Федеральное государственное автономное образовательное учреждение
высшего образования Университет ИТМО

обновлено: 8 мая 2024 г.

Методы декомпозиции задачи булевой выполнимости для синтеза и верификации дискретных управляющих моделей

Чухарев Константин

Специальность 2.3.5 (05.13.11)

Математическое и программное обеспечение вычислительных машин,
комплексов и компьютерных сетей

Диссертация
на соискание учёной степени
кандидата технических наук

Научный руководитель:
Семёнов Александр Анатольевич
к.т.н., доцент

Санкт-Петербург, Россия
2024

Оглавление

Введение

Актуальность темы. В современном мире существует множество различных приложений, в которых возникают задачи большой размерности. Особенно это актуально в контексте задач синтеза и верификации дискретных управляющих (ДУ) систем, таких как конечные автоматы и цифровые схемы, которые используются в различных областях, включая робототехнику, электронику, информационную безопасность. Основыми проблемами в области дискретных систем управления являются их синтез и верификация. Синтез ДУ-модели заключается в построении модели системы, которая обладает заданными свойствами. Верификация ДУ-модели заключается в проверке того, что модель удовлетворяет заданной спецификации. Важно отметить, что задачи синтеза и верификации ДУ-моделей являются NP-полными. Это означает, что на сегодняшний день не существует эффективных алгоритмов, способных решать эти задачи за разумное время для всех возможных входных данных. Это приводит к тому, что существующие методы не всегда применимы для решения задач большой размерности. Таким образом, развитие методов и алгоритмов, способных решать подобные задачи более эффективно, остается актуальной и важной задачей в области дискретной математики и компьютерных наук.

Распространённым подходом к *автоматическому* синтезу и верификации является *сведение* к классическим NP-полным задачам, таким как задача выполнимости булевой формулы (Boolean satisfiability problem, SAT), задача максимальной выполнимости (MaxSAT) и задача выполнимости в теориях (Satisfiability Modulo Theory, SMT), с последующим применением так называемых *решателей*, реализующих современные алгоритмы решения этих задач. Данные задачи являются «универсальными», в том смысле что они могут быть использованы для решения широкого класса задач, исключая тем самым необходимость разработки специализированных алгоритмов для каждой конкретной задачи.

Одной из центральных проблем в области синтеза и верификации ДУ-моделей является отсутствие априорных оценок времени работы алгоритмов решения задачи SAT. В рамках данной диссертационной работы предлагаются и развиваются методы и алгоритмы, которые позволяют строить такие оценки. Для этого используются специальные декомпозиционные представления булевых формул. Идеи построения декомпозиций уже выдвигались ранее, но они обладают меньшей точностью. Методы декомпозиции, предложенные в данной работе, учитывают особенности исходной

задачи, которая решается с помощью сведения к SAT, например, особенности задачи синтеза конечно-автоматных моделей или задачи проверки эквивалентности логических схем.

Резюмируя, предложенные в диссертации методы и алгоритмы позволяют повысить эффективность комбинаторных алгоритмов в применении к задачам синтеза и верификации ДУ-моделей. Эти методы могут быть использованы для решения различных задач в области автоматизированного проектирования и управления дискретными системами.

Цель работы. Целью данной работы является повышение эффективности работы полных алгоритмов решения задачи булевой выполнимости (SAT) в применении к задачам синтеза и верификации ДУ-моделей за счет оригинальных методов и техник декомпозиции булевых формул.

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

1. Разработаны оригинальные алгоритмы кодирования в SAT задач синтеза конечно-автоматных моделей с заданным поведением и свойствами, отличающиеся от существующих добавлением кодирования структуры охранных условий в виде деревье разбора соответствующих формул.
2. Разработаны оригинальные алгоритмы кодирования в SAT задачи синтеза модульных конечно-автоматных моделей с заданным поведением и свойствами, отличающиеся от существующих автоматизированным модульным разбиением.
3. Разработаны оригинальные алгоритмы кодирования в SAT задачи синтеза булевых схем и булевых формул по заданной таблице истинности, отличающиеся от существующих возможностью использования произвольных элементарных гейтов.
4. Разработаны оригинальные методы оценивания декомпозиционной трудности булевых формул, кодирующих задачи синтеза конечно-автоматных моделей и верификации булевых схем, отличающиеся от существующих учётом особенностей исходной задачи, а также низкой дисперсией времени решения подзадач.
5. С применением разработанных алгоритмов решены трудные примеры задач синтеза ДУ-моделей (как конечно-автоматных моделей, так и логических схем).

6. Разработан новый SAT решатель, использующий вероятностные лазейки для вывода новой информации при работе с трудными булевыми формулами.
7. Разработана программная библиотека для взаимодействия с SAT-решателями через API (программный интерфейс).
8. Проведены масштабные вычислительные эксперименты для подтверждения эффективности всех разработанных методов.

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

1. Новые алгоритмы синтеза конечно-автоматных и модульных конечно-автоматных моделей, основанные на сведении к проблеме булевой выполнимости (SAT).
2. Новые методы оценивания декомпозиционной трудности булевых формул, кодирующих задачи синтеза ДУ-моделей.
3. Оригинальные алгоритмы решения трудных инстансов задачи SAT, использующие объединение нескольких вероятностных лазеек.
4. Решение экстремально трудных задач синтеза ДУ-моделей при помощи разработанных алгоритмов.

Основные положения, выносимые на защиту.

1. Оригинальные алгоритмы кодирования в SAT задачи синтеза конечно-автоматных моделей, отличающиеся от существующих добавлением кодирования структуры охранных условий в виде деревье разбора соответствующих формул.
2. Оригинальные алгоритмы кодирования в SAT задачи синтеза булевых схем и булевых формул по заданной таблице истинности, отличающиеся от существующих возможностью использования произвольных элементарных гейтов.
3. Оригинальные алгоритмы оценивания декомпозиционной трудности булевых формул применительно к задачам верификации логических схем, отличающиеся от существующих учётом особенностей исходной задачи.
4. Программная библиотека[1,2] для взаимодействия с различными SAT-решателями через API (программный интерфейс), отличающаяся от существующих полнотой, гибкостью и удобством.

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

[2]Для языка Rust: `https://github.com/Lipen/sat-nexus`

Теоретическая и практическая значимость. Теоретическая значимость диссертации заключается в разработанных в ней концепциях и алгоритмах решения задач синтеза ДУ-моделей и методах построения оценок трудности таких задач. Практическая значимость диссертации состоит в том, что основные разработанные в ней алгоритмы применимы к индустриальным задачам синтеза и верификации ДУ-моделей, а также в том, что на целом ряде конкретных примеров практическая реализация и апробация разработанных алгоритмов демонстрируют лучшую эффективность в сравнении с известными подходами.

Методы и инструменты исследования. Теоретическая часть работы использует методологию дискретной математики и математической логики, теории вычислительной сложности,а также теорию эволюционных вычислений. Для синтеза конечно-автоматных моделей был использован программный комплекс fbSAT, разработанный в рамках данной диссертации[3]. При построении вычислительных задач из области проверки логической эквивалентности схем использовалась программная система Transalg[4]. Для решения конкретных инстансов задачи SAT использовались различные современные SAT-решатели, находящиеся в открытом доступе, такие как MiniSAT, Glucose, Kissat, CaDiCaL. В вычислительных экспериментах задействовался вычислительный кластер.

Соответствие специальность. Содержание научно-квалификационной работы охватывает такие направления как: синтез и верификация управляющих систем дискретной природы; разработку проблемно-ориентированных комбинаторных алгоритмов, применимых к автоматическому проектированию и верификации ДУ-моделей; разработку алгоритмов декомпозиции сложных экземпляров комбинаторных задач; разработку программных средств для эффективного взаимодействия и SAT-решателями; разработку специализированных SAT-решателей, учитывающих особенности решаемых задач. Программная библиотека, являющаяся одним из основных практических результатов диссертации, обеспечивает эффективное взаимодействие с SAT-решателями через программный интерфейс (API), а также содержит дополнительный функционал для кодирования (сведения) комбинаторных задач в SAT. Таким образом, можно утверждать, что работа соответствует паспорту специальности 2.3.5 (05.13.11) в пунктах 1 и 3.

Достоверность результатов проведённых исследований. [TODO]

[3] https://github.com/ctlab/fbSAT

[4] https://gitlab.com/transalg/transalg

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.

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.

Публикации по теме диссертации. Основные результаты по теме диссертации изложены в 9 публикациях. Из них 4 опубликовано в изданиях, индексируемых в базе цитирования Scopus.

Структура и объём работы. Диссертация состоит из введения, 3 глав, заключения и 0 приложений. Полный объём диссертации составляет 68 страниц, включая 6 рисунков и 4 таблицы. Список литературы содержит 62 наименования.

Глава 1. Обзор предметной области

В данной главе представлены основные понятия и существующие результаты.

1.1 Дискретные управляющие модели

Здесь описываются дискретные управляющие модели и их применение.

1.1.1 Конечные автоматы

Конечный автомат (КА) — это модель вычислений, которая описывает систему с конечным числом состояний и переходов между ними. Конечный автомат может быть задан в виде пятерки $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$, 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 $s_n \in F$.

1.1.2 Булевы схемы

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 $v \in V \setminus V^{\text{in}}$ 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.

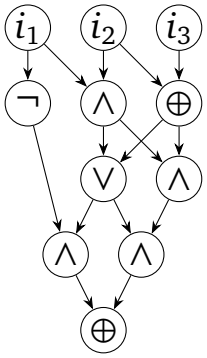

Рисунок 1 — Пример булевой схемы с тремя входами (i_1, i_2, i_3) и восьмью гейтами

Булева схема с n входами и m выходами естественным образом задает (тотальную) дискретную функцию $f\colon \{0,1\}^n \to \{0,1\}^m$, где под $\{0,1\}^k$ мы понимаем множество всех возможных двоичных слов длины $k \in \mathbb{N}^+$. Имея это в виду, мы 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. Проходя по гейтам схемы в фиксированном порядке (обычно указанном топологической сортировкой [1]) и вычисляя значения элементарных функций, сопоставленных гейтам, мы получаем 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 $X^{\text{in}} = \{x_1, \ldots, x_n\}$.

[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*.

Переменные, связанные с гейтами, мы будем называть *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:

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

Мы будем обозначать (1) как *шаблонную КНФ* для функции f. Заметим, что C_f является КНФ-формулой, полученной применением преобразований Цейтина [2] к схеме S_f.

Ниже, следуя работе [3], 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. [4; 5]. Он использует простой механизм булевой дедукции, известный как правило распространения единичного дизъюнкта (Unit Propagation — UP) [6].

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 $x_1^{\alpha_1} \wedge \cdots \wedge x_n^{\alpha_n} \wedge C_f$ для любого $\alpha = (\alpha_1, \ldots, \alpha_n)$. Лемма 1 очень полезна при доказательстве свойств, связанных с булевыми схемами и SAT.

1.2 Задачи синтеза и верификации дискретных управляющих моделей

1.2.1 Международный стандарт IEC 61499

Международный стандарт распределенных систем управления и автоматизации IEC 61499 [7] нацелен на упрощение разработки распределенных киберфизических систем. Стандарт отличается от «предыдущего» стандарта IEC 61131[8] 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).

1.2.2 Методы синтеза конечно-автоматных моделей

Задача поиска минимального детерминированного конечно автомата по примерам поведения является NP-полной задачей [9], а сложность задачи LTL-синтеза дважды экспоненциальная от размера LTL-спецификации. Не смотря на это, синтез различных типов конечно-автоматных моделей по примерам поведения и/или формальной спецификации был исследован во многих научных работах [10—21], 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 [11; 22], основанных на сведении к задаче SAT. В [11] LTL-спецификация учитывается путём применения итеративного подхода запрета контрпримеров. Существенным недостатком [11] 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 [22] решается задача синтеза вычислимых выходных алгоритмов, однако предполагается, что базовая модель автомата (то есть 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 [14; 23] 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 [24] 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. Не смотря 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 [20] и Strix [25], обладают аналогичными недостатками по отношению к рассматриваемой задаче, а именно, отсутствие минимизации охранных условий и невозможность (эффективного) учета примеров поведения.

В статье [26] предлагается метод FBCSP для синтеза конечно-автоматных моделей функциональных блоков по примерам поведения, основанный на сведении к задаче удовлетворения ограничений (*Constraint Satisfaction Problem* — CSP). 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*), то есть некорректно работают на входных данных, которые не были использованы в процессе «обучения» (синтеза). В [26] это отчасти исправляется дополнительной жадной минимизацией охранных условий, однако жадный подход не гарантирует, что охранные условия будут наименьшими. В работе [27] метод FBCSP был расширен процедурой запрета контрпримеров для учета LTL-спецификации (в дальнейшем это расширение будет называться FBCSP+LTL), аналогично работе [11]. 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 [28] 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 [28] — 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.

1.2.3 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 [29], а рассмотренные спецификация систем были составлены на языке линейной темпоральной логики (*Linear Temporal Logic* — LTL) [30], 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 [31].

1.2.4 Синтез булевых формул и схем

[TODO синтез формулы/схемы по таблице истинности для булевой функции]

1.2.5 Задача проверки эквивалентности булевых схем

Задача проверки эквивалентности булевых схем (Logical Equivalence Checking — LEC) является одной из ключевых комбинаторных проблем в автоматизации проектирования электроники. В этом разделе даны основные понятия о LEC, которые будут использованы в дальнейшем.

Рассмотрим две булевы схемы S_f, 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, то есть 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.

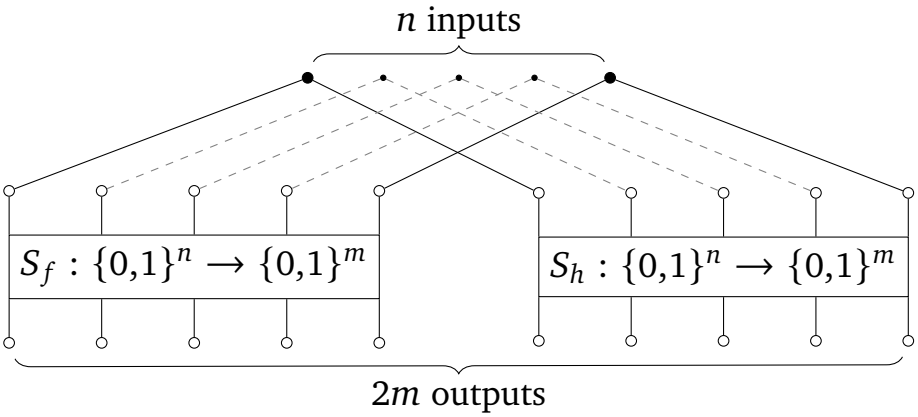

Рисунок 2 — Склеенная схема $S_{f \triangle h}$, построенная с использованием одного и того же набора входов для двух схем S_f и S_h

Используя S_f, S_h, построим новую схему, обозначаемую $S_{f \triangle h}$ (см. Рисунок 2), 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 S_f и S_h, и определяет следующую функцию:

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

Обозначим через V_f^{out} и V_h^{out} множества выходов схем S_f, 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* [32]. 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

$$C_{f \oplus h} = C_{f \triangle h} \wedge C(\mathcal{M}), \tag{3}$$

где $C_{f \triangle h}$ — шаблонная CNF для функции (2), а $C(\mathcal{M})$ выглядит следующим образом:

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

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

1.2.6 Задача генерации тестовых шаблонов для верификации булевых схем

В этом разделе рассматривается задача *автоматической* генерации тестовых шаблонов (Automatic Test Pattern Generation — ATPG) для верификации булевых схем. Сначала вводится понятие модели неисправностей (*fault model*). Затем формулируется задача генерации шаблонов (ATPG) для комбинационных (*combinational*) булевых схем. Также упоминается секвенциальная (*sequential*) постановка задачи ATPG для схем с элементами памяти, такими как триггеры (*flip-flops*). Наконец, кратко рассматриваются классические алгоритмы ATPG, работающие на структуре схемы. В разделе **??** отдельно рассматриваются методы решения задачи ATPG, основанные на сведении к задаче выполнимости (SAT), как наиболее релевантные к текущей работе.

Модель неисправности Stuck-At

После производства чипа необходимо проверить его функциональную корректность относительно спецификации схемы на уровне булевых элементов. Без этой проверки некорректный чип будет доставлен заказчикам, что может привести к неполадкам в конечном продукте. Это, конечно же, недопустимо. С другой стороны,

из-за дефектов материала, вариаций процесса во время производства и т. д. возможен широкий диапазон неисправностей. Но непосредственная проверка всех возможных физических дефектов невозможна. Поэтому вводится абстракция в виде модели неисправности. Модель неисправности Stuck-At (SAFM) [BF76] хорошо известна и широко используется на практике. В этой модели неисправности предполагается, что одна линия застряла на фиксированном значении вместо зависимости от значений входов. Когда линия застряла на значении 0, это называется неисправностью stuck-at-0 (SA0). Аналогично, если линия застряла на значении 1, это называется неисправностью stuck-at-1 (SA1).

Пример. Рассмотрим схему, показанную на рисунке 27.7(a). Когда на линии d вводится неисправность SA0, получается неисправная схема, показанная на рисунке 27.7(b). Выход элемента И отключается, и вход элемента ИЛИ постоянно принимает значение 0.

Помимо SAFM было предложено ряд других моделей неисправностей, например, клеточная модель неисправностей [Fri73], где меняется функция одного элемента, или модель мостовой неисправности [KP80], где предполагается, что две линии устанавливаются в одно значение. Эти модели неисправностей в основном охватывают статические физические дефекты, такие как обрывы или замыкания. Динамические эффекты охватываются моделями неисправностей задержки. В модели неисправности по задержке пути [Smi85] одна неисправность означает, что изменение значения вдоль пути от входов к выходам в схеме не приходит в течение времени цикла тактового сигнала. Вместо путей модель неисправности задержки на входах элементов [HRVD77, SB77] учитывает задержку в элементах.

Далее рассматривается только SAFM из-за его высокой значимости в практических применениях. Эту значимость можно объяснить двумя наблюдениями: количество неисправностей имеет порядок размера схемы, и моделирование неисправностей в SAFM относительно просто, то есть для статической модели неисправностей вычислительная сложность генерации тестовых шаблонов ниже по сравнению с динамическими моделями неисправностей.

Комбинационный ATPG

Автоматическая генерация тестовых шаблонов (ATPG) — это задача определения всех набора тестовых шаблонов для заданной схемы с учетом модели неисправностей. Тестовый шаблон для конкретной неисправности — это назначение на основные входы схемы, которое приводит к различным выходным значениям в зависимости от наличия неисправности. Вычисление булевой разности между бездефектной и неисправной схемами дает все тестовые шаблоны для конкретной неисправности. Эта конструкция аналогична схеме *miter* [32], поскольку ее можно использовать для проверки эквивалентности комбинационных схем.

Пример. Снова рассмотрим неисправность SA0 в схеме на рисунке 27.7. Назначение входов $a = 1$, $b = 1$, $c = 1$ приводит к значению выхода $f = 1$ для корректной схемы и к значению выхода $f = 0$ в случае наличия неисправности. Поэтому это назначение входов является тестовым шаблоном для неисправности SA0 на линии d. Конструкция для вычисления булевой разности бездефектной и неисправной схем показана на рисунке 27.8.

Когда тестовый шаблон существует для конкретной неисправности, эта неисправность классифицируется как *тестируемая* (*testable*). Если тестовый шаблон отсутствует, неисправность называется *избыточной* (*redundant*). Проблема классификации неисправности как тестируемой или избыточной является NP-полной. Задача ATPG заключается в классификации всех неисправностей и создании набора тестовых шаблонов, содержащего хотя бы один тестовый шаблон для каждой испытуемой неисправности.

Генерация тестовых шаблонов для схем, содержащих элементы состояния (памяти), такие как триггеры (*flip-flops*), вычислительно более сложна, потому что элементы памяти не могут быть непосредственно установлены в определенное значение. Вместо этого поведение схемы во времени должно рассматриваться во время ATPG. Было предложено ряд инструментов, которые непосредственно решают эту секвенциальную проблему, например, HITEC [NP91]. Но на практике получаемая модель часто слишком сложна для обработки с помощью инструментов ATPG. Поэтому обычно рассматривается полный режим сканирования для преодоления этой проблемы путем подключения всех элементов состояния в цепочку сканирования [WA73, EW77]. В режиме тестирования цепочка сканирования объединяет все элементы состояния в сдвиговый регистр, в нормальном режиме работы элементы

состояния управляются обычной логикой в схеме. В результате элементы состояния могут рассматриваться как основные входы и выходы для тестирования, и получается комбинационная постановка задачи ATPG, уже рассмотренная выше.

1.3 Задача булевой выполнимости

Задача выполнимости булевых формул (Boolean satisfiability problem — SAT) формулируется следующим образом [33]: для произвольной булевой формулы $\varphi(x_1, \ldots, x_n)$ необходимо определить, существует ли подстановка значений переменных X_{SAT}, при которой формула становится истинной, то есть, формально, $\exists X_{\text{SAT}} \in \{0,1\}^n : \varphi(X_{\text{SAT}}) = 1$. Если такая подстановка 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*).

Задача SAT является первой задачей, для которой была доказана NP-полнота [34]. [TODO Описание универсальности задачи SAT]

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:

$$n \text{ конъюнкций} \begin{cases} (x_1 \wedge y_1) \vee \\ (x_2 \wedge y_2) \vee \\ \ldots \\ (x_n \wedge y_n) \end{cases} \xrightarrow{\text{КНФ}} \begin{cases} (x_1 \vee x_2 \vee \cdots \vee x_n) \wedge \\ (y_1 \vee x_2 \vee \cdots \vee x_n) \wedge \\ \ldots \\ (y_1 \vee y_2 \vee \cdots \vee y_n) \end{cases} 2^n \text{ дизъюнкций}$$

С помощью преобразований Цейтина [2] 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, подвергаются либо эквивалентным логическим преобразованиям, либо преобразованиям Цейтина, то есть по итогу представляются в виде КНФ.

1.3.1 Алгоритм DPLL

[TODO Описание алгоритма DPLL]

1.3.2 Алгоритм CDCL

Conflict-Driven Clause Learning (CDCL) — это расширение классического алгоритма DPLL, включающее в себя механизмы анализа конфликтов и вывода новых дизъюнктов. Алгоритм CDCL лежит в основе многих современных SAT-решателей благодаря своей эффективности при решении экземпляров задачи SAT. Ниже приводится подробное описание алгоритма CDCL, а также псевдокод для наглядности.

[TODO Описание алгоритма CDCL]

Algorithm 1: DPLL Algorithm with Conflict Analysis and Clause Learning

Input: Boolean formula F, current assignment σ
Output: satisfying assignment or indication of unsatisfiability

```
while not all variables assigned do
    Branch on unassigned variable v
    if F becomes unsatisfiable under σ ∪ {v} then
        β ← AnalyzeConflict(F, σ, v)
        F ← F ∪ {β}
        Backtrack to previous decision level
    else
        Continue recursive exploration
```

1.3.3 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 [35]: среди текущих лидеров можно выделить MapleCOMSPS [36], Cadical [37], CryptoMiniSat [38], Glucose [39] и Plingeling [40], хотя на практике эффективность решателей может значительно отличаться, в зависимости от класса рассматриваемых задач. В некоторых случаях хорошие результаты также показывает MiniSat [41], являющийся минимальной реализацией CDCL-решателя (*Conflict-Driven Clause Learning* [42]) и служащий основой для многих других решателей (например, CryptoMiniSat и Glucose).

[TODO Инкрементальность]

1.3.4 Методы сведения задач к SAT

[TODO Пример сведения задачи раскраски графа к SAT – описание переменных и ограничений. Дополнительно – оптимизационная постановка задачи.]

[TODO LEC as SAT]

[TODO ATPG as SAT]

1.3.5 Декомпозиционная трудность

Концепция *лазеек* (*backdoors*) была введена в классической работе [43]. 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 (то есть 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 [44], что если B — некоторый SBS, то сложность C ограничена сверху значением $poly(|C|) \cdot 2^{|B|}$, где $poly(\cdot)$ — некоторый полином.

В статье [45] 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:

$$\mu_{A,B}(C) = \sum_{\beta \in \{0,1\}^{|B|}} t_A(C[\beta/B]) \tag{4}$$

Минимальное значение (4) по всем возможным множествам $B \in 2^X$ называется *декомпозиционной трудностью* (*decomposition hardness*) формулы C относительно алгоритма A.

Как показано в [45], значение (4) можно выразить с использованием математического ожидания случайной величины ξ_B, связанной с множеством B, которая задается следующим соотношением:

$$\mu_{A,B}(C) = 2^{|B|} \cdot \mathbb{E}[\xi_B] \tag{5}$$

Для оценки значения (4) можно использовать метод Монте-Карло и формулу (**??**). 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 [45] для этой цели использовались метаэвристические алгоритмы.

Выводы по главе 1

[TODO Завершение обзора]

Глава 2. Синтез конечно-автоматных моделей на основе сведения к задаче булевой выполнимости (SAT)

Данная глава посвящена решению задачи синтеза монолитных конечно-автоматных моделей логических контроллеров по примерам поведения и формальной спецификации. В разделе 2.1 предлагается метод синтеза по примерам поведения, основанный на сведении к задаче выполнимости SAT, приводится описание разработанных алгоритмов: Basic — для синтеза базовых моделей, Extended — для синтеза расширенных моделей, Complete — для учета негативных сценариев выполнения. В разделе 2.2 рассматривается задача синтеза минимальных моделей, приводится описание разработанных алгоритмов Basic-min, Extended-min, Complete-min и Extended-min-UB. В разделе 2.3 рассматривается подход индуктивного синтеза, основанного на контрпримерах (*Counterexample-Guided Inductive Synthesis* — CEGIS) [46; 47], используемый для учета при синтезе формальной спецификации, приводится описание алгоритмов CEGIS и CEGIS-min. Раздел 2.5 содержит экспериментальное сравнение разработанных методов с существующими на примере задачи синтеза конечно-автоматной модели логического контроллера, управляющего Pick-and-Place манипулятором. Раздел 2.6 посвящен применению разработанных методов для минимизации *систем переходов*, полученных с помощью программного средства для LTL-синтеза BoSy [14; 23] по исходным данным с соревнования по реактивному синтезу SYNTCOMP [48]. Все разработанные в данной работе методы реализованы в виде программного средства fbSAT [49].

2.1 Метод синтеза конечно-автоматных моделей монолитных логических контроллеров по примерам поведения

В этом разделе приводится описание разработанного метода синтеза минимальных моделей базовых функциональных блоков по примерам поведения и LTL-спецификации. 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).

2.1.1 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[1]. 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.

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

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

нулевые (несуществующие) переходы: $(\tau_{q,k} = q_0) \iff (\xi_{q,k} = \varepsilon)$. Переменная $\delta_{q,k,e,u} \in \mathbb{B}$ кодирует функцию активации охранного условия, то есть выполнение перехода при входном действии $e[u]$. Переменная $\theta_{q,k,u} \in \mathbb{B}$ кодирует таблицу истинности охранного условия, то есть значение соответствующей булевой функции на входе $u \in \mathcal{U}$. Взаимосвязь между этими переменная задается следующим образом:

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

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:

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

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:

$$nf_{q,k,e,u} \iff \neg \delta_{q,k,e,u} \wedge nf_{q,k-1,e,u},$$

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

$$(ff_{q,e,u} = k) \iff \delta_{q,k,e,u} \wedge nf_{q,k-1,e,u}.$$

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, то есть 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.

2.1.2 BFS-предикаты нарушения симметрии для состояний автомата

Дополнительно, стоит добавить ограничения нарушения симметрии (*symmetry breaking predicates*) [50], форсирующие нумерацию состояний автомата в порядке BFS-обхода (*Breadth-first search*), то есть 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:

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

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:

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

Непосредственно BFS-ограничение выглядит следующим образом:

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

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

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

Однако существуют и другие, теоретически более эффективные способы. Например, можно использовать так называемые «переменные, кодирующие порядок» (*order-encoding*) [51]. 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*» [52], «*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[2], поэтому кодирующих переменных будет столько же, сколько и при *onehot*-кодировании[3]. Детальное описание *order encoding* присутствует в [51] и в данной работе не приводится. Таким образом, BFS-предикат с использованием *order-encoded* переменной $\pi_{q_j}^{\text{BFS-}\mathcal{A}(order)} \equiv (\pi_{q_j}^{\text{BFS-}\mathcal{A}} \geqslant q_j)$ может быть сформулирован следующим образом:

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

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.

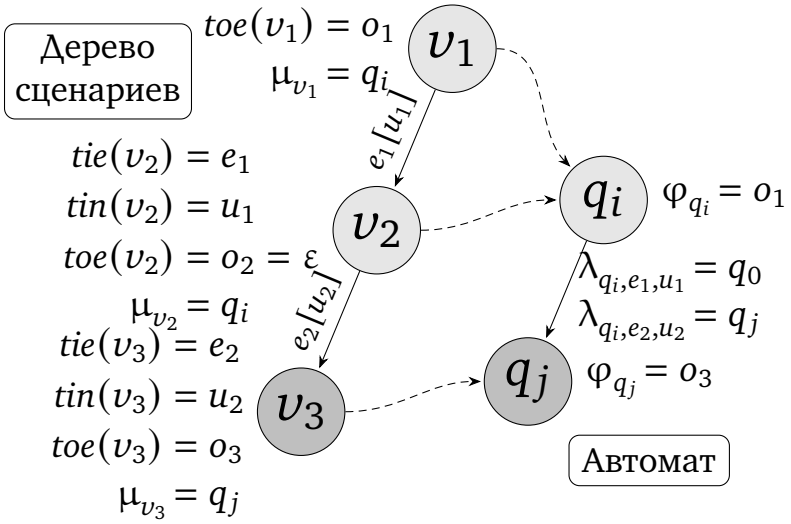

Рисунок 3 — Пример отображения дерева сценариев на автомат

2.1.3 Кодирование отображения позитивного дерева сценариев

Для обеспечения соответствия автомата дереву сценариев, необходимо построить отображение $\mu : V \to Q$ вершин дерева на состояния автомата. Пример такого отображения приведен на рисунке 3. 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:

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

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

[3]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*.

2.1.4 Кодирование ограничений на количество переходов

Для того, чтобы ограничить количество переходов в автомате, то есть закодировать ограничение вида «суммарное число *ненулевых* переходов в автомате не больше T», можно воспользоваться техникой *totalizer* (раздел **??**) и закодировать в SAT *ограничение на кардинальность* $\Phi(\mathcal{D}, 0, T)$, где $\mathcal{D} = \{\tau_{q,k} \neq q_0 \mid q \in Q, k \in [1\,..\,K]\}$ — 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.

2.1.5 Алгоритм Basic

Описанные выше переменные и ограничения позволяют синтезировать *вычислимые* конечно-автоматные модели, то есть 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 $\textsc{Basic}(\mathcal{S}^+, C) = \textsc{Basic}^*(\mathcal{S}^+, C, T = \infty)$ для случая, когда число переходов в автомате остается неограниченным. 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.

2.1.6 Кодирование структуры охранных условий

В вышеописанном 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 [53], 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 состояния $q \in Q$ 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, то есть *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*), то есть 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: $(\eta_{q,k,p} = \bullet) \implies (\eta_{q,k,p+1} = \bullet)$. Переменная $\chi_{q,k,p} \in \mathcal{X} \cup \{0\}$ кодирует входную переменную (или ее отсутствие: $\chi_{q,k,p} = 0$), ассоциированную с терминальной вершиной p. Только терминальные вершины могут иметь ассоциированные входные переменные: $(\eta_{q,k,p} = \square) \iff (\chi_{q,k,p} \neq 0)$.

Для задания структуры дерева разбора, а именно, для определения родительских связей между вершинами, используются переменные $\pi_{q,k,p} \in [0 \,..\, (p-1)]$ и

$\sigma_{q,k,p} \in \{0\} \cup [(p+1) \mathinner{..} P]]$, 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: $(\sigma_{q,k,p} = ch) \implies (\pi_{q,k,ch} = p)$. Только типизированные вершины, кроме корня ($p = 1$), имеют родительские вершины: $(\pi_{q,k,p} \neq 0) \iff (\eta_{q,k,p} \neq \bullet)$. Стоит отметить, что правый ребенок вершины дерева разбора не кодируется явно — для бинарных операторов предполагается, что он имеет номер на единицу больше левого ребенка ($c \in [(p+1) \mathinner{..} (P-1)]]$):

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

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:

$$\begin{aligned}
(\eta_{q,k,p} = \square) \wedge (\chi_{q,k,p} = x) &\implies \bigwedge\nolimits_{u \in \mathcal{U}} \left[\vartheta_{q,k,p,u} \iff u_x\right]; \\
(\eta_{q,k,p} = \wedge) \wedge (\sigma_{q,k,p} = c) &\implies \bigwedge\nolimits_{u \in \mathcal{U}} \left[\vartheta_{q,k,p,u} \iff \vartheta_{q,k,c,u} \wedge \vartheta_{q,k,c+1,u}\right]; \\
(\eta_{q,k,p} = \vee) \wedge (\sigma_{q,k,p} = c) &\implies \bigwedge\nolimits_{u \in \mathcal{U}} \left[\vartheta_{q,k,p,u} \iff \vartheta_{q,k,c,u} \vee \vartheta_{q,k,c+1,u}\right]; \\
(\eta_{q,k,p} = \neg) \wedge (\sigma_{q,k,p} = c) &\implies \bigwedge\nolimits_{u \in \mathcal{U}} \left[\vartheta_{q,k,p,u} \iff \neg\vartheta_{q,k,c,u}\right]; \\
(\eta_{q,k,p} = \bullet) &\implies \bigwedge\nolimits_{u \in \mathcal{U}} \left[\neg\vartheta_{q,k,p,u}\right].
\end{aligned}$$

2.1.7 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 2.1.2), но применяются не ко всему автомату, а к каждому дереву разбора охранного условия по-отдельности: для каждого $q \in Q, k \in [1 \mathinner{..} K]$. Переменная $\tau^{\text{BFS-}\mathcal{G}}_{i,j} \in \mathbb{B}$ $(1 \leqslant i < j \leqslant P)$

задает существование «перехода» из i-й вершины в j-ю:

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

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

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

Непосредственно BFS-ограничение задается следующим образом:

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

2.1.8 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* (раздел **??**) 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.

2.1.9 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 $\textsc{Extended}(\mathcal{S}^+, C, P) = \textsc{Extended}^*(\mathcal{S}^+, C, P, N = \infty)$ для случая, когда суммарный размер охранных условий в автомате остается неограниченным.

2.1.10 Кодирование отображения негативного дерева сценариев

Отображение $\widehat{\mu} : \widehat{V} \to Q_0$ 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, то есть 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 (то есть отображается в q_0), то это распространяется далее через потомков: $(\widehat{\mu}_{\widehat{tp}(\widehat{v})} = q_0) \implies (\widehat{\mu}_{\widehat{v}} = q_0)$. 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:

$$(\widehat{\mu}_{\widehat{v}} = q') \iff (\widehat{\lambda}_{q,e,u} = q') \wedge (\varphi_{q'} = o) \wedge \bigwedge\nolimits_{z \in \mathcal{Z}} (\gamma_{q',z,b} = b'),$$

где $\widehat{v} \in \widehat{V}^{(\text{active})}$, $\widehat{p} = \widehat{tp}(\widehat{v})$, $q = \mu_{\widehat{p}}$, $q' \in Q$, $e = \widehat{tie}(\widehat{v})$, $u = \widehat{tin}(\widehat{v})$, $o = \widehat{toe}(\widehat{v})$, $z \in \mathcal{Z}$, $b = \widehat{tov}(\widehat{p},z)$, $b' = \widehat{tov}(\widehat{v},z)$. Стоит отметить, что в этом ограничении используется « $\iff$ », 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:

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

2.1.11 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 Complete$(\mathcal{S}^+, \mathcal{S}^-, C, P) =$ Complete$^*(\mathcal{S}^+, \mathcal{S}^-, C, P, N = \infty)$ для случая, когда суммарный размер охранных условий в автомате остается неограниченным.

2.2 Синтез минимальных монолитных 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.

2.2.1 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.

2.2.2 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*.

2.2.3 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, то есть 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: $P' \leqslant N^{\text{best}}_{\min} - T_{\min}$.

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

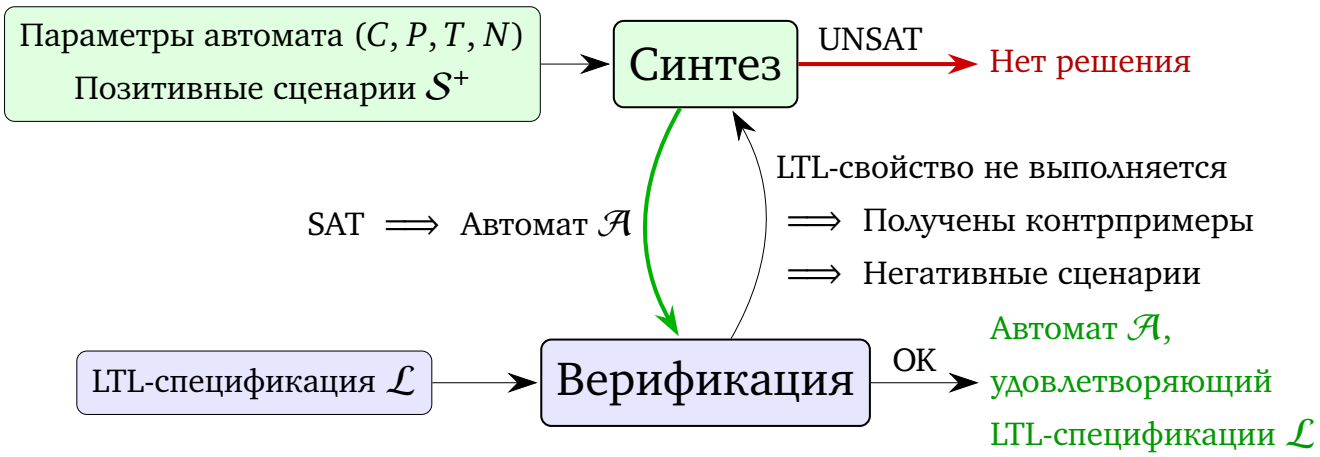

Рисунок 4 — Цикл «Синтез–Верификация» — индуктивный синтез, основанный на контрпримерах

разработанные методы остаются *точными*, то есть 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 **??**.

2.3 Индуктивный синтез, основанный на контрпримерах

Для того, чтобы производить синтез конечно-автоматных моделей не только примерам поведения в виде сценариев выполнения, но и с использованием LTL-спецификации — заданного набора LTL-свойств — в данной работе используется подход индуктивного синтеза, основанного на контпримерах (*Counterexample-Guided Inductive Synthesis* — CEGIS) [46; 47]. CEGIS является итеративным подходом и его общий вид изображен на рисунке 4. 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 [29]. 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.

2.3.1 Алгоритм CEGIS

Обозначим $\text{CEGIS}^*(\mathcal{S}^+, \mathcal{L}, C, P, T, N)$ 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 $\text{CEGIS}(\mathcal{S}^+, \mathcal{L}, C, P) = \text{CEGIS}^*(\mathcal{S}^+, \mathcal{L}, C, P, T=\infty, N=\infty)$ для случая, когда число переходов и суммарных размер охранных условий в автомате остаются неограниченными.

2.3.2 Алгоритм CEGIS-MIN

Рассмотрим автомат $\mathcal{A}$, полученный с помощью алгоритма $\text{CEGIS}(\mathcal{S}^+, \mathcal{L}, C, P)$. Если мы будем минимизировать суммарный размер охранных условий N, то 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 $\text{CEGIS}^*(\mathcal{S}^+, \mathcal{L}, C^*, P^*, T=\infty, N^*)$ производится попытка синтезировать конечный автомат, удовлетворяющий спецификации $\mathcal{L}$. Отсутствие решения (случай UNSAT на рисунке 4) означает, что заданная верхняя граница для суммарного размера охранных условий N 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. 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: $N_{\min}$.

2.4 Программное средство FBSAT

Все предложенные в данной работе методы были реализованы в виде программного средства FBSAT с использованием языка программирования Kotlin. Исходный код распространяется под лицензией GNU GPLv3 и доступен онлайн [49]. В качестве бэкенда возможно использование любого SAT-решателя, поддерживающего работу через стандартный ввод или файлы формата DIMACS [54].

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* [55] 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*) [56]. Совместно со студенткой второго курса Гречишкиной Дарьей была разработана библиотека `kotlin-jnisat` [57; 58], содержащая реализации нативных интерфейсов для современных SAT-решателей: MiniSat, CryptoMiniSat, Cadical, Glucose. 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 наиболее эффективным и робастным решателем, то есть

способен решать как простые, так и сложные задачи, в отличие от, например, решателя MiniSat, который не всегда справляется с большими экземплярами задачи SAT. Однако стоит отметить, что в большинстве случаев решатели Cadical, CryptoMiniSat и Glucose показывают схожие результаты.

2.5 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 [53].

TODO
Рисунок 5 — Pick-and-Place манипулятор

Pick-and-Place (PnP) манипулятор, изображенный на рисунке 5, 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 [53]. 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 X, поступающими от объекта управления — среды исполнения:

- `c1Home`/`c1End` — горизонтальный цилиндр I находится в крайнем левом/правом положении;

Таблица 1 — Результаты синтеза минимальной конечно-автоматной модели логического контроллера PnP-манипулятора по примерам поведения с помощью двухэтапного метода Two-stage [28] и алгоритма Extended-min-UB
TODO

- `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.

2.5.1 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 [28], где на первом этапе производится построение базовой модели, удовлетворяющей заданым сценариям выполнения, с помощью SAT-решателя, а затем охранные условия полученной модели минимизируются с помощью CSP-решателя. Стоит отметить, что превосходство данного двухэтапного метода над другими методами, например, EFSM-tools [11], уже было показано в [28], 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 [28]: $C_{\min}$ — 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 $N_{\min}$.

2.5.2 Синтез минимальной конечно-автоматной модели 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 [59] 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 **??** 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»). При этом свойства φ_1–φ_7 зафиксированы, то есть используются во всех экспериментах, а использование свойств φ_8–φ_{10} разнится. 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 [60] 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 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 [27]. 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 [53] — 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 [27], 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 обработки только одной рабочей детали. Существующий метод FBCSP+LTL [27] 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.

2.6 Экспериментальное исследование: SYNTCOMP

В этом разделе описывается применение разработанных методов к задаче синтеза системы переходов (*transition system*) [14; 23] по входным данным с соревнования по реактивному синтезу SYNTCOMP [48]. 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 [14; 23] и Strix [25]. 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[4], *система переходов* $\mathcal{T}$ это кортеж $\langle T, t_0, \Sigma = I \cup O, \tau \rangle$, 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*), $\tau : T \times 2^I \rightarrow$

[4]Здесь стоит отметить, что в данном разделе для описания системы переходов используется *оригинальная* нотация из [23]. 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 $\mathbb{B}^{|X|}$.

$2^O \times T$ — 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 (то есть 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 (`<length>` — число шагов симуляции) и команда «`show_traces -a -v`» для сохранения трассировок. Полученные трассировки были сконвертированы в сценарии выполнения в соответствии с разделом **??**.

Полученные сценарии выполнения были использованы для синтеза конечно-автоматных моделей с помощью FBSAT. На рисунке 6 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.

TODO

Рисунок 6 — 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»[5] — 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», то есть 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: $N = 52$.

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

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.

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. Разработанные методы были реализованы в виде программного средства fbSAT [49]. 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.

Таблица 2 — Темпоральные свойства для системы PnP-манипулятора

TODO

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

TODO

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 SAT-экземпляров, связанных с булевыми схемами относительно разбиений SAT. Аналогично трудности относительно задней двери, значение трудности относительно разбиения предоставляет нам некоторую верхнюю границу для сложности рассматриваемой формулы, выраженной в некоторых единицах, которые можно использовать для измерения времени работы полного SAT-решателя. Также предлагается две простые конструкции разбиения, которые показали хорошие результаты в вычислительных экспериментах.

3.1 Трудность относительно разбиения и её вероятностная оценка

Теоретическое основание для оценки трудности относительно разбиения с помощью простого алгоритма Монте-Карло заключается в том, что существует возможность 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 мы обозначаем как $\mathbb{E}[\xi_\Pi]$).

Рассмотрим произвольную CNF-формулу C над переменными X, и пусть A – полный SAT-решатель. Пусть $\Pi = \{G_1, \ldots, G_s\}$ – произвольное SAT-разбиение C. Мы можем рассматривать Π как пространство выборки [**Feller71**], где G_i, $i \in \{1, \ldots, s\}$ – элементарные события. Затем, пусть p_i – положительные числа, связанные с каждым G_i так, что $\sum_{i=1}^{s} p_i = 1$. Далее, установим $p_i = 1/s$ для каждого $i \in \{1, \ldots, s\}$, определив таким образом равномерное распределение на Π. Затем мы определяем случайную величину $\xi_\Pi \colon \Pi \to \mathbb{R}^+$ следующим образом:

$$\xi_\Pi(G \in \Pi) = t_A(G \wedge C), \tag{6}$$

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:*

$$\mu_{A,\Pi}(C) = \sum_{i=1}^{s} t_A(G_i \wedge C) \tag{7}$$

Теорема 1. *Справедливо соотношение:* $\mu_{A,\Pi}(C) = s \cdot \mathbb{E}\,[\xi_\Pi]$.

Доказательство. В контексте сказанного выше, имеется вероятностное пространство, где $\Pi = \{G_1,\ldots,G_s\}$ — пространство выборки со случайной величиной ξ_Π, определенной на нем. Пусть $Spec(\xi_\Pi) = \{\xi_1, \ldots, \xi_r\}$ $(r \leqslant s)$ – спектр ξ_Π, где $\xi_k \in \mathbb{R}^+$, $k \in \{1, \ldots, r\}$. Мы обозначаем $\#\xi_k$ количество элементарных событий в Π, для которых случайная величина ξ_Π принимает значение ξ_k. Затем 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 все аксиомы Колмогорова выполняются [**Feller71**]). Следовательно, у нас есть:

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

Используя Теорему 1, можно оценить $\mu_{A,\Pi}(C)$ с помощью метода Монте-Карло [**metropolis1949**]. 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 $p_i = 1/s$, $i \in \{1, \ldots, s\}$. 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 $\widetilde{\mu}_{A,\Pi} = s \cdot \hat{\mu}$.

Говорят, что $\widetilde{\mu}_{A,\Pi}(C)$ является (ε,δ)-приближением [**KarpLuby89**] для $\mu_{A,\Pi}(C)$, если выполняется следующее неравенство (вариация условия (**??**)):

$$\Pr\left\{\left|\mu_{A,\Pi}(C) - \widetilde{\mu}_{A,\Pi}(C)\right| \leqslant \varepsilon \cdot \mu_{A,\Pi}(C)\right\} \geqslant 1 - \delta. \tag{8}$$

Как следует из неравенства Чебышева, неравенство (8) выполняется для всех $N \geqslant \frac{Var(\xi_\Pi(C))}{\varepsilon^2 \cdot \delta \cdot E[\xi_\Pi]^2}$, где ξ_Π определена относительно (6). 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 методах разбиения, таких как стандартная техника Cube-and-Conquer, а также схема разбиения из [**CP2021**]. В следующем разделе мы представим две конструкции разбиения, которые позволяют уменьшить дисперсию $Var(\xi_\Pi)$ и, следовательно, улучшить точность оценки $\mu_{A,\Pi}(C)$.

3.2 Два новых метода разбиения SAT для CircuitSAT

В данном подразделе мы представляем два метода для построения разбиений SAT для формул, связанных с проблемами, связанными с CircuitSAT. Соответствующие конструкции могут быть применены как к LEC, так и к атакам на обратное отображение криптографических функций. Ниже приведено описание для LEC.

Рассмотрим проблему LEC для двух булевых схем S_f, S_h, специфицирующих функции $f,h : \{0,1\}^n \longrightarrow 0,1^m$. Мы определяем 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 $\lambda\colon \{0,1\}^l \longrightarrow \{0,1\}$, где $l \in \mathbb{N}^+$, и предположим, что λ не постоянная. Пусть $\neg\lambda\colon \{0,1\}^l \longrightarrow \{0,1\}$ обозначает отрицание λ. С каждым X^j, $j \in \{1, \ldots, q\}$, ассоциируем две CNF-формулы φ_1^j и φ_2^j, которые определяют функции $\lambda^j\colon \{0,1\}^{|X^j|} \longrightarrow \{0,1\}$ и $\neg\lambda^j\colon \{0,1\}^{|X^j|} \longrightarrow \{0,1\}$ соответственно.

Теорема 2. *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* (3).

Доказательство. Сначала докажем, что формула $C_{f \triangle h}$ имеет ровно 2^n удовлетворяющих назначений. Действительно, рассмотрим множество $X^{in} = \{x_1, \ldots, x_n\}$ и пусть $\alpha = (\alpha_1, \ldots, \alpha_n)$ – произвольное назначение переменных из X^{in}. Затем 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 получению значений всех переменных из этой CNF 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 как $\Lambda(X)$. Ясно, что все назначения из $\Lambda(X)$ различны. Теперь рассмотрим множество переменных $\tilde{X} = X \setminus X^{in}$. С каждым назначением $\lambda \in \Lambda(X)$

ассоциируем 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 $\gamma \notin \Lambda(\tilde{X})$. Легко видеть, что подстановка γ в $C_{f\triangle h}$ приводит к CNF $C_{f\triangle h}[\gamma/\tilde{X}]$, которая 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, любое удовлетворяющее назначение $C_{f\triangle h}$ – это назначение, сгенерированное некоторым $\alpha \in \{0,1\}^{|X^{in}|}$, и различные назначения из $\{0,1\}^{|X^{in}|}$ порождают различные удовлетворяющие назначения $C_{f\triangle h}$. Таким образом, у формулы $C_{f\triangle h}$ ровно 2^n удовлетворяющих назначений. С другой стороны, нетрудно видеть, что любое $\alpha \in \{0,1\}^{|X^{in}|}$ удовлетворяет только одной формуле G_i, $i \in \{1, \ldots, 2^{\lceil n/k \rceil}\}$ описанного выше вида. Таким образом, формулы $C_{f\triangle h}$ и $(G_1 \vee \ldots G_{2^{\lceil n/k \rceil}})$ имеют одинаковые наборы удовлетворяющих назначений, и, следовательно, формулы $C_{f\triangle h} \wedge C(M)$ и $(G_1 \vee \ldots G_{2^{\lceil n/k \rceil}}) \wedge C_{f\triangle h} \wedge C(M)$ эквивалентны по выполнению. □

Важным вопросом является выбор функций λ^j и $\neg\lambda^j$ таким образом, чтобы гарантировать малую дисперсию $Var(\xi_\Pi)$ для разбиения SAT типа, описанного ранее? Здравый смысл подсказывает, что *сбалансированная* булева функция, которая принимает значение 1 на 2^{l-1} входных словах, должна быть использована в качестве функции $\lambda\colon \{0,1\}^l \to \{0,1\}$. Очевидно, что если λ является сбалансированной функцией, то ее отрицание $\neg\lambda$ также сбалансировано. Хорошим примером сбалансированной функции для $l > 1$ является функция, заданная формулой $x_1 \oplus \ldots \oplus x_l$.

В данной работе представлен несколько неформальный анализ свойств Конструкции 1, который используется в качестве основы для Конструкции 2, которая показала лучшие результаты среди всех рассмотренных методов в экспериментах с некоторыми чрезвычайно сложными экземплярами LEC в виде SAT. Рассмотрим функцию (2) и шаблонную CNF $C_{f\triangle h}$. Множественные эксперименты показывают, что даже когда задача SAT для $C_{f\triangle h} \wedge C(\mathcal{M})$ является чрезвычайно сложной, SAT для $C_{f\triangle h}$ остается простой: любой SAT-решатель на базе CDCL, получивший на вход $C_{f\triangle h}$ и не имея никакой дополнительной информации о структуре схемы, способен найти удовлетворяющее назначение для $C_{f\triangle h}$. Это назначение можно рассматривать как сертификат удовлетворимости для $C_{f\triangle h}$. Как отмечалось ранее, CNF $C_{f\Delta h}$ имеет 2^n таких сертификатов. Таким образом, доказательство невыполнимости $C_{f\triangle h} \wedge C(\mathcal{M})$ можно рассматривать как процесс, который аннулирует 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 $2^{n-\lceil n/k \rceil}$ удовлетворяющих назначений, которые также являются сертификатами удовлетворимости. Таким образом, можно сделать два спекулятивных 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 аннулировать $2^{n-\lceil n/k \rceil}$ 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 выполнения алгоритма A на них, т.е. разбиение Π, заданное Конструкцией 1, должно иметь малую дисперсию $Var(\xi_\Pi)$.

Хотя представленные аргументы лишены строго формального доказательства, их выводы были экспериментально проверены. Ниже описана 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 $\{0,1\}^n \longrightarrow N_0^n$. 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, $i \in \{1,\ldots,n\}$ принимают значения из $\{0,1\}$:

$$a \leqslant x_n + 2 \cdot x_{n-1} + \cdots + 2^{n-1} \cdot x_1 < b \tag{9}$$

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 (9).

Конструкция 2. *Пусть $\mathcal{R}^n$ — полная система интервалов. С произвольным $I \in \mathcal{R}^n$, $I = [a,b]$, ассоциируем неравенство вида* (9) *и CNF C_I, полученную путем перевода* (9) *в SAT с использованием соответствующих техник, например, представленных в [61]. Определим $\Pi = \{C_I\}_{I \in \mathcal{R}^n}$.*

Теорема 3. *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. Подобно доказательству Теоремы 2, мы используем Лемму 1, чтобы показать, что любое назначение, удовлетворяющее $C_{f \triangle h}$, также удовлетворяет ровно одной формуле вида $G_i \wedge C_{f \triangle h}$, $i \in \{1, \ldots, s\}$. Следовательно, мы можем заключить, что $C_{f \triangle h} \wedge C(\mathcal{M})$ и $C_{f \triangle h} \wedge C(\mathcal{M}) \wedge (G_1 \vee \cdots \vee G_s)$ эквивалентны в удовлетворимости. С другой стороны, ясно, что любая CNF вида $G_i \wedge G_j$, $i \neq j$, $i,j \in \{1, \ldots, s\}$, невыполнима, что означает, что $\Pi = \{G_1, \ldots, G_s\}$ 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 [61]. 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 $j = 0, \ldots, 2^l - 1$. Следовательно, этот интервал можно указать с помощью двоичного вектора $\lambda_{l+1} \ldots \lambda_n$, где λ_q, $q \in \{l+1, \ldots, n\}$, 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) указывает на число 40. Таким образом, не существует числа $i \in \{1, \ldots, 6\}$ такого, что x_i принимает одно и то же значение во всех числах из интервала $[23,40] \cap N_0$. Поэтому в общем случае необходимо применять техники, такие как например, описанные в [61], для кодирования интервалов вида (9) в SAT.

3.3 Экспериментальное исследование

[TODO Вычислительные эксперименты и их результаты] [TODO Multipliers, Sorters, Cryptography] [TODO Статистические оценки, результаты из IEEE, доверительные интервалы, обоснование мощности выборки]

3.4 Вычислительные эксперименты

Все эксперименты, представленные в данном разделе, были проведены на кластере Университета ИТМО, где каждый узел 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 мы изучали в вычислительных экспериментах, была точность оценок сложности относительно разбиения (в смысле формулы (1)). Ситуации, когда сложность относительно разбиения меньше или близка к времени выполнения последовательного решателя SAT на исходной задаче, являются особенно интересными, потому что в таких случаях соответствующие разбиения позволяют не только точно оценить время выполнения решателя SAT на задаче (в отличие от случая последовательного решения), но и обеспечивают более эффективную стратегию решения соответствующего экземпляра LEC.

Все дополнительные материалы к настоящей статье доступны онлайн[3].

3.4.1 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» [1], «алгоритм Карацубы» [**knuth1969**] и «умножитель Дадда» [**dadda1965**]. Эти экземпляры обозначаются как $\mathtt{AvB}_k$, где `A` и `B` означают алгоритмы умножения, а k — количество бит в умножаемых числах. Например, $\mathtt{CvK}_{16}$ представляет собой

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

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

[3] `https://github.com/Lipen/FMCAD-2023-Supplementary`

экземпляр LEC для проверки эквивалентности умножения двух 16-битных чисел (16x16 умножитель) методом столбика и алгоритмом Карацубы. Такие бенчмарки известны своей сложностью для современных SAT-решателей [**kaufmann2019**; **CP2021**].

Второй класс состоит из нескольких (выполнимых) экземпляров, связанных с алгебраическим криптоанализом [**bard2009**], а именно, SAT-кодировок атаки прообраза на хеш-функцию MD4 с уменьшенным числом раундов. Эта проблема была недавно решена в [**DBLP:conf/ijcai/Zaikin2022**] с использованием подхода Cube and Conquer. Тестовые наборы MD4 служат дополнительным материалом для того, чтобы показать, что предложенная техника действительно применима к (1) совместимым тестовым наборам (2) наборам, которые не кодируют LEC.

3.4.2 Эксперименты по оценке декомпозиционной сложности

В первом наборе экспериментов мы оцениваем сложность экземпляров LEC для умножителей относительно предложенных разбиений SAT, где мы разбиваем множество входов X^{in} на непересекающиеся подмножества, называемые *чанками*, 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 $\lambda_1^j = \lambda^j$ и $\lambda_2^j = \neg\lambda^j$, были закодированы в CNF.

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

Чтобы обеспечить достоверность и актуальность наших результатов, среди всех составленных бенчмарков были выбраны только те экземпляры, которые представляют практический интерес, то есть те, которые не решаются в течение нескольких секунд, но 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 нашего подхода в решении задач с различной сложностью. Для каждого выбранного экземпляра были построены соответствующие разбиения и были решены **все** подзадачи, чтобы вычислить истинные значения математического ожидания $\mathbb{E}[\xi_\Pi]$ и дисперсии $Var(\xi_\Pi)$.

Кроме того, были рассмотрены разбиения, построенные с использованием техники Cube and Conquer (CnC). Для этой цели были построены кубы с помощью `march_cu`[4]. В частности, были подобраны значения опций `-d <depth>` и `-n <number>` таким образом, чтобы размеры полученных разбиений были аналогичны разбиению, построенному 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. Однако для некоторых «более сложных» экземпляров режим 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 эти разбиения, сгенерированные CnC, как `CnC-d*`, `CnC-n*` и `CnC-default`.

Экспериментальные результаты суммируются на Рисунках **??** и **??**. В частности, Рисунок **??** содержит подробные результаты для *всех* обсуждаемых типов разбиений на двух выбранных экземплярах ($\texttt{CvK}_{12}$ и $\texttt{CvK}_{16}$). Тем временем, Рисунок **??** представляет *лучшие* разбиения для остальных экземпляров. Для каждого разбиения сообщается среднее и стандартное отклонение («Avg ± sd»), диапазон времени выполнения подзадач («Min – max») и общее CPU-время, необходимое для решения всех подзадач. Таблица также включает строки «Sequential» для представления базовой производительности последовательного решателя SAT без применения разбиения на основе разбиения.

Графики на Рисунке **??** визуализируют дисперсию времени выполнения SAT-решателя при использовании рассматриваемых схем разбиения. Для конструкций 1 и 2 (разбиения `2-XOR` и `INT`, соотвественно) время выполнения SAT-решателя имеет относительно низкую дисперсию, что указывает на то, что мы можем точно оценить необходимое общее время выполнения, используя относительно небольшой объем выборки. Напротив, для Cube and Conquer и разбиений `2-DIS` дисперсия значительно больше из-за неравномерного распределения сложности подзадач. Для

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

последнего это можно объяснить дисбалансом функции $\lambda = a \vee b$, используемой в 2-DIS. Эти результаты подчеркивают важность выбора подходящих схем разбиения для достижения более последовательной производительности времени выполнения и точной оценки общего времени выполнения.

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

В контексте всего сказанного одной из основных проблем является точность полученных оценок $\mathbb{E}[\xi_\Pi]$. Дисперсия $Var(\xi_\Pi)$ оказывает отрицательное влияние на точность. Однако результаты в Таблице **??** 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 мы выполняем дополнительный анализ полученных результатов.

Для различных значений N мы генерируем $P = 1000$ случайных выборок размера N и вычисляем средние значения выборок $(\hat{\xi}^1, \ldots, \hat{\xi}^P)$, где каждое $\hat{\xi}^r = \frac{1}{N} \sum j = 1^N \xi_j^r$. Мы также вычисляем среднее значение средних значений выборок $\Xi(N) = \frac{1}{P} \sum_{r=1}^{P} \hat{\xi}^r$ и минимальные и максимальные значения среди $\hat{\xi}$, обозначенные $M_*(N)$ и $M^*(N)$, соответственно. Затем все значения нормализуются путем деления их на $\mathbb{E}[\xi_\Pi]$. Распределения нормализованных средних значений выборок для различных размеров выборки показаны на Рисунке **??**, где горизонтальная ось представляет размер случайной выборки N. Здесь мы рассматриваем экземпляр LEC умножителей $\mathtt{CvK}_{16}$ и два различных разбиения: `INT-65536` (предложенное разбиение на 65 536 интервалов) и `CnC-d16` (Cube and Conquer, используя `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

Таблица 4 — Экспериментальные результаты для разбиений экземпляров MD4
TODO

$M^*(N)/\mathbb{E}[\xi_\Pi]$ (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` в экземпляре $\texttt{CvK}_{16}$ достаточен для получения оценки в пределах 10%-интервала $\mathbb{E}[\xi_\Pi]$. Наоборот, достаточный размер выборки для разбиений CnC обычно значительно больше, особенно для `CnC-d16` (которое также имеет размер 65 536), он составляет как минимум $N \approx 1000$.

3.4.3 Эксперименты по поиску прообразов MD4

Чтобы показать, что предложенные конструкции применимы и к другим сложным экземплярам CircuitSAT, помимо невыполнимых LEC-бенчмарков, мы применили их к задаче поиска преобразов для хэш-функции MD4 сокращенного раунда. Эта проблема интересна тем, что лучшие известные результаты для нее были получены с помощью метода Cube-and-Conquer со специализированной стратегией поиска параметров CnC.

В наших экспериментах мы взяли две CNF (md4_40steps_11.30-32Dobb_one_constr_one обозначаемую как $\texttt{MD4}_{40}$, и md4_43steps_12Dobb_one_constr_one_hash, обозначаемую как $\texttt{MD4}_{43}$) из хранилища[5], представленного в [62]. Эти CNF соответствуют двум не слишком легким и не слишком трудным случаям криптоанализа. Используя Конструкцию 2, мы провели серию оценок времени выполнения, варьируя количество интервалов, и использовали наилучшие найденные параметры разбиения для полного решения обеих задач. Как и в [62], были решены все подзадачи, то есть процесс не останавливался как только была найдена удовлетворяющая подстановка.

Результаты этой серии экспериментов обобщены в Таблице 4. Они сравниваются с результатами, опубликованными в [62], обозначенными как `CnC` (время выполнения последних было представлено для 12 ядер CPU, поэтому мы масштабировали их для одного ядра CPU). Стоит отметить, что в статье [62] также использовалась стратегия, адаптированная к данной конкретной задаче, для поиска оптимальных параметров

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

разложения CnC, хотя время нахождения этих параметров не включено в Таблицу 4, а также использовалась вычислительная платформа с более быстрыми ядрами CPU. Тем не менее, используя наш простой метод, нам удалось решить рассмотренные задачи за время, сравнимое с временем в [62]. В Таблице 4 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 «Time». Из распределения выборочных средних для $\mathtt{MD4}_{40}$, представленного на правом графике на Рисунке **??**, 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 4 еще раз показывают, что предложенные конструкции дают небольшую дисперсию в трудности подзадач.

Выводы по главе 3

[TODO В этой главе были представлены результаты экспериментального исследования, которые показывают, что предложенные методы разбиения задачи SAT позволяют получить точные оценки время работы SAT-решателя.]

Заключение

Список литературы

1. *Cormen T. H.*, *Leiserson C. E.*, *Rivest R. L.* Introduction to Algorithms. 1st ed. MIT Press, 1990.
2. *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).
3. *Szeider S.* Backdoor Sets for DLL Subsolvers // Journal of Automated Reasoning. 2006. Oct. 5. Vol. 35, no. 1–3. P. 73–88. (Visited on 05/03/2024).
4. 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 05/03/2024).
5. *Drechsler R.*, *Junttila T. A.*, *Niemelä I.* Non-Clausal SAT and ATPG // Handbook of Satisfiability. 1st ed. 2009. P. 655–693.
6. *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).
7. *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.
8. IEC 61131-1:2003. URL: https://webstore.iec.ch/publication/4550 (visited on 06/17/2020).
9. *Gold M.* Complexity of Automaton Identification from Given Data // Information and Control. 1978. Vol. 37, no. 3. P. 302–320.
10. *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.
11. *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.

12. 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.

13. *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.

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

15. *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.

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

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

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

19. *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.

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

21. *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.

22. *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.

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

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

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

26. 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.

27. 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.

28. 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.

29. 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.

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

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

32. *Brand D.* Redundancy and Don't Cares in Logic Synthesis // IEEE Transactions on Computers. 1983. Окт. Т. С—32, № 10. С. 947—952.

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

34. *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.

35. SAT Competitions. URL: `http://www.satcompetition.org/` (visited on 06/13/2020).

36. 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.

37. CaDiCaL Simplified Satisfiability Solver. URL: `http://fmv.jku.at/cadical/` (visited on 06/13/2020).

38. *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.

39. *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.

40. Lingeling, Plingeling and Treengeling. URL: http://fmv.jku.at/lingeling/ (visited on 06/13/2020).

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

42. *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.

43. *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.

44. 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).

45. 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.

46. Combinatorial sketching for finite programs / A. Solar-Lezama [et al.] // ACM SIGOPS Operating Systems Review. 2006. Vol. 40, no. 5. P. 404–415.

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

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

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

50. *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.

51. *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.

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

53. nxtControl - nxtStudio. URL: http://www.nxtcontrol.com/en/engineering (visited on 06/17/2020).

54. Benchmarks submission guidelines. URL: http://www.satcompetition.org/2009/format-benchmarks2009.html (visited on 06/17/2020).

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

56. JNI APIs and Developer Guides. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jni/ (visited on 06/17/2020).

57. *Chukharev K.*, *Grechishkina D.* Lipen/kotlin-jnisat: JNI wrappers for SAT-solvers in Kotlin. URL: https://github.com/Lipen/kotlin-jnisat (visited on 06/17/2020).

58. *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).

59. 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.

60. *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.

61. *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.

62. *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.