вершины (*outputs*) и внутренние вершины (*gates*). *Ребро* (*дуга*) представляет собой упорядоченную пару вершин. Для каждой дуги $(u,v) \in E$, вершина u называется *родителем* v, а v — *потомком* u. Множество всех родителей вершины v обозначается как P_v. Вершина называется *входной*, если у нее нет родителей, и *выходной*, если у нее нет потомков[1]. Множества входов и выходов обозначаются как $V^{\text{in}} \subseteq V$ и $V^{\text{out}} \subseteq V$ соответственно. Любая вершина $v \in V \backslash V^{\text{in}}$ называется *гейтом* (*логическим вентилем*). В булевой схеме каждому гейту сопоставляется некоторый *логический элемент* из предопределенного набора, называемого *базисом* (например, $\{\wedge, \neg\}$). Таким образом, любой логический элемент интерпретирует некоторую элементарную булеву функцию. Пример булевой схемы представлен на Рисунке 1.

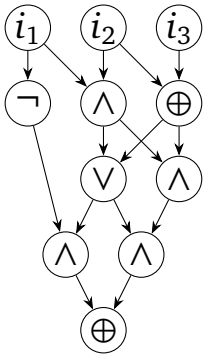


Рисунок 1 — Пример булевой схемы с тремя входами (i_1, i_2, i_3) и восьмью гейтами

[1] Здесь стоит отметить, что вполне возможны вариации данных определений. В некоторых ситуациях *входными/выходными* вершинами в схеме считаются некоторые заранее выбранные вершины, но при этом у них могут быть родители/потомки, соотвественно. В зависимости от контекста, эти родители/потомки игнорируются в соответствующих определениях, связанных с обходом вершин графа от *входов* к *выходам*.

Булева схема с n входами и m выходами естественным образом задает (тотальную) дискретную функцию $f\colon \{0,1\}^n \to \{0,1\}^m$, где $\{0,1\}^k$ — множество всех возможных двоичных слов длины $k \in \mathbb{N}^+$. Имея это в виду, будем использовать обозначение S_f для представления булевой схемы, задающей функцию f. Каждому гейту схемы S_f сопоставлена булева функция, которая соответствует логическому элементу, назначенному этому гейту.

Пусть $\alpha \in \{0,1\}^n$ произвольное слово, поданное на вход S_f. При проходе по гейтам схемы в фиксированном порядке (обычно указанном топологической сортировкой [10]) и вычислении значений элементарных функций, сопоставленных гейтам, получается значение функции f на входном слове α в качестве результата. Этот процесс называется *интерпретацией* схемы S_f на входе α.

Каждой вершине в схеме S_f сопоставим булеву переменную. Обозначим множество переменных, ассоциированных с входами V^{in} схемы S_f, как $X^{\text{in}} = \{x_1, \ldots, x_n\}$. Переменные, связанные с гейтами, называются *вспомогательными* (*auxiliary*). Пусть u — некоторая вспомогательная переменная, соответствующая гейту v, и U_v — множество переменных, связанных с вершинами из P_v. Предположим, что h_v — булева функция, соответствующая логическому элементу, назначенному гейту v, и $F(h_v)$ — булева формула над U_v, которая задает функцию h_v. Обозначим через C_v КНФ-представление формулы $F(h_v) \equiv u$.

Рассмотрим следующую КНФ:

$$C_f = \bigwedge_{v \in V \setminus V^{\text{in}}} C_v \tag{1}$$

Будем называть (1) *шаблонной КНФ* для функции f. Заметим, что C_f является КНФ-формулой, полученной применением преобразований Цейтина [11] к схеме S_f.

Ниже, следуя работе [12], будем использовать обозначение x^σ, где $\sigma \in \{0,1\}$, предполагая, что x^0 обозначает отрицательный литерал $\neg x$, а x^1 обозначает положительный литерал x, а также обозначение $\{0,1\}^{|B|}$, что означает множество всех возможных назначений переменных из B. Следующий факт был многократно установлен в литературе, например, см. [13; 14]. Он использует простой механизм булевой дедукции, известный как правило распространения единичного дизъюнкта (Unit Propagation — UP) [15].

Лемма 1. *Применение UP к КНФ-формуле* $x_1^{\alpha_1} \wedge \cdots \wedge x_n^{\alpha_n} \wedge C_f$ *для любого* $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha \in \{0,1\}^{|X^{in}|}$ *выводит (в форме единичных дизъюнкций) значения*

всех переменных, связанных с гейтами из $V \setminus V^{in}$*, включая переменные* $y_1, \ldots, y_m$*, связанные с выходами схемы* S_f*:* $y_1 = \gamma_1, \ldots, y_m = \gamma_m$*,* $f(\alpha) = \gamma = (\gamma_1, \ldots, \gamma_m)$.

Стоит отметить, что Лемма 1 в сущности означает, что процесс интерпретации схемы S_f на входном слове α может быть смоделирован последовательным применением UP к КНФ $C_f \wedge x_1^{\alpha_1} \wedge \cdots \wedge x_n^{\alpha_n}$ для любого $\alpha = (\alpha_1, \ldots, \alpha_n)$. Лемма 1 очень полезна при доказательстве свойств, связанных с булевыми схемами и SAT.

1.3. Международный стандарт IEC 61499

Международный стандарт распределенных систем управления и автоматизации IEC 61499 [16] нацелен на упрощение разработки распределенных киберфизических систем. Стандарт отличается от «предыдущего» стандарта IEC 61131[17] тем, что в IEC 61499 используется событийная модель исполнения. Этот стандарт предлагает использование так называемых *функциональных блоков* (ФБ), являющихся, по-сути, контейнерами для базовых элементов — управляющих конечных автоматов. Основные типы описываемых в стандарте IEC 61499 функциональных блоков — *базовые* и *композитные*. Функционал композитных блоков определяется сетью базовых ФБ. Базовые ФБ являются совокупностью интерфейса (описания входных и выходных событий и переменных) и управляющего конечного автомата (*Execution Control Chart* — ECC). Подробное описание формальной модели [18] базового функционого блока предоставлено в разделе 1.4.

1.4. Модель базового функционального блока

В данном разделе приводится описание формальной модели базового функционального блока [18]. Базовый функциональный блок (*Basic Function Block* — BFB) состоит из интерфейса, диаграммы управления выполнением (*Execution Control Chart* — ECC), набора алгоритмов управления (*Alg*), а также множества внутренних переменных V. Отметим, что в данной работе рассматриваются только функциональные блоки без внутренних переменных, поэтому в дальнейшем большинство упоминаний множества внутренних переменных V будут опущены.

Интерфейс функционального блока включает в себя входные и выходные события (множества E^I и E^O) и переменные (множества $\mathcal{X}$ и $\mathcal{Z}$), а также отношения ассоциации событий с переменными (W^I и W^O). Формально, $Interface = (E^I, E^O, \mathcal{X}, \mathcal{Z}, W^I, W^O)$, где $W^I \subseteq E^I \times \mathcal{X}$, $W^O \subseteq E^O \times \mathcal{Z}$. В данной работе рассматриваются ФБ, входные/выходные события которых ассоциированы со всеми

входными/выходными переменными: $W^I = E^I \times \mathcal{X}$, $W^O = E^O \times \mathcal{Z}$. Отметим, что это не влияет на вычислительную способность системы. Ассоциация событий только с «нужными» переменными значительно упрощает восприятие системы человеком и способствует инкапсуляции отдельных частей системы. Однако на текущем этапе разработки методов, предлагаемых в данной работе, полноценный учет ассоциации событий и переменных является нецелесообразным.

Алгоритмы управления функциональных блоков реализуются на языках стандарта IEC 61131-3 (SFC — *Sequential Function Chart*, LD — *Ladder Diagram*, FBD — *Function Block Diagram*, ST — *Structured Text*, IL — *Instruction List*) с использованием типов данных, описанных в IEC 61131-3 (например, булевы значений, целые числа, числа с плавающей точкой, символы и строки, объекты даты и времени, а также пользовательские типы, такие как перечисления, массивы и структуры), и могут быть представлены функциями вида $f_{alg}\colon \prod_{x\in\mathcal{X}} \mathrm{Dom}(x) \times \prod_{z\in\mathcal{Z}} \mathrm{Dom}(z) \times \prod_{v\in V} \mathrm{Dom}(v) \to \prod_{z\in\mathcal{Z}} \mathrm{Dom}(z) \times \prod_{v\in V} \mathrm{Dom}(v)$, которые изменяют значения выходных и внутренних переменных в зависимости от входных переменных. Здесь $\mathrm{Dom}(v)$ означает домен (область значений) переменной v. В данной работе рассматриваются только логические контроллеры ($\forall x \in \mathcal{X}, z \in \mathcal{Z} : \mathrm{Dom}(x) = \mathrm{Dom}(z) = \mathbb{B} = \{\top, \bot\}$) без внутренних переменных ($V = \varnothing$) и с алгоритмами управления, явно независящими от входных переменных, поэтому функции алгоритмов принимают простой вид $f_{alg}\colon \mathbb{B}^{|\mathcal{Z}|} \to \mathbb{B}^{|\mathcal{Z}|}$. Отметим, что данный вид алгоритмов обладает способностью к преобразованию любого набора значений выходных переменных, что является достаточным в задачах логического управления.

Диаграмма управления выполнением (*Execution Control Chart* — ECC) представляется в виде конечного автомата-преобразователя (*finite-state transducer*), похожего на автомат Мура, в котором переходы дополнены охранными условиями (*guard conditions*), а состояния имеют ассоциированные выходные события и описанные выше алгоритмы управления. Формально, $ECC = (ECState, s_0, ECAction, ECTran, ECTCond, PriorT)$, где $ECState = (s_0, s_1, \ldots, s_r)$ — множество состояний автомата; s_0 — начальное состояние; $ECAction\colon ECState \setminus \{s_0\} \to ECA^*$ — функция соответствия состояниям автомата действий, где $ECA = Alg \times E^O \cup Alg \cup E^O$ — множество синтаксически корректных действий автомата, $ECA^* = \bigcup_{k=0}^{\infty} ECA^k$ — множество всевозможных последовательностей действий (включая пустую последовательность действий, обозначаемую впоследствии $ECA^0 = (\varepsilon)$); $ECTran \subseteq ECState^2$ — множество переходов автомата; $ECTCond\colon ECTran \to \left[\prod_{e_i\in E^I} \mathrm{Dom}(e_i) \times \prod_{x\in\mathcal{X}} \mathrm{Dom}(x) \times \prod_{z\in\mathcal{Z}} \mathrm{Dom}(z) \times\right.$

$\prod_{v \in V} \mathrm{Dom}(v) \rightarrow \mathbb{B}]$ — функция соответствия переходам автомата охранных условий — логических функций, зависящих от входных событий и (потенциально) всех переменных; $PriorT : ECTran \rightarrow \mathbb{N}_+$ — функция приоритетов переходов.

Говорят, что ЕСС находится в *канонической форме* [18], если с каждым состоянием автомата ассоциировано не более одного действия. В данной работе в основном рассматриваются канонические конечно-автоматные модели, состояния которых имеют в точности одно выходное действие, так как такие модели в большистве случаев значительно проще для реализации, использования и анализа.

1.5. Сценарии выполнения и дерево сценариев

Желаемое поведение системы может быть описано с помощью примеров поведения. Одним из самых простых способов представления примеров поведения являются *трассировки* — последовательности входных воздействий и реакций на них конкретной системы в виде выходных действий. Зачастую отсутствие выходного действия также является допустимой реакцией, однако трассировки являются лишь последовательностями действий и не обладают какой-либо дополнительной структурой. Поэтому в данной работе для описания примеров поведения используются более структурированные объекты — так называемые *сценарии выполнения*.

Сценарий выполнения s — это последовательность *элементов сценария* $s_i = \langle e^I[\bar{x}], e^O[\bar{z}] \rangle$. Каждый элемент сценария представляет собой пару «воздействие–реакция»: входное действие $e^I[\bar{x}]$ содержит входное событие $e^I \in E^I$ и *вход* — набор значений входных переменных $\bar{x} \in \mathbb{B}^{|X|}$; выходное действие $e^O[\bar{z}]$ содержит выходное событие $e^O \in E^O$ и *выход* — набор значений выходных переменных $\bar{z} \in \mathbb{B}^{|\mathcal{Z}|}$.

Конвертация трассировок в сценарии выполнения происходит следующим образом. Каждая последовательная пара из входного и выходного действий образует *активный* элемент сценария $s_i^{(\mathrm{active})} = \langle e^I[\bar{x}], e^O[\bar{z}] \rangle$. Если в трассировке входные воздействия следуют одно за другим, без промежуточных реакций системы, то такие действия соответствуют *пассивным* элементам сценариев $s_i^{(\mathrm{passive})} = \langle e^I[\bar{x}], \varepsilon[\bar{z}^{\mathrm{prev}}] \rangle$, где $\bar{z}^{\mathrm{prev}}$ — выход, равный выходу в предыдущем элементе сценариев. Стоит отметить, что если первый элемент сценария является пассивным, то «предыдущим» выходом $\bar{z}^{\mathrm{prev}}$ считается инициализирующий набор $\bar{z}_{\mathrm{init}}$, соответствующий начальному состоянию системы, и обычно равный $\langle 0 \ldots 0 \rangle$. Также стоит отметить, что в дальнейшем пассивное выходное действие $\varepsilon[\bar{z}^{\mathrm{prev}}]$ может быть обозначено просто ε.

В качестве примера рассмотрим следующий набор сценариев исполнения:

- `in=R[10]` , `in=R[01],out=B[1]` , `in=R[11],out=A[0]` , `in=R[11],out=A[1]`
- `in=R[01],out=B[1]` , `in=R[01]` , `in=R[11],out=A[0]` , `in=R[00],out=B[1]`
- `in=R[01],out=B[1]` , `in=R[01]` , `in=R[00]` , `in=R[01],out=A[1]`

Здесь цветами выделены последовательные пары входных–выходных действий. Красным цветом отмечены (и подчеркнуты) входные действия без пары, то есть те воздействия, на которые нет реакции системы. Зеленым цветом выделены совпадающие первые пары во второй и третьей последовательностях. Остальные пары событий выделены синим. Данные трассировки соответствуют множеству сценариев выполнения $\mathcal{S} = \{s_1, s_2, s_3\}$:

$$
\begin{aligned}
s_1 &= [\,\langle \texttt{R[10]}, \varepsilon\rangle, \langle \texttt{R[01]}, \texttt{B[1]}\rangle, \langle \texttt{R[11]}, \texttt{A[0]}\rangle, \langle \texttt{R[11]}, \texttt{A[1]}\rangle\,],\\
s_2 &= [\,\langle \texttt{R[01]}, \texttt{B[1]}\rangle, \langle \texttt{R[01]}, \varepsilon\rangle, \langle \texttt{R[11]}, \texttt{A[0]}\rangle, \langle \texttt{R[00]}, \texttt{B[1]}\rangle\,],\\
s_3 &= [\,\langle \texttt{R[01]}, \texttt{B[1]}\rangle, \langle \texttt{R[01]}, \varepsilon\rangle, \langle \texttt{R[00]}, \varepsilon\rangle, \langle \texttt{R[01]}, \texttt{A[1]}\rangle\,].
\end{aligned}
\tag{2}
$$

Можно заметить, что если система является детерминированной, то последовательности одинаковых пассивных элементов сценария могут быть заменены одним пассивным элементом с общим входным действием — это является простейшей операцией *предобработки*, что позволяет сократить размеры сценариев, не искажая их смысла. Аналогично, последовательности пассивных элементов с чередующимися входными действиями могут быть заменены множеством элементов с уникальными входными действиями. Несмотря на простоту, такие техники предобработки оказываются крайне эффективными на реальных данных, изобилирующих повторяющимися пассивными элементами. В дальнейшем в данной работе считается, что все используемые сценарии выполнения подвергаются описанной предобработке, если не указано обратное.

Дерево сценариев $\mathcal{T}$ — префиксное дерево, построенное из сценариев $\mathcal{S}$. Стоит отметить, что перед построением дерева сценарии выполнения дополняются фиктивными пассивными элементами $\varepsilon[\bar{z}_{\text{init}}]$. Это приводит к тому, что у всех сценариев образуется общий префикс — добавленные фиктивные элементы соответствуют корню префиксного дерева сценариев $\mathcal{T}$. Каждая вершина в дереве и входящее в нее ребро соответствуют элементам сценария — вершины отмечены выходными действиями, а входящие ребра входными действиями. *Позитивным деревом сценариев* $\mathcal{T}^{(+)}$ называется дерево сценариев, построенное по позитивным сценариям $\mathcal{S}^{(+)}$.

Здесь и далее используется следующая нотация для дерева сценариев: V — множество вершин дерева; $\rho \in V$ — корень дерева; $tp(v) \in V$ — родитель

вершины $v \in V$ ($v \neq \rho$); $tie(v) \in E^I$ — входное событие на входящем ребре вершины $v \in V$ ($v \neq \rho$); $toe(v) \in E^O \cup \{\varepsilon\}$ — выходное событие в вершине $v \in V$, где ε обозначает пустое событие; $V^{(\text{active})} = \{v \in V \setminus \{\rho\} \mid toe(v) \neq \varepsilon\}$ — множество *активных* вершин; $V^{(\text{passive})} = \{v \in V \setminus \{\rho\} \mid toe(v) = \varepsilon\}$ — множество *пассивных* вершин; $\mathcal{U} \subseteq \mathbb{B}^{|\mathcal{X}|}$ — множество *входов* (*вход* — набор значений входных переменных), встречающихся в сценариях; $tin(v) \in \mathcal{U}$ — вход на входящем ребре вершины $v \in V$ ($v \neq \rho$); $tov(v,z) \in \mathbb{B}$ — значение выходной переменной $z \in \mathcal{Z}$ в вершине $v \in V$. У корня дерева нет родительской вершины, поэтому значения $tp(\rho)$, $tie(\rho)$ и $tin(\rho)$ неопределены. На рисунке 2 изображен пример дерева сценариев, построенного по сценариям $\mathcal{S}$ (2).

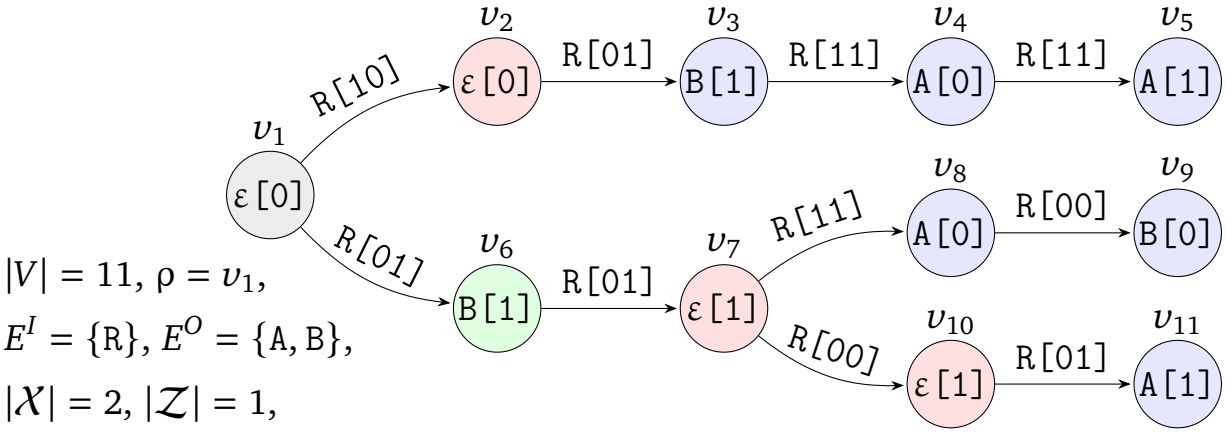


Рисунок 2 — Дерево сценариев, построенное по сценариям выполнения $\mathcal{S}$ (2)

1.6. Линейная темпоральная логика

Формальная спецификация может быть проверена с помощью верификатора (*model checker*) — специализированного программного средства, которое проверяет выполнение заданных свойств в системе и генерирует контрпримеры к нарушенным свойствам. В данной работе был использован символьный верификатор NuSMV [19], а рассмотренные спецификации систем были составлены на языке линейной темпоральной логики (*Linear Temporal Logic* — LTL) [20], полностью поддерживаемом NuSMV. Для так называемых «свойств безопасности» (*safety properties*), выражающих отсутствие нежелательного поведения (например, «с системой никогда не произойдёт ничего плохого»), контрпримером является конечная последовательность вычислительных состояний (*execution state*), приводящая к нежелательному поведению. Для так называемых «свойств живости» (*liveness properties*), выражающих присутствие желаемого поведения (например, «с системой точно произойдет что-то хорошее»),

контрпримером является бесконечная, но циклическая последовательность состояний, представляющая нежелательное циклическое поведение системы, и которая может быть представлена в виде конечного префикса с последующим циклом конечной длины [21].

1.7. Формальная верификация с использованием проверки моделей

В данной работе система специфицируется с помощью набора формул линейной темпоральной логики (LTL) [20]. LTL-формула описывает некоторые темпоральные свойства путей исполнения формальной системы. Формулы на языке LTL включают атомарные высказывания (некоторые элементарные утверждения о системе), логические связки ($\wedge$, $\vee$, $\neg$, $\rightarrow$ и другие) и темпоральные операторы: **X** — «*next*», **U** — «*until*», **G** — «*always*», **F** — «*in the future*». С помощью LTL-формул можно специфицировать свойства *безопасности* («что-то плохое никогда не происходит») и *живучести* («что-то хорошее в конечном итоге произойдёт») данной системы. Примером свойства безопасности является формула $\mathbf{G}\neg P$, которая утверждает, что некоторый предикат P всегда ложен (во всех путях исполнения). Примером свойства живучести является формула $\mathbf{G}(P \rightarrow \mathbf{F}Q)$, которая утверждает, что если предикат P истиннен, то предикат Q в конечном итоге также станет истинным в любом пути исполнения.

Проверка моделей (*model checking*) [21] — это техника, которая может использоваться для верификации заданной модели конечного автомата относительно заданной спецификации (в данном случае набора LTL-формул) и получения контрпримера траектории выполнения, если эта спецификация нарушена. Контрпримеры могут быть преобразованы в *негативные сценарии* — сценарии выполнения, представляющие нежелательное поведение автомата или системы автоматов.

Проверка моделей является одним из важнейших методов формальной верификации программ и аппаратных систем. Основная идея состоит в автоматической проверке, соответствует ли модель системы формальной спецификации. Существует два основных подхода к проверке моделей: символьная проверка моделей (*symbolic model checking*) и ограниченная проверка моделей (*Bounded Model Checking* — BMC).

1.7.1. Символьная проверка моделей

Символьная проверка моделей (*symbolic model checking*) использует булевы функции и бинарные диаграммы решений (Binary Decision Diagram — BDD) для

представления и манипулирования множествами состояний системы. Классическими инструментами символьной проверки моделей являются NuSMV [19] и SPIN [22].

NuSMV (открытый инструмент для символьной проверки моделей) позволяет описывать системы на языке SMV и проверять их на соответствие LTL и CTL (Computational Tree Logic) спецификациям — наборам свойств на указанных языках темпоральной логики. Он использует BDD для представления переходных систем и позволяет анализировать большие пространства состояний.

SPIN (Simple Promela INterpreter) предназначен для проверки моделей, описанных на языке Promela. SPIN использует оптимизации для поиска путей и проверки LTL спецификаций, а также может генерировать исходный код на языке C для исполнения моделей.

1.7.2. Ограниченная проверка моделей

Ограниченная проверка моделей (*Bounded Model Checking* — BMC) [23] используется для поиска контрпримеров ограниченной длины. BMC заключается в сведении задачи проверки модели к задаче выполнимости SAT, что позволяет эффективно искать ошибки в моделях.

BMC итеративно проверяет, существует ли трасса (сценарий выполнения), нарушающая спецификацию, длиной не более заданного числа шагов. Процесс решения BMC через сведения к SAT включает несколько этапов:

1. Построение булевой формулы, описывающей переходную систему и спецификацию.
2. Ограничение длины траектории выполнения (глубина поиска).
3. Преобразование построенной формулы в CNF (Conjunctive Normal Form) для SAT-решателя.
4. Решение полученной SAT-задачи с помощью SAT-решателя.
5. Анализ результата: если SAT-решатель нашел удовлетворяющую подстановку, она интерпретируется как контрпример; если нет, то глубина поиска (длина искомого контрпримера) увеличивается, и процесс повторяется.

Классическими инструментами для BMC являются CBMC (C Bounded Model Checker) [24] и Eldarica [25]. Эти инструменты используют современные SAT-решатели для поиска контрпримеров и позволяют анализировать сложные программы на языке C с большим пространством состояний.

1.7.3. LTL-синтез

LTL-синтез (Linear Temporal Logic synthesis) представляет собой процесс генерации моделей, которые непосредственно удовлетворяют заданной формальной спецификации в виде LTL-свойств. Этот подход позволяет автоматически строить системы, которые гарантированно выполняют все указанные свойства, без необходимости в последующей проверке.

Процесс синтеза LTL включает следующие этапы:

1. Спецификация свойств системы с использованием LTL-формул.
2. Построение автомата, эквивалентного данной LTL-формуле.
3. Генерация контроллера или системы автоматов, которые удовлетворяют построенному автомату.

Существуют различные инструменты и методы для LTL-синтеза, включая GR(1) синтез и автоматный синтез на основе игр. Эти методы активно исследуются и развиваются, так как позволяют создавать надежные системы с гарантированными свойствами.

Примерами инструментов для LTL-синтеза являются:

– BoSy [26; 27], инструмент для синтеза систем переходов по LTL-спецификациям, основанный на сведении к Quantified SAT — QSAT.
– Strix [28], инструмент для реактивного синтеза автоматов Мили, основанный на теории игр.
– SLUGS [29], который поддерживает GR(1) синтез и предоставляет эффективные алгоритмы для генерации контроллеров.
– Spectra [30], язык спецификаций и соответствующий инструмент, который позволяет описывать и синтезировать реактивные системы.

LTL-синтез и проверка моделей являются мощными инструментами для создания и анализа надежных систем, обеспечивая соответствие сложным формальным спецификациям и позволяя автоматически обнаруживать ошибки на ранних стадиях разработки.

1.8. Методы синтеза конечно-автоматных моделей

Задача поиска минимального детерминированного конечного автомата по примерам поведения является NP-полной задачей [31], а сложность задачи LTL-синтеза дважды экспоненциальная от размера LTL-спецификации [32]. Несмотря на это, синтез различных типов конечно-автоматных моделей по примерам поведения

и/или формальной спецификации был исследован во многих научных работах [26; 33—43], где используются методы, основанные на эвристическом объединении состояний (*state merging*), эволюционные алгоритмы, а также методы, основанные на применении SAT- и SMT-решателей. В данной работе рассматриваются только точные и эффективные методы, поэтому внимание уделяется методам с применением SAT-решателей.

Расширенный конечный автомат (*Extended Finite State Machine* — EFSM) является моделью, наиболее близкой к рассматриваемой в данной работе модели ECC. EFSM является объединением автомата Мили и Мура, расширенный условными переходами. Переходы в EFSM помечены входными событиями и охранными условиями — булевыми функциями от входных переменных, а состояния EFSM имеют ассоциированные выходные действия. Для синтеза EFSM по примерам поведения и LTL-спецификации существует несколько подходов [34; 44], основанных на сведении к задаче SAT. В [34] LTL-спецификация учитывается путём применения итеративного подхода запрета контрпримеров. Существенным недостатком [34] является то, что охранные условия должны быть известны заранее, а также то, что синтезируемые алгоритмы в состояниях EFSM являются лишь указаниями на некоторые внешние процедуры. В [44] решается задача синтеза вычислимых выходных алгоритмов, однако предполагается, что базовая модель автомата (его структура — состояния и переходы между ними) известна заранее или получается отдельно. В общем случае, при использовании исходных данных, получаемых при black-box тестировании системы, информация о внутреннем устройстве системы, а также о доступных внешних процедурах и их поведении, оказывается недоступной, поэтому существующие методы синтеза EFSM не подходят для решения задачи синтеза модели ECC, рассматриваемой в данной работе.

Программное средство BoSy [26; 27] реализует так называемый ограниченный синтез (*bounded synthesis*) системы переходов (*transition system*) по LTL-спецификации. Синтез «ограничен» в том смысле, что позволяет синтезировать систему заданного размера, либо гарантировать отсутствие решения заданного размера. В BoSy реализовано не только сведение задачи LTL-синтеза к SAT, но также разработано более эффективное (при рассмотренной авторами постановке задачи) сведение с использованием Quantified SAT (QSAT). При использовании SAT-кодировки, синтезируемые системы переходов являются «явными» (*explicit*) — охранные условия на переходах являются полными зависят от всех входных переменных. При использовании QSAT-кодировки, системы получаются «символьными» (*symbolic*) — охранные условия

синтезируются в виде полноценных булевых формул. Используемые в BoSy подход ограниченного синтеза позволяет синтезировать минимальные модели в терминах числа состояний, однако важный вопрос о размере охранных условий обходится стороной — синтезируемые модели, как правило, обладают огромными охранными условиями, что сильно затрудняет их восприятие человеком, а также ограничивает применимость таких моделей во встраиваемых системах. В [45] предлагается способ упрощения генерируемых моделей, заключающийся в дополнении SAT сведения специальными ограничениями для минимизации числа циклов в системе переходов, однако это слабо влияет на размеры и форму охранных условий. Также стоит упомянуть, что отличительной особенностью LTL-синтеза является то, что в качестве входных данных не используются примеры поведения, так как предполагается полнота входной спецификации — в том смысле, что она описывает все желаемое поведение системы. Несмотря на то, что примеры поведения могут быть представлены в виде LTL-свойств, этот подход становится крайне неэффективным уже на небольших наборах данных. Другие программные средства для LTL-синтеза, например G4LTL-ST [42] и Strix [28], обладают аналогичными недостатками по отношению к рассматриваемой задаче: отсутствие минимизации охранных условий и невозможность эффективного учета примеров поведения.

В статье [46] предлагается метод FBCSP для синтеза конечно-автоматных моделей функциональных блоков по примерам поведения, основанный на сведении к задаче удовлетворения ограничений (*Constraint Satisfaction Problem* — CSP) [47]. Однако методу FBCSP присущи следующие ограничения. Получаемые модели обладают *полными* охранными условиями — соответствующие булевы формулы зависят от *всех* входных переменных. Такие модели практически не обобщаются (*generalize*) — некорректно работают на входных данных, которые не были использованы в процессе «обучения» (синтеза). В [46] это отчасти исправляется дополнительной жадной минимизацией охранных условий, однако жадный подход не гарантирует, что охранные условия будут наименьшими. В работе [48] метод FBCSP был расширен процедурой запрета контрпримеров для учета LTL-спецификации (в дальнейшем это расширение будет называться FBCSP+LTL), аналогично работе [34]. При этом охранные условия в генерируемых моделях представляются в виде конъюнкции литералов входных переменных. Основным недостатком этого подхода является его низкая эффективность в тех случаях, когда темпоральная спецификация покрыта сценариями выполнения не полностью.

В работе [49] разработан двухэтапных подход: сначала генерируется *базовая* модель с использованием метода, основанного на SAT, а затем охранные условия полученной модели отдельно минимизируются с помощью CSP — деревья разбора булевых формул, соответствующих охранным условиям, кодируются в CSP, а затем минимизируется их суммарный размер. Таким образом, получаемая модель является минимальной, однако независимость двух этапов приводит к тому, что модель не является наименьшей (в терминах суммарного размера охранных условий).

Резюмируя, ни один из рассмотренных методов, каждый из которых по своему хорош при конкретной постановке задачи, не позволяет *одновременно* и *эффективно* учитывать при синтезе конечно-автоматных моделей как (1) примеры поведения, так и (2) LTL-спецификацию, а также (3) минимальность генерируемых моделей. В ходе выполнения данной работы был разработан метод, который фактически является расширением [49] — объединением двух независимых этапов в один — и вносит вклад в расширение *state-of-the-art* конечно-автоматного синтеза с применением SAT-решателей, а именно: *одновременно* поддерживает учет позитивных примеров поведения, реализует индуктивный синтез, основанный на контрпримерах — для учета LTL-спецификации, а также позволяет генерировать минимальные модели — как в терминах числа состояний, так и в терминах суммарного размера охранных условий.

1.9. Задача проверки эквивалентности булевых схем

Задача проверки эквивалентности булевых схем (Logical Equivalence Checking — LEC) является одной из ключевых комбинаторных проблем в автоматизации проектирования электроники (Electronic Design Automation — EDA) и важной частью процесса верификации цифровых схем. В этом разделе даны основные понятия о LEC, которые будут использованы в дальнейшем.

Рассмотрим две булевы схемы S_f и S_h, определяющие функции $f, h : \{0,1\}^n \to \{0,1\}^m$. Задача LEC заключается в том, чтобы определить, являются ли две заданные схемы эквивалетными — обладают одинаковыми выходами на всех возможных входах, что выражается в том, что соответствующие функции поточечно равны, $f \cong h$. Задача LEC может быть сведена к задаче выполнимости булевых формул (SAT), ниже это показано на примерах.

Используя S_f и S_h, построим новую схему, обозначаемую $S_{f \triangle h}$ (см. Рисунок 3), которая получается из S_f и S_h путем «склейки» вместе входных вершин — обозначим её $S_{f \triangle h}$. Она имеет тот же набор входов V^{in}, как и схемы S_f и S_h, и определяет

следующую функцию:

$$f \triangle h \colon \{0,1\}^n \to \{0,1\}^{2m} \tag{3}$$

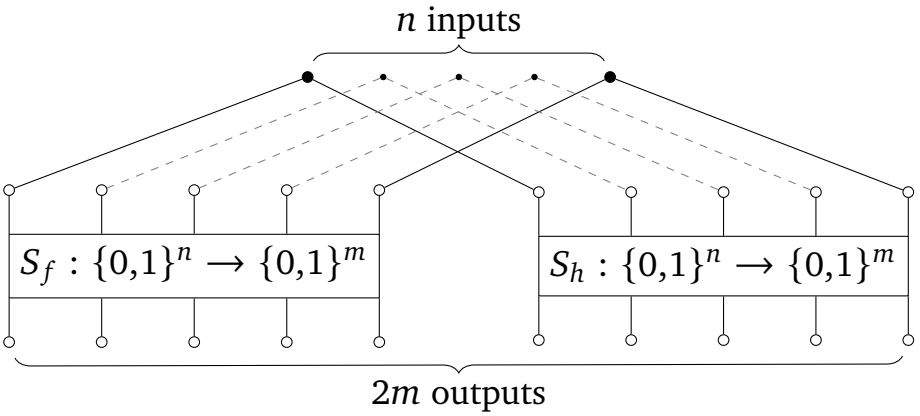


Рисунок 3 — Склеенная схема $S_{f \triangle h}$, построенная с использованием одного и того же набора входов для двух схем S_f и S_h

Обозначим через V_f^{out} и V_h^{out} множества выходов схем S_f и S_h, а через $Y_f = \{y_1^f, \ldots, y_m^f\}$ и $Y_h = \{y_1^h, \ldots, y_m^h\}$ множества переменных, связанных с вершинами из V_f^{out} и V_h^{out} соответственно, упорядоченные согласно семантике схем. Теперь рассмотрим формулу $(y_1^f \oplus y_1^h) \vee \cdots \vee (y_m^f \oplus y_m^h)$, которая задает булеву функцию $\mathcal{M} \colon \{0,1\}^{2m} \to \{0,1\}$, называемую *miter* [50]. Мы обозначим булеву схему, реализующую функцию $\mathcal{M} \circ (f \triangle h)$ как $S_{f \oplus h}$ и будем ссылаться на нее как на *miter-схему*. Рассмотрим формулу

$$C_{f \oplus h} = C_{f \triangle h} \wedge C(\mathcal{M}), \tag{4}$$

где $C_{f \triangle h}$ — шаблонная CNF для функции (3), а $C(\mathcal{M})$ выглядит следующим образом:

$$\begin{aligned} C(\mathcal{M}) = & C(w_1 \equiv (y_1^f \oplus y_1^h)) \wedge \\ & \wedge \ldots \wedge \\ & \wedge C(w_m \equiv (y_m^f \oplus y_m^h)) \wedge \\ & \wedge (w_1 \vee \ldots \vee w_m), \end{aligned}$$

где $C(f)$ — CNF-представление булевой функции, заданной формулой f. Из Леммы 1 непосредственно следует, что S_f и S_h эквивалентны тогда и только тогда, когда $C_{f \oplus h}$ невыполнима.

1.10. Задача генерации тестовых шаблонов для верификации булевых схем

Задача автоматической генерации тестовых шаблонов (Automatic Test Pattern Generation — ATPG) тесно связана с задачей проверки эквивалентности LEC [51]. Вкратце, она вытекает из следующего контекста. В процессе производства цифровых схем могут возникать некоторые дефекты. В результате дефекта один (или несколько) логических элементов (гейтов) в схеме могут «застрять» — стать постоянными и выдавать только 0 или только 1 для любой комбинации входов. Эта модель дефектов называется в литературе *моделью застревания на уровне* (*stuck-at-fault*, а конкретнее, *stuck-at-zero* или *stuck-at-one*, в зависимости от значения, на котором «застрял» гейт). Возможно, что в некоторых случаях получившаяся «дефектная» схема будет эквивалентна оригинальной. Однако проблема возникает тогда, когда её поведение отличается от оригинальной схемы. К счастью, проверка на наличие застреваний на уровне в элементах достаточно проста с помощью специальных тестов (наборов входных значений), нацеленных на выявление таких дефектов. Однако генерация этих тестов является весьма нетривиальной задачей.

Пусть S_f — это оригинальная схема, определяющая функцию $f\colon \{0,1\}^n \to \{0,1\}^m$. Обозначим через $S_{[f,v,\delta]}$ схему, полученную из S_f заменой элемента v на константу $\delta \in \{0,1\}$; эта схема определяет некоторую функцию $f'\colon \{0,1\}^n \to \{0,1\}^m$, и, следовательно, будем использовать обозначение $S_{f'} = S_{[f,v,\delta]}$. Назовём элемент v' в схеме $S_{f'}$, который застрял на уровне δ, *образом застревания элемента* v. Если $S_f \ncong S_{f'}$, то ошибка в элементе v не является критической. Предположим, что для определённого значения δ схемы S_f и $S_{f'}$ не эквивалентны. Тогда существует такой вход $\alpha_\delta \in \{0,1\}^n$, что $\gamma_\delta \neq \gamma_{\delta'}$, где $\gamma_\delta = f(\alpha_\delta)$, $\gamma'_\delta = f'(\alpha_\delta)$. Тогда $(\alpha_\delta, \gamma_\delta, \gamma'_\delta)$ представляет собой эффективно проверяемый сертификат застревания элемента v на уровне δ. Мы будем называть множество таких сертификатов для всех элементов *полным набором тестов* для рассматриваемой схемы (в контексте модели застревания на уровне). Задача построения такого набора естественным образом называется задачей автоматической генерации тестовых шаблонов (ATPG). Из приведенного описания ясно, что для построения полного набора тестов для S_f необходимо решить $2 \cdot K$ задач LEC, где K — это число элементов в S_f.

Предположим, что для оригинальной схемы S_f и конкретного элемента v построена схема $S_{f'}$. Затем для этих двух схем строится *miter*-схема $S_{f \oplus f'}$ и рассматривается выполнимость КНФ $C_{f \oplus f'}$. Однако из формулировки задачи ATPG видно, что S_f и $S_{f'}$ могут быть очень похожими. Учитывая этот факт, можно существенно

упростить задачу LEC в форме SAT для этих двух схем. Описанная ниже процедура хорошо известна в области автоматического дизайна цифровых схем (Electronic Design Automation — EDA).

Пусть V и $\tilde{V}$ — это множества вершин схем S_f и $S_{f'}$, соответственно. Существует естественная взаимно-однозначная соответствие θ между V и $\tilde{V}$: неформально, зафиксируем один и тот же порядок относительно топологической сортировки на V и $\tilde{V}$ и предположим, что $\theta(v) = \tilde{v}$, если вершины v и $\tilde{v}$ имеют одинаковый номер. Далее будем говорить, что вершины v и $\tilde{v}$, где $\tilde{v} = \theta(v)$, называются *одноименными*.

Заметим, что после объединения входов схем S_f и $S_{f'}$ при переходе к схеме $S_{f \triangle f'}$, последняя схема может содержать элементы, которые также можно объединить. Действительно, предположим, что элементы $v \in S_f$ и $\tilde{v} \in S_{f'}$ имеют одинаковые координаты — им назначены одни и те же булевы функции и они имеют одних и тех же родителей $P_v = P_{\tilde{v}}$. В этом случае можно удалить один из этих элементов и ассоциировать с оставшимся элементом потомков удаленного. В графах эта операция аналогична процедуре, используемой для объединения вершин при построении ROBDD (Reduced Ordered Binary Decision Diagram) [52], и может быть эффективно выполнена с использованием хеш-таблиц. Если схемы S_f и $S_{f'}$ не сильно отличаются, то при переходе к LEC для этих схем в форме SAT целесообразно объединить все элементы, для которых это возможно в соответствии с вышеуказанным.

Для произвольного элемента $v \in V \setminus V^{\text{in}}$ рассмотрим все пути, соединяющие v с вершинами из V^{out}, и обозначим через D_v множество всех вершин, через которые проходят эти пути (включая v). Назовем D_v *тенью* вершины v.

Лемма 2. *Пусть $v \in V \setminus V^{in}$ — произвольный элемент, а v' — образ застревания элемента v на уровне. При применении к схемам S_f и $S_{f'}$ описанной выше процедуры будут объединены все одноименные вершины, кроме (возможного) множества одноименных вершин, которые лежат в тенях D_v и $D_{v'}$.*

Доказательство этого утверждения следует непосредственно из определения процедуры объединения и факта, что $v \neq v'$ в смысле эквивалентности соответствующих координат.

Обозначим через $\tilde{S}_{f \triangle f'}$ схему, полученную из $S_{f \triangle f'}$ объединением всех возможных вершин с использованием описанной выше процедуры. Заметим, что некоторые одноименные выходы схем S_f и $S_{f'}$ также могут быть объединены. Построим для $\tilde{S}_{f \triangle f'}$ её шаблонную КНФ $\tilde{C}_{f \triangle f'}$. Обозначим через $\tilde{C}(\mathcal{M})$ КНФ-формулу, представляющую (в вышеупомянутом смысле) *miter* для одноименных выходов всех

выходных переменных S_f и $S_{f'}$, которые не были объединены. Тогда справедлива следующая теорема.

Теорема 1. *Если формула $\tilde{C}_{f\triangle f'} \wedge \tilde{C}(\mathcal{M})$ выполнима, то, решив SAT для этой формулы, получим триплет $(\alpha_\delta, \gamma_\delta, \gamma'_\delta)$ для обнаружения соответствующего дефекта в элементе v.*

Доказательство. Подобно многим другим фактам, устанавливающим взаимосвязь между свойствами схем и формулами, построенными для этих схем, это доказательство основано на Лемме 1. Из неё следует, что число присваиваний, удовлетворяющих КНФ $\tilde{C}_{f\triangle f'}$, равно 2^n, так как каждый входной вектор $\alpha \in \{0,1\}^n$ вызывает (в смысле Леммы 1) одно присваивание, удовлетворяющее $\tilde{C}_{f\triangle f'}$. Легко видеть, что для любого присваивания, удовлетворяющего $\tilde{C}_{f\triangle f'}$, существует единственный вход $\alpha \in \{0,1\}^n$, который его вызывает.

Как следует из Леммы 2, некоторый входной вектор $\alpha \in \{0,1\}^n$ может вызвать различные значения одноименных выходов только для выходных вершин, лежащих в тенях D_v и $D_{v'}$. Пусть y и y' — переменные, назначенные таким выходным вершинам. Предположим, что некоторый $\alpha \in \{0,1\}^n$ вызывает различные значения y и y'. В этом случае такое присваивание также вызовет присваивание, удовлетворяющее формулу $\tilde{C}_{f\triangle f'} \wedge \tilde{C}(\mathcal{M})$, что приведет к соответствующему триплету $(\alpha_\delta, \gamma_\delta, \gamma'_\delta)$ для обнаружения дефекта в элементе v. В противном случае, если каждое $\alpha \in \{0,1\}^n$ вызывает присваивание, в котором каждая пара выходных элементов с одинаковым номером имеют одинаковые значения, то формула $\tilde{C}(\mathcal{M})$ примет значение `False` на соответствующем присваивании, и таким образом, формула $\tilde{C}_{f\triangle f'} \wedge \tilde{C}(\mathcal{M})$ будет невыполнима. □

1.11. Задача булевой выполнимости (SAT)

Задача выполнимости булевых формул (Boolean satisfiability problem — SAT) формулируется следующим образом [53]: для произвольной булевой формулы $\varphi(x_1, \ldots, x_n)$ необходимо определить, существует ли подстановка значений переменных X_{SAT}, при которой формула становится истинной:

$$\exists X_{\text{SAT}} \in \{0,1\}^n : \varphi(X_{\text{SAT}}) = 1$$

Если такая подстановка X_{SAT} существует, то она называется *удовлетворяющей* (*satisfying assignment*; также используются термины *модель* и *интерпретация*), а

формула φ называется *выполнимой* (*SATisfiable*). В противном случае, если удовлетворяющая подстановка не существует, φ называется *невыполнимой* (*UNSATisfiable*).

Если булева формула φ представлена в конъюктивной нормальной форме (КНФ), то соответствующую задачу называют CNF-SAT. Любая булева формула может быть преобразована в эквивалентную КНФ, однако при этом размер формулы может увеличиться экспоненциально, например:

$$n \text{ конъюнкций} \left\{ \begin{array}{ll} (x_1 \wedge y_1) \vee & \\ (x_2 \wedge y_2) \vee & \\ \dots & \\ (x_n \wedge y_n) & \end{array} \right. \xrightarrow{\text{КНФ}} \left. \begin{array}{ll} (x_1 \vee x_2 \vee \dots \vee x_n) \wedge & \\ (y_1 \vee x_2 \vee \dots \vee x_n) \wedge & \\ \dots & \\ (y_1 \vee y_2 \vee \dots \vee y_n) & \end{array} \right\} 2^n \text{ дизъюнкций}$$

С помощью преобразований Цейтина [11] возможно привести любую булеву формулу в КНФ — с сохранением выполнимости (*equisatisfiable CNF*), но с добавлением новых переменных (*auxiliary variable*) — при этом размер формулы увеличится лишь линейно. В данной работе подразумевается, что все булевы выражения, кодирующие задаваемые ограничения, представляются в виде КНФ — либо подвергаются эквивалентным логическим преобразованиям, либо преобразованиям Цейтина.

1.12. Основные алгоритмы решения SAT

С учетом сказанного в предыдущем разделе, везде далее под задачей булевой выполнимости (SAT) в распознавательном варианте понимается задача распознавания выполнимости произвольной булевой формулы в КНФ. SAT является исторически первой NP-полной задачей. Данный факт был установлен С. А. Куком (без привлечения точного определения NP-полноты) в статье [54], которая является основополагающей работой для структурной теории сложности алгоритмов. Помимо распознавательного варианта далее нас будет интересовать поисковый вариант SAT — когда требуется распознать выполнимость КНФ, и в случае ее выполнимости найти произвольный выполняющий набор. Данная задача, соответственно, NP-трудна. Сказанное означает, что — в предположении $P \neq NP$ — SAT не может быть решена в общем случае за полиномиальное время. При всем этом существует масса разнообразных аргументов в пользу того, что SAT (как и многие другие NP-трудные задачи) не является сложной в большинстве своих частных случаев. Именно этот факт позволяет использовать современные алгоритмы решения SAT в задачах, комбинаторная размерность которых может быть колоссальной. Так, в символьной верификации удается успешно решать SAT в отношении КНФ, включающих миллионы

дизъюнктов. Число переменных, встречающихся в таких КНФ, может исчисляться десятками и сотнями тысяч. Существуют два больших класса алгоритмов решения SAT — полные и неполные. Неполные алгоритмы плохо подходят или не подходят совсем для решения задач, в которых требуется доказывать невыполнимость (соответственно, например, для символьной верификации). Однако они вполне могут использоваться для обращения функций.

В настоящей работе основным вычислительным инструментом являются полные алгоритмы решения SAT. Именно такие алгоритмы позволяют точно решать задачи верификации автоматов и булевых схем — находить решение, если оно существует, либо гарантировать его отсутствие при заданных параметрах, что в том числе позволяет точно решать задачу минимизации.

1.12.1. Неполные алгоритмы решения SAT

Неполные алгоритмы решения SAT не гарантируют ответ SAT/UNSAT за конечное время для произвольной КНФ. Существует целый ряд различных концепций, лежащих в основе таких алгоритмов. Статья [55] представляет собой детальный обзор по данному вопросу, содержащий ключевые ссылки. В дальнейшем нас будут интересовать только те неполные алгоритмы решения SAT, в основе которых лежит идеология локального поиска или (в отдельных случаях) эволюционные стратегии. В таких алгоритмах задача SAT в отношении КНФ C над множеством из n переменных, состоящей из m дизъюнктов, рассматривается в форме задачи максимизации псевдобулевой функции следующего вида:

$$f_C\colon \{0,1\}^n \to \{0,1,\ldots,m\} \tag{5}$$

Псевдобулевой называется [56] любая функция следующего вида:

$$f\colon \{0,1\}^n \to \mathbb{R} \tag{6}$$

На произвольном наборе $\alpha \in \{0,1\}^n$ значение функции (5) равно числу дизъюнктов в C, которые на этом наборе обращаются в единицу. Фактически в данном случае рассматривается задача SAT в оптимизационной постановке — известная как MaxSAT.

Рассматривая $\{0,1\}^n$ в роли пространства поиска, можно ввести на нем *функцию окрестностей* [57] $\aleph\colon \{0,1\}^n \to 2^{\{0,1\}^n}$. Проще всего для этой цели использовать метрику Хэмминга [58], в рамках которой для произвольной точки $\alpha \in \{0,1\}^n$ ее

окрестность Хэмминга радиуса $r \geqslant 1$, определяется следующим образом:

$$\aleph_r(\alpha) = \{\alpha' \in \{0,1\}^n \mid \rho_H(\alpha,\alpha') \leqslant r\} \tag{7}$$

В (7) через $\rho_H(\alpha,\alpha')$ обозначено расстояние Хэмминга между словами α и α'. Чаще всего рассматриваются окрестности радиуса 1. В такой постановке для решения SAT и MaxSAT может использоваться, без преувеличения, огромный арсенал методов локального поиска. Проиллюстрируем общую идею, лежащую в основе таких методов, на примере простейшего алгоритма, известного как «восхождение к вершине» (*Hill Climbing* — далее HC) [59]. Опишем вариант HC, применимый для максимизации произвольной псевдобулевой функции вида (6).

1. Выбираем (вообще говоря, произвольным образом, например, случайно в соответствии с равномерным распределением на $\{0,1\}^n$) стартовую точку $\alpha_0 \in \{0,1\}^n$, вычисляем $f(\alpha_0)$; считаем α_0 текущей точкой, а $f(\alpha_0)$ текущим значением f.
2. Пусть $\alpha \in \{0,1\}^n$ — текущая точка.
   Обходим в некотором порядке $\aleph_1(\alpha) \setminus \{\alpha\}$, вычисляя для каждой точки α' из данного множества $f(\alpha')$. Если найдена такая точка $\alpha' \in \aleph_1(\alpha) \setminus \{\alpha\}$, что $f(\alpha') > f(\alpha)$, перейти на шаг 3, в противном случае перейти на шаг 4.
3. $\alpha \leftarrow \alpha'$, $f(\alpha) \leftarrow f(\alpha')$, перейти на шаг 2.
4. $(\alpha,f(\alpha))$ — локальный максимум функции f на $\{0,1\}^n$ (поскольку для любой $\alpha' \in \aleph_1(\alpha) \setminus \{\alpha\}$ имеет место $f(\alpha') \leqslant f(\alpha)$). В этом случае алгоритм либо останавливается и выдает в качестве ответа $(\alpha, f(\alpha))$, либо запускает некоторую процедуру выхода из локального максимума.

Для выхода из локальных экстремумов можно дополнять приведенный выше базовый алгоритм различными техниками, основанными на эвристических и метаэвристических соображениях. Зачастую, выйдя из точки локального экстремума, можно оказаться в точке с худшим значением f. Такого сорта «выпрыгивания» из локальных экстремумов могут осуществляться в процессе поиска неоднократно. В этом случае обычно хранится точка с самым лучшим значением функции f, достигнутым за все историю поиска. Такое значение называется *рекордом* (*best known value* — BKV). Современные техники выхода из локальных экстремумов позволяют даже при решении весьма трудных задач многократно улучшать рекорд в процессе поиска. Если некоторое число попыток выхода из локальных экстремумов не дает улучшения текущего рекорда f^*, достигнутого в точке α^*, то алгоритм останавливается и выдает в качестве ответа пару (α^*, f^*).

Предположим, что Hill Climbing применяется к задаче максимизации произвольной функции вида (5). Легко понять, что даже если дополнить HC какой-либо процедурой выхода из точек локального максимума, это не позволит безошибочно распознавать невыполнимость невыполнимых КНФ. С другой стороны, если рассматривается КНФ с большим числом выполняющих наборов, то HC (даже в самом простом варианте) может случайно натолкнуться на такой набор за приемлемое время. Известный пример данного типа — успешное использование HC для решения SAT в отношении КНФ, кодирующих задачу размещения k ферзей на шахматной доске размерности $k \times k$ [60]. Однако, к сожалению, HC и известные его модификации напрямую неприменимы к КНФ, кодирующим обращение интересных с практической точки зрения криптографических функций. Причем это верно даже в отношении КНФ с огромным числом выполняющих наборов — например, для КНФ, кодирующих задачи обращения криптографических хеш-функций.

Если некоторый алгоритм локального поиска, решающий задачу максимизации функции вида (5), слишком долго не может улучшить текущий рекорд $(\alpha, f_C(\alpha))$, где $f_C(\alpha) < m$, то данный алгоритм можно остановить с ответом UNSAT в отношении КНФ C. Этот ответ может оказаться ошибочным. Однако существуют специальные техники рандомизации локального поиска, использование которых позволяет оценивать вероятность ошибки указанного типа и даже снижать эту вероятность за счет выполнения алгоритмом большого числа некоторых случайных шагов. Один из самых известных примеров такого рода — алгоритм, предложенный У. Шёнингом в [61]. Данный алгоритм решает задачу о выполнимости произвольной k-КНФ — такой КНФ, где каждый дизъюнкт которой имеет длину $k \geqslant 2$. Алгоритм Шёнинга пытается улучшить значение функции (5), достигнутое в точке $\alpha \in \{0,1\}^n$, за счет случайного выбора и модификации тех дизъюнктов в КНФ C, которые обращаются в ноль на наборе α. Получаемый в результате процесс интерпретируется в рамках хорошо изученной в теории вероятностей модели случайных блужданий [62]. Как результат, если алгоритм Шёнинга, сделав $O(n \cdot (2 - \frac{2}{k})^n)$ простых случайных действий, не находит выполняющий набор, то вероятность ошибиться, заключив, что C невыполнима, не превосходит $1 - \frac{1}{poly(n)}$, где через $poly(\cdot)$ обозначен некоторый полином. Если повторение упомянутого выше списка действий, допустим, $n \cdot p(n)$ раз не дает выполняющий набор, то заключение о невыполнимости C будет справедливым с вероятностью, которая близка к $1 - e^{-n}$. Идеи, лежащие в основе алгоритма Шёнинга, позволили для SAT в отношении 3-КНФ построить нетривиальные верхние

оценки сложности, которые долгое время оставались лучшими среди аналогичных по смыслу оценок (см. [63]).

Алгоритмы, в которых базовые схемы локального поиска (например, HC) дополняется различными стратегиями рандомизации, относятся к классу, известному как *Stochastic Local Search methods* (SLS). К сожалению, имеющиеся на сегодняшний день SLS-алгоритмы плохо подходят для обращения криптографических функций. Как будет показано далее, лучшие такие алгоритмы позволяют успешно решать задачи криптоанализа лишь весьма простых генераторов ключевого потока.

1.12.2. Полные алгоритмы решения SAT

Алгоритм решения SAT называется полным, если для любой КНФ C он за конечное число шагов выдает верный ответ вида SAT/UNSAT. Как и в ситуации с неполными, для построения полных алгоритмов решения SAT можно использовать множество различных базовых концепций. Так, довольно естественно решать SAT, основываясь на широко применяемой в комбинаторной оптимизации идеологии ветвей, границ и отсечений [64]. Хорошо известна сводимость SAT к задаче 0-1 целочисленного линейного программирования (0-1-ЦЛП), после осуществления которой можно использовать для решения полученной задачи из семейства 0-1-ЦЛП богатый набор программных средств. Также довольно просто перейти от SAT к задаче поиска решений системы алгебраических уравнений степени не выше двух над полем $\mathrm{GF}(2)$. К таким системам можно применять алгоритм Б. Бухбергера (он же «метод баз Грёбнера») [65], а также специальные техники работы с разреженными квадратичными системами над $\mathrm{GF}(2)$ (см., например, [66; 67]).

Многие авторы сходятся во мнении, что лучших результатов в решении трудных вариантов SAT из целого ряда областей, среди которых символьная верификация и криптоанализ, удается добиться за счет использования алгоритмов, относящихся к направлению, которое правильнее всего назвать «Вычислительная логика». Ранние алгоритмы из данного класса лежали в основе первых программных реализаций систем автоматического доказательства теорем (Automated Theorem Proving — ATP). Одним из таких алгоритмов был «метод Девиса-Патнема» (Davis-Putnam method) [68]. Усовершенствованная версия данного алгоритма, известная как DPLL (от фамилий Davis, Putnam, Logemann, Loveland) [69], до настоящего момента продолжает использоваться в основе высокоэффективных полных SAT-решателей. DPLL представляет собой обход дерева поиска, в рамках которого выход из тупиковых ветвей организован в форме процедуры, называемой *хронологическим бэктрекингом*

(*chronological backtracking*). Остановимся подробнее на тех деталях DPLL, которые потребуются нам в дальнейшем.

1.12.3. Алгоритм DPLL

Пусть C — произвольная КНФ над множеством переменных X. Пусть $S \subset L_X$ — произвольное множество литералов над переменными из X, которое не содержит контрарных литералов. Пусть X_S — множество, включающее все те переменные из X, литералы над которыми содержатся в S, то есть $X_S \subseteq X$. Рассмотрим отображение $\sigma\colon X_S \to \{0,1\}$, заданное по следующему правилу:

$$\sigma(x \in X_S) = \begin{cases} 1 & \text{если } x \in S \\ 0 & \text{если } \neg x \in S \end{cases} \tag{8}$$

Иными словами, отображение (8) связывает с выбираемыми из L_X литералами значения соответствующих переменных: выбор литерала x интерпретируется как присвоение переменной x значения 1, выбор же $\neg x$ соответствует принятию x значения 0. Множество S будем называть списком литералов, выбранных из L_X. Теперь опишем собственно алгоритм DPLL.

На начальном шаге список выбранных литералов пуст. Выберем (вообще говоря, произвольный) литерал $l_1 \in L_X$, поместим его в список S_1 и рассмотрим КНФ $l_1 \wedge C$. Удалим из данной КНФ каждый дизъюнкт вида $(l_1 \vee D)$, а из каждого дизъюнкта вида $(\neg l_1 \vee D)$ удалим литерал $\neg l_1$ (здесь через D обозначен произвольный непустой дизъюнкт над X). Обозначим результирующую КНФ через C'. Будем говорить, что данная КНФ получена из $l_1 \wedge C$ в результате применения правила *единичного дизъюнкта* (Unit Propagation rule — UP [70]) к C и литералу l_1. Заметим, что если в C содержался дизъюнкт вида $D = (\neg l_1 \vee l')$, то удаление из D литерала $\neg l_1$ дает единичный дизъюнкт, состоящий из литерала l'. В описанной ситуации литерал l_1 называют *угаданным*, а про литерал l' говорят, что он был *выведен по правилу единичного дизъюнкта*. К КНФ C' и литералу l' можно снова применить правило единичного дизъюнкта. Если в результате применения UP вывелось несколько литералов, они все ставятся в очередь, после чего к ним и соответствующим КНФ последовательно применяется UP.

Пусть $S_k = \{l_1, \ldots, l_k\}$ — список угаданных литералов. Обозначим через $\widetilde{S}_k$ список всех литералов, которые были выведены по правилу UP в соответствии с описанной выше процедурой. Пусть C_k — полученная в результате КНФ. Проанализируем несколько ситуаций, которые могут при этом возникнуть.

1. Пусть $\widetilde{S}_k$ не содержит контрарных литералов, а C_k содержит только единичные дизъюнкты. Тогда C выполнима. Следовательно, существует выполняющий C набор, в котором значения части (или всех) переменных определяются при помощи отображения (8), применяемого к литералам из $S_k \cup \widetilde{S}_k$.
2. Пусть $\widetilde{S}_k$ не содержит контрарных литералов, а C_k содержит дизъюнкты длины не менее 2. Пусть $\widetilde{C}_k$ — КНФ, составленная из этих дизъюнктов, а $\widetilde{X}_k$ — множество переменных, встречающихся в $\widetilde{C}_k$. Выберем из $L_{\widetilde{X}_k}$ произвольный литерал l_{k+1}, построим список $S_{k+1} = S_k \cup \{l_{k+1}\}$ и применим UP к $\widetilde{C}_k$ и l_{k+1}.
3. Список $\widetilde{S}_k$ содержит контрарные литералы — пару вида $(l,\neg l)$ для некоторого $l \in L_X$. В этой ситуации будем говорить, что список угаданных литералов S_k породил конфликт. Сам по себе конфликт еще не означает, что исходная КНФ невыполнима — вполне возможно, что она выполнима, но были угаданы такие литералы, что сочетание соответствующих им в смысле отображения (8) значений переменных не встречается ни в одном из выполняющих C наборов.

Пусть $S_k = \{l_1, \ldots, l_k\}$ — список угаданных литералов, такой что после угадывания l_k по UP был выведен конфликт. В этой ситуации перейдем от списка S_k к списку $S'_k = \{l_1, \ldots, \neg l_k\}$, помечая литерал $\neg l_k$ как «инвертированный». Предположим, что для некоторого k оба списка $S_k = \{l_1, \ldots, l_k\}$ и $S'_k = \{l_1, \ldots, \neg l_k\}$ порождают конфликты. Пусть l_r — ближайший предшествующий l_k литерал в списке S_k, который ранее не был инвертирован. Тогда новым списком является $S'_r = \{l_1, \ldots, \neg l_r\}$ (везде здесь предполагалось, что $k,r \geqslant 2$).

Описанная выше процедура перехода к списку S'_r называется хронологическим (обычным) бэктрекингом. Можно заметить, что процесс бэктрегинга соответствует обходу с возвратом бинарного дерева специального вида. Вершинам этого дерева приписаны переменные из X. Из произвольной вершины, которой приписана переменная x, выходит два ребра, символизирующие литералы x и $\neg x$. Корню данного дерева приписана переменная, над которой берется литерал l_1. Произвольная ветвь соответствует некоторому списку угаданных литералов. Если такой список порождает конфликт, то соответствующую ветвь назовем тупиковой.

Если для некоторого k списки S_k и S'_k порождают конфликты, и при этом все литералы, предшествующие l_k, включая первый, были инвертированы, то каждая ветвь описанного выше дерева поиска является тупиковой. Легко понять, что данный факт означает невыполнимость КНФ C. Также в силу всего сказанного

выше можно заметить, что число вершин в таком дереве поиска не превосходит $M = 2^{n+1} - 1$. Отметим, что в реальности это число может быть существенно меньше за счет большого числа литералов, выведенных по UP. Таким образом, применив UP не более M раз, либо достигнем ситуации, описанной в пункте 1, и это будет означать, что C выполнима, либо докажем невыполнимость C. Все сказанное означает полноту алгоритма DPLL.

1.12.4. Концепция CDCL

Концепция решения SAT, известная сегодня как Conflict Driven Clause Learning (CDCL), включает в себя ряд важных техник, дополняющих алгоритм DPLL. Первая и главная из них позволяет записывать информацию о конфликте, который возник в процессе обхода дерева поиска, в виде специальным образом построенного дизъюнкта. Такой дизъюнкт называется *конфликтным* (conflict-induced clause). Если C — исходная КНФ, а D — конфликтный дизъюнкт, то имеет место $C \rightarrow D$, то есть D — это логическое следствие (импликация) из C. Соответственно, КНФ C выполнима тогда и только тогда, когда выполнима КНФ $C \wedge D$.

Конфликтные дизъюнкты можно строить различными способами. Приведем простейший пример. Пусть список угаданных литералов $S_k = \{l_1, \ldots, l_k\}$ породил конфликт в рамках алгоритма DPLL. Построим следующий конфликтный дизъюнкт: $D = (\neg l_1 \vee \ldots \vee \neg l_k)$. Данный дизъюнкт запрещает одновременный выбор всех литералов из списка S_k. Рассмотрим КНФ $C \wedge D$ (C — исходная КНФ). Если при применении к КНФ $C \wedge D$ алгоритма DPLL используется список угаданных литералов $S_{k-1} = \{l_1, \ldots, l_{k-1}\}$, то единичный дизъюнкт $\neg l_k$ будет выведен по правилу UP. В этом случае говорят, что вывод литерала $\neg l_k$ индуцирован конфликтом. Очень важно, что в данном случае литерал $\neg l_k$ не угадывается, а выводится по правилу UP на основе информации, полученной в результате анализа конфликта.

Впервые идея использовать конфликтные дизъюнкты для записи информации о тупиковых ветвях в DPLL-поиске была предложена в конференционной статье Ж. Маркеша-Сильвы и К. Сакаллы в 1996 году [71]. В более детальном виде она была представлена этими же авторами в журнальной статье [72]. По сути, именно в этих двух работах были заложены основы концепции CDCL. Еще одна важная техника, которая была описана в [71; 72], заключается в использовании для представления процесса решения SAT специальных графов, называемых *графами вывода* (*implication graph* — IG). Граф вывода позволяет эффективно выявлять литералы (причем, что важно, как угаданные, так и выведенные по UP), которые ответственны за

рассматриваемый конфликт. Графы вывода очень информативны. Разные способы обхода IG соответствуют различным эвристикам формирования конфликтных дизъюнктов. Некоторые такие эвристики также были приведены в [71; 72].

Основное концептуальное отличие CDCL от DPLL заключается в том, что CDCL использует память для хранения информации о ходе поиска в форме конфликтных дизъюнктов. Это позволяет вместо хронологического бэктрекинга в ряде случаев осуществлять *нехронологический бэктрекинг* (*non-chronological backtracking*), называемый также *бэкджампингом* (backjumping). Бэкджампинг — это ситуация, когда после анализа конфликта откат в списке угаданных литералов происходит не к ближайшему (от конфликта) литералу, который не был ранее инвертирован, а к угаданному еще раньше (иногда существенно раньше). Во многих случаях бэкджампинг позволяет эффективно отсекать значительные части дерева поиска, запрещая при помощи конфликтных дизъюнктов последующий поиск в этих областях. Первым SAT-решателем, фактически использующим CDCL, был GRASP [71].

Следующий шаг был сделан в работах [73; 74], где были введены еще несколько важных техник, дополняющих базовый CDCL. Так, в [73] был описан механизм выбора порядка угадывания литералов, основанный на их «мере конфликтности». Соответствующая эвристика получила называние VSIDS (Variable State Independent Decaying Sum). Также в [73] была описана весьма эффективная техника итеративного применения правила UP, использующая так называемые «ленивые» (*lazy*) структуры данных, известные как *watched literals*, применяются в настоящее время в большинстве CDCL SAT-решателей. Еще одно важное достижение [73] — экспериментальная аргументация пользы рестартов. В статье [74] было проведено детальное исследование различных способов построения конфликтных дизъюнктов за счет анализа графов вывода. На основе результатов работ [73; 74] был построен решатель zchaff — первый по-настоящему высокоэффективный SAT-решатель, базирующийся на концепции CDCL. С использованием zchaff еще в 2002 году удавалось решать задачи криптоанализа некоторых поточных шифров существенно быстрее простого перебора.

В 2003 году в работе [75] были описаны общие принципы построения высокоскоростного CDCL SAT-решателя с эффективно модифицируемой архитектурой. Исходный код соответствующего решателя, получившего название MiniSat, был представлен авторами в открытом доступе. Решатель MiniSat на протяжении многих лет остается де-факто стандартом программной основы эффективных SAT-решателей как широкого профиля, так и нацеленных на конкретную прикладную область.

Еще одной важной частью MiniSat стали процедуры периодической чистки баз конфликтных дизъюнктов. Здесь следует отметить, что грамотно реализованный CDCL в процессе работы может генерировать огромные массивы конфликтной информации. Чрезмерное число конфликтных дизъюнктов увеличивает число срабатываний правила UP и, как следствие, приводит к падению эффективности вывода. Соответственно, часть конфликтной информации можно попытаться удалить. Однако неудачное удаление конфликтных дизъюнктов может привести к их повторной генерации. В данном контексте особенно ценны эвристики, которые позволяют удалять большое число слабо релевантных конфликтных дизъюнктов. Первые относительно нетривиальные такие эвристики были предложены в статье [76].

Алгоритм 1: CDCL — расширенный алгоритм DPLL с анализом конфликтов и изучением дизъюнктов

Входные данные: булева формула F, текущая модель σ (изначально пустая)
Результат: SAT и удовлетворяющая модель σ, либо UNSAT

```
1  while не все переменные назначены do
2      Выбираем неозначенную переменную v
3      if формула F становится невыполнимой при σ ∪ {v} then
4          β ← АнализКонфликта(F, σ, v)
5          F ← F ∪ {β}
6          Возврат на предыдущий уровень решения
7      else
8          Продолжение рекурсивного поиска
9  if все переменные назначены then
10     return (SAT, σ)
11 else
12     return UNSAT
```

Алгоритм CDCL (Conflict-Driven Clause Learning) представляет собой расширение классического алгоритма DPLL (Davis-Putnam-Logemann-Loveland), включающее в себя механизмы анализа конфликтов и изучение новых дизъюнктов. Его псевдокод представлен в виде Алгоритма 1. Входными данными алгоритма являются булева формула F и (изначально пустая) модель σ (означивание переменных). Алгоритм рекурсивно ветвится по переменным — выбирает очередную неназначенным переменную и пытается присвоить ей какое-то значение (можно выбирать это значение

эвристически [73]; простейший же подход — всегда выбирать отрицательный литерал). Если при добавлении означенной переменной в текущую модель σ формула F становится невыполнимой — возникает конфликт, то выполняется анализ этого конфликта, в результате чего выводится новый дизъюнкт β. Этот дизъюнкт добавляется к формуле F для предотвращения повторения той же конфликтной ситуации в будущем, после чего происходит возврат на предыдущий уровень решений для продолжения поиска. Если конфликтов не возникает, алгоритм продолжает рекурсивный поиск. Процесс повторяется, пока не будут назначены все переменные, что приводит к нахождению удовлетворяющей модели. При получении конфликта на нулевом уровне рекурсии алгоритм возвращает *UNSAT* — доказывает невыполнимость исходной формулы F.

Алгоритмы решения SAT, основанные на CDCL, оказались весьма удачно приспособленными для применения к ним различных концепций распараллеливания. Основными двумя такими концепциями являются так называемый «портфолио-подход» (*portfolio approach*) и «подход на основе разбиений» (*partitioning approach*). Детальное сравнение эффективности этих двух подходов предпринято в диссертации А. Хиваринена [77].

Портфолио-подход предполагает запуск нескольких копий решателя на исходном пространстве поиска, при этом каждая копия начинает работу, используя некоторый набор значений входных параметров SAT-решателя (разным копиям соответствуют различные наборы значений параметров). В процессе работы копии решателей могут обмениваться друг с другом конфликтными дизъюнктами. Для достижения высокой скорости такой обмен обычно организуется через оперативную память вычислительного устройства с использованием технологий многопоточного программирования. Соответствующая техника получила название *clause sharing* («обмен конфликтными дизъюнктами»). Одна из первых эффективных реализаций обмена конфликтными дизъюнктами была представлена в [78].

Подход на основе разбиений (*SAT-partitioning*) предполагает разбиение пространства поиска (фактически, множества $\{0,1\}^n$, где n — число переменных в КНФ) на непересекающиеся подобласти, которые обрабатываются независимо друг от друга. Данный подход позволяет организовать решение SAT в параллельной среде со слабо связанными или даже независимыми рабочими процессами (в частности, в грид-средах). Как будет показано далее, подход на основе разбиений дает хорошие результаты при решении SAT-задач, кодирующих криптоанализ блочных и поточных

шифров, поскольку естественным образом ассоциируется с атаками, относящимися к классу «угадывай и определяй» (*guess and determine*) [3].

Скажем здесь несколько слов по поводу теоретических аргументов эффективности CDCL. Соответствующие результаты относятся к теории сложности пропозициональных доказательств. В этой области исследуется задача доказательства невыполнимости невыполнимой формулы в КНФ. Пусть C — произвольная невыполнимая КНФ и x_C — двоичное слово, представляющее C в некоторой «разумной» системе кодирования. Пусть $\Sigma_U \subset \{0,1\}^*$ — множество слов x_C по всем возможным невыполнимым КНФ C. Пусть A — произвольный полный алгоритм решения SAT. Любой такой алгоритм называется также системой пропозиционального доказательства (*propositional proof system*). Получив на вход x_C, алгоритм A выдает двоичное слово s, которое будем называть A-доказательством невыполнимости C (см., например, [79]). Рассмотрим функцию $\omega_A\colon \{0,1\}^* \to \{0,1\}^*$, определенную следующим образом. Если слово является A-доказательством невыполнимости некоторой КНФ C, то ω_A сопоставляет этому слову слово x_C. В противном случае выходом ω_A является двоичный код символа $\varnothing$. Несложно понять, что для одной и той же КНФ C могут существовать различные A-доказательства ее невыполнимости (особенно хорошо это видно на примере метода резолюций). С произвольным $x_C \in \Sigma_U$ свяжем длину кратчайшего A-доказательства невыполнимости C. Можно заметить, что если для некоторого A функция длины кратчайшего A-доказательства по всем $x_C \in \Sigma_U$ растет как полином от $|x_C|$, то NP = coNP (см. [80]). Однако для целого ряда алгоритмов несложно указать примеры бесконечных семейств невыполнимых КНФ с полиномиально растущей длиной кратчайшего A-доказательства на этих КНФ.

Пусть теперь A и B — два полных алгоритма решения SAT. Пусть C — произвольная формула из Σ_U. Если существует полиномиальный алгоритм, который произвольное A-доказательство невыполнимости C преобразует в B-доказательство ее невыполнимости, то говорят, что система доказательств B полиномиально моделирует систему доказательств A. Если A полиномиально моделирует B, а B полиномиально моделирует A, то данные системы доказательств называются полиномиально эквивалентными. Если на некотором (бесконечном) семействе противоречий $\Sigma'_U \subset \Sigma_U$ длина кратчайшего A-доказательства растет как полином от длины КНФ, а длина кратчайшего B-доказательства как экспонента, то B не может полиномиально моделировать A. Если при этом A полиномиально моделирует B, то разумно считать систему A мощнее системы B.

В серии работ конца 90-х–начала 00-х годов были получены результаты, касающиеся сложности доказательств в системах, связанных с методом резолюций [81]. В данном контексте важнейший для нас результат содержится в статьях [82; 83], где было показано, что «общая резолюция» (*general resolution*) в ее пропозициональном варианте полиномиально эквивалентна алгоритму CDCL с рестартами. Данный факт, в частности, означает, что CDCL имеет экспоненциальную сложность. Действительно, в работе [84] было установлено, что общая резолюция экспоненциальна на семействе логических противоречий, известных как «формулы Дирихле» (*Pigeon Hole Principle formulas* — PHP_n^{n+1} [85]). В силу сказанного выше, это означает, что и CDCL будет иметь на PHP_n^{n+1} экспоненциальную сложность. В то же время, CDCL является более мощной системой доказательств, чем DPLL [82; 83; 85].

1.12.5. SAT-решатели

На практике для решения задачи SAT используются специализированные программные средства — SAT-*решатели*. Несмотря на то, что задача SAT имеет экспоненциальную оценку сложности (при условии, что $P \neq NP$), современные SAT-решатели способны решать формулы с миллионами переменных за обозримое время. Для выбора наиболее эффективного SAT-решателя можно руководствоваться результатами соревнования SAT Comptetition [86]: среди текущих лидеров можно выделить MapleCOMSPS [87], CaDiCaL [88], CryptoMiniSat [89], Glucose [76] и Plingeling [90], хотя на практике эффективность решателей может значительно отличаться, в зависимости от класса рассматриваемых задач. В некоторых случаях хорошие результаты также показывает MiniSat [75], являющийся минимальной реализацией CDCL-решателя (*Conflict-Driven Clause Learning* [71]) и служащий основой для многих других решателей (например, CryptoMiniSat и Glucose).

1.13. Ограничения на кардинальность

Ограничение на кардинальность некоторого заданного множества булевых переменных E обозначается $\Psi(E, \mu, \rho)$ и заключается в следующем: минимум μ, но максимум ρ переменных из E должны иметь истинное значение. Такое ограничение может быть выражено в виде псевдо-булевого ограничения $\mu \leqslant \sum_{e \in E} bool2int(e) \leqslant \rho$, где μ и ρ — нижняя и верхняя границы, а $bool2int(e)$ обозначает функцию, которая переводит булевы значения `true` и `false` переменнойЁe в целые числа 1 и 0, соответственно.

Для того, чтобы закодировать такое ограничение в SAT — сформировать формулу в конъюктивной нормальной форме (КНФ), которая выполнима тогда и только тогда, когда выполняется ограничение на кардинальность $\Psi(E, \mu, \rho)$. В данной работе используется техника *totalizer* [91], которая заключается в формировании унарной записи числа, представляющего сумму элементов заданного множества булевых переменных.

В оригинальной статье [91] используется следующая нотация: множество *входных переменных* обозначается $E = \{e_1, \ldots, e_n\}$, где n — их число; множество *выходных переменных*, соответствующих *битам* в унарной записи суммы, обозначается $S = \{s_1, \ldots, s_n\}$ и состоит из n новых переменных; множество *связующих переменных* обозначается L. Взаимосвязь между этими множествами может быть представлена с помощью бинарного дерева следующим образом. Начнем построение бинарного дерева с корня: отметим эту вершину числом n, а затем будем итеративно добавлять к каждому листу дерева, отмеченному число $m > 1$, двух потомков, отмеченных числами $\lfloor m/2 \rfloor$ и $(m - \lfloor m/2 \rfloor)$. В результате будет получено бинарное дерево с n листьями, отмеченных единицами. С каждым листом дерева i ассоциируется множество $\{e_i\}$, содержащее одну соответствующую входную переменную $e_i \in E$. С корнем ассоциируется все множество выходных переменных S. С каждой внутренней вершиной дерева (кроме корня), отмеченной числом m, ассоциируется множество из m новых переменных, которые впоследствии будут иметь смысл *битов* в унарной записи суммы числа, представляемого соответствующим поддеревом. Совокупность всех новых переменных во внутренних вершинах дерева образует множество связующих переменных L. На рисунке 4 изображен пример бинарного дерева, построенного для $n = 5$.

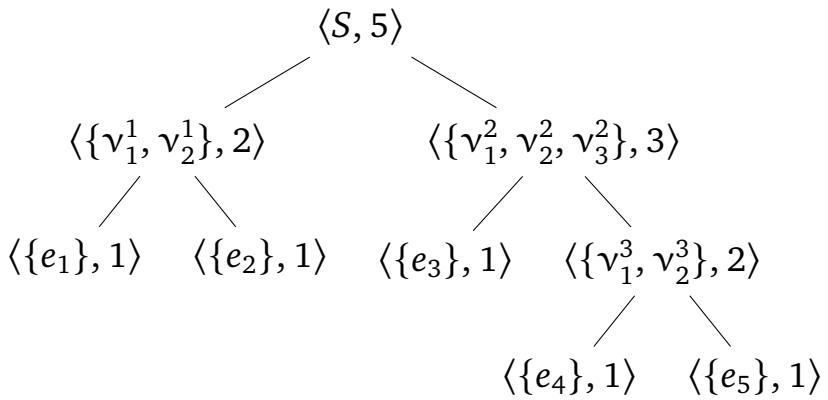


Рисунок 4 — Бинарное дерево для $n = 5$ со связующими переменными $L = \{\nu_1^1, \nu_2^1, \nu_1^2, \nu_2^2, \nu_3^2, \nu_1^3, \nu_2^3\}$

Рассмотрим внутреннюю вершину r, имеющую двух потомков a и b. Эта вершина r представляет собой унарную сумму $\alpha + \beta$, где α и β — числа, которыми отмечены вершины a и b, соответственно. Пусть $R = \{r_1, \ldots, r_m\}$ — множество переменных, ассоциированных с вершиной r. Аналогично, $A = \{a_1, \ldots, a_{m_1}\}$ и $B = \{b_1, \ldots, b_{m_2}\}$ — множества, ассоциированные с потомками a и b. Для вершины r объявляется следующий набор конъюнктов:

$$\bigwedge_{\substack{0 \leqslant \alpha \leqslant m_1 \\ 0 \leqslant \beta \leqslant m_2 \\ 0 \leqslant \sigma \leqslant m \\ \alpha+\beta=\sigma}} \left[C_1(\alpha, \beta, \sigma) \wedge C_2(\alpha, \beta, \sigma) \right],$$

где используется следующая нотация:

$$\begin{aligned} a_0 = b_0 = r_0 &= 1 \\ a_{m_1+1} = b_{m_2+1} = r_{m+1} &= 0 \\ C_1(\alpha, \beta, \sigma) &= \overline{a_\alpha} \vee \overline{b_\beta} \vee r_\sigma \\ C_2(\alpha, \beta, \sigma) &= a_{\alpha+1} \vee b_{\beta+1} \vee \overline{r_{\sigma+1}} \end{aligned}$$

Отметим, что $C_1(\alpha, \beta, \sigma)$ отвечает за соотношение $\sigma \geqslant \alpha + \beta$, в то время как $C_2(\alpha, \beta, \sigma)$ отвечает за соотношение $\sigma \leqslant \alpha + \beta$.

Множество всех C_1 и C_2 для всех троек вершин r, a, b в бинарном дереве образует так называемый *сумматор* (*totalizer*) $\Phi(E)$. Для того, чтобы полученный сумматор представлял желаемый интервал, соответствующий ограничению на кардинальность $\mu \leqslant |\{e_i \in E \mid e_i = \top\}| \leqslant \rho$, необходимо объявить так называемый *компаратор* $\Omega(S, \mu, \rho)$, где $S = \{s_1, \ldots, s_n\}$ — множество из $n = |E|$ *вспомогательных* переменных, создаваемых дополнительно во время кодирования:

$$\Omega(S, \mu, \rho) = \bigwedge_{1 \leqslant i \leqslant \mu} s_i \bigwedge_{\rho+1 \leqslant j \leqslant n} \overline{s_j}$$

В результате, ограничение на кардинальность $\Psi(E, \mu, \rho)$ представляется в виде конъюнкции сумматора $\Phi(E)$ и компаратора $\Omega(S, \mu, \rho)$.

1.14. Разбиение задачи SAT

Для эффективного решения сложных задач SAT часто целесообразно использовать некоторые методы разделения исходной задачи на более простые. Естественный способ декомпозиции задачи SAT на подзадачи называется «разбиением» (*partitioning approach*) [77].

Рассмотрим произвольную КНФ-формулу C над множеством булевых переменных X и множество $\Pi = \{G_1, \ldots, G_s\}$, где G_i, $i \in \{1, \ldots, s\}$, являются некоторыми булевыми формулами над X. Будем говорить, что Π является *разбиением (partitioning) задачи SAT для* C, если выполняются следующие условия:

1. Формулы C и $C \wedge (G_1 \vee \cdots \vee G_s)$ эквивалентны по выполнимости.
2. Для любых $i \neq j \in \{1, \ldots, s\}$, формула $C \wedge G_i \wedge G_j$ невыполнима.

1.14.1. Необходимые основы теории вероятности

Ниже будут использоваться некоторые вероятностные рассуждения для оценки сложности кодировок SAT для LEC задач с использованием разбиения SAT. Ниже описаны некоторые основные факты из теории вероятностей.

Пусть ξ — некоторая случайная величина с конечным набором числовых значений (спектром) $\mathrm{Spec}(\xi) = \{\xi_1, \ldots, \xi_M\}$ и вероятностным распределением $P_\xi = \{p_1, \ldots, p_M\}$. В дальнейшем будем считать, что $0 < \xi_i < \infty$ для каждого $i \in \{1, \ldots, M\}$. Тогда, математическое ожидание (среднее значение) величины ξ определяется как $\mathbb{E}[\xi] = \sum_{i=1}^{M} \xi_i p_i$. Во многих практических приложениях знание $\mathbb{E}[\xi]$ оказывается весьма важным. Однако, часто невозможно точно вычислить значение $\mathbb{E}[\xi]$ за разумное время. В таких случаях можно вместо этого оценить $\mathbb{E}[\xi]$ с некоторой заранее заданной точностью ε. Соответствующие алгоритмы используют случайные выборки и традиционно относятся к методу Монте-Карло [92].

Более точно, пусть $\xi^1, \ldots, \xi^N$ — независимые наблюдения случайной величины ξ. Тогда неравенство Чебышёва [62] утверждает (см., например, [93]):

$$\Pr\left\{\left|\mathbb{E}[\xi] - \frac{1}{N}\sum_{j=1}^{N}\xi^j\right| \leqslant \varepsilon \cdot \mathbb{E}[\xi]\right\} \geqslant 1 - \delta, \tag{9}$$

где $\delta = \frac{\mathrm{Var}[\xi]}{\varepsilon^2 \cdot N \cdot \mathbb{E}^2[\xi]}$, а $\mathrm{Var}[\xi]$ обозначает дисперсию случайной величины ξ. Из (9) следует, что для конечных $\mathbb{E}[\xi]$ и $\mathrm{Var}[\xi]$ математическое ожидание $\mathbb{E}[\xi]$ может быть аппроксимировано (в смысле (9)) значением $\frac{1}{N}\sum_{j=1}^{N}\xi^j$ с любыми заранее заданными $\varepsilon, \delta \in (0{,}1)$ путём увеличения числа наблюдений N.

1.14.2. Декомпозиционная трудность

Концепция *лазеек* (*backdoors*) была введена в классической работе [94]. В частности, множество переменных B в произвольной КНФ-формуле C является

сильной лазейкой (Strong Backdoor Set — SBS) для C относительно некоторого полиномиального алгоритма P (называемого вспомогательным решателем (*subsolver*)), если формула $C[\beta/B]$ решается с помощью P (получается ответ SAT/UNSAT за полиномиальное время) для любого $\beta \in \{0,1\}^{|B|}$. Здесь через $C[\beta/B]$ обозначается формула, полученная подстановкой значений β в переменные из B в C. Можно заметить [95], что если B — некоторый SBS, то сложность C ограничена сверху значением $poly(|C|) \cdot 2^{|B|}$, где $poly(\cdot)$ — некоторый полином.

В статье [93] было предложено использовать полный детерминированный SAT-решатель A в качестве вспомогательного решателя, вместо традиционного полиномиального алгоритма P. Для оценки производительности решателя, введём следующие обозначения. Пусть $t_A(C)$ обозначает время работы A на КНФ-формуле C. Сложность формулы C относительно множества B и солвера A может быть определена следующим образом:

$$\mu_{A,B}(C) = \sum_{\beta \in \{0,1\}^{|B|}} t_A(C[\beta/B]) \tag{10}$$

Минимальное значение (10) по всем возможным множествам $B \in 2^X$ называется *декомпозиционной трудностью* (*decomposition hardness*) формулы C относительно алгоритма A.

Как показано в [93], значение (10) можно выразить с использованием математического ожидания случайной величины ξ_B, связанной с множеством B, которая задается следующим соотношением:

$$\mu_{A,B}(C) = 2^{|B|} \cdot \mathbb{E}[\xi_B] \tag{11}$$

Для оценки значения (10) можно использовать метод Монте-Карло и формулу (9). Это сводит задачу оценки сложности декомпозиции к задаче псевдо-булевой *black-box* оптимизации, которая включает перебор различных множеств B и оценку сложности C относительно каждого B в попытке минимизировать это значение в пространстве 2^X. В [93] для этой цели использовались метаэвристические алгоритмы.

1.14.3. Вероятностный подход к оцениванию трудности булевых формул

Предлагаемые в главе 3 конструкции для декомпозиции формул, кодирующих трудные примеры LEC, основаны на концепции *декомпозиционной трудности*, предложенной в [93]. Данная концепция в свою очередь базируется на понятии *лазейки*, введенном в [94].

Пусть рассматривается произвольная формула C в КНФ над множеством переменных X. Для произвольного $B \subseteq X$ через $\{0,1\}^{|B|}$ обозначается множество всех возможных наборов значений переменных из B. Пусть P — некоторый полиномиальный алгоритм, который получает на вход произвольную КНФ, а на выход выдает ответ из следующего множества $\{\text{SAT}, \text{UNSAT}, \text{INDET}\}$, ответ INDET соответствует ситуации, когда P не может за отведенное время решить, выполнима ли рассматриваемая КНФ. Если применение алгоритма P к формуле C выдает ответ из множества $\{\text{SAT}, \text{UNSAT}\}$, то будем обозначать данный факт через $C \in \Sigma(P)$. Если же результатом применения P к C является ответ INDET, то обозначим данную ситуацию через $C \notin \Sigma(P)$.

Через $C[\beta/B]$ обозначим формулу, полученную из C в результате подстановки набора β значений переменных из B. Тогда множество B называется сильной лазейкой (Strong Backdoor Set, SBS), если для любого $\beta \in \{0,1\}^{|B|}$ имеет место $C[\beta/B] \in \Sigma(P)$.

В статье [95] было отмечено, что любое SBS B, существенно меньшее X, дает нетривиальную верхнюю оценку трудности формулы C, поскольку существует алгоритм со сложностью $poly(|C|) \cdot 2^{|B|}$, определяющий выполнимость C, для некоторого полинома $poly(\cdot)$.

На основании этой идеи в статье [93] был предложен подход к оцениванию трудности произвольных булевых формул, используя их декомпозиционные представления, называемые также разбиениями (*partitioning* [77]). Более того, оказалось, что оценивать декомпозиционную трудность можно при помощи вероятностных алгоритмов, традиционно относимых к методу Монте-Карло [92]. Кратко изложим здесь основную суть данного подхода.

Прежде всего напомним понятие разбиения. Согласно [77] разбиение (*partitioning*) произвольной КНФ над множеством переменных X — это множество формул $\Pi = \{G_1, \ldots, G_s\}$ над X, такое что выполнены два следующих требования:

1. Для любых $i \neq j \in \{1, \ldots, s\}$, формула $C \wedge G_i \wedge G_j$ невыполнима.
2. Формула C выполнима тогда и только тогда, когда выполнима формула $C \wedge (G_1 \vee \ldots \vee G_s)$.

Если Π — некоторое разбиение, то на задачу о выполнимости C, можно смотреть как на семейство аналогичных задач для формул вида $C \wedge G_i$, где $i \in \{1, \ldots, s\}$. Для решения последних можно использовать параллельные вычисления. SAT задачи для формул вида $C \wedge G_i$ могут быть существенно проще SAT для C: если,

например, множество Π — это множество из 2^k различных кубов над переменными $B = \{\widetilde{x}_1, \ldots, \widetilde{x}_k\}$.

Если Π — некоторое разбиение C, то по аналогии с понятием лазейки можно определить трудность формулы C относительно Π и некоторого алгоритма A решения SAT. Если рассмотреть в роли A некоторый полный SAT-решатель, то иногда удается найти относительно небольшие по размеру разбиения, которые вполне можно использовать для решения трудных индустриальных примеров SAT (в том числе верификационной природы). Таким образом, имеет смысл определить трудность C относительно разбиения Π как следующую величину:

$$\mu_{A,\Pi}(C) = \sum_{G \in \Pi} t_A(G \wedge C),$$

где через $t_A(C)$ обозначено время работы полного SAT-решателя A на формуле C.

Возникает вопрос: как для конкретного разбиения Π вычислить, или хотя бы оценить величину $\mu_{A,\Pi}(C)$ в ситуации, когда s велико? Именно для этой цели может быть использован метод Монте-Карло.

Зададим на Π равномерное распределение, приписав каждому $G_i \in \Pi$ вероятность $1/s$, где $i \in \{1, \ldots, s\}$, и получим таким способом некоторое пространство элементарных исходов. С каждым $G_i \in \Pi$ свяжем значение случайной величины $\xi_\Pi \colon \Pi \longrightarrow R^+$, которое на произвольном $G \in \Pi$ равно $t_A(G \wedge C)$. Пусть $\mathrm{Spec}(\xi_\Pi) = \{\xi_1, \ldots, \xi_r\}$ — спектр величины ξ_Π, а $P(\xi_\Pi) = \{p_1, \ldots, p_r\}$ — закон распределения данной величины. Как показано в [93], имеет место следующий факт:

$$\mu_{A,\Pi}(C) = s \cdot \mathbb{E}[\xi_\Pi],$$

где $\mathbb{E}[\xi_\Pi]$ — математическое ожидание величины ξ_Π. В соответствии с методом Монте-Карло можно оценить величину $\mu_{A,\Pi}(C)$ через значение выборочного среднего $\overline{\xi}_\Pi = \frac{1}{N} \cdot \sum_{j=1}^{N} \xi^j$, где ξ^j — это независимые наблюдения величины ξ_Π. Использование неравенства Чебышёва [62] даёт следующее соотношение:

$$\Pr\left\{(1-\varepsilon)\,\mathbb{E}[\xi_\Pi] \leqslant \overline{\xi}_\Pi \leqslant (1+\varepsilon)\,\mathbb{E}[\xi_\Pi]\right\} \geqslant 1-\delta, \tag{12}$$

справедливое для любых фиксированных $\varepsilon, \delta \in (0,1)$ и натурального числа N (число наблюдений), связанных следующим образом:

$$\delta = \frac{\mathrm{Var}[\xi_\Pi]}{\varepsilon^2 N\, \mathbb{E}^2[\xi_\Pi]} \tag{13}$$

Таким образом, с увеличением N точность оценивания $\mu_{A,\Pi}(C)$ величиной $\overline{\xi}_{\Pi}$ будет возрастать. Следует особо отметить, что не существует полных гарантий точности таких оценок: при большой дисперсии и малом N получаемые оценки могут быть сколь угодно неточны. Тем не менее, можно использовать стандартные статистические аргументы точности получаемых оценок. Один метод такого рода описан в [93] и основан на периодическом увеличении объема выборки N до тех пор, пока не выполнится неравенство

$$N \geqslant \frac{s^2(\xi_{\Pi})}{\delta\varepsilon^2\overline{\xi}_{\Pi}^2}, \tag{14}$$

в котором $s^2(\xi_{\Pi})$ — выборочная дисперсия. Неравенство (14) является статистическим аналогом неравенства

$$N \geqslant \frac{\mathrm{Var}[\xi_{\Pi}]}{\varepsilon^2\delta\,\mathbb{E}^2[\xi_{\Pi}]} \tag{15}$$

Заметим, что условие (12) при фиксированных $\varepsilon,\delta \in (0,1)$ имеет место для любого N, для которого выполнено (15).

Также возможно построение доверительных интервалов для $\mu_{A,\Pi}(C)$ с использованием центральной предельной теоремы. Здесь $\sigma = \sqrt{\mathrm{Var}[\xi]}$ обозначает стандартное отклонение, γ — уровень доверия, $\gamma = \Phi(\delta_{\gamma})$, где $\Phi(\cdot)$ — нормальная кумулятивная функция распределения. Это означает, что при рассмотренных предположениях значение выборочного среднего $\overline{\xi}_{\Pi} = \frac{1}{N}\cdot\sum_{j=1}^{N}\xi^j$ является хорошим приближением $\mathbb{E}[\xi]$, когда количество наблюдений N достаточно велико. Для любого заданного N качество этого приближения зависит от значения $\mathrm{Var}[\xi]$. На практике для оценки $\mathrm{Var}[\xi]$ используется несмещённая выборочная дисперсия $s^2 = \frac{1}{N-1}\sum_{j=1}^{N}(\xi^j - \overline{\xi})^2$. В этом случае вместо (13) применяется следующая формула [96]:

$$\Pr\left\{\left|\mathbb{E}[\xi] - \overline{\xi}_{\Pi}\right| < \frac{s\cdot t_{\gamma,N-1}}{\sqrt{N}}\right\} \geqslant \gamma,$$

где $t_{\gamma,N-1}$ — это квантиль распределения Стьюдента с $N-1$ степенями свободы, соответствующий уровню доверия γ. Если, например, $\gamma = 0.999$ и $N \geqslant 10000$, то $t_{\gamma,N-1} \approx 3.29$.

Глава 2. Методы синтеза и верификации моделей автоматных программ на основе сведения к задаче булевой выполнимости (SAT)

Данная глава посвящена синтезу и верификации моделей автоматных программ — конечных автоматов и булевых схем — на основе сведения к задаче булевой выполнимости (SAT). В разделе 2.2 рассматривается задача синтеза булевой формулы по таблице истинности, приводится описание методов синтеза минимальных булевых формул, основанных на сведении к SAT и использовании *инкрементальных* SAT-решателей, их реализации и результатов экспериментального сравнения. В разделе 2.3 предлагается метод синтеза конечно-автоматных моделей по примерам поведения, основанный на сведении к SAT, приводится описание разработанных алгоритмов: Basic — для синтеза базовых моделей, Extended — для синтеза расширенных моделей, Complete — для учета негативных сценариев выполнения. В разделе 2.4 рассматривается задача синтеза минимальных конечно-автоматных моделей, приводится описание разработанных алгоритмов Basic-min, Extended-min, Complete-min и Extended-min-UB. В разделе 2.5 рассматривается подход индуктивного синтеза, основанного на контрпримерах (*Counterexample-Guided Inductive Synthesis* — CEGIS) [97; 98], используемый для учета при синтезе формальной спецификации, приводится описание алгоритмов CEGIS и CEGIS-min. Раздел 2.7 содержит экспериментальное сравнение разработанных методов с существующими на примере задачи синтеза конечно-автоматной модели логического контроллера, управляющего Pick-and-Place манипулятором. Раздел 2.8 посвящен применению разработанных методов для минимизации *систем переходов*, полученных с помощью программного средства для LTL-синтеза BoSy [26; 27] по исходным данным с соревнования по реактивному синтезу SYNTCOMP [99]. Следующие разделы посвящены решению задачи синтеза модульных конечно-автоматных моделей с различными видами композиции модулей: (1) параллельная (раздел 2.9), (2) последовательная (раздел 2.10) и (3) произвольная (раздел 2.11). Раздел 2.14 содержит экспериментальное исследование, нацеленное на определение эффективности и применимости разработанных методов модульного синтеза. В качестве модельной системы используется система Pick-and-Place манипулятора, уже рассмотренная ранее в разделе 2.7. Несмотря на то, что оригинальная система фактически является монолитной, разработанные методы позволяют синтезировать распределенную модульную систему, обладающую схожим поведением. Все разработанные в данной работе методы реализованы в виде программного средства fbSAT [100].

2.1. Программная библиотека `kotlin-satlib` для взаимодействия с SAT-решателями

Для записи и передачи описанного выше сведения в SAT-решатель в ходе выполнения данной работы была разработана специализированная программная библиотека `kotlin-satlib`[1]. Разработанная библиотека выполнена в виде библиотеки на языке Kotlin и состоит из нескольких модулей: модуль взаимодействия с SAT-решателями через их нативные интерфейсы с помощью технологии JNI (*Java Native Interface*), модуль для упрощенной записи распространенных видов ограничений с использованием преобразований Цейтина [11], модуль для манипуляции переменными с ограниченными доменами (например, целочисленными), а также модуль для манипуляции многомерными массивами SAT-переменных. В последующих разделах приведено описание этих модулей.

2.1.1. Модуль взаимодействия с SAT-решателями на основе технологии JNI

Практически все современные SAT-решатели предоставляют нативный программный интерфейс для взаимодействия с ними, однако так как подавляющее большинство решателей написаны на языках C/C++ (ввиду строгих требований к эффективности реализации), их прямое использование в языках более высокого уровня весьма затруднено. В данной работе в качестве целевой платформы используется JVM (*Java Virtual Machine*), а именно, языки программирования Java и Kotlin. Подавляющее большинство взаимодействий с нативным программным обеспечением из JVM так или иначе выполняется с помощью технологии JNI[2] (*Java Native Interface*). Данная технология позволяет описывать нативные методы (в Java — с помощью ключевого слова `native`, в Kotlin — с помощью ключевого слова `external`) в Java/Kotlin классах, реализация которых выполняется на нативном языке (обычно на C/C++), где уже можно получить доступ к другому нативному коду, например, к SAT-решаелям. Полученные нативные реализации компилируются в динамические библиотеки (на GNU/Linux — *shared object library*, на Windows — *dynamic-link library*) и становятся доступными для виртуальной машины (JVM) во времени исполнения программы.

Сразу стоит отметить существование библиотеки jnisat на языке Java — программной обертки для решателей PicoSAT [101] и MiniSAT [75], использующей описанную технологию JNI. Именно эта библиотека легла в основу фреймворка

[1] `https://github.com/Lipen/kotlin-satlib`

[2] https://docs.oracle.com/javase/8/docs/technotes/guides/jni

`kotlin-satlib`: собственная реализация была выполнена на языке Kotlin с поддержкой вызова кода из Java, был добавлен общий унифицированный интерфейс для SAT-решателей, а также была добавлена поддержка таких современных SAT-решателей, как MiniSat [75], Glucose [76], CryptoMiniSat [89], CaDiCaL [88] и Kissat [102].

2.1.2. Модуль записи ограничений с использованием преобразований Цейтина

Практически все ограничения, определенные в разделе про кодирование задачи синтеза булевой формулы, не были представлены в конъюнктивной нормальной форме (КНФ), что затрудняет их прямую передачу в SAT-решатель — предварительно необходимо конвертировать все ограничения в набор дизъюнктов КНФ. Однако стоит учитывать, что некоторые выражения, например, вида $(x_1 \wedge y_1) \vee \ldots \vee (x_N \vee y_N)$, при конвертации их в эквивалентные КНФ имеют экспоненциальный размер (по числу дизъюнктов) относительно исходного. Для решения этой проблемы можно воспользоваться неэквивалетными преобразованиями, которые сохраняют выполнимость формулы, так как при решении задачи SAT нас интересует только получаемая модель или же доказательство отсутствия решения. Одними из наиболее известных таких преобразований являются преобразования Цейтина [11], основная идея которых — введение дополнительных переменных, кодирующих и заменяющих собой логические вентили в формуле, постепенно сводя ее к КНФ, размер которой растет полиномиально. Например, для выражения $A \wedge B$ преобразование Цейтина вводит новую переменную $C \leftrightarrow A \wedge B$, кодируемую в КНФ следующим образом:

$$(\neg A \vee \neg B \vee C) \wedge (A \vee \neg C) \wedge (B \vee \neg C)$$

Добавление новых переменных, безусловно, увеличивает теоретическую сложность задачи, однако на практике добавление дополнительных структурных ограничений может даже помочь ускорить процесс решения. К тому же, современные SAT-решатели способны эффективно манипулировать миллионами переменных, поэтому применение преобразований Цейтина в случаях, когда наивная конвертация ограничений в КНФ приводила бы к экспоненциальному росту размера задачи, является целесообразным.

Разработанный фреймворк `kotlin-satlib` содержит модуль `Ops`, производящий такие преобразования Цейтина автоматически. Модуль `Ops` также содержит множество типичных видов ограничений, например, функция `implyIffOr(x1: Lit,`

`x2: Lit, xs: Iterable<Lit>`) позволяет задать ограничение вида

$$x_1 \to \left(x_2 \leftrightarrow \bigvee_{x_i \in \mathtt{xs}} x_i\right)$$

Дополнительные перегрузки этих функций с альтернативными контейнерами используемых литералов, например, `implyIffOr(x1: Lit, x2: Lit, vararg xs: Lit)`, образуют небольшой предметно-ориентированный язык (*Domain Specific Language* — DSL), позволяющий пользователю сконцентрироваться на *моделировании* задачи и использовать практически произвольные виды ограничений, а не на механических действиях, связанных с кодированием дополнительных переменных и конвертацией ограничений в КНФ, необходимую для SAT-решателя.

2.1.3. Модуль манипуляции переменными с ограниченным доменом

Большинство переменных, определенных в разделе про кодирование задачи синтеза булевой формулы, не были логическими, а имели либо целочисленное значение из некоторого известного диапазона, либо некоторое значение из заданного множества допустимых, например, $\tau_p \in \{\wedge, \vee, \neg, \bot\}$. Стоит сразу отметить, что SAT-решатели поддерживают только логические переменные, однако на практике моделирование исходной задачи с использованием переменных с ограниченными доменами, а не только логических, зачастую оказывается гораздо более выразительным и позволяет смотреть на задачу с интуитивной стороны. Основной подход к непосредственному кодированию переменных с ограниченным доменом — так называемое onehot-кодирование, при котором каждому возможному значению переменной из домена сопоставляется логическая переменная. Например, истинность логическая переменная $\tau_{p,\wedge}$ соответствует равенству $\tau_p = \wedge$. При этом необходимо также добавить ограничение на то, что ровно один из *onehot*-литералов может иметь истинное значение — это может быть сделано в виде комбинации ограничений AtLeastOne (не менее одного) и AtMostOne (не более одного). Ограничение AtLeastOne является тривиальным и представляет из себя один дизъюнкт, содержащий все объявляенные литералы. А вот ограничение AtMostOne может быть закодировано множеством способов — от простых и интуитивных [103] до комплексных и эффективных [104], чему посвящены многочисленные исследования.

Стоит отметить, что *onehot*-кодирование — далеко не единственный способ представления переменных в виде набора литералов, хотя и самый интуитивный и удобный в большинстве ситуаций. Среди альтернативных способов кодирования

можно выделить следующие [105]: порядковое (*order encoding*) [106], бинарное (*binary encoding*), комбинированное (*onehot+binary encoding*) [105]. Все такие способы обеспечивают некоторое представление исходной переменной с ограниченным доменом в SAT-решателе в виде набора литералов, однако различаются возможностями их использования в различных контекстах: *onehot*-кодирование используется, когда необходимо получить доступ к значению переменной (здесь под «доступом» подразумевается использование переменной в декларативном процессе построения сведения); order-кодирование используется, если необходимо закодировать порядок между, например, целочисленными переменными; *binary*-кодирование позволяет эффективно кодировать арифметические операции в сведении; гибридное *onehot+binary*-кодирование обеспечивает комбинацию возможностей этих двух способов кодирования.

Описанные выше детали кодирования переменных с ограниченными доменами должны быть учтены при построении сведения, поэтому возникает желание автоматизировать эти действия. Разработанный фреймворк `kotlin-satlib` содержит отдельный модуль `Vars`, предназначенный для манипуляции такими переменными. Основой модуля является класс `DomainVar<T>`, хранящий набор литералов, соответствующих onehot-кодировке переменной со значениями произвольного типа `T`. При создании экземпляра такой переменной имеется возможность указать один из способов кодирования: *onehot* или *onehot+binary*. Класс `DomainVar<T>` также содержит инфиксные методы `eq(value: T)` и `neq(value: T)`, позволяющие обращаться к литералу, соответствующему значению `value`. Например, пусть `v: DomainVar<Int>`, тогда простое и удобное в использовании выражение ‘`v eq 5`’ соответствует литералу, кодирующему $v = 5$.

2.1.4. Модуль манипуляции массивами SAT переменных

Некоторые переменные сведения могут быть объявлены с множественными индексами, что подразумевает их хранение в многомерном массиве. Ввиду того, что стандартные многомерные массивы, доступные в Java (например, `int[][][]`) и Kotlin (например, `Array<Array<Array<Int»>`) не обеспечивают необходимой гибкости при их инициализации и использовании, была разработана собственная эффективная реализация многомерных массивов, оформленная в виде отдельной библиотеки MultiArray[3] на языке Kotlin.

[3] `https://github.com/Lipen/MultiArray`

Библиотека MultiArray включает в себя набор интерфейсов и их реализаций для многомерных массивов, индексируемых с нуля или единицы (по выбору), хранящих либо значения произвольного типа, либо (в отдельных эффективных специализациях) целочисленные и булевы значения. Основные интерфейсы, предоставляемые библиотекой — изменяемые (`MutableMultiArray<T>`) и неизменяемые (`MultiArray<out T>`) многомерные массивы. Неизменяемая версия ковариантна по типу `T`, что позволяет обращаться с такими массивами более гибко. Внутри многомерных массивов данные хранятся в одном линейном массиве, а индексация происходит с помощью так называемых *strides of array* («шаги массива»). Такой подход позволяет получить максимальную эффективность в совокупности с удобством по сравнению со стандартными массивами, встроенными в языки Java и Kotlin.

2.2. Синтез булевых формул с помощью инкрементальных SAT-решателей

Задача синтеза булевой формулы заключается в построении логической формулы, зависящей от N переменных $x_1 \dots x_N$ (возможно, не от всех — некоторые переменные могут не использоваться в полученной формуле), по заданной таблице истинности, с использованием заданных логических операций. Заданная таблица истинности может быть как полной (размера 2^N), так и частичной — для некоторых наборов переменных (в дальнейшем также называемых «входами») значение логической функции может быть не определено. Соответствующая логическая функция имеет ровно один логических выход, значения для которого на различных входах и записаны в таблице истинности. Стоит отметить, что список допустимых логических операций может варьироваться, также как и возможность применения операций к подвыражениям, в зависимости от желаемого результата (например, формулы в так называемой нормальной форме отрицания (*negation normal form*) могут содержать логическое отрицание, применяемое только к переменным, но не к комплексным выражениям). В данной работе рассматривается задача синтеза булевой формулы, содержащей следующие логические операции, без дополнительных ограничений на их применимость к подвыражениям: $\wedge$ (логическое И), $\vee$ (логическое ИЛИ) и $\neg$ (логическое отрицание).

В данной работе рассматривается задача синтеза минимальной булевой формулы — формулы минимального размера, удовлетворяющей заданной таблице истинности. Несмотря на простоту формулировки, задача синтеза минимальной булевой формулы по полной или частичной таблице истинности является NP-трудной [107].

На практике данная задача обычно решается с помощью эвристических подходов, не гарантирующих минимального ответа. Наиболее часто используемым подходом является использование метода Espresso [108], реализация которого доступна в виде одноименного программного средства. Несмотря на то, что этот эвристический подход был разработан относительно давно, его успех до сих пор не был существенно преодолен — многие современные подходы в той или иной степени являются модификациями Espresso, например, BOOM-II [109]. Метод Espresso позволяет синтезировать *минимальные* булевы формулы по заданным полным или частичным таблицам истинности, включая возможность синтезировать функции с множественными выходами. В процессе минимизации могут использоваться различные оптимизационные критерии, например, суммарное число логических вентилей или число использованных литералов. Отличительной особенностью данного метода является его высокая эффективность. Однако стоит отметить, что получаемое с помощью Espresso решение не является *точным* — наименьшим — возможно существование меньшего решения, даже при использовании большого числа итераций. В тех случаях, когда требуется *точное* решение — наименьшая из возможных булевых формул, необходимо использование других подходов, например, программирование в ограничениях, а именно, сведение к задаче выполнимости.

Для логической формулы может быть построено дерево разбора — укорененное (*rooted*) дерево, во внутренних узлах которого находятся логические операции, а вершины-листья отмечены переменными $x_1 \dots x_N$. Связи между вершинами соответствуют применению соответствующих операций к вершинам-потомкам. Каждое поддерево такого дерева разбора может рассматриваться как некоторое подвыражение исходной формулы. Размер дерева разбора — число вершин, из которых оно состоит (включая вершины-листья). Размер логической формулы — размер соответствующего дерева разбора. В дальнейшем размер дерева разбора или булевой формулы будет обозначаться как P.

Сведение задачи к SAT обычно выглядит как декларативное описание с помощью логических переменных и ограничений структуры желаемого решения и его взаимодействия с исходными данными. Вкратце, в случае рассматриваемой задачи синтеза булевой формулы от N переменных, необходимо закодировать структуру дерева разбора синтезируемой формулы заданного размера P, а также логические значения каждой вершины дерева разбора (каждая вершина соответствует некоторому подвыражению; корень дерева соответствует всей формуле) на различных входах. После этого необходимо добавить ограничение на соответствие значений

синтезируемой функции значениям в заданной таблице истинности. Полученную в результате такого сведения SAT-формулу (в КНФ) необходимо решить с помощью SAT-решателя для получения либо искомой булевой формулы заданного размера P, либо доказательства ее несуществования для заданного P.

Для нахождения минимальной булевой формулы необходимо каким-либо образом определить минимальное значение P, при котором решение существует. Для этого в данной работе используется перебор параметра P снизу вверх, начиная с единицы — таким образом, первое найденное решение будет минимальным из возможных. При этом в данной работе используется два подхода: (1) итеративный подход, при котором SAT-решатель перезапускается на каждом шаге для каждого нового значения P; и (2) инкрементальный подход, при котором на очередной итерации перебора параметра P сведение расширяется только теми ограничениями, которые зависят от нового значения P, а вызовы SAT-решателя производятся с использованием предположений (*assumptions*), что позволяет *не перезапускать* SAT-решатель даже после получения UNSAT — сообщения об отсутствии решения при заданных ограничениях.

Рассмотрим подробнее составляющие сведения к SAT. Параметр P отвечает за размер синтезируемой формулы — число вершин дерева разбора. Вершины дерева разбора нумеруются последовательно, начиная с корневой, имеющей индекс 1. В общем случае порядок индексации вершин не имеет значения, однако на практике использование нумерации вершин дерева в порядке BFS-обхода (*Breadth-First Search* — поиск в ширину) позволяет существенно сократить размер сведения, а также избавиться от большого числа изоморфных решений, что положительно влияет на эффективность метода — это так называемое «нарушение симметрий» [110], широко используемое при решении задач с помощью методов программирования в ограничениях. Для обеспечения BFS-нумерации необходимо, во-первых, чтобы для любой пары смежных вершин дерева (родитель–потомок) номер родительской вершины был меньше номера потомка; а во-вторых, чтобы вершины-потомки нумеровались по порядку, без пропусков индексов. В рассматриваемой задаче вершины могут иметь не более двух потомков — с номерами c и $(c+1)$, где $c > p$.

Каждая вершина дерева может быть либо одной из допустимых логических операций ($\wedge, \vee, \neg$), либо терминалом, что кодируется с помощью переменной $\tau_p \in \{\wedge, \vee, \neg, \bot\}$, где $p \in [1..P]$, а $\bot$ соответствует вершине-терминалу. Переменная $\chi_p \in [0..N]$ кодирует номер переменной (от 1 до N), которой соответствует

вершина p. Только вершины-терминалы имеют ассоциированные переменные:

$$(\tau_p = \bot) \leftrightarrow (\chi_p = 0)$$

Переменная $\pi_p \in [0..(p-1)]$, где $p \in [1..P]$, кодирует номер родительской вершины для вершины p. Как было упомянуто выше, номер родителя при использовании BFS-нумерации должен быть меньше номера самой вершины p, поэтому доменом этой переменной является диапазон от 0 до $(p-1)$. При этом $\pi_p = 0$ означает, что у вершины p в дереве нет родителя — это выполняется только для корневой вершины.

Переменная $\sigma_p \in \{0\} \cup [(p+1)..P]$, где $p \in [1..P]$ кодирует номер левого потомка вершины p. Взаимосвязь между переменными π и σ кодируется следующим образом:

$$(\sigma_p = c) \rightarrow (\pi_c = p)$$

В том случае, если тип вершины p — терминал, то такая вершина не имеет потомков:

$$(\tau_p = \bot) \rightarrow (\sigma_p = 0)$$

В том случае, если вершина p имеет тип бинарной операции ($\wedge$ или $\vee$), то вершина с номером $(\sigma_p + 1)$ неявно считается правым потомком вершины p:

$$\Big((\tau_p \in \{\wedge, \vee\}) \wedge (\sigma_p = c)\Big) \rightarrow (\pi_{c+1} = p)$$

Переменная $\vartheta_{p,u} \in \mathbb{B}$ кодирует логическое значение вершины $p \in [1..P]$ на входе $u \in U$. Значение корневой вершины соответствует значению всей синтезируемой функции и должно совпадать со значением, указанным в заданной таблице истинности. Значения вершин рассчитываются исходя из их типа, что декларативно можно описать следующими ограничениями:

$$(\tau_p = \bot) \wedge (\chi_p = x) \rightarrow \bigwedge\nolimits_{u \in U} \left[\vartheta_{p,u} \leftrightarrow u_x\right]$$
$$(\tau_p = \wedge) \wedge (\sigma_p = c) \rightarrow \bigwedge\nolimits_{u \in U} \left[\vartheta_{p,u} \leftrightarrow (\vartheta_{c,u} \wedge \vartheta_{c+1,u})\right]$$
$$(\tau_p = \vee) \wedge (\sigma_p = c) \rightarrow \bigwedge\nolimits_{u \in U} \left[\vartheta_{p,u} \leftrightarrow (\vartheta_{c,u} \vee \vartheta_{c+1,u})\right]$$
$$(\tau_p = \neg) \wedge (\sigma_p = c) \rightarrow \bigwedge\nolimits_{u \in U} \left[\vartheta_{p,u} \leftrightarrow \neg\vartheta_{c,u}\right]$$

2.2.1. Экспериментальное исследование

Был произведен синтез минимальных формул для всех 256 булевых функций от $X = 3$ переменных с использованием двух подходов поиска минимального значения параметра P — размера дерева разбора синтезируемой формулы: (1) итеративный

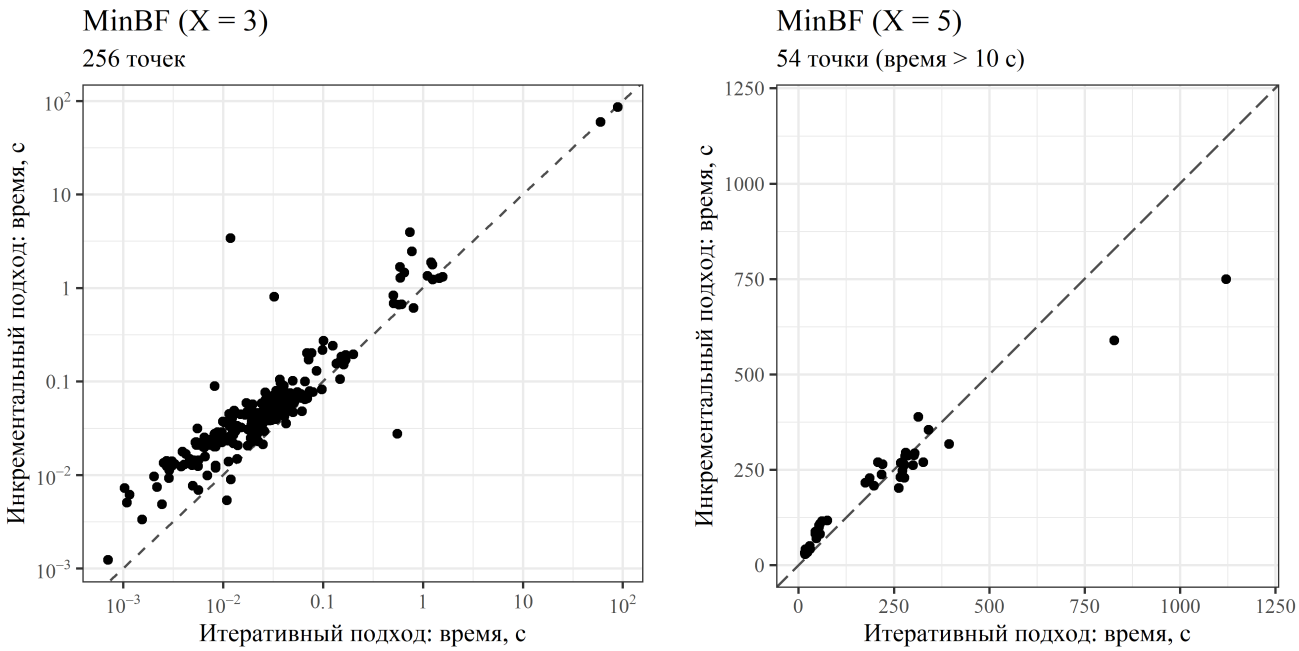


Рисунок 5 — Графики сравнения времени работы алгоритма синтеза минимальной булевой формулы (слева — от трех переменных, справа — от пяти переменных) для двух подходов: итеративный (горизонтальная ось) и инкрементальный (вертикальная ось). Время работы указано в секундах (слева — на логарифмической шкале). Каждая точка на графике соответствует булевой функции. На правом графике показаны только точки со временем работы, превышающим 10 секунд.

перебор с перезапуском SAT-решателя на каждом шаге (суммарное время — 171 с), (2) инкрементальное расширение сведения с использованием предположений на каждом шаге (суммарное время — 185 с). Результаты проведенного экспериментального сравнения представлены на Рисунке 5 (слева), где оси соответствуют времени (в секундах, в логарифмической шкале) поиска минимальной булевой формулы двумя подходами, а каждая точка (всего 256 точек) соответствует отдельной булевой функции от $X = 3$ переменных. Пунктирная красная линия (*baseline*) соответствует равенству времени работы двух подходов. Скопление точек сосредоточено около базовой линии, что свидетельствует о том, что оба подхода позволяют решать поставленную задачу примерно одинаково эффективно.

Можно заметить, что большинство минимальных формул для функций от трех переменных были найдены менее, чем за одну секунду — сравнение на таких масштабах времени в контексте решения NP-трудных задач не является целесообразным. Поэтому были проведен дополнительный набор экспериментов на данных большей размерности и более «сложных» булевых функциях — от $X = 5$ переменных. Результаты приведены на Рисунке 5 (справа), где показаны только

точки со временем работы, превышающим 10 секунд (всего 54 точки). Данный график — а именно, точки в правой части графика под базовой линией — позволяет судить о том, что инкрементальный подход действительно обеспечивает лучшую производительность в рассмотренной задаче синтеза минимальной булевой формулы.

2.3. Метод синтеза конечно-автоматных моделей по примерам поведения, основанный на сведении к SAT

В этом разделе приводится описание разработанного метода синтеза минимальных конечно-автоматных моделей базовых функциональных блоков по примерам поведения. Расширение метода, позволяющее также учитывать формальную спецификацию в виде LTL-формул, рассматривается в разделе 2.5. Сначала рассматривается решение базовой задачи (алгоритм Basic) — синтеза моделей с использованием только сценариев выполнения — приводится описание переменных и ограничений, составляющих сведение к задаче SAT и предлагается итеративный подход к синтезу минимальных моделей. Следом, сведение к SAT расширяется (алгоритм Extended) кодированием структуры деревьев разбора произвольных булевых формул охранных условий, что приводит к возможности учета их суммарного размера при минимизации. В заключение, решается задача синтеза модели, не только удовлетворяющей заданным примерам поведения, но и лишенной нежелательного поведения, выражаемого в виде негативных сценариев выполнения (алгоритм Complete).

2.3.1. Кодирование структуры автомата

Сведение обозначенной задачи синтеза к задаче SAT заключается в построении булевой формулы, которая истина тогда и только тогда, когда существует конечный автомат размера $|Q| = C$, удовлетворяющий заданному набору позитивных сценариев выполнения $\mathcal{S}^{(+)}$. Для этого необходимо рассмотреть процесс проверки соответствия автомата дереву сценариев и закодировать его в SAT[4]. При этом также необходимо закодировать структуру синтезируемого объекта — конечного автомата, а точнее, ECC. Здесь и далее в этом разделе предполагается, что $b \in \mathbb{B} = \{\top, \bot\}$, $q \in Q$, $k \in [1..K]$, $e \in E^I$, $u \in \mathcal{U}$, $v \in V$, если не указано иное.

[4]Здесь и далее фраза «закодировать в SAT» означает построение соответствующей булевой формулы в КНФ, кодирующей структуру и требуемые ограничения задачи синтеза.

Искомый автомат состоит из C состояний, каждое из которых имеет ассоциированное *действие* (выходное событие и алгоритм) и не более K выходящих (*outgoing*) переходов, упорядоченных в порядке их приоритета. Выходное событие состояния $q \in Q$ кодируется с помощью переменной $\varphi_q \in E^O \cup \{\varepsilon\}$. Так как алгоритм является функцией, независимо изменяющей значения выходных переменных, то он кодируется с помощью переменной $\gamma_{q,z,b} \in \mathbb{B}$, где $q \in Q$ — состояние автомата, $z \in \mathcal{Z}$ — выходная переменная, $b \in \mathbb{B}$ — текущее значение выходной переменной. Каждый переход в автомате ассоциирован с *охранным условием* — парой из входного события и булевой формулы, зависящей от входных переменных $\mathcal{X}$ соответствующего базового функционального блока. Переменная $\tau_{q,k} \in Q_0 = Q \cup \{q_0\}$ кодирует конец k-го перехода из состояния $q \in Q$. «Переходы» в фиктивное состояние q_0 называются *нулевыми* (*null-transitions*) и означают отсутствие перехода в автомате. Входное событие на переходе кодируется с помощью переменной $\xi_{q,k} \in E^I \cup \{\varepsilon\}$. Так как каждый переход должен обладать входным событием, то ε-событием отмечены только *нулевые* (несуществующие) переходы: $(\tau_{q,k} = q_0) \leftrightarrow (\xi_{q,k} = \varepsilon)$. Переменная $\delta_{q,k,e,u} \in \mathbb{B}$ кодирует функцию активации охранного условия — выполнение перехода при входном действии $e[u]$. Переменная $\theta_{q,k,u} \in \mathbb{B}$ кодирует таблицу истинности охранного условия — значения соответствующей булевой функции на входах $u \in \mathcal{U}$. Взаимосвязь между этими переменная задается следующим образом:

$$\delta_{q,k,e,u} \leftrightarrow \Big((\xi_{q,k} = e) \wedge \theta_{q,k,u} \Big)$$

В соответствии со стандартом IEC 61499, переходы ECC обладают приоритетом. Переменная $ff_{q,e,u} \in [0..K]$ кодирует индекс перехода, который выполняется *первым* при входном действии $e[u]$ — только этот переход будет учтен в момент исполнения ECC, даже если следующие переходы также выполняются. При этом $ff_{q,e,u} = 0$ означает, что *ни один* переход не выполняется при входном действии $e[u]$. Некоторый k-й переход выполняется *первым* тогда и только тогда, когда (1) он выполняется ($\delta_{q,k,e,u} = \top$) и (2) не выполняются все предыдущие ($k' < k$) переходы. Наивный способ кодирования переменной ff выглядит следующим образом:

$$(ff_{q,e,u} = k) \leftrightarrow \Big(\delta_{q,k,e,u} \wedge \bigwedge_{1 \leqslant k' < k} \neg \delta_{q,k',e,u} \Big)$$

Более эффективный способ кодирования заключается в определении специальной переменной $nf_{q,k,e,u} \in \mathbb{B}$ для кодирования того факта, что все переходы с 1 по k-й не выполняются. При этом можно заметить, что такая переменная может быть

определена рекурсивно:

$$nf_{q,k,e,u} \leftrightarrow \left(\neg\delta_{q,k,e,u} \wedge nf_{q,k-1,e,u}\right),$$

где следует считать, что $nf_{q,0,e,u} = \bot$. Исходя из этого, эффективный способ кодирования переменной ff выглядит следующим образом:

$$(ff_{q,e,u} = k) \leftrightarrow \left(\delta_{q,k,e,u} \wedge nf_{q,k-1,e,u}\right)$$

Рассмотрим сценарий выполнения $s \in \mathcal{S}^{(+)}$ и автомат $\mathcal{A}$, изначально находящийся в стартовом состоянии q_{init}. Автомат последовательно обрабатывает входные действия из сценария и, возможно (если выполняется какой-либо переход — соответствующее охранное условие становится истинным), изменяет состояние, продуцируя выходные действия. Когда автомат находится в состоянии q и обрабатывает входное действие $e^I[\bar{x}]$, он либо (1) переходит в состояние q', либо (2) «игнорирует» входное действие, оставаясь в том же состоянии. Такое поведение описывается переменной $\lambda_{q,e,u} \in Q_0$, где $\lambda_{q,e,u} = q_0$ соответствует второму (2) случаю. Заметим, что в первом случае автомат может перейти по переходу-петле и остаться в исходном состоянии $q' = q$, что, однако, отличается от случая $q' = q_0$, при котором не происходит генерации выходного действия, ассоциированного с состоянием q.

2.3.2. BFS-предикаты нарушения симметрии для состояний автомата

Дополнительно, стоит добавить ограничения нарушения симметрии (*symmetry breaking predicates*) [110], форсирующие нумерацию состояний автомата в порядке BFS-обхода (*breadth-first search*) — в том порядке, в котором они были бы посещены при выполнении поиска в ширину из стартового состояния. Такие ограничения позволяют существенно сократить пространство поиска, что позитивно влияет на время решения задачи SAT. Стоит отметить, что данные ограничения не влияют на корректность решения — если решение существует, то оно будет найдено независимо от нумерации состояний автомата.

Суть BFS-предиката нарушения симметрии заключается в следующем наблюдении относительно дерева BFS-обхода: родителем каждой вершины может быть только вершина с меньшим номером, а потомки каждой вершины следуют в строгом возрастающем порядке — номера соседних (*sibling*) вершин отличаются на 1. Из этого следует, что для каждой вершины $i > 1$ верно следующее: родитель соседней $(i + 1)$ вершины либо совпадает с родителем вершины i, либо имеет больший номер. Для кодирования такого ограничения в SAT, необходимо объявить

следующие переменные. Переменная $\tau^{\text{BFS-}\mathcal{A}}_{q_i,q_j} \in \mathbb{B}$ $(q_i,q_j \in Q)$ кодирует наличие любого перехода из q_i в q_j в автомате:

$$\tau^{\text{BFS-}\mathcal{A}}_{q_i,q_j} \leftrightarrow \bigvee_{k\in[1..K]} (\tau_{q_i,k} = q_j)$$

Переменная $\pi^{\text{BFS-}\mathcal{A}}_{q_j} \in \{q_1,\ldots,q_{j-1}\}$ $(q_j \in Q)$ кодирует родителя вершины q_j в дереве BFS-обхода:

$$(\pi^{\text{BFS-}\mathcal{A}}_{q_j} = q_i) \leftrightarrow \left(\tau^{\text{BFS-}\mathcal{A}}_{q_i,q_j} \wedge \bigwedge_{r<i} \neg\tau^{\text{BFS-}\mathcal{A}}_{q_r,q_j}\right)$$

Непосредственно BFS-ограничение выглядит следующим образом:

$$(\pi^{\text{BFS-}\mathcal{A}}_{q_j} = q_i) \rightarrow (\pi^{\text{BFS-}\mathcal{A}}_{q_{j+1}} \geqslant q_i)$$

Можно заметить, что в BFS-ограничении присутствует отношение $(\pi^{\text{BFS-}\mathcal{A}}_{q_{j+1}} \geqslant q_i)$. Наивный способ кодирования такого ограничения в SAT выглядит следующим образом:

$$(\pi^{\text{BFS-}\mathcal{A}}_{q_j} = q_i) \rightarrow \bigwedge_{r<i} (\pi^{\text{BFS-}\mathcal{A}}_{q_{j+1}} \neq q_r)$$

Однако существуют и другие, теоретически более эффективные способы. Например, можно использовать так называемые «переменные, кодирующие порядок» (*order-encoding*) [106]. Перед тем, как перейти к их описанию, необходимо напомнить, что привычные переменные с ограниченным доменом, например, $x \in \{2,3,5\}$, кодируются следующим образом, называемым в разных источниках как «*onehot*», «*sparse encoding*», «*direct encoding*» [111], «*pairwise encoding*»: для каждого значения из домена создается отдельная булева переменная, кодирующая равенство переменной этому значению, например, $x \bowtie_{onehot} \{x_2,x_3,x_5\}$, где $x_2 \equiv (x = 2)$, $x_3 \equiv (x = 3)$, $x_5 \equiv (x = 5)$. Аналогичным образом определяются и *order-encoded* переменные, однако кодируют они не равенство, а отношение порядка, например, $x \bowtie_{order} \{x'_2,x'_3,x'_4,x'_5\}$, где $x'_i \equiv (x \geqslant i)$. Заметим, что на практике домены переменных являются непрерывными[5], поэтому кодирующих переменных будет столько же, сколько и при *onehot*-кодировании[6]. Детальное описание *order encoding* присутствует в [106] и в данной работе не приводится. Таким образом, BFS-предикат

[5]Под «непрерывным» доменом здесь подразумевается дискретная последовательность без «пропусков», что в случае численных доменов выражается в виде интервала [low..high].

[6]Можно заметить, что в рассмотренном примере переменная $x'_2 \equiv (x \geqslant 2)$ всегда истинна, а значит ее можно не вводить, поэтому корректнее говорить, что *order-encoded* переменных всегда на одну меньше *onehot*.

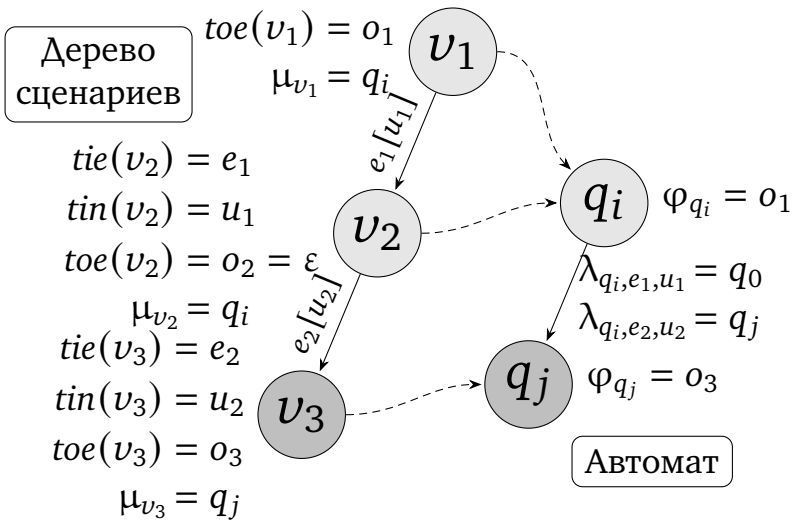


Рисунок 6 — Пример отображения дерева сценариев на автомат

с использованием *order-encoded* переменной $\pi_{q_j}^{\text{BFS-}\mathcal{A}\text{ (order)}} \equiv (\pi_{q_j}^{\text{BFS-}\mathcal{A}} \geqslant q_j)$ может быть сформулирован следующим образом:

$$(\pi_{q_j}^{\text{BFS-}\mathcal{A}} = q_i) \longrightarrow \pi_{q_i}^{\text{BFS-}\mathcal{A}\text{ (order)}}$$

К сожалению, данное усовершенствование не привело к видимым изменениям производительности сведения (результаты экспериментального исследования не приводятся), поэтому здесь и далее в данной работе считается, что используется наивный способ кодирования BFS-предиката нарушения симметрии.

2.3.3. Кодирование отображения позитивного дерева сценариев

Для обеспечения соответствия автомата дереву сценариев, необходимо построить отображение $\mu\colon V \longrightarrow Q$ вершин дерева на состояния автомата. Пример такого отображения приведен на рисунке 6. Переменная $\mu_v \in Q$ кодирует *удовлетворяющее состояние*, в котором автомат оказывается после обработки вершины дерева $v \in V$. Корень дерева ρ отображается в стартовое состояние автомата: $\mu_\rho = q_{\text{init}}$. Пассивные вершины ($toe(v) = \varepsilon$) соответствуют ситуации, когда автомат должен проигнорировать входное действие, не изменяя своего состояния, что может быть выражено с помощью следующих ограничений: $\mu_v = \mu_p$ и $\lambda_{q,e,u} = q_0$, где $v \in V^{(\text{passive})}$, $p = tp(v)$, $q = \mu_p$, $e = tie(v)$, $u = tin(v)$. Активные вершины ($toe(v) \neq \varepsilon$) соответствуют ситуации, когда автомат должен отреагировать на входное действие и продуцировать определенное (непустое) выходное действие,

что может быть закодировано следующим образом:

$$(\mu_v = q') \rightarrow \left((\lambda_{q,e,u} = q') \wedge (\varphi_{q'} = o) \wedge \bigwedge_{z \in Z} (\gamma_{q',z,b} = b')\right),$$

где $v \in V^{(\text{active})}$, $p = tp(v)$, $q = \mu_p$, $q' \in Q$, $e = tie(v)$, $u = tin(v)$, $o = toe(v)$, $z \in Z$, $b = tov(p,z)$, $b' = tov(v,z)$.

2.3.4. Кодирование ограничений на количество переходов

Для того, чтобы ограничить количество переходов в автомате — закодировать ограничение вида «суммарное число *ненулевых* переходов в автомате не больше T», можно воспользоваться техникой *totalizer* (раздел 1.13) и закодировать в SAT *ограничение на кардинальность* $\Phi(\mathcal{D}, 0, T)$, где $\mathcal{D} = \{\tau_{q,k} \neq q_0 | q \in Q, k \in [1..K]\}$ — множество интересующих переменных, T — верхняя граница для суммарного числа ненулевых переходов в автомате. Стоит отметить, что на практике интерес представляет задача минимизации числа переходов в автомате, рассматриваемая в данной работе далее, поэтому нижняя граница принимается равной нулю. Техника *totalizer* позволяет кодировать сразу две границы, что может быть полезно при иных постановках задачи — например, возможно синтезировать автомат с точным (заранее известным) числом переходов T^*, для чего необходимо закодировать обе границы, равные T^* — однако такие задачи в данной работе не рассматриваются.

2.3.5. Алгоритм Basic

Описанные выше переменные и ограничения позволяют синтезировать *вычислимые* конечно-автоматные модели — модели, способные реагировать на входные воздействия, генерируя выходные действия. Обозначим $\textsc{Basic}^*(\mathcal{S}^{(+)}, C, T)$ процедуру нахождения автомата, который удовлетворяет заданному набору позитивных сценариев выполнения $\mathcal{S}^{(+)}$, и в котором C состояний и суммарно не более T переходов. Данная процедура состоит из (1) построения позитивного дерева сценариев $\mathcal{T}^{(+)}$, (2) формирования сведения к SAT (кодирование структуры автомата, отображения дерева сценариев и ограничения на кардинальность) и (3) вызова SAT-решателя. Результатом работы данной процедуры является либо искомый конечный автомат, либо доказательство отсутствия автомата заданного размера. Также введем обозначение для случая, когда число переходов в автомате остается неограниченным:

$$\textsc{Basic}(\mathcal{S}^{(+)}, C) = \textsc{Basic}^*(\mathcal{S}^{(+)}, C, T = \infty)$$

Стоит отметить, что параметр K — максимальное число переходов из каждого состояния — здесь и далее принимается равным $K = C \cdot |E^I|$, так как при меньших значениях возможно отсутствие решения из-за слишком сильных ограничений на искомую модель, а дополнительный перебор подходящего значения K является обременительной задачей.

2.3.6. Кодирование структуры охранных условий

В описанном выше сведении охранные условия представляются в виде таблиц истинности соответствующих булевых формул — посредством переменной θ. Однако такие охранные условия сложны для восприятия человеком, а также неприменимы в средствах разработки систем управления, таких так Matlab или nxtSTUDIO [112], где охранные условия должны быть явно представлены в виде булевых формул. Поэтому сведение расширяется кодированием деревьев разбора произвольных булевых формул, зависящих от входных переменных $\mathcal{X}$.

Каждое дерево разбора охранного условия на k-м переходе ($k \in [1..K]$) из состоянияЁ$q \in Q$ состоит из P вершин, где P является параметром разрабатываемого метода. Каждая вершина типизирована и может быть либо булевым оператором, либо терминальной вершиной, соответствующей входной переменной. Здесь стоит отметить, что параметр P является «глобальным» для всего автомата — *все* охранные условия состоят из P вершин. Так как существуют булевы формулы, для записи которых достаточно менее P вершин в дереве разбора, некоторые вершины могут быть «нетипизированными» (*none-typed*) — не включаться в дерево. Размер дерева разбора определяется как число *типизированных* вершин в нем. Здесь и далее будут использованы следующие обозначения, если не указано иное: $p \in [1..P]$, $x \in \mathcal{X}$, $u \in \mathcal{U}$.

Переменная $\eta_{q,k,p} \in \{\square, \wedge, \vee, \neg, \bullet\}$ кодирует тип вершины дерева разбора p, где «$\square$» обозначает терминальные вершины, «$\wedge$», «$\vee$», «$\neg$» — логические операторы, а «$\bullet$» — нетипизированные вершины. Без потери общности можно задать ограничение на то, что нетипизированные вершины имеют наибольшие номера в дереве:

$$(\eta_{q,k,p} = \bullet) \rightarrow (\eta_{q,k,p+1} = \bullet)$$

Переменная $\chi_{q,k,p} \in \mathcal{X} \cup \{0\}$ кодирует входную переменную, ассоциированную с терминальной вершиной p, или ее отсутствие ($\chi_{q,k,p} = 0$). Только терминальные вершины могут иметь ассоциированные входные переменные:

$$(\eta_{q,k,p} = \square) \leftrightarrow (\chi_{q,k,p} \neq 0)$$

Для задания структуры дерева разбора, а именно, для определения родительских связей между вершинами, используются переменные $\pi_{q,k,p} \in [0..(p-1)]$ и $\sigma_{q,k,p} \in \{0\} \cup [(p+1)..P]$, кодирующие, соответственно, родителя и *левого* ребенка вершины p (либо их отсутствие: $\pi_{q,k,p} = 0$, $\sigma_{q,k,p} = 0$). Взаимосвязь между этими переменными задается следующим образом:

$$(\sigma_{q,k,p} = ch) \rightarrow (\pi_{q,k,ch} = p)$$

Только типизированные вершины, кроме корня ($p = 1$), имеют родительские вершины:

$$(\pi_{q,k,p} \neq 0) \leftrightarrow (\eta_{q,k,p} \neq \bullet)$$

Стоит отметить, что правый ребенок вершины дерева разбора не кодируется явно — для бинарных операторов предполагается, что он имеет номер на единицу больше левого ребенка ($c \in [(p+1)..(P-1)]$):

$$\Big((\eta_{q,k,p} \in \{\wedge, \vee\}) \wedge (\sigma_{q,k,p} = c)\Big) \rightarrow (\pi_{q,k,c+1} = p)$$

Переменная $\vartheta_{q,k,p,u} \in \mathbb{B}$ кодирует значение подформулы — булевого выражения, соответствующего поддереву с корнем p — на входе u. Значение корня дерева разбора соответствует значению всей булевой формулы охранного условия, а значит, можно переиспользовать переменную $\theta_{q,k,u}$, объявленную ранее:

$$\theta_{q,k,u} \equiv \vartheta_{q,k,1,u}$$

Значения терминальных вершин соответствуют значениям ассоциированных входных переменных; значения вершин-операторов могут быть вычислены на основе значений вершин-потомков; значения нетипизированных вершин для определенности принимаются равными `False`, однако это является лишь технической деталью реализации — значения нетипизированных вершин впоследствии не используются:

$$\Big((\eta_{q,k,p} = \square) \wedge (\chi_{q,k,p} = x)\Big) \rightarrow \bigwedge_{u \in \mathcal{U}} \big[\vartheta_{q,k,p,u} \leftrightarrow u_x\big]$$

$$\Big((\eta_{q,k,p} = \wedge) \wedge (\sigma_{q,k,p} = c)\Big) \rightarrow \bigwedge_{u \in \mathcal{U}} \big[\vartheta_{q,k,p,u} \leftrightarrow (\vartheta_{q,k,c,u} \wedge \vartheta_{q,k,c+1,u})\big]$$

$$\Big((\eta_{q,k,p} = \vee) \wedge (\sigma_{q,k,p} = c)\Big) \rightarrow \bigwedge_{u \in \mathcal{U}} \big[\vartheta_{q,k,p,u} \leftrightarrow (\vartheta_{q,k,c,u} \vee \vartheta_{q,k,c+1,u})\big]$$

$$\Big((\eta_{q,k,p} = \neg) \wedge (\sigma_{q,k,p} = c)\Big) \rightarrow \bigwedge_{u \in \mathcal{U}} \big[\vartheta_{q,k,p,u} \leftrightarrow \neg\vartheta_{q,k,c,u}\big]$$

$$(\eta_{q,k,p} = \bullet) \rightarrow \bigwedge_{u \in \mathcal{U}} \neg\vartheta_{q,k,p,u}$$

2.3.7. BFS-предикаты нарушения симметрии для охранных условий

Дополнительно, стоит добавить ограничения нарушения симметрии, форсирующие BFS-нумерацию вершин дерева разбора охранного условия. Фактически, они аналогичны BFS-ограничениям для состояний автомата (раздел 2.3.2), но применяются не ко всему автомату, а к каждому дереву разбора охранного условия по-отдельности: для каждого $q \in Q$, $k \in [1..K]$. Переменная $\tau_{i,j}^{\text{BFS-}\mathcal{G}} \in \mathbb{B}$ $(1 \leqslant i < j \leqslant P)$ задает существование «перехода» из i-ой вершины в j-ую:

$$\tau_{i,j}^{\text{BFS-}\mathcal{G}} \leftrightarrow (\pi_{q,k,j} = i)$$

Переменная $\pi_j^{\text{BFS-}\mathcal{G}} \in [1..(j-1)]$ $(j \in [2..P])$ кодирует родителя j-й вершины в дереве BFS-обхода:

$$(\pi_j^{\text{BFS-}\mathcal{G}} = i) \leftrightarrow \left(\tau_{i,j}^{\text{BFS-}\mathcal{G}} \wedge \bigwedge_{r<i} \neg\tau_{r,j}^{\text{BFS-}\mathcal{G}}\right)$$

Непосредственно BFS-ограничение задается следующим образом:

$$(\pi_j^{\text{BFS-}\mathcal{G}} = i) \rightarrow \bigwedge_{r<i} (\pi_{j+1}^{\text{BFS-}\mathcal{G}} \neq r)$$

2.3.8. Кодирование ограничений на суммарный размер охранных условий

Для того, чтобы ограничить размер охранных условий в автомате, то есть закодировать ограничение вида «суммарное число *типизированных* вершин в деревьях разбора булевых формул, соответствующих охранным условиям на переходах автомата не больше N», можно воспользоваться техникой *totalizer* (см. раздел 1.13) и закодировать в SAT *ограничение на кардинальность* $\Phi(\mathcal{H}, 0, N)$, где $\mathcal{H} = \{\eta_{q,k,p} \neq \bullet \mid q \in Q, k \in [1..K], p \in [1..P]\}$ — множество интересующих переменных, N — верхняя граница для суммарного размера охранных условий в автомате.

2.3.9. Алгоритм Extended

Обозначим Extended$^*(\mathcal{S}^{(+)}, C, P, N)$ процедуру нахождения автомата, который удовлетворяет заданному набору позитивных сценариев выполнения $\mathcal{S}^{(+)}$, и в котором C состояний, максимальный размер охранного условия P, а суммарный размер охранных условий не больше N. Стоит отметить, что параметр T — число ненулевых переходов в автомате — здесь и далее не рассматривается. Данная процедура состоит из (1) построения позитивного дерева сценариев $\mathcal{T}^{(+)}$, (2) формирования

сведения к SAT (кодирование структуры автомата и охранных условий, отображения дерева сценариев и ограничения на кардинальность) и (3) вызова SAT-решателя. Также введем обозначение для случая, когда суммарный размер охранных условий в автомате остается неограниченным:

$$\textsc{Extended}(\mathcal{S}^{(+)}, C, P) = \textsc{Extended}^{*}(\mathcal{S}^{(+)}, C, P, N = \infty)$$

Стоит отметить, что минимизация параметра T — числа *ненулевых* переходов в автомате — в данном случае не производится, потому что если сначала минимизировать T, а затем N, то полученное решение не будет наименьшим, а если наоборот — сначала N, а затем T — то это не повлияет на уже полученное минимальное значение $N_{\min}$.

2.3.10. Кодирование отображения негативного дерева сценариев

Отображение $\widehat{\mu}\colon \widehat{V} \to Q_0$ вершин негативного дерева сценариев $\mathcal{T}^{(-)}$ на состояния автомата очень похоже на отображение позитивного дерева, однако главным отличием является то, что негативное дерево представляет нежелательное поведение системы, включая нежелательное циклическое поведение, которое необходимо запретить.

Переменная $\widehat{\mu}_{\widehat{v}} \in Q_0$ кодирует *удовлетворяющее* состояние (либо его отсутствие: $\widehat{\mu}_{\widehat{v}} = q_0$). Стоит отметить, что автомат может не обладать поведением, заданным в негативном дереве сценариев — его поведение может отличаться от записанного в вершине дерева $\widehat{v} \in \widehat{V}$ — в этом (и только в этом) случае $\widehat{\mu}_{\widehat{v}} = q_0$. Если некоторая вершина $\widehat{v} \in \widehat{V}$ никуда не отображается (отображается в «фиктивное» состояние q_0), то это распространяется далее через потомков:

$$(\widehat{\mu}_{\widehat{tp}(\widehat{v})} = q_0) \to (\widehat{\mu}_{\widehat{v}} = q_0)$$

Корень негативного дерева $\widehat{\rho}$ отображается в стартовое состояние автомата:

$$\widehat{\mu}_{\widehat{\rho}} = q_{\text{init}}$$

Пассивные вершины дерева сценариев описывают поведение, когда автомат игнорирует входное действие и не изменяет своего состояния, значит, если автомат соответствует такому поведению, то пассивная вершина отображается — аналогично позитивному дереву — в то же состояние, что и ее родитель, иначе же вершина никуда не отображается: $(\widehat{\mu}_{\widehat{v}} = \widehat{\mu}_{\widehat{tp}(\widehat{v})}) \vee (\widehat{\mu}_{\widehat{v}} = q_0)$, где $\widehat{v} \in \widehat{V}^{(\text{passive})}$.

В свою очередь активные вершины дерева сценариев описывают поведение, когда автомат должен отреагировать на входное воздействие определенным образом,

значит, если автомат соответствует такому поведению, то отображение активной вершины аналогично позитивному дереву, иначе же вершина никуда не отображается:

$$(\widehat{\mu}_{\widehat{v}} = q') \leftrightarrow \left((\widehat{\lambda}_{q,e,u} = q') \wedge (\varphi_{q'} = o) \wedge \bigwedge_{z \in \mathcal{Z}} (\gamma_{q',z,b} = b')\right),$$

где $\widehat{v} \in \widehat{V}^{(\text{active})}$, $\widehat{p} = \widehat{tp}(\widehat{v})$, $q = \widehat{\mu}_{\widehat{p}}$, $q' \in Q$, $e = \widehat{tie}(\widehat{v})$, $u = \widehat{tin}(\widehat{v})$, $o = \widehat{toe}(\widehat{v})$, $z \in \mathcal{Z}$, $b = \widehat{tov}(\widehat{p},z)$, $b' = \widehat{tov}(\widehat{v},z)$. Стоит отметить, что в этом ограничении используется эквивалентность («↔»), а не импликация («⟶»), что позволяет не рассматривать отдельно определение для случая $q' = q_0$, при котором было бы необходимо учитывать различные варианты несоответствия поведения автомата поведению, записанному в вершине негативного дерева.

В заключение, необходимо запретить в автомате нежелательное циклическое поведение, представляемое с помощью *обратных ребер*. Для этого необходимо, чтобы вершины негативного дерева, являющиеся началом и концом обратного ребра, отображались в различные состояния, либо же не отображались вовсе:

$$\bigwedge_{\widehat{v} \in \widehat{V}} \bigwedge_{\widehat{v}' \in \widehat{tbe}(\widehat{v})} \left[(\widehat{\mu}_{\widehat{v}} \neq \widehat{\mu}_{\widehat{v}'}) \vee (\widehat{\mu}_{\widehat{v}} = \widehat{\mu}_{\widehat{v}'} = q_0)\right]$$

2.3.11. Алгоритм Complete

Обозначим Complete$^*(\mathcal{S}^{(+)}, \mathcal{S}^{(-)}, C, P, N)$ процедуру нахождения автомата, который удовлетворяет заданному набору позитивных сценариев выполнения $\mathcal{S}^{(+)}$ и не удовлетворяет набору негативных сценариев $\mathcal{S}^{(-)}$, и в котором C состояний, максимальный размер охранного условия P, а суммарный размер охранных условий не больше N. Данная процедура состоит из (1) построения позитивного дерева сценариев $\mathcal{T}^{(+)}$ и негативного дерева сценариев $\mathcal{T}^{(-)}$, (2) формирования сведения к SAT (кодирование структуры автомата и охранных условий, отображения позитивного и негативного дерева сценариев, а также ограничения на кардинальность) и (3) вызова SAT-решателя. Также введем обозначение для случая, когда суммарный размер охранных условий в автомате остается неограниченным:

$$\text{Complete}(\mathcal{S}^{(+)}, \mathcal{S}^{(-)}, C, P) = \text{Complete}^*(\mathcal{S}^{(+)}, \mathcal{S}^{(-)}, C, P, N = \infty)$$

2.4. Синтез минимальных конечно-автоматных моделей

Разработанные в данной работе методы синтеза монолитных конечно-автоматных моделей зависят от трех параметров: число состояний автомата C,

максимальный размер каждого охранного условия P и суммарный размер всех охранных условий N. В реальности эти параметры неизвестны заранее, а их оценки, полученные какими-либо сторонними способами, могут быть далеки от оптимальных. На практике гораздо большей ценностью обладают модели меньших размеров, ввиду их эффективности и простоты. Обе задачи — *автоматизация* поиска параметров и их *минимизация* — могут быть решены одновременно путем применения *итеративного* подхода к синтезу минимальных моделей, рассматриваемому в текущем разделе.

2.4.1. Алгоритм Basic-min

Для быстрой оценки минимального числа состояний автомата, удовлетворяющего заданным сценариям выполнения $\mathcal{S}^{(+)}$, используется алгоритм Basic$(\mathcal{S}^{(+)}, C)$ с итеративным перебором параметра C «снизу вверх». После нахождения минимального числа состояний $C_{\min}$ производится минимизация числа переходов в автомате с использованием алгоритма Basic$^*(\mathcal{S}^{(+)}, C_{\min}, T)$ с итеративным перебором параметра T «сверху вниз», начиная с синтеза неограниченной модели: $T = \infty$. Псевдокод полученного алгоритма Basic-min$(\mathcal{S}^{(+)})$ представлен в листинге **??** вместе со вспомогательными функциями Basic-minC и Basic-minT, которые непосредственно выполняют минимизацию каждого параметра: C и T.

2.4.2. Алгоритмы Extended-min и Complete-min

Пусть параметр P — максимальный размер охранного условия в автомате — известен, а число состояний C оценено с помощью алгоритма Basic-minC. Последующая минимизация суммарного размера охранных условий в автомате производится аналогично алгоритму Basic-minT: с использованием алгоритма Extended$^*(\mathcal{S}^{(+)}, C, P, N)$ путем итеративного перебора параметра N «сверху вниз», начиная с синтеза неограниченной модели: $N = \infty$. Псевдокод полученного алгоритма Extended-min$(\mathcal{S}^{(+)}, P)$ приведен в листинге **??**. Алгоритм Complete-min$(\mathcal{S}^{(+)}, \mathcal{S}^{(-)}, P)$ определяется аналогично алгоритму Extended-min: $\mathcal{S}^{(+)}$ и $\mathcal{S}^{(-)}$ — наборы позитивных и негативных сценариев выполнения, P — максимальный размер охранного условия, внутри используется алгоритм Complete*.

2.4.3. Алгоритм Extended-min-UB

На данном этапе возникает закономерный вопрос — как выбрать подходящее значение параметра P? Можно заметить, что решение задачи синтеза существует

только если параметр P достаточно большой для того, чтобы охранные условия в автомате обладали достаточной выразительностью для представления желаемого поведения автомата. Самый простой способ перебора параметра P — «снизу вверх», начиная с $P = 1$, до тех пор, пока не будет найдено (с помощью алгоритма Extended-minN) решение с суммарным размером охранных условий $N = N^*_{\min}$ для некоторого $P = P^*$. Однако при этом может существовать некоторое $P' > P^*$, при использовании которого будет найдено еще меньшее решение: $N'_{\min} < N^*_{\min}$. Поэтому для нахождения глобально-наименьшего автомата в терминах N, необходимо продолжать поиск для $P > P^*$. Однако при этом возникает вопрос: в какой момент необходимо остановить перебор параметра P и считать найденное ранее решение оптимальным?

Для ответа на этот вопрос, рассмотрим некоторый момент перебора, когда $P = P'$. Заметим, что в лучшем случае все охранные условия в автомате имеют размер 1, кроме одного, имеющего размер P'. Также заметим, что в лучшем случае в автомате ровно $T_{\min}$ переходов, где значение $T_{\min}$ определено с помощью алгоритма Basic-minT. Исходя из этого, в лучшем случае суммарный размер охранных условий в автомате равен $N'_{\min} = T_{\min} - 1 + P'$. Обозначим $N^{\text{best}}_{\min}$ лучшее (наименьшее) значение, найденное в текущий момент. Так как перебор параметра P производится с целью нахождения $N'_{\min} < N^{\text{best}}_{\min}$, то есть $T_{\min} - 1 + P' < N^{\text{best}}_{\min}$, то из этого следует, что верхняя граница для параметра P есть $P' \leqslant N^{\text{best}}_{\min} - T_{\min}$.

Процесс перебора P до теоретической верхней границы может потребовать значительного количества времени, поэтому предлагается следующая эвристика для ускорения этого процесса. Рассмотрим два последовательных значения P' и $P'' = P' + 1$, а также соответствующим им значения $N'_{\min}$ и $N''_{\min}$. Равенство $N'_{\min} = N''_{\min}$ говорит о том, что процесс поиска оптимального P находится в локальном минимуме — *на плато*. Если при увеличении P'' равенство сохраняется, то в таком случае увеличивается *ширина плато*, равная $P'' - P'$. Обозначим w критическое значение ширины плато, при достижении которого останавливается перебор P. Выбор подходящего значения w обеспечивает компромисс между временем выполнения и глобальной минимальностью полученного решения. На практике, значение $w = 2$ является оптимальным. Стоит отметить, что при использовании данной эвристики разработанные методы остаются *точными* — синтезированные автоматы так же соответствуют заданному поведению.

Обозначим Extended-min-UB($\mathcal{S}^{(+)}$,w) процесс синтеза минимальной модели, удовлетворяющей заданным сценариям выполнения $\mathcal{S}^{(+)}$, с автоматическим

перебором параметра P с учетом значения критической ширины плато w: если $w = 0$, то первое найденное решение считается оптимальным; если $w > 0$, то используется описанная выше эвристика; если $w = \infty$, то перебор производится до теоретической верхней границы, также описанной выше. Псевдокод алгоритма Extended-min-UB$(\mathcal{S}^{(+)}, w)$ приведен в листинге **??**.

2.5. Индуктивный синтез, основанный на контрпримерах

Для того, чтобы производить синтез конечно-автоматных моделей не только примерам поведения в виде сценариев выполнения, но и с учётом LTL-спецификации — заданного набора LTL-свойств — в данной работе используется подход индуктивного синтеза, основанного на контпримерах (*Counterexample-Guided Inductive Synthesis* — CEGIS) [97; 98]. CEGIS является итеративным подходом и его общий вид изображен на рисунке 7. На каждой итерации производится (1) синтез модели (конечного автомата $\mathcal{A}$) с помощью алгоритма Complete, а затем (2) верификация — проверка выполнения заданных LTL-свойств с помощью верификатора NuSMV [19]. Если какие-то LTL-свойства не выполняются, верификатор генерирует *контрпримеры*, которые затем конвертируются в *негативные сценарии* и учитываются на следующей итерации CEGIS. В конечном итоге будет получен автомат $\mathcal{A}$, полностью удовлетворяющий заданной LTL-спецификации $\mathcal{L}$, либо же будет доказано его отсутствие при заданных параметрах C,P,T,N — в этом случае необходимо повторить процесс CEGIS с другими значениями параметров, например, ослабить ограничения на размер автомата.

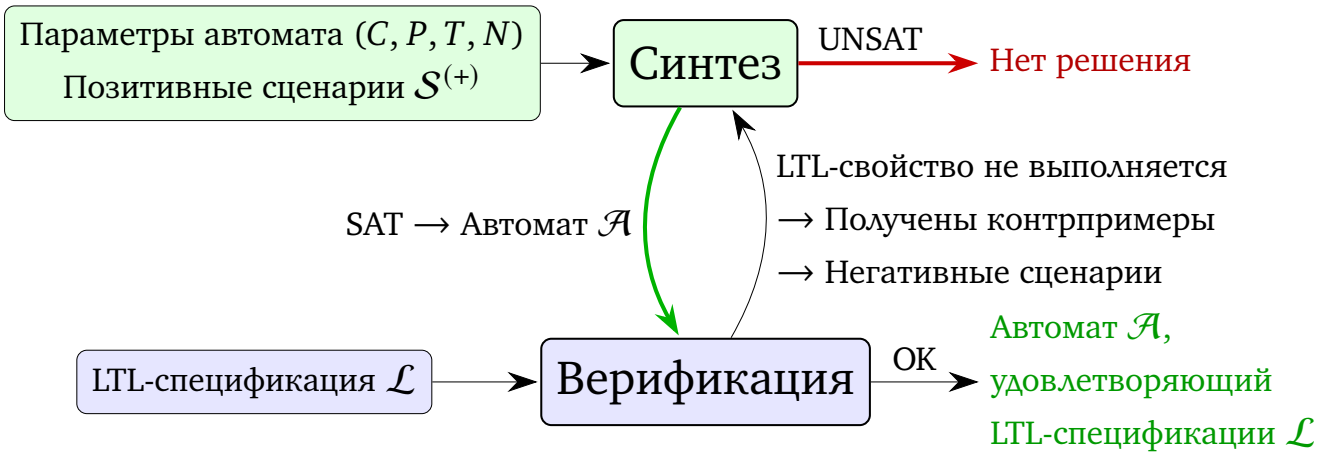


Рисунок 7 — Цикл «Синтез–Верификация» — индуктивный синтез, основанный на контрпримерах

2.5.1. Алгоритм CEGIS

Обозначим CEGIS$^*(\mathcal{S}^{(+)}, \mathcal{L}, C, P, T, N)$ процедуру, реализующую индуктивный синтез, основанный на контрпримерах, для нахождения конечного автомата $\mathcal{A}$, который удовлетворяет заданному набору позитивных сценариев выполнения $\mathcal{S}^{(+)}$ и LTL-спецификации $\mathcal{L}$, и в котором C состояний, суммарно не более T переходов, максимальный размер охранного условия P, а суммарный размер охранных условий не больше N. Также введем обозначение для случая, когда число переходов и суммарных размер охранных условий в автомате остаются неограниченными:

$$\text{CEGIS}(\mathcal{S}^{(+)}, \mathcal{L}, C, P) = \text{CEGIS}^*(\mathcal{S}^{(+)}, \mathcal{L}, C, P, T = \infty, N = \infty)$$

2.5.2. Алгоритм CEGIS-min

Рассмотрим автомат $\mathcal{A}$, полученный с помощью алгоритма CEGIS$(\mathcal{S}^{(+)}, \mathcal{L}, C, P)$. При попытке минимизации суммарного размер охранных условий N, автомат в общем случае перестанет удовлетворять заданной LTL-спецификации $\mathcal{L}$, однако уже учтеные ранее негативные сценарии продолжат не выполняться. Поэтому, для синтеза минимальных моделей в данной работе предлагается поддерживать минимальную модель на каждой итерации CEGIS. Как правило, запуск процесса CEGIS начинается с оценки параметров автомата с помощью алгоритма Extended-min-UB — обозначим полученные оценки C^*, P^* и N^*. Следом, с помощью алгоритма CEGIS$^*(\mathcal{S}^{(+)}, \mathcal{L}, C^*, P^*, T = \infty, N^*)$ производится попытка синтезировать конечный автомат, удовлетворяющий спецификации $\mathcal{L}$. Отсутствие решения (случай UNSAT на рисунке 7) означает, что заданная верхняя граница N для суммарного размера охранных условий слишком мала, поэтому необходимо ослабить это ограничение (например, взять значение $N' = N + 1$) и повторить CEGIS. Стоит отметить, что это является единственным моментом, когда прерывается процесс *инкрементального* решения с помощью SAT-решателя, потому как ослабление ограничений требует изъятия дизъюнктов из КНФ, а это является невозможным без полного перезапуска. Обозначим CEGIS-min$(\mathcal{S}^{(+)}, \mathcal{L}, C, P)$ алгоритм, реализующий индуктивный синтез для нахождения *минимальной* конечно-автоматной модели, которая удовлетворяет заданному набору позитивных сценариев выполнения $\mathcal{S}^{(+)}$ и LTL-спецификации $\mathcal{L}$, и в которой C состояний, максимальный размер охранного условия P, а суммарный размер охранных условий является минимальным: $N_{\min}$.

2.6. Программное средство fbSAT для синтеза и верификации конечно-автоматных моделей

Все предложенные в данной работе методы синтеза и верификации конечно-автоматных моделей были реализованы в виде программного средства fbSAT с использованием языка программирования Kotlin. Исходный код распространяется под лицензией GNU GPLv3 и доступен онлайн [100]. В качестве бэкенда возможно использование любого SAT-решателя, поддерживающего работу через стандартный ввод или файлы формата DIMACS [113].

Стоит отметить, что существующие текстовые интерфейсы общения с SAT-решателями не позволяют использовать возможность решать последовательные SAT задачи *инкрементально*, без потери процесса решения при перезапуске. В ходе выполнения работы была реализована обертка *incremental-cryptominisat* [114] для SAT-решателя CryptoMiniSat, позволяющая формулировать и решать инкрементальные задачи через текстовый интерфейс с использованием формата iCNF (расширенный формат DIMACS).

Альтернативой является *нативный интерфейс*, однако его поддержка требует отдельной реализации для каждого SAT-решателя. В данной работе для этих целей была использована технология JNI (*Java Native Interface*) [115]. Совместно с Гречишкиной Дарьей была разработана библиотека `kotlin-satlib`[7] [116], содержащая реализации нативных интерфейсов для современных SAT-решателей: MiniSAT[8], Glucose[9], Kissat[10], CaDiCaL[11]. Библиотека написана на языке Kotlin с поддержкой возможности ее использования из других JVM языков, например, из Java.

При проведении экспериментов в данной работе был использован SAT-решатель CaDiCaL посредством реализованного нативного интерфейса. Данный выбор обоснован тем, что CaDiCaL является одним из наиболее эффективных и робастных решателей — он способен решать как простые, так и сложные задачи, в отличие от, например, решателя MiniSat, который не всегда справляется с большими экземплярами задачи SAT. Стоит также отметить, что в некоторых случаях решатели CryptoMiniSat и Glucose показывают схожие результаты.

[7] `https://github.com/Lipen/kotlin-satlib`
[8] `https://github.com/niklasso/minisat`
[9] `https://github.com/audemard/glucose`
[10] `https://github.com/arminbiere/kissat`
[11] `https://github.com/arminbiere/cadical`

2.7. Экспериментальное исследование: Pick-and-Place манипулятор

В данном разделе приводится экспериментальное исследование, посвященное применению разработанных методов к задаче синтеза конечно-автоматной модели логического контроллера, управляющего Pick-and-Place (PnP) манипулятором. Синтезированные модели верифицируются программно и проверяются вручную в виртуальной среде исполнения nxtSTUDIO [112].

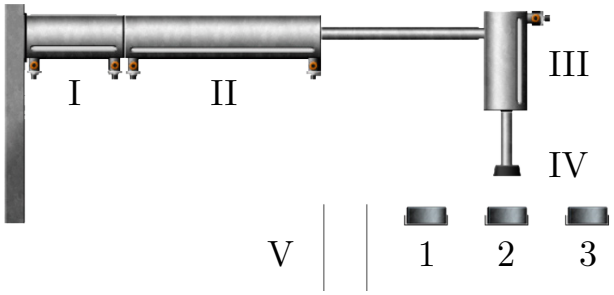


Рисунок 8 — Pick-and-Place манипулятор

Pick-and-Place (PnP) манипулятор, изображенный на рисунке 8, состоит из двух горизонтальных пневматических цилиндров (I, II), одного вертикального цилиндра (III) и захватывающего устройства с вакуумной присоской (IV) для подъема рабочий деталей. Когда рабочая деталь оказывается во входном лотке (1, 2, 3), горизонтальные цилиндры располагают актюатор над деталью, вертикальный цилиндр опускает захватывающее устройство, который в свою очередь захватывает деталь, после чего вся система аналогичным образом приходит в движение для переноса захваченной детали в выходной лоток (V). Данная система управления реализована в соответствии со стандартом IEC 61499 с использованием функциональных блоков в среде моделирования nxtSTUDIO [112]. Логический контроллер, осуществляющий управление, выполнен в виде базового функционального блока, интерфейс которого включает в себя одно входное событие `REQ` (*request*), одно выходное событие `CNF` (*confirmation*), а также десять входных и семь выходных переменных. Контроллер PnP-манипулятора оперирует следующими входными сигналами $\mathcal{X}$, поступающими от объекта управления — среды исполнения:

- `c1Home`/`c1End` — горизонтальный цилиндр I находится в крайнем левом/правом положении;
- `c2Home`/`c2End` — горизонтальный цилиндр II находится в крайнем левом/-правом положении;
- `vcHome`/`vcEnd` — вертикальный цилиндр III находится в крайнем верхнем/-нижнем положении;

– pp1/pp2/pp3 — рабочая деталь находится на входном лотке 1/2/3;
– vac — вакуумная присоска включена.

В свою очередь контроллер PnP-манипулятора может посылать следующие сигналы $\mathcal{Z}$ объекту управления:

– c1Extend/c1Retract — удлинить/втянуть цилиндр I;
– c2Extend/c2Retract — удлинить/втянуть цилиндр II;
– vcExtend — удлинить цилиндр III;
– vacuum_on/vacuum_off — включить/выключить ваккумную присоску.

2.7.1. Синтез минимальной конечно-автоматной модели по примерам поведения

Для исследования эффективности и практической применимости разработанных методов синтеза минимальных моделей по примерам поведения, производится сравнение с двухэтапным подходом из [49], где на первом этапе с помощью SAT-решателя строится базовая модели, удовлетворяющая заданым сценариям выполнения, а затем охранные условия полученной модели минимизируются с помощью CSP-решателя. Стоит отметить, что превосходство данного двухэтапного метода над другими методами, например, EFSM-tools [34], уже было показано в [49], поэтому сравнение происходит только с двухэтапным методом, впоследствии называемым «Two-stage».

Для синтеза минимальной конечно-автоматной модели контроллера PnP-манипулятора по заданным сценариям выполнения $\mathcal{S}^{(+)}$ был применен алгоритм Extended-min-UB$(\mathcal{S}^{(+)}, w)$ с различными значениями параметра w — ширины плато для поиска локального минимума: $w = 0$ для случая, когда первое найденное решение считается финальным, $w = \infty$ для нахождения глобально-минимального решения, а также $w = 2$ для случая использования предложенной эвристики. Результаты эксперимента представлены в Таблице 1, где $\mathcal{S}^{(+)}$ — набор сценариев выполнения, $|\mathcal{T}^{(+)}|$ — размер дерева сценариев; «Время, с» — время работы в секундах, $N_{\min}$ — минимальный суммарный размер охранных условий; для метода Two-stage [49]: $C_{\min}$ — минимальное число состояний, $T_{\min}$ — минимальное число переходов; для метода Extended-min-UB: минимальное число состояний опущено, так как совпадает с $C_{\min}$ для двухэтапного метода, P — максимальный размер каждого охранного условия, T — число переходов (не было минимизировано), w — критическая ширина плато для предложенной эвристики. Результаты свидетельствуют о том, что разработанный метод Extended-min-UB способен генерировать

более компактные конечные автоматны, чем двухэтапный метод, при этом значения эвристического параметра $w = 2$ достаточно для получения оптимального результата в терминах $N_{\min}$.

Таблица 1 — Результаты синтеза минимальной конечно-автоматной модели логического контроллера PnP-манипулятора по примерам поведения с помощью двухэтапного метода Two-stage [49] и алгоритма Extended-min-UB

$\mathcal{S}^{(+)}$	$\lvert\mathcal{T}^{(+)}\rvert$	Two-stage [49]				Extended-min-UB ($w = 0$)				Extended-min-UB ($w = 2$)				Extended-min-UB ($w = \infty$)			
		Время, с.	$C_{\min}$	$T_{\min}$	N_{sum}	Время, с.	P	T	$N_{\min}$	Время, с.	P	T	$N_{\min}$	Время, с.	P	T	$N_{\min}$
$\mathcal{S}^{(1)}$	24	8	6	8	15	3	3	8	14	4	3	8	14	5	3	8	14
$\mathcal{S}^{(10)}$	234	3	8	17	36	17	3	18	38	58	5	16	25	87	5	16	25
$\mathcal{S}^{(39)}$	960	13	8	15	32	41	3	18	38	124	5	16	25	162	5	16	25
$\mathcal{S}^{(49)}$	2939	36	8	18	60	602	5	18	44	4305	6	16	39	51666	6	16	39

2.7.2. Синтез минимальный конечно-автоматных моделей по примерам поведения и LTL-спецификации

Следующий эксперимент посвящен учету формальной спецификации с помощью применения индуктивного синтеза, основанного на контрпримерах. Для использования LTL-свойств *живости* (*liveness*) верификация моделей с помощью NuSMV проводилась в замкнутом цикле [117] с заранее подготовленной формальной моделью объекта управления — PnP-манипулятора. Эта модель определяет состояние объекта управления в зависимости от команд управления контроллера — синтезированной конечно-автоматной модели. Набор использованых LTL-свойств представлен в таблице 2 и включает в себя как свойства безопасности φ_1–φ_6 («система не окажется в нежелательном состояния»), так и свойства живости φ_7–φ_{10} («что-то полезное когда-нибудь произойдет»). Свойства φ_1–φ_7 «зафиксированы» — используются во всех экспериментах, в то время как использование свойств φ_8–φ_{10} варьируется. Стоит отметить, что эти три свойства определяют тот факт, что если рабочая деталь (1–3) размещается на входном лотке, то она когда-нибудь будет обработана. Однако в оригинальной системе PnP-манипулятора [118] выполняется *только* свойство φ_8 , касающееся первой детали — если в первом входном лотке всегда присутствует рабочая деталь (при ее подъеме на ее месте в этот же момент появляется новая), то рабочие детали во втором и третьем входных лотках никогда не будут обработаны, что нарушает свойства живости φ_9 и φ_{10} . Поэтому

Таблица 2 — Темпоральные свойства для системы PnP-манипулятора

	LTL-свойство и его описание
	Фиксированные свойства
φ_1	$\mathbf{G}(\neg(\texttt{c1Extend} \wedge \texttt{c1Retract}))$ Цилиндр I не должен одновременно сжиматься и расширяться.
φ_2	$\mathbf{G}(\neg(\texttt{c2Extend} \wedge \texttt{c2Retract}))$ Аналогичное свойство для цилиндра II.
φ_3	$\mathbf{G}(\neg(\texttt{vacuum\_on} \wedge \texttt{vacuum\_off}))$ Аналогичное свойство для вакуумной присоски IV.
φ_4	$\mathbf{G}(\neg\texttt{vcHome} \wedge \neg\texttt{vcEnd} \rightarrow \texttt{c1Home} \vee \texttt{c1End})$ Пока вертикальный цилиндр III находится в промежуточном положении, горизонтальный цилиндр I должен оставаться в своем крайнем положении.
φ_5	$\mathbf{G}(\neg\texttt{c1Home} \wedge \neg\texttt{c1End} \rightarrow \texttt{vcHome} \vee \texttt{vcEnd})$ Пока горизонтальный цилиндр I находится в промежуточном положении, вертикальный цилиндр III должен оставаться в своем крайнем положении.
φ_6	$\mathbf{G}(\texttt{all\_home} \wedge \neg\texttt{pp1} \wedge \neg\texttt{pp2} \wedge \neg\texttt{pp3} \wedge \neg\texttt{lifted} \rightarrow \mathbf{X}(\neg\texttt{c1Extend} \wedge \neg\texttt{c2Extend} \wedge \neg\texttt{vcExtend}))$ Если все цилиндры манипулятора находятся в своих начальных положениях, то, пока нет необходимости в обработке (переносе) рабочей детали, не должно возникать команд управления для начала движения.
φ_7	$\mathbf{G}(\texttt{lifted} \rightarrow \mathbf{F}(\texttt{dropped}))$ Если рабочая деталь была поднята со входного лотка, то она должна быть когда-нибудь опущена в выходной лоток V.
	Вариабельные свойства
φ_8	$\mathbf{G}(\texttt{pp1} \rightarrow \mathbf{F}(\texttt{vp1}))$ Рабочая деталь во входном лотке 1 должна быть обработана в будущем.
φ_9	$\mathbf{G}(\texttt{pp2} \rightarrow \mathbf{F}(\texttt{vp2}))$ Рабочая деталь во входном лотке 2 должна быть обработана в будущем.
φ_{10}	$\mathbf{G}(\texttt{pp3} \rightarrow \mathbf{F}(\texttt{vp3}))$ Рабочая деталь во входном лотке 3 должна быть обработана в будущем.

каждое дополнительное (по отношению к φ_1–φ_7) свойство φ_8–φ_{10} рассматривается отдельно от остальных, при этом предполагается, что рабочие детали появляются только на соответствующих входных лотках. Для эксперимента с использованием дополнительного LTL-свойства φ_9 был использован специальный набор сценариев $\mathcal{S}^{(1)\prime\prime}$, состоящий из одного сценария, описывающего обработку детали во втором входном лотке. Аналогично, для свойства φ_{10} был использован специальный набор сценариев $\mathcal{S}^{(1)\prime\prime\prime}$, состоящий из одного сценария, описывающего обработку детали в третьем входном лотке.

Проведенное экспериментальное сравнение включало в себя три метода: два разработанных метода CEGIS и CEGIS-MIN, входящие в состав FBSAT, а также расширение метода FBCSP для учета LTL-спецификации, называемое впоследствии FBCSP+LTL [48]. Для обоих разработанных методов параметры C^* и P^* были предварительно оценены с помощью алгоритма EXTENDED-MIN-UB с параметром $w = 2$, время работы было учтено в суммарном времени работы алгоритма CEGIS. Дополнительно, синтезированные модели были вручную протестированы в nxtSTUDIO [112] — загружены в симуляционную среду и проверены на соответствие желаемому поведению. Результаты данного экспериментального исследования представлены в таблице 3, где «Дополнительное LTL-свойство» — одно из свойств φ_8–φ_{10}, использованное в дополнение к свойствам φ_1–φ_7, $\mathcal{S}^{(+)}$ — набор использованых позитивных сценариев выполнения $\mathcal{S}^{(+)}$, N_{init} — начальный минимальный суммарный размер охранных условий (для автомата, полученного с помощью алгоритма EXTENDED-MIN-UB($\mathcal{S}^{(+)}, w$)), «Время, с» — время работы в секундах, P — максимальный размер охранного условия, N — финальный суммарный размер охранных условий (при использовании алгоритма CEGIS-MIN это значения является минимальным), «#iter» — число итераций CEGIS.

Анализируя полученные результаты, можно заметить, что модели, найденные с помощью подхода CEGIS всегда имеют больший размер (в терминах суммарного размера охранных условий N), нежели модели, построенные только по сценариям выполнения. Это объясняется тем, что используемые сценарии выполнения не полностью покрывают рассмотренную LTL-спецификацию. Поэтому алгоритм CEGIS-MIN всегда находит наименьшее решение и во всех случаях превосходит FBCSP+LTL [48], как по времени работы, так и по размеру моделей. Наиболее интересным результатом является то, что CEGIS-MIN позволяет эффективно синтезировать модели по наборам сценариев $\mathcal{S}^{(1)}$, $\mathcal{S}^{(1)\prime\prime}$ и $\mathcal{S}^{(1)\prime\prime\prime}$ — эти сценарии «не покрывают» соответствующие свойства живости φ_8–φ_{10} в том смысле, что эти сценарии описывают процесс

Таблица 3 — Результаты применения подхода CEGIS к синтезу конечно-автоматной модели логического контроллера PnP-манипулятора по примерам поведения и LTL-спецификации

Дополнительное LTL-свойство	$\mathcal{S}^{(+)}$	N_{init}	CEGIS-MIN				CEGIS			FBCSP+LTL [48]		
			Время, с.	#iter	P	$N_{\min}$	Время, с.	#iter	N	Время, с.	#iter	N
$\mathbf{G}(\mathtt{pp1} \rightarrow \mathbf{F}(\mathtt{vp1}))$	$\mathcal{S}^{(1)}$	14	55	273	3	16	19	79	33	>12h	>500	–
	$\mathcal{S}^{(10)}$	25	242	20	5	28	60	2	118	613	10	40
	$\mathcal{S}^{(39)}$	25	425	70	5	28	142	20	128	1019	2	41
$\mathbf{G}(\mathtt{pp2} \rightarrow \mathbf{F}(\mathtt{vp2}))$	$\mathcal{S}^{(1)''}$	14	63	193	3	16	37	188	29	>12h	>500	–
$\mathbf{G}(\mathtt{pp3} \rightarrow \mathbf{F}(\mathtt{vp3}))$	$\mathcal{S}^{(1)'''}$	14	116	353	3	16	400	1450	34	>12h	>500	–

обработки только одной рабочей детали. Существующий метод FBCSP+LTL [48] не справился в этих случаях, в то время как разработанный метод CEGIS-MIN с легкостью преуспел. Также стоит отметить, что алгоритм CEGIS позволяет синтезировать модели быстрее, однако не обеспечивает минимальности охранных условий.

2.8. Экспериментальное исследование: SYNTCOMP

В этом разделе описывается применение разработанных методов к задаче синтеза системы переходов (*transition system*) [26; 27] по входным данным с соревнования по реактивному синтезу SYNTCOMP [99]. Один из треков соревнования SYNTCOMP — трек последовательного синтеза (*sequential synthesis track*) — посвящен задаче синтеза системы переходов по заданной LTL-спецификации, также известной как задача LTL-синтеза. Существует множество различных программных средств, осуществляющих LTL-синтез, среди которых можно выделить BoSy [26; 27] и Strix [28]. Стоит отметить, что среди всех доступных программных средств только BoSy ограничивает размеры (число состояний) генерируемых систем переходов, однако BoSy не минимизирует размеры охранных условий, что значительно затрудняет анализ получаемых систем человеком. Также стоит отметить, что на текущий момент разработанное в данной работе программное средство FBSAT неприменимо в явном виде к задаче LTL-синтеза, так как для FBSAT необходимым условием является наличие некоторого множества позитивных сценариев выполнения $\mathcal{S}^{(+)}$. Несмотря на это, FBSAT может быть применен для *минимизации* систем переходов, генерируемых BoSy.

Формально[12], *система переходов* $\mathcal{T}$ это кортеж $(T, t_0, \Sigma = I \cup O, \tau)$, где T — множество состояний, $t_0 \in T$ — стартовое состояние, Σ — алфавит системы, I — множество пропозициональных переменных, управляемых окружением (*входы*), O — множество пропозициональных переменных, управляемых системой (*выходы*), $\tau\colon T \times 2^I \to 2^O \times T$ — функция перехода, отображающая состояние t и входной набор $\boldsymbol{i} \in 2^I$ в выходной набор $\boldsymbol{o} \in 2^O$ и новое состояние t'. Можно заметить сходство систем переходов и конечно-автоматных моделей ECC базовых функциональных блоков. Если система переходов обладает семантикой, схожей с семантикой автомата Мура (выходы в функции перехода зависят от состояния системы), то такая система может быть смоделирована в виде ECC, а значит и синтезирована с помощью FBSAT.

Наборы данных (*инстансы*) на соревновании SYNTCOMP представляют собой JSON-файлы с описанием интерфейса системы и набора инвариантов — свойств на языке LTL, которые должны выполняться в каждый момент времени работы системы. В листинге **??** приведен пример инстанса `lilydemo19`, описывающего систему с семантикой автомата Мили. Данная система оперирует входами {`ec`, `ets`} и выходами {`fl`, `hl`}. Логика работы системы описывается с помощью LTL-свойств, указанных в поле `guarantees`, а дополнительные глобальные ограничения (предположения) записаны в поле `assumptions`.

При выполнении данной работы был использован набор из 136 инстансов с соревнования SYNTCOMP 2018. Для получения конечно-автоматных моделей в формате NuSMV по имеющимся LTL-спецификациям было использовано программное средство BoSy (input-symbolic, QBF-encoding, максимальное число состояний: 10, максимальное время работы: 1 час), в результате чего только только для 97 из 136 инстансов были получены результирующие модели. В листинге **??** приведена NuSMV модель для инстанса `lilydemo19`. Все полученные модели были просимулированы с помощью NuSMV с целью получения случайных сценариев выполнения различных длин. Для этого была использована команда ‘`simulate -r -k <length>`’ для симуляции (где ‘`<length>`’ — число шагов симуляции) и команда ‘`show_traces -a -v`’ для сохранения трассировок.

[12]Здесь стоит отметить, что в данном разделе для описания системы переходов используется *оригинальная* нотация из [27]. Эта нотация может конфликтовать с другими частями данной работы — следует считать, что все объявления в данном разделе действуют только здесь. Также стоит отметить, что в оригинальной нотации для обозначения *множества всех наборов значений* пропозициональных переменных используется нотация 2^X, где X — множество пропозициональных переменных, однако более корректным обозначением было бы $\mathbb{B}^{|X|}$.

Полученные сценарии выполнения были использованы для синтеза конечно-автоматных моделей с помощью FBSAT. На рисунке 9 представлена синтезированная модель для описанного выше инстанса `lilydemo19`. Для синтеза было использовано пять сценариев длины 10 (`scenarios-k5-l10`), алгоритм EXTENDED-MIN-UB($w = 0$), время синтеза составило менее секунды. Модель состоит из $C = 2$ состояний и $T = 4$ переходов, а охранные условия имеют суммарный размер $N = 6$. Дополнительный этап верификации подтвердил соответствие синтезированной системы исходной LTL-спецификации, указанной во входном файле `lilydemo19.json`. Как можно заметить, синтезированная модель полностью эквивалентна исходной NuSMV модели, поэтому необходимо рассмотрение более «сложного» инстанса.

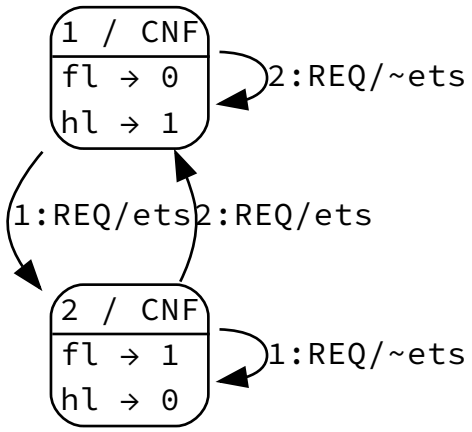


Рисунок 9 — Конечно-автоматная модель для инстанса `lilydemo19`, синтезированная с помощью FBSAT

Рассмотрим инстанс `full_arbiter_3` — данная система оперирует входами $\{r_0, r_1, r_2\}$ и выходами $\{g_0, g_1, g_2\}$. Полученная с помощью BoSy система переходов $\mathcal{T}_{\text{original}}$ изображена на рисунке **??** и состоит из $C = 8$ состояний и $T = 28$ переходов, а суммарный размер охранных условий $N = 147$. На этом этапе можно предположить, что полученная модель не является минимальной, а значит, возможно применение FBSAT для синтеза минимальной модели, также соответствующей исходной LTL-спецификации — для этого был использован алгоритм CEGIS-MIN. Стоит заметить, что для более эффективного синтеза необходимо полное покрытие состояний сценариями выполнения. Поэтому было использовано 20 сценариев, каждый длины 20 (`scenarios-k20-l20`).

Стоит отметить, что формальное определение системы переходов, данное выше, не обязывает функцию переходов τ быть детерминированной, однако FBSAT всегда генерирует детерминированные модели. Также стоит отметить, что формальное определение системы переходов не включает в себя функцию приоритета переходов,

которая присутствует в определении ECC. Для того, чтобы модели, синтезируемые fbSAT, соответствовали моделям, получаемым с помощью BoSy, в fbSAT было добавлено ограничение на «дизъюнктивные переходы»[13] — в каждом состоянии $q \in Q$ для каждого входа $u \in \mathcal{U}$ выполняется не более одного перехода. В результате была синтезирована модель $\mathcal{A}_{\text{deterministic}}$, изображенная на рисунке **??**, с тем же числом состояний и переходов, что и $\mathcal{T}_{\text{original}}$, однако с меньшим суммарным размером охранных условий: $N = 105$.

Если же не использовать введенное ограничение на «дизъюнктивные переходы», а использовать fbSAT в оригинальном виде, то синтезируемые модели будут детерминированными ECC (из-за функции приоритета переходов), но недетерминированными системами переходов. В таком случае результирующая модель $\mathcal{A}_{\text{non-deterministic}}$, изображенная на рисунке **??**, обладает наименьшим суммарным размером охранных условий: $N = 52$.

Полученные результаты показывают, что предложенный подход к явному кодированию деревьев разбора булевых формул, соответствующих охранным условиям на переходах автомата, позволяет существенно сократить суммарный размер охранных условий в автомате. Стоит также отметить, что данный подход применим не только *после* LTL-синтеза — возможно расширить SAT-/QBF-сведение в BoSy предложенной кодировкой для охранных условий для их минимизации непосредственно *в процессе* синтеза.

2.9. Метод синтеза модульных конечно-автоматных моделей с параллельной композицией модулей по примерам поведения

В этом разделе рассматривается задача синтеза модульной конечно-автоматной модели с параллельной композицией модулей. Аналогично монолитному синтезу (раздел 2.3), исходная задача сводится к задаче выполнимости SAT. Полученное сведение состоит из двух компонент: (1) определение переменных и (2) определение ограничений.

2.9.1. Сведение к SAT: переменные

При параллельной композиции для описания каждого модуля $m \in [1 \,..\, M]$ используются те же переменные, что и при монолитном синтезе — ко всем переменным добавляется верхний индекс (m). Здесь и далее фраза «модуль m» означает

[13] Флаг `--encode-disjunctive-transitions` в fbSAT

«модуль с номером m». Дополнительно, объявляется переменная $\zeta_z \in [1\,..\,M]$, кодирующая модуль, управляющий выходной переменной $z \in Z$. В данной постановке задачи рассматривается ситуация, когда каждый из модулей «управляет» своим собственным набором выходных переменных $\mathcal{Z}^{(m)} \subset \mathcal{Z}$, причём эти наборы попарно не пересекаются. Стоит отметить, что в общем случае эти наборы неизвестны заранее, а определяются динамически в ходе решения задачи SAT. Возможна также ситуация, когда полное или частичное распределение переменных по модулям известно — в таком случае разработанный метод позволяет это учесть путём введения дополнительных ограничений на описанную переменную ζ.

2.9.2. Сведение к SAT: ограничения

При модульном синтезе ограничения так же, как и при монолитном синтезе, делятся на две группы: (1) ограничения, кодирующие структуру синтезируемого (модульного) автомата, (2) ограничения, кодирующие соответствие поведения синтезируемого (модульного) автомата дереву сценариев.

Для кодирования структуры каждого модуля, входящего в состав модульного автомата, используются те же ограничения, что и при монолитном синтезе. Дополнительно, вводятся следующие ограничения:

– каждая выходная переменная $z \in \mathcal{Z}$ управляется ровно одним модулем:

$$\mathrm{EO}_{m\in[1..M]}(\zeta_z = m)$$

– каждый модуль $m \in [1\,..\,M]$ управляет хотя бы одной выходной переменной:

$$\mathrm{ALO}_{z\in\mathcal{Z}}(\zeta_z = m)$$

– если выходная переменная $z \in \mathcal{Z}$ не управляется модулем $m \in [1\,..\,M]$, то алгоритмы во всех состояниях $q \in Q$ модуля m «ничего не делают» — переводят $\top$ в $\top$ и $\bot$ в $\bot$:

$$(\zeta_z \neq m) \rightarrow \bigwedge_{q\in Q}\left[(\gamma^{(m)}_{q,z,\top} = \top) \wedge (\gamma^{(m)}_{q,z,\bot} = \bot)\right]$$

В свою очередь ограничения, кодирующие соответствие поведения синтезируемого модульного автомата дереву сценариев, отличаются от аналогичных при монолитном синтезе. Для отображения пассивных вершин $V^{(\text{passive})}$ дерева сценариев используются те же ограничения, что и при монолитном синтезе. А при отображении активных вершин $V^{(\text{active})}$ дерева сценариев $\mathcal{T}^{(+)}$ на состояния Q

каждого модуля $m \in [1\,..\,M]$ необходимо дополнительно учитывать контролируемые этим модулем выходные переменные:

$$\bigwedge_{m\in[1..M]} \bigwedge_{v\in V^{(\text{active})}} \bigwedge_{q\in Q} \Bigg[(\mu_v^{(m)} = q) \leftrightarrow \Big((\lambda_{q',e,u}^{(m)} = q) \wedge (\varphi_q^{(m)} = o) \wedge \bigwedge_{z\in Z} \big[(\zeta_z = m) \rightarrow (\gamma_{q,z,\,tov(p,z)}^{(m)} = tov(v,z)) \big] \Big) \Bigg], \tag{16}$$

где $p = tp(v)$, $q' = \mu_p^{(m)}$, $e = tie(v)$, $u = tin(v)$, $o = toe(v)$.

Дополнительно, необходимо гарантировать корректность выходных событий модулей для соответствия дереву сценариев. Для этого рассмотрим обработку модульным автоматом некоторого входного действия $e^I[\bar{x}]$ из некоторой *активной* вершины v дерева сценариев ($toe(v) \neq \varepsilon$). Каждый параллельный модуль $m \in [1\,..\,M]$, входящий в состав модульного автомата, в ответ на входное действие генерирует выходное действие $e^O_{(m)}[\bar{z}_{(m)}]$. Возникает необходимость некоторым образом объединить эти выходные действия в одно $e^O[\bar{z}]$, которое будет являться финальной реакцией модульного автомата на исходное входное действие $e^I[\bar{x}]$. Фактически, необходимо использовать специальный функциональный блок, реализующий объединение событий и называемый E_MERGE в IEC 61499. Существует несколько способов такого объединения: (1) синхронное *rendezvous*, (2) асинхронное *rendezvous*, (3) *гибридное*.

Синхронное *rendezvous* подразумевает ожидание завершения работы всех модулей, при этом корректной считается ситуация, когда выходные события всех модулей совпадают. При формировании выходного действия $e^O[\bar{z}]$ используется общее выходное событие e^O, а обновленные значения выходных переменных $\bar{z}_{(m)}$ просто конкатенируются в $\bar{z}$, так как множества $\mathcal{Z}^{(m)}$ попарно не пересекаются. Стоит отметить, что в этом случае ни один модуль не должен игнорировать входное действие — не должен генерировать пустое событие $e^O = \varepsilon$.

E_MERGE с асинхронным *rendezvous* генерирует выходное действие равное входному, не ожидая завершения работы всех модулей При этом модульный автомат, использующий такой тип объединения событий, в ответ на некоторое входное действие $e^I[\bar{x}]$ может реагировать несколькими выходными действиями. Корректной считается следующая ситуация: каждый модуль генерируют либо выходное событие e^O, либо пустое событие ε; хотя бы один модуль генерирует непустое событие; объединение обновленных значений выходных переменных равно $\bar{z}$. Здесь $e^O[\bar{z}]$ — выходное действие, записанное в обрабатываемой активной

вершине дерева сценариев. Такая ситуация нарушает соответствие поведения модульного автомата дереву сценариев, однако может являться допустимой в реальном мире, так как последовательность выходных действий в ответ на одно входное все равно приводит систему к корректному состоянию с точки зрения значений выходных переменных.

Гибридный способ объединения событий является совокупностью синхронного и асинхронного *rendezvous*. Гибридный `E_MERGE` генерирует ровно одно выходное событие («асинхронное» поведение), при этом непустых входных событий может быть несколько, и все они учитываются для генерации совместного события. Это достигается тем, что гибридный блок ожидает завершения работы всех модулей («синхронное» поведение) лишь некоторое ограниченное время. При этом считается, что все неответившие модули игнорировали входное действие, а значит, продуцировали пустое выходное событие ε. Корректной считается ситуация, когда каждый модуль генерируют либо выходное событие e^O, либо пустое событие ε, а объединение обновленных значений выходных переменных равно $\bar{z}$, где $e^O[\bar{z}]$ — выходное действие, записанное в обрабатываемой активной вершине дерева сценариев.

В приведенном выше ограничении (16), кодирующем отображение активных вершин дерева на состояния модулей, используется синхронный вариант объединения выходных событий.

2.9.3. Алгоритмы Parallel-Basic и Parallel-Extended

Обозначим Parallel-Basic$^*(\mathcal{S}^{(+)}, M, C, T)$ алгоритм, реализующий синтез модели, удовлетворяющей заданным сценариям выполнения $\mathcal{S}^{(+)}$ и состоящей из M *параллельно* соединенных модулей, каждый из которых состоит из C состояний (суммарное число состояний равно $M \cdot C$), а суммарное число переходов во всех модулях не больше T. Также введём обозначение для случая, когда число переходов в автомате остается неограниченным:

$$\text{Parallel-Basic}(\mathcal{S}^{(+)}, M, C) = \text{Parallel-Basic}^*(\mathcal{S}^{(+)}, M, C, T = \infty)$$

Отметим, что модификации сведения для синтеза расширенных моделей не отличаются от описанных в главе **??** — структура охранных условий кодируется независимо от количества модулей и их композиции. Поэтому здесь и далее алгоритмы, реализующие синтез расширенных моделей, будут объявляться без

пояснений:

$$\textsc{Parallel-Extended}(\mathcal{S}^{(+)}, M, C, P) =$$
$$= \textsc{Parallel-Extended}^{*}(\mathcal{S}^{(+)}, M, C, P, N = \infty)$$

2.10. Метод синтеза конечно-автоматной модели модульного логического контроллера с последовательной композицией модулей по примерам поведения

Последовательная композиция модулей значительно отличается от параллельной, так как модули в таком случае перестают быть независимыми — каждый следующий модуль зависит от предыдущего. В данной работе рассматривалась постановка задачи синтеза модульного контроллера со следующими особенностями и допущениями: (1) каждый модуль зависит от всех входных переменных *внешнего* (модульного) автомата, (2) значения выходных переменных каждого модуля зависят от предыдущих значений выходных переменных предыдущего модуля (для первого модуля «предыдущим» считается он сам), (3) входное событие обрабатывается (происходит генерация ответного выходного события) только последним модулем, до которого оно неизменно передается от модуля к модулю.

2.10.1. Сведение к SAT: переменные

Аналогично параллельной композиции, структура каждого модуля кодируется так же, как и при монолитном синтезе, с учетом допущения о том, что только последний модуль реагирует на входные события, а все предыдущие модули просто неизменно передают полученное входное событие далее — это достигается тем, что множества выходных событий каждого модуля приравниваются множеству входных событий:

$$\forall m \in [1..(M-1)] : E^{O(m)} = E^{I}$$

Дополнительно, объявляется переменная $\nabla^{(m)}_{v,z} \in \mathbb{B}$, кодирующая «вычисленное» значение выходной переменной $z \in \mathcal{Z}$ при отображении каждой вершины $v \in V$ дерева сценариев — значение в каждый момент времени после обработки соответствующей вершины дерева.

2.10.2. Сведение к SAT: ограничения

Для определения переменной $\nabla^{(m)}$ объявляются следующие ограничения. Изначально, в момент времени после «обработки» фиктивного корня ρ дерева

сценариев, вычисленные значения всех выходных переменных $z \in \mathcal{Z}$ во всех модулях $m \in [1\,..\,M]$ равны `False`:

$$\bigwedge_{m\in[1..M]} \bigwedge_{z\in\mathcal{Z}} (\nabla_{\rho,z}^{(m)} = \bot)$$

В следующие моменты времени, при обработке вершин $v \in V \setminus \{\rho\}$, вычисляемые значения выходных переменных определяются в соответствии с работой алгоритмов в состояниях, в которых в эти моменты времени находятся модули — в тех состояниях, в которые отображаются рассматриваемые вершины дерева сценариев: $\mu_v^{(m)}$. Для первого модуля:

$$\bigwedge_{v\in V\setminus\{\rho\}} \bigwedge_{z\in\mathcal{Z}} \left[\nabla_{v,z}^{(m)} \leftrightarrow \gamma^{(m)}_{\mu_v^{(m)},z,\,tov(tp(v),z)}\right]$$

Для последующих модулей:

$$\bigwedge_{m\in[2..M]} \bigwedge_{v\in V\setminus\{\rho\}} \bigwedge_{z\in\mathcal{Z}} \left[\nabla_{v,z}^{(m)} \leftrightarrow \gamma^{(m)}_{\mu_v^{(m)},z,\nabla_{v,z}^{(m-1)}}\right]$$

При отображении активных вершин дерева необходимо учесть, что все модули, кроме последнего, передают входное событие следующему модулю. Стоит отметить, что при этом реакция модулей на входные события не изменяется — входные события так же играют свою роль в охранных условиях на переходах автомата. Также необходимо учесть, что значения выходных переменных ограничиваются для соответствия дереву сценариев только для последнего модуля — все предыдущие модули могут производить произвольные операции над выходными переменными в рамках существующего сведения.

$$\bigwedge_{v\in V^{(\text{active})}} \bigwedge_{m\in[1..(M-1)]} \bigwedge_{q\in Q} \left[(\mu_v^{(m)} = q) \to \left((\lambda^{(m)}_{q',e,u} = q) \wedge (\varphi_q^{(m)} = e)\right)\right],$$

$$\bigwedge_{v\in V^{(\text{active})}} \bigwedge_{q\in Q} \left[(\mu_v^{(M)} = q) \to \left((\lambda^{(M)}_{q',e,u} = q) \wedge (\varphi_q^{(M)} = o) \wedge \bigwedge_{z\in\mathcal{Z}} (\nabla_{v,z}^{(M)} = tov(v,z))\right)\right],$$

где $p = tp(v)$, $q' = \mu_p^{(m)}$, $e = tie(v)$, $u = tin(v)$.

При отображения пассивных вершин достаточно закодировать тот факт, что все модули игнорируют входное воздействие — автоматы не изменяют своего состояния:

$$\bigwedge_{v\in V^{(\text{passive})}} \bigwedge_{m\in[1..M]} (\lambda^{(m)}_{q',e,u} = q_0),$$

где $p = tp(v)$, $q' = \mu_p^{(m)}$, $e = tie(v)$, $u = tin(v)$.

2.10.3. Алгоритмы Consecutive-Basic и Consecutive-Extended

Аналогично разделу 2.9.3, обозначим Consecutive-Basic$^*(\mathcal{S}^{(+)}, M, C, T)$ алгоритм, реализующий синтез модульной модели с *последовательной* композицией модулей. Также введём обозначение для случая, когда число переходов в автомате остается неограниченным:

$$\begin{aligned}&\text{Consecutive-Basic}(\mathcal{S}^{(+)}, M, C) = \\ &= \text{Consecutive-Basic}^*(\mathcal{S}^{(+)}, M, C, T = \infty)\end{aligned}$$

В соответствии с разделом 2.9.3, алгоритм для синтеза расширенных моделей объявляется без пояснений:

$$\begin{aligned}&\text{Consecutive-Extended}(\mathcal{S}^{(+)}, M, C, P) = \\ &= \text{Consecutive-Extended}^*(\mathcal{S}^{(+)}, M, C, P, N = \infty)\end{aligned}$$

2.11. Метод синтеза модульных конечно-автоматных моделей с произвольной композицией модулей по примерам поведения

Наиболее общей формулировкой задачи синтеза конечно-автоматных моделей логических контроллеров является модульный синтез с произвольной композицией модулей. Каждый модуль рассматривается как черный ящик с набором входных и выходных контактов для входных/выходных событий/переменных. Содержимое черного ящика — монолитный контроллер, структура которого кодируется так же, как при монолитном синтезе. Совокупность модулей является внутренним содержимым модульного контроллера, так же обладающего входными и выходными *контактами*. Контакты соединяются *связями*, что обеспечивает взаимодействие модулей друг с другом. При такой постановке задачи, разрабатываемый метод обладает максимальными возможностями для синтеза комплексных модульных структур, и является свободным от ограничений, присущих рассмотренным ранее параллельной и последовательной композициям. Стоит отметить, что для корректных вычислений выходных событий и значений выходных переменных необходимо знать порядок вычисления модулей, что возможно только если граф связей модулей является ацикличным. Данное ограничение не уменьшает вычислительной способности модульного контроллера, поэтому в данной работе предполагается, что граф связей модулей является ацикличным, а также то, что модули упорядочены в порядке их номеров: от 1 до M.

2.11.1. Сведение к SAT: переменные

Рассмотрим подробнее, как устроено сведение для модульного синтеза с произвольной композицией. Обозначим $\Pi^{(m)}_{in\ var} = \{(m-1)\cdot|\mathcal{X}|+1,\ldots,m\cdot|\mathcal{X}|\}$ множество входных контактов для переменных, считая, что контакты нумеруются натуральными числами, начиная с единицы, а нумерация является сквозной среди всех модулей $m \in [1\,..\,M]$ для всех контактов одного типа. Аналогично, обозначим $\Pi^{(m)}_{out\ var}$ множество выходных контактов для переменных, а также обозначим множества входных и выходных контактов для событий $\Pi^{(m)}_{in\ ev}$ и $\Pi^{(m)}_{out\ ev}$. Дополнительно, обозначим множества контактов для внешнего (модульного) автомата следующим образом: $\Pi^{(ext)}_{in\ var}$, $\Pi^{(ext)}_{out\ var}$, $\Pi^{(ext)}_{in\ ev}$, $\Pi^{(ext)}_{out\ ev}$. Отметим, что, например, множество $\Pi^{(ext)}_{in\ var}$ внешних входных контактов для переменных соответствует выходным переменным $\mathcal{Z}$, так как эти контакты являются входами для входящих в них связей из выходных контактов. При этом нумерация внешних контактов продолжает нумерацию модульных того же типа, например, $\Pi^{(ext)}_{in\ var} = \{M\cdot|\mathcal{X}|+1,\ldots,M\cdot|\mathcal{X}|+|\mathcal{Z}|\}$. Для упрощения используемой нотации обозначим множества *всех* входных/выходных контактов для переменных/событий $\Pi_{in\ var}$, $\Pi_{out\ var}$, $\Pi_{in\ ev}$, $\Pi_{out\ ev}$, где, например:

$$\Pi_{in\ var} = \Pi^{(ext)}_{in\ var} \cup \bigcup_{m\in[1..M]} \Pi^{(m)}_{in\ var}$$

Следующий шаг — определение графа соединений контактов модулей. Введём переменные $\pi^{var}_p \in \Pi_{out\ var}$ и $\pi^{ev}_p \in \Pi_{out\ ev}$, обозначающие родителя входного контакта p для переменных/событий. Можно заметить, что родителем входного контакта может быть только выходной контакт. Если контакт является родителем другого, то они соединены *связью*, и обновление значения родителя приводит к обновлению значения ребенка. Пользуясь допущением об ацикличности графа соединений модулей, определим, что родителем входного контакта p может быть выходной контакт из модуля с меньшим номером, либо внешний выходной контакт, например, для некоторого контакта $p \in \Pi_{in\ var}$ из модуля m:

$$\pi^{var}_p \in \Pi^{(ext)}_{out\ var} \cup \bigcup_{m'<m} \Pi^{(m')}_{out\ var}$$

Для каждого контакта необходимо кодировать его вычисленное значение в каждый момент времени в процессе отображения вершин дерева сценариев на состояния модулей. Для этого вводятся переменные $\nabla^{in}_{v,p'}$ и $\nabla^{out}_{v,p''}$, кодирующие значения входных и выходных контактов $p' \in \Pi_{in\ var}$ и $p'' \in \Pi_{out\ var}$ в момент времени после обработки вершины v дерева сценариев.

Важно отметить, что в отличие от рассмотренных ранее методов, при произвольной композиции каждый модуль может получать на вход произвольные наборы значений входных переменных. Всего существует $2^{|\mathcal{X}|}$ таких наборов, и каждый из них должен быть корректно обработан каждым модулем. Поэтому при кодировании структуры автомата необходимо учитывать множество всех возможных различных входов $\mathcal{U}^* = \mathbb{B}^{|\mathcal{X}|} = \{\langle 0 \ldots 0\rangle, \ldots, \langle 1 \ldots 1\rangle\}$, а не только множество входов $\mathcal{U} \subseteq \mathcal{U}^*$, встречающихся в дереве сценариев. Введём переменную $\partial_v^{(m)} \in \mathcal{U}^*$, обозначающую вход в момент времени после обработки вершины v дерева сценариев. Заметим, что вычисленные значения $\nabla_{v,p}^{in}$ входных контактов являются битами в бинарной записи номера входа $u \in \mathcal{U}^*$, учитывая, что входы нумеруются с нуля. Для взаимосвязи этих двух переменных была использована техника кодирования целочисленных SAT переменных OneHot+Binary [105]: числовое значение переменной $\partial_v^{(m)}$ соответствует номеру входа $u \in \mathcal{U}^*$, а биты соответствующего двоичного числа — значениям переменной $\nabla_{v,p}^{in}$.

2.11.2. Сведение к SAT: ограничения

Для определения переменных $\nabla_{v,p}^{in}$ и $\nabla_{v,p}^{out}$ объявляются следующие ограничения:

1. Взаимосвязь родителя контакта с его потомком:
$$\bigwedge_{v \in V} \bigwedge_{p \in \Pi_{in\ var}} \left[\nabla_{v,p}^{out} \leftrightarrow \nabla_{v,\pi_p^{var}}^{in}\right]$$
2. Начальные значения всех выходных контактов равны `False`:
$$\bigwedge_{p \in \Pi_{out\ var}} (\nabla_{1,p}^{out} = \bot)$$
3. Значения входных переменных, записанные в вершинах дерева (кроме корня):
$$\bigwedge_{v \in V \setminus \{\rho\}} \bigwedge_{p \in \Pi_{out\ var}^{(ext)}} (\nabla_{v,p}^{out} = tiv(v, pinToInputVar(p))),$$
где $pinToInputVar\colon \Pi_{out\ var}^{(ext)} \to \mathcal{X}$ — функция, возвращающая входную переменную $x \in \mathcal{X}$, соответствующую внешнему выходному контакту $p \in \Pi_{out\ var}^{(ext)}$.
4. Значения входных контактов, у которых нет родителей, принимаются равными `False`:
$$\bigwedge_{p \in \Pi_{in\ var}} \left[(\pi_p^{var} = 0) \to \bigwedge_{v \in V} (\nabla_{v,p}^{in} = \bot)\right]$$

При отображении активных вершин дерева сценариев на состояния модулей необходимо учитывать новую переменную $\partial^{(m)}$:

$$\bigwedge_{m\in[1..M]}\bigwedge_{v\in V}\left[(\mu_v^{(m)}=q)\rightarrow(\lambda_{q',u}^{(m)}=q)\right],$$

где $p = tp(v)$, $q' = \mu_p^{(m)}$, $u = \partial_v^{(m)}$.

Вычисляемые значения выходных контактов модулей зависят от своих предыдущих значений и определяются с помощью алгоритмов в состояниях модулей:

$$\bigwedge_{m\in[1..M]}\bigwedge_{v\in V}\bigwedge_{p\in\Pi_{out\ var}}\bigwedge_{z\in\mathcal{Z}}\left[\nabla_{v,p}^{out}\leftrightarrow ITE(\nabla_{v',p}^{out},\gamma_{q,z,\top}^{(m)},\gamma_{q,z,\bot}^{(m)})\right],$$

где $v' = tp(v)$, $q = \mu_v^{(m)}$. Также необходимо ограничить значения внешних входных контактов (напомним, что эти контакты соответствуют выходным переменным внешнего модульного автомата) в соответствии со значениями в дереве сценариев:

$$\bigwedge_{v\in V}\bigwedge_{p\in\Pi_{in\ var}^{(ext)}}\left[\nabla_{v,p}^{in}=tov(v,pinToOutputVar(p))\right],$$

где $pinToOutputVar(p)\colon \Pi_{in\ var}^{(ext)}\rightarrow\mathcal{Z}$ — функция, возвращающая выходную переменную $z\in\mathcal{Z}$, соответствующую внешнему входному контакту $p\in\Pi_{in\ var}^{(ext)}$.

При отображении пассивных вершин достаточно закодировать тот факт, что все модули игнорируют входное действие, записанное в вершине дерева сценариев. При этом необходимо учесть нововведенную переменную $\partial^{(m)}$:

$$\bigwedge_{m\in[1..M]}\bigwedge_{v\in V^{(\text{passive})}}\left[(\mu_v^{(m)}=\mu_{tp(v)}^{(m)})\wedge(\lambda_{q',u}^{(m)}=0)\right],$$

где $q' = \mu_{tp(v)}^{(m)}$, $u = \partial_v^{(m)}$. Также, значения выходных контактов не изменяются, так как модули не генерируют выходных событий:

$$\bigwedge_{v\in V^{(\text{passive})}}\bigwedge_{p\in\Pi_{out\ var}}(\nabla_{v,p}^{out}=\nabla_{tp(v),p}^{out})$$

2.11.3. Алгоритмы Arbitrary-Basic и Arbitrary-Extended

Аналогично разделу 2.9.3, обозначим Arbitrary-Basic$^*(\mathcal{S}^{(+)},M,C,T)$ алгоритм, реализующий синтез модульной модели с *произвольной* композицией модулей. Также введём обозначение для случая, когда число переходов в автомате остается

неограниченным:

$$\textsc{Arbitrary-Basic}(\mathcal{S}^{(+)}, M, C) = \\ = \textsc{Arbitrary-Basic}^{*}(\mathcal{S}^{(+)}, M, C, T = \infty)$$

В соответствии с разделом 2.9.3, алгоритм для синтеза расширенных моделей объявляется без пояснений:

$$\textsc{Arbitrary-Extended}(\mathcal{S}^{(+)}, M, C, P) = \\ = \textsc{Arbitrary-Extended}^{*}(\mathcal{S}^{(+)}, M, C, P, N = \infty)$$

2.12. Синтез минимальных модульных конечно-автоматных моделей

Для синтеза *минимальных* модульных моделей в данной работе используется тот же итеративный подход, что и при монолитном синтезе (раздел 2.4).

Алгоритм $\textsc{Parallel-Basic-min}(\mathcal{S}^{(+)}, M)$ позволяет синтезировать базовую конечно-автоматную модель композитного функционального блока с параллельной композицией заданного числа модулей M с минимальным числом состояний и переходов. При минимизации предполагается, что все модули имеют одинаковое число состояний, а число переходов считается суммарно во всех модулях. Возможно также рассмотрение такой постановки задачи, когда размеры каждого модуля заданы явно (или в виде некоторых интервальных оценок), однако в общем случае эти значения абсолютно неизвестны заранее. В результате минимальная модульная модель содержит $M \cdot C_{\min}$ состояний (по $C_{\min}$ в каждом модуле) и $T_{\min} = T_{\min}^{(1)} + \cdots + T_{\min}^{(M)}$ переходов. Здесь стоит отметить, что используемая нотация $T_{\min}^{(m)}$ обозначает *возможно неминимальное* число переходов в модуле $m \in [1 \mathinner{..} M]$, входящего в состав *минимальной* модульной модели с *минимальным* суммарным числом переходов $T_{\min}$.

Аналогично, алгоритм $\textsc{Parallel-Extended-min}(\mathcal{S}^{(+)}, M, P)$ позволяет синтезировать расширенную модульную модель с параллельной композицией M модулей с минимальным числом состояний $M \cdot C_{\min}$, минимальным суммарным размером охранных условий $N_{\min}$ и максимальным размером каждого охранного условия P.

Аналогичным образом определяются алгоритмы для синтеза базовых и расширенных модульных моделей с последовательной и произвольной композиций модулей:

- $\textsc{Consecutive-Basic-min}(\mathcal{S}^{(+)}, M)$: базовая/последовательная;
- $\textsc{Consecutive-Extended-min}(\mathcal{S}^{(+)}, M, P)$: расширенная/последовательная;
- $\textsc{Arbitrary-Basic-min}(\mathcal{S}^{(+)}, M)$: базовая/произвольная;

– Arbitrary-Extended-min($\mathcal{S}^{(+)}, M, P$): расширенная/произвольная.

2.13. От монолитного к распределённому синтезу

Предлагаемое расширение позволяет осуществлять синтез моделей распределённых систем и основывается на двух идеях. Первая идея заключается в том, чтобы для каждого модуля в распределённой системе независимо кодировать задачу монолитного синтеза в SAT. Это описано в разделе 2.13.1. Вторая идея связана с адаптацией подхода CEGIS, описанного и использованного в разделе 2.5, с целью сделать его применимым для распределённого синтеза.

Сложность использования CEGIS для распределённого синтеза заключается в том, что поскольку спецификация написана для системы *в целом*, негативные сценарии, полученные из контрпримеров, также описывают (негативное) поведение системы в целом. Раздел 2.13.2 описывает структуру этих сценариев. Проблема заключается в том, что заранее неизвестно, какие именно модули ведут себя неправильно в конкретном негативном сценарии, и, следовательно, неизвестно, в каком модуле необходимо запретить данный негативный сценарий.

Наивный подход заключался бы в том, чтобы запретить каждый негативный сценарий для всех модулей, но это было бы некорректно: поведение, которое является неправильным для одного из модулей, может быть совершенно нормальным для другого. Таким образом, этот наивный подход привёл бы к невыполнимости соответствующих булевых формул, и искомая распределённая система не была бы синтезирована.

Ключевая идея для преодоления этой проблемы заключается в том, чтобы убедиться, что каждый негативный сценарий запрещён хотя бы для одного модуля. Это описано в разделе 2.13.3. Наконец, в разделе 2.13.4 обсуждается алгоритм синтеза минимальных автоматов с точки зрения их параметров, например, количества состояний и суммарного размера охранных условий.

2.13.1. Сведение к SAT для распределённого синтеза по примерам поведения

В данном подразделе представлено сведение к SAT, основа которого уже была описана в разделе **??**. Для каждого модуля независимо объявляется описанный набор переменных и ограничений — ко всем переменным в сведении добавляется индекс модуля. Таким образом, соответствующие ограничения остаются теми же, но теперь они также квантифицированы по всем модулям. Например, переменная $\tau_{q,k}^{(m)} \in Q_0$

обозначает конечное состояние k-го перехода из состояния q в автомате для m-го модуля, а переменная $\xi_{q,k}^{(m)} \in E^I \cup \{\varepsilon\}$ обозначает входное событие, связанное с этим переходом. Ограничение

$$(\tau_{q,k}^{(m)} = q_0) \leftrightarrow (\xi_{q,k}^{(m)} = \varepsilon)$$

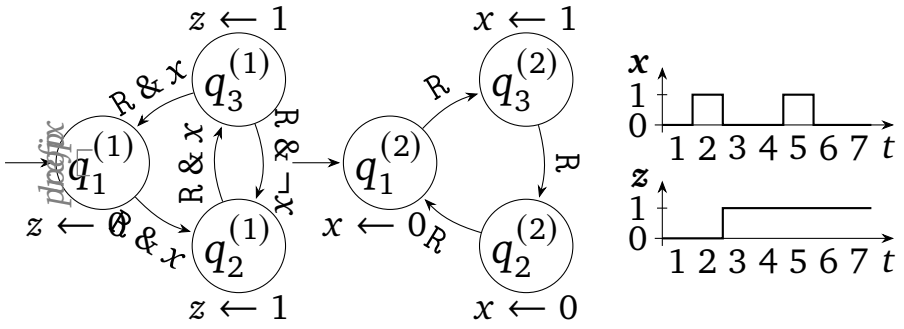
обеспечивает, что только переходы в q_0 отмечены входным событием ε. Мы также добавляем индексы модулей к параметрам редукции. Например, $C^{(m)}$ обозначает количество состояний в $\mathcal{A}^{(m)}$ — автомате для m-го модуля. Все остальные части кодировки из раздела 2.3 расширяются аналогичным образом, за исключением негативного отображения, которое будет разобрано ниже.

Различие с позитивными сценариями заключается в том, что теперь у нас есть *набор* деревьев позитивных сценариев $\{\mathcal{T}^{(+)(m)}\}_{m\in[1..M]}$, по одному для каждого модуля распределённой системы. Переменная $\mu_v^{(m)} \in Q^{(m)}$ обозначает состояние, в котором автомат для m-го модуля завершает обработку последовательности входных данных, образованной следованием по позитивному дереву от корня до вершины v (поскольку дерево детерминированное, для каждой вершины v существует только один путь от корня). Эта переменная представляет отображение $\mu^{(m)} : V^{(m)} \longrightarrow Q^{(m)}$ между вершинами дерева $\mathcal{T}^{(+)(m)}$ и состояниями автомата $\mathcal{A}^{(m)}$, а ограничения на это отображение аналогичны описанным в разделе 2.3.3.

2.13.2. Составное негативное дерево сценариев

Составной негативный сценарий представляет собой последовательность *составных элементов сценария*. Каждый составной элемент c_i является M-кортежем элементов сценария для каждого синтезированного модуля: $c_i = (s_i^{(m)})_{m\in[1..M]}$, где каждый $s_i^{(m)} = \langle i[\bar{x}], o[\bar{z}]\rangle$.

Рассмотрим систему из двух взаимосвязанных модулей, изображённую на Рисунке 10. Граф и таблица на Рисунке 10 демонстрируют последовательность состояний выполнения этой системы, иллюстрируя нарушение LTL-свойства **GF**$\neg z$. Составной негативный сценарий, полученный из этой последовательности выполнения, показан



t	$q^{(1)}$	$q^{(2)}$	x	z
1	q_1	q_1	0	0
2	q_1	q_3	1	0
3	q_2	q_2	0	1
4	q_2	q_1	0	1
5	q_2	q_3	1	1
6	q_3	q_2	0	1
7	q_2	q_1	0	1

Рисунок 10 — Циклическая распределенная система

ниже (начало цикла отмечено подчеркиванием).

$$
\begin{aligned}
&[(\langle \mathtt{R[0]}, \varepsilon\mathtt{[0]}\rangle, \langle \mathtt{R}, \mathtt{C[1]}\rangle);\\
&\underline{(\langle \mathtt{R[1]}, \mathtt{C[1]}\rangle, \langle \mathtt{R}, \mathtt{C[0]}\rangle);}\\
&(\langle \mathtt{R[0]}, \varepsilon\mathtt{[1]}\rangle, \langle \mathtt{R}, \mathtt{C[0]}\rangle);\\
&(\langle \mathtt{R[0]}, \varepsilon\mathtt{[1]}\rangle, \langle \mathtt{R}, \mathtt{C[1]}\rangle);\\
&(\langle \mathtt{R[1]}, \mathtt{C[1]}\rangle, \langle \mathtt{R}, \mathtt{C[0]}\rangle);\\
&(\langle \mathtt{R[0]}, \mathtt{C[1]}\rangle, \langle \mathtt{R}, \mathtt{C[0]}\rangle)]
\end{aligned}
$$

Одним из важных отличий от монолитного случая является то, что в распределённом синтезе предполагается, что входное событие может быть пустым: $i \in E^I \cup \{\varepsilon\}$. Это соответствует случаю, когда один из модулей не получает никаких входных данных в какой-то момент времени. Также предполагается, что в допустимых сценариях $i = \varepsilon$ соответствует $o = \varepsilon$ — пока модуль не получает входных событий, он остаётся в состоянии простоя и ничего не продуцирует. Следует отметить, что в негативных сценариях, полученных из контрпримеров модели, может случиться так, что все синтезированные модули не получают входного события. Это происходит когда все модули находятся в состоянии простоя, ожидая входные события из окружающей среды. Такие элементы фильтруются, так как они не содержат полезной информации для синтеза.

Составное негативное дерево сценариев $\mathcal{CT}^{(-)}$ (далее просто *составное дерево*) — это негативное дерево сценариев, построенное из набора составных негативных сценариев аналогичным образом, как описано в Разделе 1.5. Каждая вершина дерева $\mathcal{CT}^{(-)}$ и её входное ребро соответствуют *составному элементу сценария*. Множество вершин дерева $\mathcal{CT}^{(-)}$ обозначается как $\widehat{\mathcal{CV}}$, и для составного дерева

повторно используются вспомогательные функции негативного дерева, такие как $\widehat{tp}(\widehat{v})$, $\widehat{tie}(\widehat{v})$ и другие.

Проекция составного элемента сценария на модуль m — это просто его m-й элемент. *Проекция* $\mathcal{T}^{(-)(m)}$ составного дерева $\mathcal{CT}^{(-)}$ на модуль m представляет собой дерево, изоморфное $\mathcal{CT}^{(-)}$, каждая вершина которого является проекцией элемента сценария в соответствующей вершине дерева $\mathcal{CT}^{(-)}$ на модуль m.

2.13.3. Отображение составного негативного дерева сценариев

Переменная $\widehat{\mu}_{\widehat{v}}^{(m)}$ обозначает удовлетворяющее состояние в автомате $\mathcal{A}^{(m)}$ для вершины $\widehat{v} \in \widehat{\mathcal{CV}}$, при этом $\widehat{\mu}_{\widehat{v}}^{(m)} = q_0$ в то случае, когда автомат не проявляет данного поведения, и, следовательно, не существует удовлетворяющего состояния для $\widehat{v}$ («фиктивное» состояние q_0 является общим для всех модулей). Для каждого модуля корневая вершина составного негативного дерева отображается на начальное состояние соответствующего автомата:

$$\widehat{\mu}_{\widehat{\rho}}^{(m)} = q_{\text{init}}^{(m)}$$

Проекция составного дерева может содержать вершины с $i = \varepsilon$ на входящем ребре — такие вершины называются *пустыми*. Среди непустых вершин различают пассивные ($\widehat{toe}(\widehat{v}) = \varepsilon$) и активные вершины ($\widehat{toe}(\widehat{v}) \neq \varepsilon$) с такими же свойствами, как и для обычных негативных деревьев. Множества пустых, пассивных и активных вершин составного негативного дерева обозначаются как $\widehat{\mathcal{CV}}_{(\text{idle})}$, $\widehat{\mathcal{CV}}_{(\text{pass})}$ и $\widehat{\mathcal{CV}}_{(\text{act})}$ соответственно.

Ограничения для пассивных и активных вершин такие же, как в сведении для монолитного синтеза, описанного в разделе **??**. Каждая *пустая* вершина $\widehat{v} \in \widehat{\mathcal{CV}}_{(\text{idle})}$ отображается на то же состояние, что и её родительская вершина:

$$(\widehat{\mu}_{\widehat{v}}^{(m)} = \widehat{\mu}_{\widehat{tp}(\widehat{v})}^{(m)}),$$

Единственное отличие пустых вершин от пассивных вершин заключается в том, что не требуется явно кодировать, что автомат игнорирует входное действие, так как фактически он его не получает.

Аналогично монолитному случаю, необходимо обеспечить, чтобы если вершина $\widehat{v} \in \widehat{\mathcal{CV}}$ не отображается (отображается в фиктивное состояние q_0), то это свойство передаётся вниз по дереву:

$$(\widehat{\mu}_{\widehat{tp}(\widehat{v})}^{(m)} = q_0) \rightarrow (\widehat{\mu}_{\widehat{v}}^{(m)} = q_0)$$

Наконец, требуется запретить циклическое поведение, описанное в контрпримерах, но таким образом, чтобы каждый цикл был запрещён хотя бы в одном модуле. Для этого добавляется дизъюнкция по $m \in [1 \,..\, M]$ для каждого цикла в $\mathcal{CT}^{(-)}$:

$$\bigwedge_{\widehat{v'} \in \widehat{tbe}(\widehat{v})} \bigvee_{m \in [1..M]} \left[(\widehat{\mu}_{\widehat{v}}^{(m)} \neq \widehat{\mu}_{\widehat{v'}}^{(m)}) \vee (\widehat{\mu}_{\widehat{v}}^{(m)} = \widehat{\mu}_{\widehat{v'}}^{(m)} = q_0) \right]$$

Таким образом, на каждой итерации CEGIS в SAT-решатель добавляются ограничения на то, что данный негативный сценарий запрещён хотя бы для одного модуля. По сути, этот подход представляет собой неявный поиск, делегированный SAT-решателю.

2.13.4. Нахождение минимального распределённого контроллера

Конечной целью является нахождение *минимального* распределённого контроллера, соответствующего заданной спецификации $\mathcal{L}$. Минимальные модели имеют большую практическую ценность — они гораздо более понятны для человека и более эффективны в аппаратной реализации. Однако, если начать минимизацию (*e.g.*, путём добавления дополнительных ограничений) решения — набора модулей $\{\mathcal{A}^{(m)}\}$ — найденного с помощью CEGIS, то полученное минимальное решение может (и, на практике, будет) не соответствовать спецификации $\mathcal{L}$, хотя уже полученные негативные сценарии не будут удовлетворены, как и ожидалось. Поэтому шаг минимизации должен быть включён в цикл CEGIS. Более того, генерация минимальных моделей на каждой итерации CEGIS позволяет выполнять проверку модели гораздо быстрее, поскольку меньшие модели имеют меньшую пространство состояний. В дальнейшем метод CEGIS с включённой минимизацией будет называться CEGIS-min.

Для обеспечения минимизации синтезированных автоматов по общему числу состояний выполняются следующие расширения редукции к SAT. Во-первых, параметры $C^{(m)}$ — максимальное число состояний в каждом модуле $m \in [1 \,..\, M]$ — должны быть известны до начала процесса вывода, и решение (если найдено) будет иметь точно заданный размер (число состояний). Самая простая стратегия для нахождения удовлетворяющего решения с минимальным общим числом состояний будет итеративной стратегией — линейный восходящий поиск. Однако, в случае $M > 1$ (любого не монолитного синтеза) неясно, как выполнять итерацию параметров — эффективно, одновременно сохраняя минимальность — и это само по себе становится проблемой. Одним из возможных способов является итерация всех $C^{(m)}$ вместе до нахождения какого-либо решения (все «слишком малые» значения приводят к

UNSAT), а затем попытка уменьшить некоторые из $C^{(m)}$. Однако, последнее — попытки синтеза с различными значениями $C^{(m)}$ — требует *перезапусков* SAT-решателя, поскольку объявленные переменные (с индексом $q \in Q^{(m)}$) и ограничения зависят от $C^{(m)}$ *статически* — этот параметр не может быть легко изменён с сохранением выполнимости формулы. Таким образом, требуется *динамический* способ постепенного «отключения» некоторых состояний в модулях. Это можно сделать, используя подход «используемых состояний», предложенный в [119]. Основная идея заключается в том, чтобы отметить каждое состояние как *используемое* или *неиспользуемое*, что позволяет минимизировать общее количество *используемых* состояний, кодируя это с помощью техники *totalizer* [91].

Переменная $\alpha_q \in \mathbb{B}$ обозначает, является ли состояние q *используемым* автоматом — достижимо ли оно из начального состояния q_{init}. Начальное состояние всегда является *используемым*. (Не начальное) состояние является *используемым* тогда и только тогда, когда у него есть входной переход:

$$\alpha_{q_{\text{init}}} \wedge \bigwedge_{q \in Q \setminus \{q_{\text{init}}\}} \left[\alpha_q \leftrightarrow \bigvee_{\substack{q' \in Q \\ k \in [1..K]}} (\tau_{q',k} = q) \right]$$

Дополнительно, следующее ограничение обеспечивает, что неиспользуемые состояния имеют наибольшие индексы:

$$\neg \alpha_{q_i} \rightarrow \neg \alpha_{q_{i+1}}$$

В контексте распределённого синтеза эта переменная объявляется для каждого модуля $m \in [1 \,..\, M]$: $\alpha_q^{(m)}$. Общее число используемых состояний во всех модулях обозначается C^{used} и кодируется с помощью *totalizer* [91].

Описанные ранее ограничения нарушения симметрии (разделы 2.3.2 и 2.3.7) требуют, чтобы состояния автомата были пронумерованы в порядке обхода BFS — были включены в дерево обхода BFS. Это подразумевает, что, согласно определению переменной π_q^{BFS}, все состояния (кроме начального) имеют входной переход — родителя состояния q в дереве обхода BFS. Следовательно, это подразумевает, что все состояния *используются*, по определению. Для того чтобы уменьшить пространство поиска, используя ограничения BFS, и позволить состояниям быть *неиспользуемыми*, BFS-предикаты должны быть изменены следующим образом. Модифицированная переменная $\pi_{q_j}^{\text{BFS-mod}} \in \{q_0, q_1, \ldots, q_{j-1}\}$ ($j \in [2..C]$) включает q_0 в своё множество значений, что означает отсутствие родителя для состояния $q_j \in Q$ в дереве обхода

BFS. Только *неиспользуемые* состояния не имеют родителя:

$$\neg\alpha_{q_j} \leftrightarrow (\pi_{q_j}^{\text{BFS-mod}} = q_0)$$

Переменная $\tau_{q_i,q_j}^{\text{BFS}} \in \mathbb{B}$ и все остальные ограничения — определения переменных τ^{BFS} и $\pi^{\text{BFS-mod}}$, а также самого BFS-предиката — не требуют дополнительных модификаций.

Основная процедура, реализующая алгоритм CEGIS-MIN, представлена на Листинге 2. Сначала оценивается параметр D: *общее* верхнее значение для числа состояний в каждом модуле. Оценка выполняется с помощью простого линейного восходящего поиска. На каждой итерации вызывается функция `infer`$(***)$, где $***$ обозначает общие аргументы $(\{\mathcal{S}^{(+)(m)}\}, \mathcal{CS}^{(-)}, \{C^{(m)} = D\}, P)$. Процедура `infer` отвечает за:

1. Построение дерева позитивных сценариев $\{\mathcal{S}^{(+)(m)}\}$ и дерева составного негативного сценария для $\mathcal{CS}^{(-)}$.
2. Сведение задачи синтеза к SAT путём объявления ограничений, описанных в данной работе.
3. Делегирование полученной формулы SAT-решателю для непосредственного решения задачи SAT.
4. Восстановление ответа — построение набора автоматов (модулей) $\{\mathcal{A}^{(m)}\}$, удовлетворяющих заданным сценариям (точнее, удовлетворяющих позитивным сценариям и неудовлетворяющих негативным), из удовлетворяющего присваивания, найденного SAT-решателем.

Далее начинается фактический цикл CEGIS. Сначала минимизируется C^{used} с использованием нисходящего подхода. Здесь вызывается `infer` с дополнительным аргументом-предикатом «$C^{\text{used}} < \texttt{UB}$», где `UB` обозначает верхнюю границу для общего числа использованных (достижимых) состояний во всех синтезированных автоматах. Этот предикат представляет собой *ограничение на кардинальность*, которое кодируется с использованием техники *totalizer* [91], описанной в разделе 1.13. Затем минимизируется N^{used} с использованием того же подхода. После этого проводится проверка модели синтезированного распределённого контроллера, представленного автоматами $\{\mathcal{A}^{(m)}\}$, на соответствие заданной LTL-спецификации $\mathcal{L}$ с использованием проверщика моделей NuSMV [19], и получается набор составных негативных сценариев $\mathcal{CS}_{\text{new}}^{(-)}$ — если такие есть, они будут учтены на следующей итерации, иначе CEGIS завершается, и $\{\mathcal{A}^{(m)}\}$ является искомым решением —

Алгоритм 2: Distributed-CEGIS-min($\{\mathcal{S}^{(+)(m)}\}$, $\mathcal{L}$, P)

Вход: множество позитивных сценариев $\{\mathcal{S}^{(+)(m)}\}_{m\in[1..M]}$,
LTL-спецификация $\mathcal{L}$, максимальный размер дерева разбора P.

Результат: модульный автомат $\{\mathcal{A}^{(m)}\}_{m\in[1..M]}$ удовлетворяющий
LTL-спецификации $\mathcal{L}$.

```
// Compound negative scenarios
```
1 $\mathcal{CS}^{(-)} \leftarrow \varnothing$

```
// Common arguments (for «infer»)
```
2 $*** := (\{\mathcal{S}^{(+)(m)}\}, \mathcal{CS}^{(-)}, \{C^{(m)} = D\}, P)$

3 **label** *start*

```
// Estimate D
```
4 **for** $D \leftarrow 1$ **to** ∞ **do**
5 $\quad \{\mathcal{A}^{(m)}\} \leftarrow$ `infer`$(***)$
6 $\quad$ **if** $\{\mathcal{A}^{(m)}\} \neq$ `null` **then break**

7 **while** `true` **do**
```
    // Minimize C^used
```
8 $\quad \{\mathcal{A}^{(m)}\} \leftarrow$ `infer`$(***, C^{\text{used}} < \infty)$
9 $\quad$ **if** $\{\mathcal{A}^{(m)}\} =$ `null` **then**
10 $\quad\quad$ **goto** *start*
11 $\quad$ **while** $\{\mathcal{A}^{(m)}\} \neq$ `null` **do**
12 $\quad\quad C^{\text{used}}_{\text{min}} \leftarrow$ `numberOfUsedStates`$(\{\mathcal{A}^{(m)}\})$
13 $\quad\quad \{\mathcal{A}^{(m)}\} \leftarrow$ `infer`$(***, C^{\text{used}} < C^{\text{used}}_{\text{min}})$

```
    // Minimize N^used
```
14 $\quad \{\mathcal{A}^{(m)}\} \leftarrow$ `infer`$(***, C^{\text{used}} = C^{\text{used}}_{\text{min}}, N^{\text{used}} < \infty)$
15 $\quad$ **while** $\{\mathcal{A}^{(m)}\} \neq$ `null` **do**
16 $\quad\quad N^{\text{used}}_{\text{min}} \leftarrow$ `numberOfParseTreeNodes`$(\{\mathcal{A}^{(m)}\})$
17 $\quad\quad \{\mathcal{A}^{(m)}\} \leftarrow$ `infer`$(***, C^{\text{used}} = C^{\text{used}}_{\text{min}}, N^{\text{used}} < N^{\text{used}}_{\text{min}})$

18 $\quad \{\mathcal{A}^{(m)}\} \leftarrow$ `infer`$(***, C^{\text{used}} = C^{\text{used}}_{\text{min}}, N^{\text{used}} = N^{\text{used}}_{\text{min}})$
19 $\quad \mathcal{CS}^{(-)}_{\text{new}} \leftarrow$ `performModelChecking`$(\{\mathcal{A}^{(m)}\}, \mathcal{L})$
20 $\quad$ **if** $\mathcal{CS}^{(-)}_{new} = \varnothing$ **then** `// No counterexamples`
21 $\quad\quad$ **return** $\{\mathcal{A}^{(m)}\}$

22 $\quad \mathcal{CS}^{(-)} \leftarrow \mathcal{CS}^{(-)} \cup \mathcal{CS}^{(-)}_{\text{new}}$

системой из M автоматов, удовлетворяющей заданным примерам поведения и LTL-спецификации.

2.14. Экспериментальное исследование: модульный синтез

В данном разделе приводится экспериментальное исследование, посвященное применению разработанных методов для синтеза модульных конечно-автоматных моделей логических контроллеров с различными композициями модулей, входящих в состав композитного функционального блока: (1) параллельной, (2) последовательной и (3) произвольной. В качестве примера была использована система PnP-манипулятора, уже рассмотренная ранее в разделе 2.7. Несмотря на то, что исходная модель контроллера PnP-манипулятора является монолитной, модульный синтез позволяет синтезировать распределенную систему, реализующую схожее поведение.

Стоит отметить, что приведенное экспериментальное исследование не включает в себя сравнение с существующими методами, так как двухэтапный метод «Two-stage» [49], использованный в главе **??** неприменим к задаче модульного синтеза, а среди доступных инструментов для задачи модульного синтеза можно выделить только EFSM-tools [34], однако данное решение обладает следующими недостатками: (1) низкая эффективность (по критериям времени работы и размерам моделей) на задаче монолитного синтеза, что подтверждено в [49], (2) модульное разбиение должно быть известно заранее, так как его необходимо указывать вручную, в то время как разработанные в данной работе методы позволяют получать модульное разбиение автоматически.

В первом эксперименте были использованы методы для синтеза *базовой* модульной модели, состоящей из двух модулей: $M = 2$. В качестве входных данных были использованы наборы сценариев $\mathcal{S}^{(1)}$, $\mathcal{S}^{(4)}$ и $\mathcal{S}^{(39)}$, где верхний индекс означает число сценариев в наборе. В экспериментальном сравнении были использованы алгоритмы Parallel-Basic-min, Consecutive-Basic-min и Arbitrary-Basic-min. Результаты первого эксперимента представлены в таблице 4, где $\mathcal{S}^{(+)}$ — набор сценариев выполнения, $|\mathcal{T}^{(+)}|$ — размер дерева сценариев, $C_{\min}$ — минимальное число состояний (одинаковые значения для каждого модуля), $T_{\min}$ — минимальное число переходов (указаны по-отдельности для каждого модуля: $T^{(1)}_{\min}+T^{(2)}_{\min}$), «Время, с» — время работы в секундах. Полученные результаты показывают, что методы синтеза минимальных модульных моделей с параллельной и последовательной композициями модулей обладают хорошей мастшабируемостью — время работы на большом

наборе сценариев $\mathcal{S}^{(39)}$ можно субъективно считать небольшим. Что же касается произвольной композиции — результаты показывают, что эта задача является непростой и плохо мастабируется с ростом размера сценариев. Низкую эффективность метода синтеза с произвольной композицией можно объяснить тем, что сведение (формула в КНФ для SAT-решателя) получается значительных размеров (для $\mathcal{S}^{(39)}$ $\approx$ 2.5 миллиона переменных, $\approx$ 45 миллионов дизъюнктов), что соответствует большому пространству поиска, перебираемому в процессе минимизации. В таких случаях обычно используются техники нарушения симметри (помимо уже рассмотренных в разделах 2.3.2 и 2.3.7 BFS-предикатов нарушения симметрии), однако в данной работе их исследование и разработка не проводились.

Таблица 4 — Результаты синтеза базовых модульных конечно-автоматных моделей логического контроллера PnP-манипулятора

$\mathcal{S}^{(+)}$	$\lvert\mathcal{T}^{(+)}\rvert$	Parallel-Basic-min			Consecutive-Basic-min			Arbitrary-Basic-min		
		$C_{\min}$	$T_{\min}$	Время, с	$C_{\min}$	$T_{\min}$	Время, с	$C_{\min}$	$T_{\min}$	Время, с
$\mathcal{S}^{(1)}$	24	4	4+6	1.4	4	4+5	1.3	4	6+4	69
$\mathcal{S}^{(4)}$	94	5	8+6	5.9	5	5+9	51	5	6+8	1081
$\mathcal{S}^{(39)}$	960	5	8+6	58	5	8+7	80	5	6+8	7055

Аналогичным образом было проведено экспериментальное исследование методов синтеза *расширенных* модульных моделей, охранные условия которых представлены в виде явных произвольных булевых формул. При синтезе были использованы заранее выбранные значения параметра P. В экспериментальном сравнении были использованы алгоритмы Parallel-Extended-min и Consecutive-Extended-min. Алгоритм Arbitrary-Extended-min не был рассмотрен, так как задача синтеза расширенной модели значительно сложнее синтеза базовой, а алгоритм Arbitrary-Basic-min показал крайне низкую эффективность в предыдущем эксперименте. Результаты второго эксперимента представлены в таблице 5, где $\mathcal{S}^{(+)}$ — набор сценариев выполнения, $\lvert\mathcal{T}^{(+)}\rvert$ — размер дерева сценариев, $C_{\min}$ — минимальное число состояний в каждом модуле (одинаковые значения для каждого модуля), P — максимальный размер охранного условия, T — число переходов (не было минимизировано) (указаны по-отдельности для каждого модуля: $T^{(1)}+T^{(2)}$), $N_{\min}$ — минимальный суммарный размер охранных условий (указаны по-отдельности для каждого модуля: $N^{(1)}_{\min}+N^{(2)}_{\min}$), «Время, с» — время работы в секундах.

Таблица 5 — Результаты синтеза расширенных модульных конечно-автоматных моделей логического контроллера PnP-манипулятора

$\mathcal{S}^{(+)}$	$\lvert\mathcal{T}^{(+)}\rvert$	Parallel-Extended-min					Consecutive-Extended-min				
		$C_{\min}$	P	T	$N_{\min}$	Время, с	$C_{\min}$	P	T	$N_{\min}$	Время, с
$\mathcal{S}^{(1)}$	24	4	3	6+7	12+13	4	4	3	6+7	10+11	6.5
$\mathcal{S}^{(4)}$	94	5	3	15+11	35+27	37	5	3	12+12	18+20	4043
$\mathcal{S}^{(39)}$	960	5	5	12+10	33+18	1130	—	5	—	—	>6ч

Анализируя полученные результаты, можно прийти к выводу, что разработанные методы синтеза расширенных модульных моделей работают корректно, однако в целом неэффективно. Можно заметить, что в случае параллельной и произвольной композиций основным результатом методов синтеза базовых моделей являются не сами модели, а модульное разбиение — соответствие переменных модулям, которые ими управляют. При переходе к синтезу расширенной модели уже полученное модульное разбиение может быть зафиксировано, что положительно скажется на скорости решения ввиду того, что пространство поиска будет меньше, однако финальная модель в общем случае уже не будет минимальной.

Выводы по главе 2

В данной главе была рассмотрена задача синтеза монолитных конечно-автоматных моделей логических контроллеров по примерам поведения и формальной спецификации. Для решения этой задачи были разработаны методы, основанные на сведении к задаче выполнимости SAT и применении SAT-решателей. Отдельное внимание было уделено решению задачи синтеза минимальных моделей. Работоспособность и эффективность разработанных методов были проверены в ходе экспериментального исследования, посвященному синтезу модели логического контроллера, управляющего Pick-and-Place манипулятором. Дополнительно, разработанные методы были применены для минимизации конечно-автоматных моделей, получаемых в ходе LTL-синтеза с помощью программного средства BoSy по исходным данным с соревнования по реактивному синтезу SYNTCOMP. Также в данной главе была рассмотрена и решена задача синтеза минимальных конечно-автоматных моделей модульных логических контроллеров с различными видами композиции

модулей: (1) параллельной, (2) последовательной и (3) произвольной. Разработанные методы были проверены в ходе экспериментального исследования, посвященного синтезу конечно-автоматной модели контроллера Pick-and-Place манипулятора. Все разработанные методы были реализованы в виде программного средства fbSAT [100].

Глава 3. Методы оценивания декомпозиционной трудности булевых формул в применении к задачам тестирования и верификации логических схем

В данной главе описывается общий подход к оценке трудности примеров SAT, связанных с булевыми схемами, относительно одного класса методов разбиения этих примеров на подформулы. Такого рода оценка трудности, фактически представляет собой некоторую верхнюю границу сложности рассматриваемой формулы, выраженной в некоторых единицах, которые можно использовать для измерения времени работы полного SAT-решателя. Также предлагаются две конструкции SAT-разбиений, которые показали хорошие результаты в вычислительных экспериментах.

3.1. Трудность относительно разбиения и вероятностный алгоритм её оценки

Легко видеть, что можно связать с определенным разбиением SAT Π и конкретным полным SAT-решателем A случайную величину ξ_Π так, что общее время работы A на всех SAT-экземплярах из разбиения Π может быть выражено через математическое ожидание ξ_Π, которое обозначается как $\mathbb{E}[\xi_\Pi]$. Этот факт дает теоретические основания для оценки трудности относительно разбиения с помощью простого алгоритма Монте-Карло.

Рассмотрим произвольную КНФ C над переменными X, и пусть A — полный SAT-решатель. Пусть $\Pi = \{G_1, \ldots, G_s\}$ — произвольное разбиение C. Можно рассматривать Π как пространство элементарных событий [62], где каждое G_i — элементарное событие. Далее, пусть p_i — положительные числа, связанные с каждым G_i таким образом, что $\sum_{i=1}^{s} p_i = 1$. Установим $p_i = 1/s$ для каждого $i \in \{1, \ldots, s\}$, задав таким образом равномерное распределение на Π. Затем определим случайную величину $\xi_\Pi \colon \Pi \to \mathbb{R}^+$ следующим образом:

$$\xi_\Pi(G \in \Pi) = t_A(G \wedge C), \tag{17}$$

где $t_A(G \wedge C)$ обозначает время, затраченное A на определение выполнимости формулы $G \wedge C$.

Определение 1. *Трудность C относительно алгоритма A и разбиения Π определяется как значение $\mu_{A,\Pi}(C)$:*

$$\mu_{A,\Pi}(C) = \sum_{i=1}^{s} t_A(G_i \wedge C) \tag{18}$$

Теорема 2. $\mu_{A,\Pi}(C) = s \cdot \mathbb{E}[\xi_\Pi]$.

Доказательство. В контексте сказанного выше, имеем вероятностное пространство, где $\Pi = \{G_1, \ldots, G_s\}$ — пространство элементарных событий, а ξ_Π — определенная на нём случайная величина. Пусть $\mathrm{Spec}(\xi_\Pi) = \{\xi_1, \ldots, \xi_r\}$ $(r \leqslant s)$ — спектр ξ_Π, где $\xi_k \in \mathbb{R}^+$, $k \in \{1, \ldots, r\}$. Обозначим через $\#\xi_k$ число элементарных событий в Π, для которых случайная величина ξ_Π принимает значение ξ_k. Тогда вероятность события $\{\xi_\Pi = \xi_k\}$ равна $p_k = \#\xi_k / s$, и ξ_Π имеет закон распределения $P(\xi_\Pi) = \{p_1, \ldots, p_k\}$ (поскольку выполняются все аксиомы Колмогорова [62]). Таким образом:

$$\sum_{i=1}^{s} t_A(G_i \wedge C) = \sum_{k=1}^{r} \#\xi_k \cdot \xi_k = s \cdot \sum_{k=1}^{r} \frac{\#\xi_k}{s} \cdot \xi_k = s \cdot \mathbb{E}[\xi_\Pi] \qquad \square$$

Используя Теорему 2, можно оценить $\mu_{A,\Pi}(C)$ с помощью метода Монте-Карло [92]. Сначала проведем N независимых вероятностных экспериментов, где каждый эксперимент включает выбор $G \in \Pi$ с учетом вероятностного распределения $P = \{p_1, \ldots, p_s\}$, с $p_i = 1/s$, где $i \in \{1, \ldots, s\}$. Затем вычислим выборочное среднее ξ_Π как $\hat{\mu} = \frac{1}{N} \sum_{j=1}^{N} \xi^j$, где ξ^j — значение ξ_Π, наблюдаемое в j-ом эксперименте. Наконец, оценим $\mu_{A,\Pi}(C)$ как $\widetilde{\mu}_{A,\Pi} = s \cdot \hat{\mu}$.

Будем говорить, что $\widetilde{\mu}_{A,\Pi}(C)$ является (ε,δ)-приближением [120] для $\mu_{A,\Pi}(C)$, если выполняется следующее неравенство (вариация условия (9)):

$$\Pr\left\{\left|\mu_{A,\Pi}(C) - \widetilde{\mu}_{A,\Pi}(C)\right| \leqslant \varepsilon \cdot \mu_{A,\Pi}(C)\right\} \geqslant 1 - \delta \tag{19}$$

Как следует из неравенства Чебышёва, неравенство (19) выполняется для всех $N \geqslant \frac{\mathrm{Var}[\xi_\Pi(C)]}{\varepsilon^2 \cdot \delta \cdot E[\xi_\Pi]^2}$, где ξ_Π определена относительно (17). Однако на практике $Var(\xi_\Pi(C))$ может быть очень большим, и использование малых значений N может привести к недостаточной точности оценки $\mu_{A,\Pi}(C)$. Эта проблема возникает в таких методах разбиения, как стандартная техника Cube-and-Conquer, а также схема разбиения из [93]. В следующем разделе представлены две конструкции построения разбиений SAT, которые позволяют уменьшить дисперсию $\mathrm{Var}[\xi_\Pi]$ и, следовательно, улучшить точность оценки $\mu_{A,\Pi}(C)$.

3.2. Два новых метода разбиения SAT для CircuitSAT

В данном подразделе представленны два метода для построения разбиений SAT, нацеленных, главным образом, на задачи из области CircuitSAT. Соответствующие конструкции могут быть применены как к LEC, так и к задачам обращения криптографических функций. Ниже приведено описание для LEC.

Рассмотрим задачу LEC для двух булевых схем S_f, S_h, задающих функции $f,h : \{0,1\}^n \to 0,1^m$. Определим первую конструкцию следующим образом:

Конструкция 1. *Рассмотрим множество переменных* $X^{in} = \{x_1, \ldots, x_n\}$*, связанных с входами схем* $S_f, S_h, S_{f \triangle h}$*. Затем выберем целое число* k *такое, что* $1 < k < n$ *и разделим* X^{in} *на* $q = \lceil n/k \rceil$ *попарно непересекающихся множеств* X^j*, где* $j \in \{1, \ldots, q\}$*. Если* n *делится на* k*, то каждое множество* X^j *содержит* k *переменных. В противном случае разделим* X^{in} *на* $q - 1$ *множества* $X^1, \ldots, X^{q-1}$ *по* k *переменных каждое и множество* X^q *размером* r*, так что* $n = k \cdot \lfloor \frac{n}{k} \rfloor + r$*, где* $r \in \{1, \ldots, k-1\}$*.*

Рассмотрим произвольную булеву функцию $\lambda\colon \{0,1\}^l \to \{0,1\}$, где $l \in \mathbb{N}^+$, и предположим, что λ не зафиксирована. Пусть $\neg\lambda\colon \{0,1\}^l \to \{0,1\}$ обозначает отрицание λ. С каждым X^j, $j \in \{1, \ldots, q\}$, свяжем две КНФ φ_1^j и φ_2^j, которые определяют функции $\lambda^j\colon \{0,1\}^{|X^j|} \to \{0,1\}$ и $\neg\lambda^j\colon \{0,1\}^{|X^j|} \to \{0,1\}$ соответственно.

Теорема 3. *Пусть* φ^j *обозначает обе формулы* φ_1^j *и* φ_2^j*. Множество* Π *всех* $2^{\lceil n/k \rceil}$ *возможных формул вида* $\varphi^1 \wedge \cdots \wedge \varphi^{\lceil n/k \rceil}$ *формирует SAT-разбиение формулы* (4)*.*

Доказательство. Сначала докажем, что формула $C_{f \triangle h}$ имеет ровно 2^n выполняющих наборов. Действительно, рассмотрим множество $X^{\text{in}} = \{x_1, \ldots, x_n\}$ и пусть $\alpha = (\alpha_1, \ldots, \alpha_n)$ — произвольное назначение переменных из X^{in}. Также рассмотрим формулу $\Phi(X, \alpha) = x_1^{\alpha_1} \wedge \ldots \wedge x_n^{\alpha_n} \wedge C_{f \triangle h}$ над множеством переменных X. Из Леммы 1 следует, что применение UP к $\Phi(X, \alpha)$ приведет к выводу значений всех переменных из этой КНФ без конфликтов. Для каждого $\alpha \in \{0,1\}^n$ построим такое назначение переменных из X и скажем, что это назначение генерируется соответствующим α. Обозначим построенное множество назначений над X через $\Lambda(X)$. Можно заметить, что все назначения из $\Lambda(X)$ различны.

Следом, рассмотрим множество переменных $\tilde{X} = X \setminus X^{in}$. С каждым назначением $\lambda \in \Lambda(X)$ свяжем часть λ, содержащую назначения переменных из $\tilde{X}$. Обозначим построенное множество назначений как $\Lambda(\tilde{X})$. Пусть $\gamma \in \{0,1\}^{|\tilde{X}|}$ — произвольное назначение переменных из $\tilde{X}$ такое, что $\gamma \notin \Lambda(\tilde{X})$. Подстановка γ в $C_{f \triangle h}$ приводит к тому, что результирующая КНФ $C_{f \triangle h}[\gamma/\tilde{X}]$ является невыполнимой, поскольку для любого $\alpha = (\alpha_1, \ldots, \alpha_n)$ применение UP к формуле $x_1^{\alpha_1} \wedge \cdots \wedge x_n^{\alpha_n} \wedge C_{f \triangle h}[\gamma/\tilde{X}]$ приведет к конфликту. Таким образом, любой выполняющий набор $C_{f \triangle h}$ — это назначение, порождённое некоторым $\alpha \in \{0,1\}^{|X^{in}|}$, и различные назначения из $\{0,1\}^{|X^{in}|}$ порождают различные выполняющие наборы $C_{f \triangle h}$. Следовательно, у формулы $C_{f \triangle h}$ ровно 2^n выполняющих наборов. С другой стороны, любое $\alpha \in \{0,1\}^{|X^{in}|}$ выполняет ровно одну формулу G_i описанного выше вида, где $i \in \{1, \ldots, 2^{\lceil n/k \rceil}\}$. Следовательно, формулы $C_{f \triangle h}$ и $(G_1 \vee \ldots G_{2^{\lceil n/k \rceil}})$ имеют одинаковые выполняющие

наборы. Следовательно, формулы $C_{f\triangle h} \wedge C(M)$ и $(G_1 \vee \ldots G_{2^{\lceil n/k \rceil}}) \wedge C_{f\triangle h} \wedge C(M)$ равновыполнимы. □

Важный вопрос состоит в том, как выбрать функции λ^j и $\neg\lambda^j$ таким образом, чтобы гарантировать малую дисперсию $\mathrm{Var}[\xi_\Pi]$ для SAT-разбиения описанного выше типа? Здравый смысл подсказывает, что имеет смысл использовать *сбалансированнаые* булевы функции — функции, которые принимают значение 1 на 2^{l-1} входных словах, в качестве функции $\lambda\colon \{0,1\}^l \to \{0,1\}$. Заметим, что отрицание сбалансированной функции также является сбалансированной функцией. Хорошим примером такого рода функции для $l > 1$ является функция, задаваемая формулой $x_1 \oplus \ldots \oplus x_l$.

Далее проведем несколько неформальный анализ свойств Конструкции 1, и используем его результаты в качестве основы для Конструкции 2, которая показала налучшие результаты среди всех рассмотренных методов в экспериментах с некоторыми чрезвычайно сложными примерами LEC в виде SAT. Рассмотрим функцию (3) и шаблонную КНФ $C_{f\triangle h}$. Многочисленные эксперименты показывают, что даже когда SAT для $C_{f\triangle h} \wedge C(\mathcal{M})$ является чрезвычайно сложной, SAT для $C_{f\triangle h}$ остается простой: любой CDCL SAT-решатель, получив на вход $C_{f\triangle h}$ и не имея никакой дополнительной информации о структуре схемы, способен найти выполняющий набор для $C_{f\triangle h}$. Этот набор можно рассматривать как сертификат выполнимости для $C_{f\triangle h}$. Как было отмечено выше, всего для КНФ $C_{f\Delta h}$ существует 2^n таких сертификатов. Таким образом, доказательство невыполнимости $C_{f\triangle h} \wedge C(\mathcal{M})$ можно рассматривать как процесс, который опровергает все эти сертификаты. Более того, если функции λ^j сбалансированы для каждого $j \in \{1, \ldots, \lceil n/k \rceil\}$, то каждая формула вида $\varphi^1 \wedge \cdots \wedge \varphi^{\lceil n/k \rceil} \wedge C_{f\triangle h}$ имеет $2^{n-\lceil n/k \rceil}$ выполняющих наборов, которые также являются сертификатами удовлетворимости. Таким образом, можно сделать два предположения:

1. Алгоритму A намного легче доказать невыполнимость формулы $\varphi^1 \wedge \cdots \wedge \varphi^{\lceil n/k \rceil} \wedge C_{f\triangle h} \wedge C(\mathcal{M})$, потому что ему необходимо опровергнуть $2^{n-\lceil n/k \rceil}$ сертификатов вместо 2^n.
2. Для сбалансированных функций λ^j, $j \in \{1, \ldots, \lceil n/k \rceil\}$, все $2^{\lceil n/k \rceil}$ различных формул вида $\varphi^1 \wedge \cdots \wedge \varphi^{\lceil n/k \rceil} \wedge C_{f\triangle h} \wedge C(\mathcal{M})$ должны быть более или менее схожи по времени работы алгоритма A на них — разбиение Π, заданное Конструкцией 1, должно иметь малую дисперсию $\mathrm{Var}[\xi_\Pi]$.

Хотя представленные аргументы лишены строго формального доказательства, на практике их выводы экспериментально подтверждаются. Ниже опишем еще одну конструкцию, при разработке которой были учтены указанные выше свойства.

Основная идея описанной ниже конструкции заключается в том, чтобы рассмотреть произвольное назначение переменных из $X^{\text{in}} = \{x_1, \ldots, x_n\}$ в качестве коэффициентов двоичного представления числа из $N_0^n = \{0,1,\ldots,2^n - 1\}$. Таким образом, существует взаимно однозначное отображение вида $\{0,1\}^n \to N_0^n$. Для произвольных $a,b \in N_0^n$ назовем множество чисел $\{q \in N_0^n \mid a \leqslant q \leqslant b\}$ *интервалом* и обозначим такой интервал как $[a,b]$. Рассмотрим множество булевых векторов из $\{0,1\}^n$, которые являются двоичными представлениями чисел из $[a,b]$, как множество решений следующего целочисленного неравенства, предполагая, что x_i принимают значения из $\{0,1\}$:

$$a \leqslant x_n + 2 \cdot x_{n-1} + \cdots + 2^{n-1} \cdot x_1 < b \tag{20}$$

Скажем, что множество $\mathcal{R}^n$, образованное интервалами описанного вида, является *полной системой интервалов*, если никакие два интервала из $\mathcal{R}^n$ не пересекаются и любое число из N_0^n принадлежит какому-либо интервалу в $\mathcal{R}^n$. Это означает, что любая полная система интервалов порождает разбиение $\{0,1\}^n$ на непересекающиеся подмножества, образованные решениями соответствующих неравенств (20).

Конструкция 2. *Пусть $\mathcal{R}^n$ — полная система интервалов. С произвольным интервалом $I = [a,b] \in \mathcal{R}^n$ ассоциируем неравенство вида* (20) *и КНФ C_I, полученное путем кодирования* (20) *в SAT с использованием соответствующих техник, например, представленных в [121]. Определим* $\Pi = \{C_I\}_{I \in \mathcal{R}^n}$.

Теорема 4. *Множество $\Pi = \{C_I\}_{I \in \mathcal{R}^n}$, полученное с использованием Конструкции 2, формирует SAT-разбиение формулы $C_{f \triangle h} \wedge C(\mathcal{M})$.*

Доказательство. Аналогично доказательству Теоремы 3, используем Лемму 1, чтобы показать, что любое назначение, удовлетворяющее $C_{f \triangle h}$, также удовлетворяет ровно одну формулу вида $G_i \wedge C_{f \triangle h}$, где $i \in \{1, \ldots, s\}$. Следовательно, можно заключить, что $C_{f \triangle h} \wedge C(\mathcal{M})$ и $C_{f \triangle h} \wedge C(\mathcal{M}) \wedge (G_1 \vee \cdots \vee G_s)$ равновыполнимы. С другой стороны, можно заметить, что любая КНФ вида $G_i \wedge G_j$ (где $i \neq j \in \{1, \ldots, s\}$) невыполнима. Значит, $\Pi = \{G_1, \ldots, G_s\}$ является SAT-разбиением формулы $C_{f \triangle h} \wedge C(\mathcal{M})$. □

Заметим, что в случае, когда $\mathcal{R}^n$ формируется интервалами равного размера 2^l, где $1 \leqslant l < n$, можно указать любой интервал $I \in \mathcal{R}^n$ без использования

кодировок из [121]. Действительно, в этом конкретном случае интервал с номером $k \in \{1, \ldots, 2^{n-l}\}$ состоит из чисел вида $(k-1) \cdot 2^l + j$, где $j \in \{0, \ldots, 2^l - 1\}$. Следовательно, этот интервал можно указать с помощью двоичного вектора $\lambda_{l+1} \ldots \lambda_n$, где λ_q являются коэффициентами из $\{0,1\}$ в двоичном представлении числа $(k-1) \cdot 2^l$ из n бит.

Заметим, что описанный подход не применим к интервалам произвольного вида. В самом деле, рассмотрим небольшой пример: для $n = 6$ вектор (010111) указывает на число 23, а вектор (101000) соответствует числу 40. Тогда, не существует числа $i \in \{1, \ldots, 6\}$ такого, что x_i принимает одно и то же значение во всех числах из интервала $[23,40] \subseteq \mathbb{N}_0$. Поэтому в общем случае необходимо применять техники, такие как например, описанные в [121], для кодирования интервалов вида (20) в SAT.

3.3. Вычислительные эксперименты

Все эксперименты, представленные в данном разделе, были проведены на кластере Университета ИТМО, каждый узел которого оснащен двумя 18-ядерными процессорами Intel Xeon E5-2695 v4 и 128 ГБ оперативной памяти. В качестве SAT-решателя были использованы Kissat[1] (версия 3.0.0) и CaDiCaL[2] (версия 1.9.5) из-за их высокой производительности, широких возможностей по настройке и программному взаимодействию через API. Это было согласовано даже при использовании подхода CnC с инкрементальными решателями (включая те, которые интегрированы в репозиторий CnC и последнюю версию CaDiCaL). Kissat выделялся при решении кубов на всех тестовых наборах и решателях.

Основным вопросом, который исследовался в вычислительных экспериментах, была точность оценок сложности относительно разбиения (в смысле формулы (1)). Ситуации, когда сложность относительно разбиения меньше или близка к времени работы последовательного решателя на исходной задаче, являются особенно интересными, потому что в таких случаях соответствующие разбиения позволяют не только точно оценить время решения задачи (в отличие от случая последовательного решения), но и обеспечивают более эффективную стратегию решения соответствующего экземпляра LEC.

[1] https://github.com/arminbiere/kissat

[2] https://github.com/arminbiere/cadical

3.3.1. Тестовые данные

Мы рассматриваем два класса тестовых наборов (бенчмарков). Первый класс состоит из (невыполнимых) экземпляров задачи LEC для схем, представляющих алгоритмы умножения, такие как «умножение столбиком», «дерево Уоллеса» [10], «алгоритм Карацубы» [122] и «умножитель Дадда» [123]. Эти экземпляры обозначаются как $\texttt{AvB}_k$, где A и B означают алгоритмы умножения, а k — количество бит в умножаемых числах. Например, $\texttt{CvK}_{16}$ представляет собой экземпляр LEC для проверки эквивалентности умножения двух 16-битных чисел (16x16 умножитель) методом столбика и алгоритмом Карацубы. Известно, что такого рода тесты крайне сложны для современных SAT-решателей [93; 124].

Второй класс состоит из нескольких (выполнимых) экземпляров, связанных с алгебраическим криптоанализом [3], а именно, SAT-кодировок атаки поиска прообраза для хеш-функции MD4 с уменьшенным числом раундов. Эта проблема была недавно решена в [125] с использованием подхода Cube-and-Conquer. Набор тестовых примеров на основе MD4 служит для того, чтобы показать, что предложенная техника применима к (1) выполнимым тестам (2) тестам не из области LEC.

3.3.2. Эксперименты по оценке декомпозиционной сложности

В первом наборе экспериментов оценивается сложность экземпляров LEC для умножителей относительно предложенных разбиений SAT, где множество входов X^{in} разбивается на непересекающиеся подмножества, называемые *чанками* (*chunks*), в соответствии с Конструкцией 1. Мы рассматриваем следующие виды функций λ^j:

- 2-XOR: $\lambda^1 = x_1 \oplus x_2$, 3-XOR: $\lambda^1 = x_1 \oplus x_2 \oplus x_3$, *и т.д.*;
- 2-DIS: $\lambda^1 = x_1 \vee x_2$;
- 3-MAJ: $\lambda^1 = \mathit{majority}(x_1, x_2, x_3)$, где $\mathit{majority}(a, b, c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$.

Функции λ^j для $j > 1$ определены на соответствующих непересекающихся чанках входов, например, $\lambda^2 = x_4 \oplus x_5 \oplus x_6$ для 3-XOR. Во всех случаях формулы, соответствующие $\lambda^j_1 = \lambda^j$ и $\lambda^j_2 = \neg\lambda^j$, были закодированы в КНФ.

Аналогично, для разбиений SAT, построенных в соответствии с Конструкцией 2, используется обозначение INT-s, где s обозначает количество интервалов, например, INT-65536 соответствует разбиению на 65 536 подзадач.

Чтобы обеспечить достоверность и актуальность представленных результатов, среди всех составленных бенчмарков были выбраны только те экземпляры, которые

представляют практический интерес — не решаются за несколько секунд, но при этом могут быть решены за разумное время. Для тестовых наборов алгоритмов сортировки были выбраны экземпляры, которые кодируют LEC для $k = 9$ и $l = 4$. Среди умножителей были выбраны экземпляры двух разных уровней сложности (умножители 12x12 и 16x16, например, $\texttt{CvK}_{12}$ или $\texttt{KvW}_{16}$) для демонстрации гибкости предложенного подхода к решению задач с различной сложностью. Для каждого выбранного тестового примера были построены соответствующие разбиения и были решены **все** подзадачи, чтобы вычислить истинные значения математического ожидания $\mathbb{E}[\xi_\Pi]$ и дисперсии $\mathrm{Var}[\xi_\Pi]$.

Кроме того, были рассмотрены разбиения, построенные с использованием техники Cube-and-Conquer (CnC). Для этой цели были построены кубы с помощью `march_cu`[3]. В частности, были подобраны значения опций `-d <depth>` и `-n <number>` таким образом, чтобы размеры полученных разбиений были аналогичны разбиениям, построенным с помощью методов, предложенных в данной работе. Для некоторых «простых» экземпляров также был запущен `march_cu` с параметрами по умолчанию, которые могут рассматриваться как базовый уровень в этом экспериментальном исследовании. Отметим, что для некоторых «более сложных» тестов `march_cu` в режиме по умолчанию не смог выдать результаты (т.е. набор кубов) за разумное время (24 часа), что привело к пропуску этих экспериментов. Затем все полученные кубы были независимо решены параллельно. Далее в этом документе будем обозначать разбиения, сгенерированные в идеологии CnC, как `CnC-d*`, `CnC-n*` и `CnC-default`.

Результаты экспериментов кратко представлены на Рисунке 11 и в Таблице 6. В частности, Рисунок 11 содержит подробные результаты для *всех* обсуждаемых типов разбиений на двух выбранных экземплярах ($\texttt{CvK}_{12}$ и $\texttt{CvK}_{16}$). В то же время, Таблица 6 представляет *лучшие* разбиения для остальных экземпляров. Для каждого разбиения приводится среднее и стандартное отклонение («Avg ± sd»), диапазон времени выполнения подзадач («Min – max») и общее время (CPU Time), необходимое для решения всех подзадач. Таблица также включает строки «Sequential» для представления базовой производительности последовательного решателя SAT.

Графики на Рисунке 11 визуализируют дисперсию времени выполнения SAT-решателя при использовании рассматриваемых схем разбиения. Для конструкций 1 и 2 (разбиения `2-XOR` и `INT`, соотвественно) время работы SAT-решателя имеет относительно низкую дисперсию, что указывает на то, что можно точно оценить необходимое общее время выполнения, используя относительно небольшой объем

[3] `https://github.com/marijnheule/CnC`

Таблица 6 — Экспериментальные результаты для SAT-разбиений для задачи проверки эквивалентности (LEC) умножителей

Instance	Partitioning (type / size)	Avg ± sd time, s	Min – max time, s	Total time, s
CvW_{12}	Sequential	—	—	6 612
	CnC-n2850 / 261	14 ± 31	0,6 – 299	3 583
	3-MAJ / 256	15 ± 3,4	3,5 – 25	3 832
	INT / 256	11 ± 3,0	0,6 – 17	2 809
DvC_{12}	Sequential	—	—	3 299
	CnC-d8 / 256	9,7 ± 20	1,2 – 275	2 474
	3-XOR / 256	14 ± 1,0	12 – 17	3 571
	INT / 256	8,9 ± 2,2	0,4 – 13	2 272
DvK_{12}	Sequential	—	—	36 224
	CnC-d8 / 256	17 ± 12	0,2 – 65	4 361
	3-XOR / 256	23 ± 1,1	21 – 27	5 880
	INT / 256	17 ± 1,7	10 – 22	4 348
DvW_{12}	Sequential	—	—	10 373
	CnC-n2650 / 293	13 ± 35	0,6 – 374	3 880
	3-MAJ / 256	17 ± 3,2	4,4 – 27	4 393
	INT / 256	13 ± 3,7	0,6 – 20	3 308
KvW_{12}	Sequential	—	—	37 339
	CnC-d8 / 256	20 ± 22	0,2 – 227	5 047
	3-XOR / 256	25 ± 1,0	22 – 29	6 267
	INT / 256	19 ± 2,1	10 – 24	4 838
CvW_{16}	Sequential	—	—	>864 000
	CnC-n4500 / 51 350	27 ± 150	0,05 – 13 531	1 392 714
	2-XOR / 65 536	22 ± 1,9	16 – 31	1 418 199
	INT / 65 536	19 ± 2,3	0,01 – 27	875 966
DvC_{16}	Sequential	—	—	>864 000
	CnC-d16 / 65 534	12 ± 36	0,01 – 2 797	809 516
	2-XOR / 65 536	20 ± 2,1	12 – 30	1 302 856
	INT / 65 536	9,1 ± 2,3	0,01 – 16	597 104
DvK_{16}	Sequential	—	—	>864 000
	CnC-d16 / 65 535	25 ± 28	0,01 – 918	1 640 023
	2-XOR / 65 536	35 ± 1,8	28 – 45	2 301 014
	INT / 65 536	20 ± 2,6	0,01 – 31	1 327 640
DvW_{16}	Sequential	—	—	>864 000
	CnC-n4300 / 54 347	24 ± 113	0,01 – 8 062	1 304 536
	2-XOR / 65 536	24 ± 2,2	18 – 36	1 595 146
	INT / 65 536	14 ± 3,7	0,01 – 27	885 362
KvW_{16}	Sequential	—	—	>864 000
	CnC-n7500 / 70 469	31 ± 109	0,2 – 11 293	2 159 531
	2-XOR / 65 536	36 ± 1,7	29 – 48	2 344 223
	INT / 65 536	22 ± 2,9	0,01 – 31	1 423 759

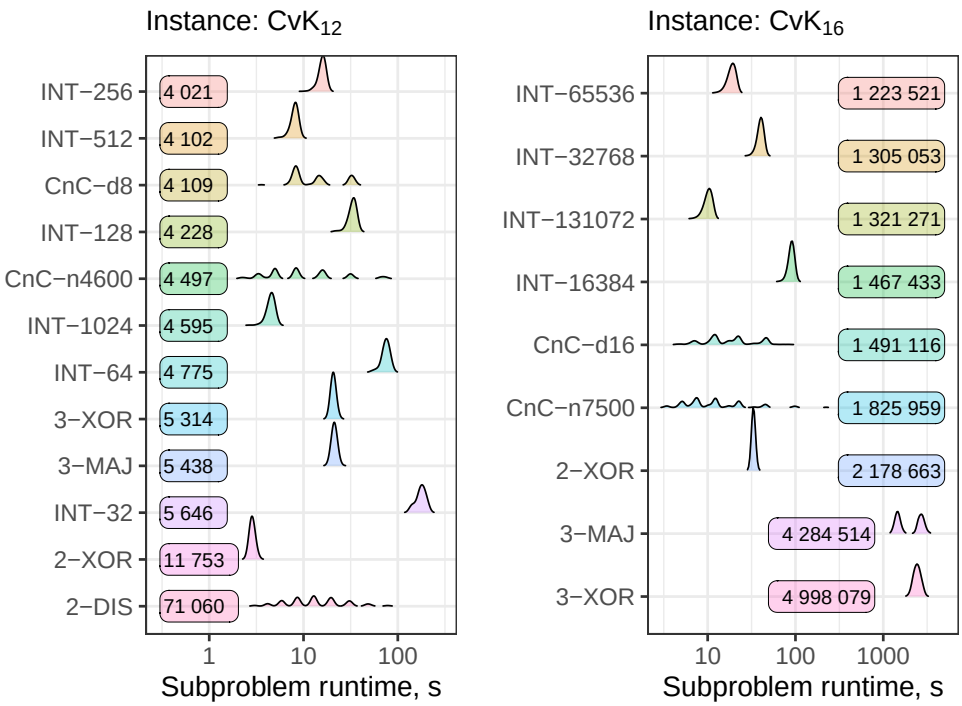
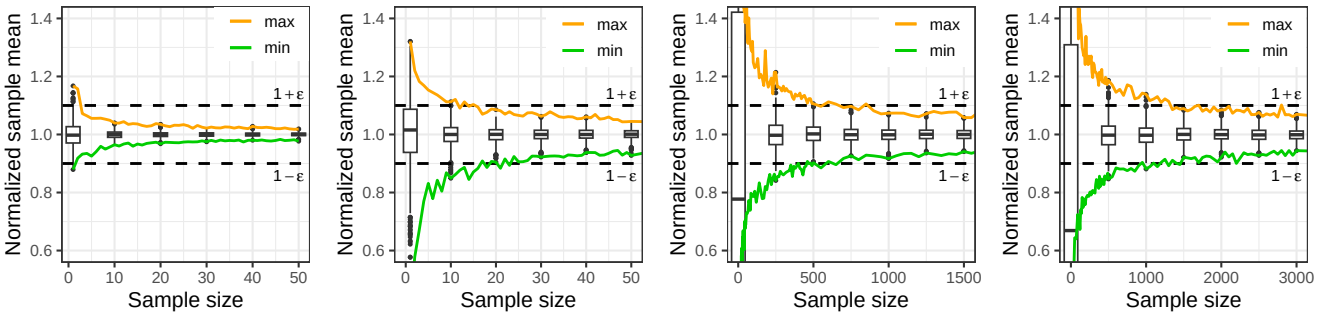


Рисунок 11 — Распределения времени выполнения SAT-решателя на подзадачах в различных разбиениях экземпляров LEC $\texttt{CvK}_{12}$ (слева) и $\texttt{CvK}_{16}$ (справа). Цифры рядом с графиками плотности указывают общее время выполнения (в секундах), и все разбиения упорядочены по общему времени (наименьшее время сверху)

выборки. Напротив, для Cube-and-Conquer и разбиений `2-DIS` дисперсия значительно больше из-за неравномерного распределения сложности подзадач. Для последнего это можно объяснить несбалансированностью функции $\lambda = a \vee b$, используемой в `2-DIS`. Эти результаты подчеркивают важность выбора подходящих схем разбиения для достижения построения точной оценки общего времени выполнения.

Экспериментальные результаты показывают, что оценка для разбиения не всегда согласуется с временем работы последовательного решателя на исходной задаче. При этом, интересно, что общее время, необходимое для решения всех подзадач для умножителей, существенно меньше времени, необходимого для последовательного решения соответствующих тестовых примеров. В частности, для 16-битных умножителей однопоточный решатель не смог завершить работу даже после 10 дней, в то время как все подзадачи в разбиении были решены за разумное и *предсказуемое* время, соответствующее оценке. Для «Sequential» строки в таблицах следует уточнить, что решатель работал на одном ядре, в то время как все остальные эксперименты вычислялись параллельно, и представленное общее время CPU является суммой времени работы на всех подзадачах.



а) $\mathtt{CvK}_{16}$, 2-XOR / 65 536 б) $\mathtt{CvK}_{16}$, INT / 65 536 в) $\mathtt{CvK}_{16}$, CnC-d16 / 65 536 г) $\mathtt{MD4}_{40}$, INT / 110 000

Рисунок 12 — Распределения выборочных средних для различных размеров выборок N для экземпляра LEC $\mathtt{CvK}_{16}$ и задачи нахождения прообраза MD4 $\mathtt{MD4}_{40}$

В контексте всего сказанного выше, одной из основных проблем является точность полученных оценок $\mathbb{E}[\xi_\Pi]$, так как дисперсия $\mathrm{Var}[\xi_\Pi]$ оказывает отрицательное влияние на точность. Однако результаты в Таблице 6 показывают, что предложенные методы SAT-разбиения дают очень низкую дисперсию на рассматриваемых бенчмарках. Это является значительным преимуществом предложенного метода, так как он позволяет использовать небольшую случайную выборку для получения надежной оценки $\mathbb{E}[\xi_\Pi]$. Для демонстрации этого проведем дополнительный анализ полученных результатов.

Для различных значений N сгенерируем $P = 1000$ случайных выборок размера N и вычислим средние значения выборок $(\hat{\xi}^1, \ldots, \hat{\xi}^P)$, где каждое $\hat{\xi}^r = \frac{1}{N}\sum j = 1^N \xi_j^r$. Также вычислим среднее значение средних значений выборок $\Xi(N) = \frac{1}{P}\sum_{r=1}^{P} \hat{\xi}^r$ и минимальные и максимальные значения среди $\hat{\xi}$, которые обозначены как $M_*(N)$ и $M^*(N)$, соответственно. Далее все значения нормализуем путем деления их на $\mathbb{E}[\xi_\Pi]$. Распределения нормализованных средних значений для различных размеров выборки показаны на Рисунке 12, где горизонтальная ось представляет размер случайной выборки N. Здесь рассматривается тестовый пример, кодирующий LEC задачу для умножителей $\mathtt{CvK}_{16}$ и два различных разбиения: `INT-65536` (предложенное разбиение на 65 536 интервалов) и `CnC-d16` (Cube-and-Conquer, построенное при помощи ‘`march_cu -d 16`’). На графиках показаны нормализованные линии для минимальных и максимальных значений, представленных как $M_*(N)/\mathbb{E}[\xi_\Pi]$ (зеленая линия, внизу) и $M^*(N)/\mathbb{E}[\xi_\Pi]$ (оранжевая линия, сверху), соответственно. Результаты показывают, что выборочное среднее $\hat{\xi}$ является надежной оценкой $\mathbb{E}[\xi_\Pi]$, даже когда N гораздо меньше общего размера разбиения. Например, размер выборки $N \approx 30$ из общего числа 65 536 подзадач для

разбиения INT-65536 для теста CvK_{16} достаточен для получения оценки в пределах 10%-интервала $\mathbb{E}[\xi_\Pi]$. Наоборот, достаточный размер выборки для разбиений CnC обычно значительно больше, в частности, для CnC-d16 (которое также имеет размер 65 536), он составляет как минимум $N \approx 1000$.

3.3.3. Эксперименты по поиску прообразов MD4

Для того, чтобы показать, что предложенные конструкции применимы и к другим сложным экземплярам CircuitSAT, помимо невыполнимых LEC-бенчмарков, они были применены к задаче поиска прообразов для хеш-функции MD4 с уменьшенным числом раундов. Эта задача интересна тем, что лучшие известные результаты для нее были получены с помощью метода Cube-and-Conquer со специализированной стратегией поиска параметров CnC.

В проведённых экспериментах были взяты две CNF из репозитория[4], представленного в [125]: md4_40steps_11.30-32Dobb_one_constr_one_hash, обозначаемая как $\mathrm{MD4}_{40}$, и md4_43steps_12Dobb_one_constr_one_hash, обозначаемая как $\mathrm{MD4}_{43}$. Эти CNF соответствуют двум не слишком легким и не слишком трудным задачам криптоанализа. Используя Конструкцию 2, были получены оценки времени выполнения для различного числа интервалов. Наилучшие найденные параметры разбиения были использованы для полного решения обеих задач. Как и в [125], были решены все подзадачи — процесс не останавливался как только был найден выполняющий набор.

Результаты этой серии экспериментов обобщены в Таблице 7. Они сравниваются с результатами, опубликованными в [125], обозначенными как CnC (время выполнения последних было замерено для 12 ядер CPU, поэтому они были масштабированны для одного ядра CPU). Стоит отметить, что в статье [125] также использовалась стратегия, адаптированная к данной конкретной задаче, для поиска оптимальных параметров разложения CnC, хотя время нахождения этих параметров не включено в Таблицу 7, а также использовалась вычислительная платформа с более быстрыми ядрами CPU. Тем не менее, используя предложенный метод, удалось решить рассмотренные задачи за время, сравнимое с временем в [125].

В Таблице 7 строки, помеченные «(est.)», соответствуют *оценкам* времени работы, построенным по случайным выборкам размером $N = 1000$, а строки, помеченные «(full)», соответствуют решению всех подзадач из построенного разбиения. Оцененное время работы также отмечено звездочкой в колонке «Время». Из распределения

[4] https://github.com/olegzaikin/MD4-CnC

Таблица 7 — Экспериментальные результаты для разбиений экземпляров MD4

Inst.	Partitioning (type / size)	Avg ± sd time, s	Min – max time, s	Time, s
$\mathtt{MD4}_{40}$	CnC / 400 509	26 ± 274	0,01 – 39 087	10 550 880
(est.)	INT / 70k	182 ± 210	0,08 – 1 927	12 747 000*
(est.)	INT / 110k	105 ± 113	0,08 – 1 382	11 511 500*
(est.)	INT / 150k	84 ± 86	0,09 – 659	12 480 000*
(full)	INT / 110k	112 ± 124	0,06 – 2 210	12 250 123
$\mathtt{MD4}_{43}$	CnC / 54 611	31 ± 52	0,01 – 2 236	1 686 960
(est.)	INT / 10k	356 ± 380	7 – 5 109	3 561 400*
(est.)	INT / 30k	105 ± 90	0,1 – 842	3 205 800*
(est.)	INT / 50k	65 ± 52	0,08 – 371	3 241 000*
(full)	INT / 30k	112 ± 104	0,07 – 1 910	3 349 802

выборочных средних для $\mathtt{MD4}_{40}$, представленного на правом графике на Рисунке 12, видно, что выбранного значения размера выборки ($N = 1000$) достаточно для построения точных оценок времени работы. Расхождения между реальным временем решения и расчетным временем в Таблице 7 еще раз показывают, что предложенные конструкции дают небольшую дисперсию в трудности подзадач.

Выводы по главе 3

В данной главе описывается общий подход к оценке сложности примеров SAT, связанных с булевыми схемами, относительно методов разбиения этих примеров на подформулы. Представлена теоретическая база для оценки сложности на основе разбиения, введена случайная величина для выражения времени работы SAT-решателя на подформулах, и предложен алгоритм Монте-Карло для практической оценки этой сложности. Было установлено, что декомпозиционная трудность формулы может быть вычислена через математическое ожидание, однако отмечено, что высокая дисперсия случайной величины может потребовать значительного числа экспериментов для точной оценки.

Также предложены две новые конструкции разбиения SAT для задач CircuitSAT, применимые к задачам логической эквивалентности и обращения криптографических функций. Первая конструкция основывается на разделении входных переменных на непересекающиеся подмножества и построении для каждого подмножества двух КНФ, представляющих функцию и её отрицание. Этот метод использует сбалансированные

булевые функции для уменьшения дисперсии и повышения точности оценки. Вторая конструкция основывается на разделении входных переменных на интервалы и построении для каждого интервала КНФ, представляющих функцию и её отрицание. Вторая конструкция, опираясь на анализ и учёт недостатков первой, продемонстрировала лучшие результаты в экспериментах с сложными примерами LEC. Основное преимущество предложенных методов заключается в том, что они позволяют SAT-решателю опровергать меньшее количество сертификатов, значительно улучшая эффективность решения задач. Представлены результаты экспериментального исследования, которые показывают, что предложенные методы декомпозиции задачи SAT позволяют получить точные оценки время работы SAT-решателя.

ЗАКЛЮЧЕНИЕ

В данной диссертационной работе была достигнута поставленная цель — повышение эффективности полных алгоритмов решения задачи булевой выполнимости (SAT) применительно к синтезу и верификации моделей автоматных программ. Разработаны оригинальные методы и техники декомпозиции булевых формул, что позволило существенно сократить время работы алгоритмов.

В ходе исследования созданы новые алгоритмы кодирования задач синтеза конечных автоматов и булевых схем, отличающиеся явным кодированием структуры охранных условий в виде деревьев разбора формул. Предложены методы декомпозиции булевых формул, обеспечивающие точную оценку декомпозиционной трудности задач благодаря низкой дисперсии времени решения подзадач. Масштабные вычислительные эксперименты подтвердили практическую значимость предложенных методов. Разработанные методы успешно применены для решения сложных примеров синтеза и верификации моделей автоматных программ, включая конечные автоматы и логические схемы.

Созданная программная библиотека `kotlin-satlib` предоставляет унифицированный интерфейс для работы с современными SAT-решателями, облегчая процесс моделирования и решения задач синтеза и верификации. Библиотека включает модули для интеграции с SAT-решателей через технологию JNI, упрощения записи ограничений с использованием преобразований Цейтина, манипуляции переменными с конечными доменами и работы с многомерными массивами SAT-переменных.

Результаты исследования демонстрируют, что предложенные методы и алгоритмы могут быть эффективно интегрированы в существующие системы синтеза и верификации моделей автоматных программ. Они также могут быть адаптированы для решения других задач, связанных с булевой выполнимостью, что открывает возможности для дальнейших исследований и разработок.

Перспективы дальнейшего развития темы включают углубленное исследование методов декомпозиции булевых формул, разработку новых алгоритмов для специфических классов задач и расширение функциональности программной библиотеки для поддержки большего числа SAT-решателей и разнообразных типов задач. Таким образом, проделанная работа вносит значительный вклад в область синтеза и верификации моделей автоматных программ, предлагая более эффективные и гибкие инструменты для решения сложных задач.

Список литературы

1. *Turing A. M.* On Computable Numbers, with an Application to the Entscheidungsproblem // Proceedings of the London Mathematical Society. 1937. Vol. s2–42, no. 1. P. 230–265.
2. *Kroening D.* Software Verification // Handbook of Satisfiability. Vol. 336 / ed. by A. Biere [et al.]. 2nd ed. IOS Press, 2021. P. 791–818. (Frontiers in Artificial Intelligence and Applications).
3. *Bard G. V.* Algebraic Cryptanalysis. Boston, MA : Springer US, 2009.
4. *Prestwich S.* CNF Encodings // Handbook of Satisfiability. Washington, 2021. P. 75–100.
5. *Gomes C. P.*, *Sabharwal A.* Exploiting Runtime Variation in Complete Solvers // Handbook of Satisfiability. Vol. 185 / ed. by A. Biere [et al.]. IOS Press, 2009. P. 271–288. (Frontiers in Artificial Intelligence and Applications).
6. *Hachtel G. D.*, *Somenzi F.* Logic Synthesis and Verification Algorithms. New York : Springer New York, NY, 1996. 596 p.
7. *Shannon C. E.* A Symbolic Analysis of Relay and Switching Circuits // Transactions of the American Institute of Electrical Engineers. 1938. Dec. Vol. 57, no. 12. P. 713–723.
8. *Шестаков В. И.* Алгебра двухполюсных схем, построенных исключительно из двухполюсников (алгебра А-схем). 1941.
9. *Arora S.*, *Barak B.* Computational Complexity: A Modern Approach. Cambridge : Cambridge University Press, 2009.
10. *Cormen T. H.*, *Leiserson C. E.*, *Rivest R. L.* Introduction to Algorithms. 1st ed. MIT Press, 1990.
11. *Tseitin G. S.* On the Complexity of Derivation in Propositional Calculus // Studies in Constructive Mathematics and Mathematical Logic, Part II / ed. by A. Slisenko. Steklov Mathematical Institute, 1970. P. 115–125. (Seminars in Mathematics).
12. *Szeider S.* Backdoor Sets for DLL Subsolvers // Journal of Automated Reasoning. 2006. Oct. 5. Vol. 35, no. 1–3. P. 73–88. (Visited on 05/01/2024).

13. Circuit Complexity and Decompositions of Global Constraints / C. Bessière [et al.] // IJCAI'09: 21st International Joint Conference on Artificial Intelligence. Pasadena, CA, United States, 07/2009. P. 412–418. (Constraints, Satisfiability, and Search). URL: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00382608 (visited on 05/01/2024).

14. *Drechsler R., Junttila T. A., Niemelä I.* Non-Clausal SAT and ATPG // Handbook of Satisfiability. 1st ed. 2009. P. 655–693.

15. *Marques-Silva J., Lynce I., Malik S.* Conflict-Driven Clause Learning SAT Solvers // Handbook of Satisfiability. Vol. 185 / ed. by A. Biere [et al.]. IOS Press, 2009. P. 131–153. (Frontiers in Artificial Intelligence and Applications).

16. *Vyatkin V.* IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review // IEEE Transactions on Industrial Informatics Information. 2011. Vol. 7, no. 4. P. 768–781.

17. IEC 61131-1:2003. URL: https://webstore.iec.ch/publication/4550 (visited on 05/01/2024).

18. *Dubinin V., Vyatkin V.* Towards a Formal Semantic Model of IEC 61499 Function Blocks // IEEE International Conference on Industrial Informatics. IEEE, 2006. P. 6–11.

19. NuSMV: a new symbolic model checker / A. Cimatti [et al.] // International Journal on Software Tools for Technology Transfer. 2000. Vol. 2, no. 4. P. 410–425.

20. *Manna Z., Pnueli A.* Temporal Verification of Reactive Systems: Safety. Springer-Verlag, 1995. P. 512.

21. *Clarke E. M., Grumberg O., Peled D.* Model Checking. MIT Press, 1999. 330 p.

22. *Holzmann G.* The Model Checker SPIN // IEEE Transactions on Software Engineering. 1997. Май. Т. 23, № 5. С. 279—295.

23. Symbolic Model Checking without BDDs / A. Biere [et al.] // Tools and Algorithms for the Construction and Analysis of Systems / ed. by W. R. Cleaveland. Berlin, Heidelberg : Springer, 1999. P. 193–207. (Lecture Notes in Computer Science).

24. *Kroening D., Tautschnig M.* CBMC – C Bounded Model Checker // Tools and Algorithms for the Construction and Analysis of Systems. Т. 8413 / под ред. E. Ábrahám, K. Havelund ; под ред. D. Hutchison [и др.]. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014. С. 389—391.

25. *Hojjat H.*, *Rummer P.* The ELDARICA Horn Solver // 2018 Formal Methods in Computer Aided Design (FMCAD) (2018 Formal Methods in Computer Aided Design (FMCAD)). Austin, TX : IEEE, 10/2018. P. 1–7.

26. *Faymonville P.*, *Finkbeiner B.*, *Tentrup L.* BoSy: An Experimentation Framework for Bounded Synthesis // Computer Aided Verification. Springer, 2017. P. 325–332.

27. Encodings of Bounded Synthesis / P. Faymonville [et al.] // Tools and Algorithms for the Construction and Analysis of Systems. 2017. P. 354–370.

28. *Meyer P. J.*, *Sickert S.*, *Luttenberger M.* Strix: Explicit Reactive Synthesis Strikes Back! // Computer Aided Verification. Springer International Publishing, 2018. P. 578–586.

29. *Ehlers R.*, *Raman V.* Slugs: Extensible GR(1) Synthesis // Computer Aided Verification. Vol. 9780 / ed. by S. Chaudhuri, A. Farzan. Cham : Springer International Publishing, 2016. P. 333–339.

30. *Maoz S.*, *Ringert J. O.* Spectra: A Specification Language for Reactive Systems // Software and Systems Modeling. 2021. Oct. Vol. 20, no. 5. P. 1553–1586.

31. *Gold M.* Complexity of Automaton Identification from Given Data // Information and Control. 1978. Vol. 37, no. 3. P. 302–320.

32. *Rosner R.* Modular Synthesis of Reactive Systems. Wezimann Institute of Science, 1992. PhD thesis.

33. *Heule M. J. H.*, *Verwer S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications. Springer Berlin Heidelberg, 2010. P. 66–79.

34. *Ulyantsev V.*, *Buzhinsky I.*, *Shalyto A.* Exact finite-state machine identification from scenarios and temporal properties // International Journal on Software Tools for Technology Transfer. 2018. Vol. 20, no. 1. P. 35–55.

35. Efficient Symmetry Breaking for SAT-Based Minimum DFA Inference / I. Zakirzyanov [et al.] // Language and Automata Theory and Applications. Springer International Publishing, 2019. P. 159–173.

36. *Buzhinsky I.*, *Vyatkin V.* Automatic Inference of Finite-State Plant Models From Traces and Temporal Properties // IEEE Transactions on Industrial Informatics Information. 2017. Vol. 13, no. 4. P. 1521–1530.

37. *Tsarev F.*, *Egorov K.* Finite State Machine Induction Using Genetic Algorithm Based on Testing and Model Checking // Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation. ACM, 2011. P. 759–762.

38. *Giantamidis G.*, *Tripakis S.* Learning Moore Machines from Input-Output Traces // Formal Methods. Springer International Publishing, 2016. P. 291–309.

39. *Avellaneda F.*, *Petrenko A.* FSM Inference from Long Traces // Formal Methods. Springer, 2018. P. 93–109.

40. FSM inference and checking sequence construction are two sides of the same coin / A. Petrenko [et al.] // Software Quality Journal. 2019. P. 651–674.

41. *Neider D.*, *Topcu U.* An Automaton Learning Approach to Solving Safety Games over Infinite Graphs // Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2016. P. 204–221.

42. G4LTL-ST: Automatic Generation of PLC Programs / C.-H. Cheng [et al.] // Computer Aided Verification. Springer International Publishing, 2014. P. 541–549.

43. *Smetsers R.*, *Fiterău-Broştean P.*, *Vaandrager F.* Model Learning as a Satisfiability Modulo Theories Problem // Language and Automata Theory and Applications. Springer International Publishing, 2018. P. 182–194.

44. *Walkinshaw N.*, *Taylor R.*, *Derrick J.* Inferring extended finite state machine models from software executions // Empirical Software Engineering. 2015. Vol. 21, no. 3. P. 811–853.

45. *Finkbeiner B.*, *Klein F.* Bounded Cycle Synthesis // Computer Aided Verification. Springer International Publishing, 2016. P. 118–135.

46. CSP-based inference of function block finite-state models from execution traces / D. Chivilikhin [et al.] // 15th IEEE International Conference on Industrial Informatics. 2017. P. 714–719.

47. *Montanari U.* Networks of Constraints: Fundamental Properties and Applications to Picture Processing // Information Sciences. 1974. Jan. 1. Vol. 7. P. 95–132.

48. Counterexample-guided inference of controller logic from execution traces and temporal formulas / D. Chivilikhin [et al.] // 23rd IEEE International Conference on Emerging Technologies and Factory Automation. IEEE, 2018. P. 91–98.

49. Function Block Finite-State Model Identification Using SAT and CSP Solvers / D. Chivilikhin [et al.] // IEEE Transactions on Industrial Informatics. 2019. Vol. 15, no. 8. P. 4558–4568.

50. *Brand D.* Redundancy and Don't Cares in Logic Synthesis // IEEE Transactions on Computers. 1983. Oct. Vol. C–32, no. 10. P. 947–952.

51. *Drechsler R.*, *Junttila T.*, *Niemelä I.* Non-Clausal SAT and ATPG // Handbook of Satisfiability. 2-е изд. IOS Press, 2021. С. 1047—1086.

52. *Bryant R. E.* Graph-Based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers. 1986. Aug. Vol. C–35, no. 8. P. 677–691.

53. Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications / A. Biere [et al.]. IOS Press, 2009.

54. *Cook S. A.* The Complexity of Theorem-Proving Procedures // Proceedings of the Third Annual ACM Symposium on Theory of Computing. ACM, 1971. P. 151–158.

55. *Kautz H.*, *Sabharwal A.*, *Selman B.* Incomplete Algorithms // Handbook of Satisfiability. IOS Press, 2009. P. 185–203.

56. *Boros E.*, *Hammer P. L.* Pseudo-Boolean Optimization // Discrete Applied Mathematics. 2002. Nov. 15. Vol. 123, no. 1. P. 155–225.

57. *Burke E. K.*, *Kendall G.* Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. 2nd ed. New York Heidelber Dordrecht London : Springer, 2014. 716 p.

58. *MacWilliams F. J.*, *Sloane N. J. A.* The Theory of Error Correcting Codes. Transferred to digital printing. Amsterdam : Elsevier, 2007. 762 p. (North Holland Mathematical Library ; 16).

59. *Russell S. J.*, *Norvig P.* Artificial Intelligence: A Modern Approach. 4th ed. Hoboken : Pearson, 2021. (Pearson Series in Artificial Intelligence).

60. *Gu J.* Efficient Local Search for Very Large-Scale Satisfiability Problems // ACM SIGART Bulletin. 1992. Jan. Vol. 3, no. 1. P. 8–12.

61. *Schoning T.* A Probabilistic Algorithm for K-SAT and Constraint Satisfaction Problems // 40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039) (40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)). 10/1999. P. 410–414.

62. *Feller W.* An Introduction to Probability Theory and Its Applications. Vol. 2. 2nd ed. John Wiley & Sons, Inc., 1971.

63. *Dantsin E., Hirsch E. A.* Worst-Case Upper Bounds // Handbook of Satisfiability. IOS Press, 2009. P. 403–424.

64. *Papadimitriou C. H., Steiglitz K.* Combinatorial Optimization: Algorithms and Complexity. Englewood Cliffs, N.J : Prentice Hall, 1982. 496 p.

65. *Buchberger B.* Bruno Buchberger's PhD Thesis 1965: An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal // Journal of Symbolic Computation. 2006. Mar. 1. Vol. 41, no. 3. P. 475–511. (Logic, Mathematics and Computer Science: Interactions in Honor of Bruno Buchberger (60th Birthday)).

66. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization / G. Goos [et al.] // Advances in Cryptology — CRYPTO' 99. Vol. 1666 / ed. by M. Wiener. Berlin, Heidelberg : Springer Berlin Heidelberg, 1999. P. 19–30.

67. *Courtois N. T., Pieprzyk J.* Cryptanalysis of Block Ciphers with Overdefined Systems of Equations // Advances in Cryptology — ASIACRYPT 2002. Vol. 2501 / ed. by Y. Zheng ; red. by G. Goos, J. Hartmanis, J. Van Leeuwen. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002. P. 267–287.

68. *Davis M., Putnam H.* A Computing Procedure for Quantification Theory // Journal of the ACM. 1960. July. Vol. 7, no. 3. P. 201–215.

69. *Davis M., Logemann G., Loveland D.* A Machine Program for Theorem-Proving // Communications of the ACM. 1962. July. Vol. 5, no. 7. P. 394–397.

70. *Dowling W. F., Gallier J. H.* Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae // The Journal of Logic Programming. 1984. Oct. Vol. 1, no. 3. P. 267–284.

71. *Marques-Silva J. P., Sakallah K. A.* GRASP—A new search algorithm for satisfiability // Proceedings of International Conference on Computer Aided Design. IEEE Comput. Soc. Press, 1996. P. 220–227.

72. *Marques-Silva J., Sakallah K.* GRASP: A Search Algorithm for Propositional Satisfiability // IEEE Transactions on Computers. 1999. May. Vol. 48, no. 5. P. 506–521.

73. Chaff: Engineering an Efficient SAT Solver / M. Moskewicz [et al.] // Proceedings of the 38th Design Automation Conference (Proceedings of the 38th Design Automation Conference). 06/2001. P. 530–535.

74. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver / L. Zhang [et al.] // IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281). 11/2001. P. 279–285.

75. *Eén N., Sörensson N.* An Extensible SAT-solver // Theory and Applications of Satisfiability Testing. Springer Berlin Heidelberg, 2003. P. 502–518.

76. *Audemard G., Simon L.* Predicting Learnt Clauses Quality in Modern SAT Solvers // Proceedings of the 21st International Jont Conference on Artifical Intelligence. Morgan Kaufmann Publishers Inc., 2009. P. 399–404.

77. *Hyvärinen A.* Grid Based Propositional Satisfiability Solving : PhD thesis / Hyvärinen Antti. School of Science : Aalto University, 2011. 107 p. URL: `http://lib.tkk.fi/Diss/2011/isbn9789526043685/isbn9789526043685.pdf` (visited on 05/01/2024).

78. *Hamadi Y., Jabbour S., Sais J.* Control-Based Clause Sharing in Parallel SAT Solving // Autonomous Search / ed. by Y. Hamadi, E. Monfroy, F. Saubion. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. P. 245–267.

79. *Razborov A. A.* Proof Complexity of Pigeonhole Principles // Developments in Language Theory. Vol. 2295 / ed. by W. Kuich, G. Rozenberg, A. Salomaa ; red. by G. Goos, J. Hartmanis, J. Van Leeuwen. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002. P. 100–116.

80. *Cook S. A., Reckhow R. A.* The Relative Efficiency of Propositional Proof Systems // Journal of Symbolic Logic. 1979. Mar. Vol. 44, no. 1. P. 36–50.

81. *Robinson J. A.* A Machine-Oriented Logic Based on the Resolution Principle // Journal of the ACM. 1965. T. 12, № 1. C. 23—41.

82. *Beame P., Kautz H., Sabharwal A.* Understanding the Power of Clause Learning // International Joint Conference on Artificial Intelligence. 2003.

83. *Beame P., Kautz H., Sabharwal A.* Towards Understanding and Harnessing the Potential of Clause Learning // Journal of Artificial Intelligence Research. 2004. Dec. 1. Vol. 22. P. 319–351.

84. *Haken A.* The Intractability of Resolution // Theoretical Computer Science. 1985. Jan. 1. Vol. 39. P. 297–308. (Third Conference on Foundations of Software Technology and Theoretical Computer Science).

85. *Ben-Sasson E.*, *Impagliazzo† R.*, *Wigderson‡ A.* Near Optimal Separation Of Tree-Like And General Resolution // COMBINATORICA. 2004. Sept. Vol. 24, no. 4. P. 585–603.

86. SAT Competitions. URL: http://www.satcompetition.org/ (visited on 05/01/2024).

87. Learning Rate Based Branching Heuristic for SAT Solvers / J. Liang [et al.] // Theory and Applications of Satisfiability Testing - SAT 2016. Springer International Publishing, 2016. P. 123–140.

88. CaDiCaL Simplified Satisfiability Solver. URL: https://github.com/arminbiere/cadical (visited on 05/01/2024).

89. *Soos M.*, *Nohl K.*, *Castelluccia C.* Extending SAT Solvers to Cryptographic Problems // Theory and Applications of Satisfiability Testing. Springer Berlin Heidelberg, 2009. P. 244–257.

90. Lingeling, Plingeling and Treengeling. URL: http://fmv.jku.at/lingeling/ (visited on 05/01/2024).

91. *Bailleux O.*, *Boufkhad Y.* Efficient CNF Encoding of Boolean Cardinality Constraints // Principles and Practice of Constraint Programming. Springer Berlin Heidelberg, 2003. P. 108–122.

92. *Metropolis N.*, *Ulam S.* The Monte Carlo Method // Journal of the American Statistical Association. 1949. Sept. Vol. 44, no. 247. P. 335–341.

93. Evaluating the Hardness of SAT Instances Using Evolutionary Optimization Algorithms / A. Semenov [et al.] // (27th International Conference on Principles and Practice of Constraint Programming). 2021. P. 18.

94. *Williams R.*, *Gomes C. P.*, *Selman B.* Backdoors to Typical Case Complexity // Proceedings of the 18th International Joint Conference on Artificial Intelligence. Vol. 3 (IJCAI). San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 08/09/2003. P. 1173–1178.

95. Measuring the Hardness of SAT Instances / C. Ansótegui [и др.] // Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1. Chicago, Illinois : AAAI Press, 13.07.2008. С. 222—228. (AAAI'08).

96. *Wilks S. S.* Mathematical Statistics. 2nd ed. New York : Wiley, 1962. 644 p. (A Wiley Publication in Mathematical Statistics).

97. Combinatorial Sketching for Finite Programs / A. Solar-Lezama [et al.] // ACM SIGOPS Operating Systems Review. 2006. Vol. 40, no. 5. P. 404–415.

98. Counterexample Guided Inductive Synthesis modulo Theories / A. Abate [et al.] // Computer Aided Verification. Springer International Publishing, 2018. P. 270–288.

99. The 5th Reactive Synthesis Competition (SYNTCOMP 2018): Benchmarks, Participants & Results / S. Jacobs [et al.]. 2019. arXiv: 1904.07736 [cs.LO].

100. *Chukharev K.* fbSAT Tool / Computer Technologies Laboratory. URL: https://github.com/ctlab/fbSAT (дата обр. 29.04.2024).

101. *Biere A.* PicoSAT Essentials // Journal on Satisfiability, Boolean Modeling and Computation / ed. by E. Speckenmeyer [et al.]. 2008. May 1. Vol. 4, no. 2–4. P. 75–97.

102. The Kissat SAT Solver. URL: https://github.com/arminbiere/kissat (visited on 05/01/2024).

103. *Walsh T.* SAT v CSP // 6th International Conference on Principles and Practice of Constraint Programming. Springer Berlin Heidelberg, 2000. P. 441–456.

104. *Nguyen V.-H., Mai S. T.* A New Method to Encode the At-Most-One Constraint into SAT // Proceedings of the Sixth International Symposium on Information and Communication Technology (SoICT 2015: The Sixth International Symposium on Information and Communication Technology). Hue City Viet Nam : ACM, 12/03/2015. P. 46–53.

105. *Björk M.* Successful SAT Encoding Techniques // Journal on Satisfiability, Boolean Modeling and Computation. 2009. Vol. 7. P. 189–201.

106. *Petke J., Jeavons P.* The Order Encoding: From Tractable CSP to Tractable SAT // Theory and Applications of Satisfiability Testing - SAT 2011. Vol. 6695. Springer Berlin Heidelberg, 2011. P. 371–372.

107. What's Hard About Boolean Functional Synthesis? / S. Akshay [et al.] // Computer Aided Verification / ed. by H. Chockler, G. Weissenbacher. Cham : Springer International Publishing, 2018. P. 251–269.

108. Logic Minimization Algorithms for VLSI Synthesis. Vol. 2 / R. K. Brayton [et al.]. Boston, MA : Springer US, 1984. (The Kluwer International Series in Engineering and Computer Science).

109. *Fiser P., Kubatova H.* Flexible Two-Level Boolean Minimizer BOOM-II and its Applications // 9th EUROMICRO Conference on Digital System Design (DSD'06) (9th EUROMICRO Conference on Digital System Design (DSD'06)). 08/2006. P. 369–376.

110. *Ulyantsev V., Zakirzyanov I., Shalyto A.* BFS-Based Symmetry Breaking Predicates for DFA Identification // Language and Automata Theory and Applications. Springer International Publishing, 2015. P. 611–622.

111. *Walsh T.* SAT v CSP // 6th International Conference on Principles and Practice of Constraint Programming. Springer Berlin Heidelberg, 2000. P. 441–456.

112. nxtControl - nxtStudio. URL: http://www.nxtcontrol.com/en/engineering (visited on 05/01/2024).

113. Benchmarks submission guidelines. URL: http://www.satcompetition.org/2009/format-benchmarks2009.html (visited on 05/01/2024).

114. *Chukharev K.* Wrapper for incremental SAT solving using Cryptominisat. URL: https://github.com/Lipen/incremental-cryptominisat (visited on 06/17/2020).

115. JNI APIs and Developer Guides. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jni/ (visited on 05/01/2024).

116. *Гречишкина Д., Чухарев К.* Программный интерфейс для SAT-решателей на основе технологии JNI // Сборник тезисов докладов конгресса молодых ученых. Электронное издание. СПб: Университет ИТМО, 2020. URL: https://kmu.itmo.ru/digests/article/4456 (дата обр. 17.06.2020).

117. Closed-Loop Modeling in Future Automation System Engineering and Validation / V. Vyatkin [et al.] // IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews. 2009. Vol. 39, no. 1. P. 17–28.

118. *Patil S.*, *Vyatkin V.*, *Sorouri M.* Formal verification of Intelligent Mechatronic Systems with decentralized control logic // IEEE Conference on Emerging Technologies and Factory Automation. IEEE, 2012. P. 1–7.

119. Automatic State Machine Reconstruction from Legacy PLC Using Data Collection and SAT Solver / D. Chivilikhin [et al.] // IEEE Transactions on Industrial Informatics. 2020. Vol. 16, issue 12. P. 7821–7831.

120. *Karp R. M.*, *Luby M.*, *Madras N.* Monte-Carlo Approximation Algorithms for Enumeration Problems // Journal of Algorithms. 1989. Sept. Vol. 10, no. 3. P. 429–448.

121. *Eén N.*, *Sörensson N.* Translating Pseudo-Boolean Constraints into SAT // Journal on Satisfiability, Boolean Modeling and Computation / ed. by D. Le Berre, L. Simon. 2006. Mar. 1. Vol. 2, no. 1–4. P. 1–26.

122. *Knuth D. E.* The Art of Computer Programming: Volume 2: Seminumerical Algorithms : in 7 vols. Vol. 2. 3rd ed. Massachusetts : Addison-Wesley, 1997. 762 p. URL: `https://www-cs-faculty.stanford.edu/~knuth/taocp.html` (visited on 05/01/2024).

123. *Dadda L.* Some schemes for parallel multipliers // Alta Frequenza. 1965. May. Vol. 34, no. 5. P. 349–356.

124. *Kaufmann D.*, *Biere A.*, *Kauers M.* Verifying Large Multipliers by Combining SAT and Computer Algebra // 2019 Formal Methods in Computer Aided Design (FMCAD). 2019. P. 28–36.

125. *Zaikin O.* Inverting 43-Step MD4 via Cube-and-Conquer //. Vol. 3 (Thirty-First International Joint Conference on Artificial Intelligence). International Joint Conferences on Artificial Intelligence Organization, 07/16/2022. P. 1894–1900.