Министерство образования и науки Российской Федерации
Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

На правах рукописи

Ульянцев Владимир Игоревич

Генерация конечных автоматов с использованием программных средств решения задач выполнимости и удовлетворения ограничений

Специальность 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
доктор технических наук,
профессор А. А. Шалыто

Санкт-Петербург – 2015 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5

ГЛАВА 1. ЗАДАЧИ ВЫПОЛНИМОСТИ И УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ, ГЕНЕРАЦИЯ КОНЕЧНЫХ АВТОМАТОВ 15

1.1. Задачи выполнимости и удовлетворения ограничений 15

1.1.1. Методы решения задач выполнимости и удовлетворения ограничений 16

1.1.2. Программные средства решения SAT и CSP 21

1.2. Конечные автоматы 26

1.3. Методы генерации конечных автоматов 33

1.3.1. Эвристические методы генерации конечных автоматов 34

1.3.2. Методы генерации, основанные на метаэвристических алгоритмах 39

1.3.3. Методы генерации, основанные на сведении к задачам из класса трудных в NP 43

1.4. Задачи, решаемые в диссертационной работе 49

Выводы по главе 1 52

ГЛАВА 2. ТЕОРЕТИЧЕСКАЯ ОЦЕНКА СЛОЖНОСТИ ПОСТАВЛЕННЫХ ЗАДАЧ ГЕНЕРАЦИИ УПРАВЛЯЮЩИХ АВТОМАТОВ 53

2.1. Доказательство принадлежности поставленной задачи классу NP-трудных 53

2.2. Условие принадлежности рассматриваемой задачи классу NP 56

Выводы по главе 2 57

ГЛАВА 3. ГЕНЕРАЦИЯ ДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ ПО ОБУЧАЮЩИМ СЛОВАРЯМ 58

3.1. Метод генерации по безошибочным обучающим словарям 58

3.1.1. Структура метода 58

3.1.2. Предикаты нарушения симметрии 60

3.2. Метод генерации по зашумленным обучающим словарям 65

3.2.1. Структура метода 66

3.2.2. Предикаты обработки ошибочных пометок 67

3.3. Реализация и экспериментальные исследования разработанных методов генерации 70
3.3.1. Реализация разработанных методов генерации ДКА 70
3.3.2. Экспериментальные исследования метода генерации ДКА по безошибочным обучающим словарям 73
3.3.3. Экспериментальные исследования метода генерации ДКА по зашумленным обучающим словарям 77
Выводы по главе 3 79
ГЛАВА 4. ГЕНЕРАЦИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ 81
4.1. Метод генерации управляющих конечных автоматов по безошибочным сценариям работы 81
4.1.1. Структура метода 82
4.1.2. Построение дерева сценариев и графа совместимости 83
4.1.3. Ограничения на целочисленные переменные 89
4.1.4. Предикаты нарушения симметрии 92
4.2. Метод генерации по зашумленным сценариям работы 95
4.2.1. Структура метода 95
4.2.2. Ограничения на целочисленные переменные 97
4.3. Реализация и экспериментальные исследования методов 100
4.3.1. Программное средство генерации конечных управляющих автоматов 100
4.3.2. Экспериментальные исследования метода генерации управляющих автоматов по безошибочным сценариям работы 102
4.3.3. Экспериментальные исследования метода генерации управляющих автоматов по зашумленным сценариям работы 106
Выводы по главе 4 108
ГЛАВА 5. ВНЕДРЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ 110
5.1. Внедрение разработанных предикатов нарушения симметрии в средство *DFASAT* 110
5.2. Внедрение в образовательный процесс 111

Выводы по главе 5 111

ЗАКЛЮЧЕНИЕ 112

СПИСОК ИСТОЧНИКОВ 113

Печатные издания на русском языке 113

Печатные издания на английском языке 114

Ресурсы сети интернет 120

Публикации автора 122

Статьи в рецензируемых изданиях из перечней ВАК или Scopus 122

Другие публикации по теме 123

ПРИЛОЖЕНИЕ 1. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ ДЛЯ ЭВМ 125

ПРИЛОЖЕНИЕ 2. АКТ, ПОДТВЕРЖДАЮЩИЙ ВНЕДРЕНИЕ И ИСПОЛЬЗОВАНИЕ РЕЗУЛЬТАТОВ ДИССЕРТАЦИОННОЙ РАБОТЫ В ПРОГРАММНОЕ СРЕДСТВО *DFASAT* 129

ВВЕДЕНИЕ

Актуальность проблемы. Теория автоматов является одним из важнейших разделов дискретной математики и информатики. Многие компоненты аппаратного и программного обеспечения моделируются конечными автоматами, а при проектировании событийных систем все чаще программный код для целевой платформы генерируется по созданной автоматной модели.

Основными достоинствами при проектировании и анализе программ с использованием конечных автоматов являются наглядность их представления в виде диаграмм состояний и повышение уровня автоматизации процесса верификации методом проверки моделей (model checking). Данная проверка применяется при создании ответственных систем, программирование которых недостаточно сопровождать лишь экспертным анализом, статическим и динамическим тестированием.

Для повышения надежности и уменьшения степени влияния человеческого фактора на этапе разработки используемых в ответственных системах автоматных моделей применяются методы их *генерации*. Особенно актуальна разработка методов генерации автоматных моделей для тех задач, в которых их построение вручную затруднительно или невозможно. Существующие методы генерации можно классифицировать по типу обучения (с заранее заданными данными и интерактивные), по типам ограничений на искомый автомат (достижение им заданного значения целевой функции приспособленности, его полное или частичное соответствие заданным примерам поведения или темпоральным свойствам, и т. д.), по типу искомой автоматной модели (детерминированная, недетерминированная, вероятностная).

Настоящая диссертация посвящена решению задач автоматизированной генерации детерминированных автоматных моделей по заранее заданным *примерам поведения*, которым должен

соответствовать искомый автомат. Помимо задач генерации и анализа программ такие задачи встречаются в грамматическом выводе, биоинформатике, лингвистике, распознавании речи, робототехнике. Заданные примеры поведения при этом могут быть получены как автоматизированно (протоколы работы существующей программы, биологические последовательности, обработанные аудиозаписи речи, примеры поведения робота, и т. д.), так и построены вручную специалистом.

Известно, что в общем случае задача генерации детерминированного конечного автомата (ДКА) с наименьшим числом состояний по *обучающим словарям* (автомат должен допускать слова из S_+ и не допускать слова из S_-) полна в классе NP. Задача нахождения ДКА с числом состояний, приближенным к наименьшему, также относится к классу NP-полных. Это указывает на актуальность разработки и реализации практически применимых методов генерации. Существует три основных подхода к генерации автоматных моделей по заданным примерам поведения: эвристические алгоритмы слияния состояний (state merging), применение метаэвристических алгоритмов и использование программных средств решения классических NP-полных задач. Первые два подхода являются *неточными* – в общем случае ими не гарантируется нахождение искомого ответа за конечное время или его отсутствие. Рассмотрим более подробно третий подход.

Использование существующего программного средства решения некоторой задачи A возможно после *сведения* к ней решаемой задачи B – однозначного представления (*кодирования*) экземпляра x задачи B в виде экземпляра $\mathcal{F}(x)$ задачи A. По определению, каждая NP-полная задача за полиномиальное время сводится к другой, однако построение конкретной функции $\mathcal{F}$ является нетривиальной задачей, а существующие сведения зачастую носят теоретический характер и неэффективны. Данное сведение

применяется на практике, если задача A «комбинаторно проще» задачи B, и для решения задачи A существуют эффективные алгоритмы и программные средства. В качестве таких задач используются такие классические задачи, как, например, задачи выполнимости булевой формулы, удовлетворения ограничений, выполнимости формул в теориях (satisfiability modulo theories – SMT), раскраски графа.

Задача *выполнимости булевых формул* (задача выполнимости, Boolean satisfiability – SAT), для которой и была впервые доказана NP-полнота, заключается в нахождении выполняющей подстановки значений переменных для заданной булевой формулы. Для решения этой задачи уже несколько десятилетий продолжается развитие алгоритмов и программных средств, ежегодно проходят конференции, посвященные теоретическим и практическим вопросам их разработки, а также соревнования среди существующих реализаций методов решения. В настоящее время программные средства способны находить за минуты работы персонального компьютера выполняющую подстановку (или продемонстрировать ее отсутствие) для формул, соответствующих практическим задачам. Такие формулы при этом могут состоять из миллионов дизъюнктов.

Указанная производительность программных средств делает актуальной разработку алгоритмов решения иных NP-трудных задач, основанных на сведении к задаче выполнимости (*пропозициональном кодировании*). При таком подходе сначала для экземпляра x решаемой задачи строится соответствующая булева формула $\mathcal{F}(x)$, затем она подается на вход программному средству для решения SAT, и, наконец, в случае нахождения выполняющей подстановки формируется ответ на исходную задачу x. Во-первых, при таком подходе границы применимости алгоритма, основанного на сведении, будут со временем расширяться *без его изменения* пропорционально производительности программных

средств для решения SAT. Во-вторых, в отличие от неточных алгоритмов подход обладает точностью – в случае отсутствия искомого решения программное средство, основанное на *поиске с возвратом* (backtracking), сообщит о невыполнимости соответствующей булевой формулы.

Аналогичным образом используются программные средства решения задачи *удовлетворения ограничений* (обобщенная задача выполнимости, constraint satisfaction problem – CSP), которая заключается в подборе таких допустимых значений переменных $x_1, \dots, x_n$ (не обязательно булевых), что выполняются все заданные ограничения на эти переменные. Существующие программные средства поддерживают ограничения многих типов, что позволяет использовать выразительную силу языка CSP и представлять комбинаторные задачи более наглядно и «естественно», в отличие, например, от языка SAT – булевых формул.

Также стоит отметить, что некоторые возникающие на практике ограничения на целочисленные переменные затруднительно выразить в виде булевой формулы полиномиальной длины, что актуализирует для данных задач применение программных средств решения CSP. Задача SAT является частным случаем задачи CSP, однако за счет простоты модели позволяет применять более эффективные эвристики решения. Таким образом, достоинством сведения к SAT является более высокая эффективность (на настоящий момент) программных средств, а достоинствами сведения к CSP являются наглядность представления в виде ограничений и применимость для более широкого класса задач за счет выразительной силы языка.

Вернемся к задачам генерации конечных автоматов, решаемых в настоящей диссертации. Известно сведение, строящее булеву формулу $\mathcal{F}(x)$, выполнимую только в случае существования ДКА с заданным числом состояний C, удовлетворяющего заданным обучающим словарям S_+ и S_- (в данном случае $x = \langle S_+, S_-, C \rangle$). Помимо гарантированной

точности генерации, которой не обладают эвристические и метаэвристические алгоритмы решения задачи, подход, основанный на сведении, превосходит их по скорости работы на практических задачах.

В настоящее время известно лишь описанное в предыдущем абзаце применение программных средств решения SAT для генерации конечных автоматов. В то же время для других задач генерации конечных автоматов по примерам поведения известны только неточные алгоритмы решения. Такими задачами являются, например, задача генерации ДКА по *зашумленным* обучающим словарям (с конечным числом ошибок в метках допуска/недопуска слов), а также задачи генерации *управляющих конечных автоматов* по сценариям работы. Сценарии могут быть как безошибочными, так и зашумленными – содержать конечное число ошибочных последовательностей выходных воздействий. Решение таких задач актуально не только при моделировании, но и при построении программных систем с использованием технологии *автоматного программирования*, а также систем визуального проектирования управляющих программ (например, *MATLAB/Stateflow*, *IBM Rational Rhapsody*).

Из изложенного следует, что развитие существующих и разработка новых точных методов генерации конечных автоматов по примерам поведения с использованием программных средств решения SAT и CSP **актуальны.**

В соответствии с паспортом специальности 05.13.11 – «Математическое обеспечение вычислительных машин, комплексов и компьютерных сетей» диссертация относится к области исследований «1. Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования».

Цель диссертационной работы – разработка точных методов генерации конечных автоматов по примерам поведения с использованием программных средств решения задач выполнимости и удовлетворения ограничений.

Основные задачи диссертационной работы:

1. Доказать NP-трудность задачи генерации управляющих конечных автоматов заданного размера по сценариям работы. Данный теоретический результат указывает на целесообразность разработки эффективных методов для задач генерации управляющих автоматов, основанных на сведении к задаче CSP.
2. Разработать точные методы генерации ДКА по безошибочным и зашумленным обучающим словарям с использованием программных средств решения задачи выполнимости. Реализовать методы в программном средстве *DFAInducer* с открытым кодом. Выполнить сравнение программных средств *DFAInducer* и *DFASAT* (Делфтский технический университет, Нидерланды).
3. Разработать точные методы генерации управляющих автоматов по безошибочным и зашумленным сценариям работы с использованием программных средств решения задачи удовлетворения ограничений. Реализовать методы в программном средстве *EFSMTools* с открытым кодом.
4. Внедрить разработанные методы генерации ДКА после их публикации в *DFASAT*.

Научная новизна. В работе получены следующие новые научные результаты, которые выносятся на защиту.

1. Доказательство NP-трудности задачи построения управляющих конечных автоматов по сценариям работы.
2. Точные методы генерации ДКА по безошибочным и зашумленным обучающим словарям. Новизна метода генерации по безошибочным

словарям заключается в разработанных *предикатах нарушения симметрии*, обеспечивающих статистически значимое преимущество скорости работы по сравнению с *DFASAT*. Метод построения ДКА по зашумленным словарям является *точным*, в отличие от известных.

3. Точные методы генерации управляющих конечных автоматов по сценариям работы. Метод генерации управляющих автоматов по безошибочным сценариям работы включает в себя новые алгоритмы построения дерева сценариев и его графа совместимости. Известные методы генерации, как по безошибочным, так и по зашумленным сценариям, являются неточными.

Методология и методы исследования. Методологическую основу диссертации составили принципы формализации, обобщения, дедуктивного и индуктивного обоснования утверждений, проведение экспериментальных исследований и анализ их результатов. В работе используются методы теории автоматов, дискретной математики, теории сложности и математической статистики.

Достоверность научных положений, выводов и практических рекомендаций, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, а также результатами экспериментов по использованию предложенных в диссертации методов.

Теоретическое значение работы состоит в доказанной NP-трудности задачи генерации управляющих конечных автоматов по сценариям работы и разработанных сведениях задач генерации конечных автоматов к задачам SAT и CSP.

Практическое значение работы. Разработанные методы и реализованные программные средства позволяют автоматизированно и *точно* решать задачи генерации конечных автоматов. С их использованием

можно повысить надежность процессов проектирования и анализа программ, уменьшить влияние человеческого фактора.

Внедрение результатов работы. Разработанное сведение задачи генерации ДКА по обучающим словарям было использовано при реализации программного средства *DFASAT*. Также результаты использовались в учебном процессе кафедры «Компьютерные технологии» Университета ИТМО в рамках курса «Теория автоматов и программирование», при руководстве и выполнении выпускных квалификационных работ студентов кафедры.

Апробация результатов работы. Основные результаты диссертационной работы докладывались на следующих научных и научно-практических конференциях:

- Всероссийская научная конференция по проблемам информатики СПИСОК. 2011-2014, Матмех СПбГУ;
- Всероссийский конгресс молодых ученых. 2011-2013, Университет ИТМО;
- Всероссийское совещание по проблемам управления. 2014, Институт проблем управления РАН, Москва;
- 14th IFAC Symposium «Information Control Problems in Manufacturing – INCOM'12». 2012, Бухарест, Румыния;
- International Conference on Machine Learning and Applications. 2011 – Гонолулу, США. 2014 – Детройт, США;
- 9th International Conference on Language and Automata Theory and Applications. 2015, Ницца, Франция.

Личный вклад автора. Решение задач диссертации, разработанные методы и алгоритмы принадлежат лично автору.

Публикации. По теме диссертации опубликовано 14 работ, в том числе шесть публикаций в изданиях из перечней ВАК или Scopus.

Свидетельства о регистрации программ для ЭВМ. В рамках диссертационной работы получены четыре свидетельства о регистрации программ для ЭВМ:

- № 2012 616462 от 18.07.2012 г. «Программное средство для построения графа совместимости вершин дерева сценариев работы программы»;
- № 2012 660438 от 20.11.2012 г. «Программное средство для построения КНФ-формулы по графу совместимости вершин дерева сценариев работы программы»;
- № 2013 619840 от 17.10.2013 г. «Программный комплекс для построения и тестирования управляющих конечных автоматов»;
- № 2015 619224 от 27.08.2015 г. «Программное средство преобразования полученных методами машинного обучения управляющих автоматов в формат *MATLAB/Stateflow*».

Участие в научно-исследовательских работах. Результаты диссертации использовались при выполнении следующих НИР, которыми руководил автор:

- «Разработка методов автоматизированного построения надежного программного обеспечения на основе автоматного подхода по обучающим примерам и темпоральным свойствам» (грант РФФИ «Мой первый грант», 2014, 2015);
- «Построение управляющих автоматов с помощью методов решения задачи удовлетворения ограничений» (программа «У.М.Н.И.К.», 2012).

Полученные результаты также использовались при проведении НИР:

- «Разработка метода машинного обучения на основе алгоритмов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов» (ФЦП «Научные и научно-

педагогические кадры инновационной России» на 2009–2013 годы, 2011-2013);

- «Разработка методов построения управляющих конечных автоматов по обучающим примерам на основе решения задачи удовлетворения ограничений» (ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы, 2012, 2013);
- «Технология разработки программного обеспечения систем управления ответственными объектами на основе методов машинного обучения и конечных автоматов» (научно-исследовательская работа в рамках проектной части государственного задания в сфере научной деятельности, Минобрнауки, 2014-2016).

Структура диссертации. Диссертация изложена на 129 страницах и состоит из введения, пяти глав, заключения и двух приложений. Список источников содержит 101 наименование. Работа проиллюстрирована 20 рисунками и тремя таблицами.

ГЛАВА 1. ЗАДАЧИ ВЫПОЛНИМОСТИ И УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ, ГЕНЕРАЦИЯ КОНЕЧНЫХ АВТОМАТОВ

В настоящей главе приводятся результаты обзора работ, посвященных задачам выполнимости булевых формул и удовлетворения ограничений, методам их решения, а также конечным автоматам и методам их генерации. На основании результатов обзора формулируются задачи, решаемые в диссертации.

1.1. Задачи выполнимости и удовлетворения ограничений

Задача *выполнимости булевых формул* (задача выполнимости, Boolean satisfiability – SAT) заключается в нахождении выполняющей подстановки значений переменных для заданной булевой формулы. Подстановка называется *выполняющей*, если формула после ее применения становится истинной. В случае существования такой подстановки формула называется *выполнимой*, при ее отсутствии формула является тождественной ложью и называется *невыполнимой*.

Задача выполнимости является исторически первой NP-полной задачей. В теореме Кука-Левина (независимо была доказана американским ученым С. А. Куком [30] и советским математиком Л. А. Левиным [6]) NP-полнота задачи выполнимости булевых формул, записанных в конъюнктивной нормальной форме (КНФ), доказывается непосредственно сведением к ней языка задачи BH_{1N} любой недетерминированной машины Тьюринга с полиномиальным временем работы [30]. NP-полнота задачи позволяет говорить о том, что из существования эффективного метода ее решения в общем случае следует равенство классов P и NP. Такие методы в настоящее время не известны.

Задача удовлетворения ограничений (constraint satisfaction problem – CSP), также именуемая *обобщенной задачей выполнимости*, расширяет задачу SAT для переменных с произвольными значениями взамен булевых.

Пусть имеется набор переменных $x_1, \dots, x_n$, для каждой из которых задано множество допустимых значений. Задача удовлетворения ограничений состоит в подборе таких допустимых значений переменным $x_1, \dots, x_n$, что выполняются все заданные ограничения на эти переменные. Таким образом, экземпляр задачи CSP представляет собой тройку $\langle X, D, \mathcal{C} \rangle$, где X – набор переменных, D – множество их наборов допустимых значений, $\mathcal{C}$ – множество ограничений на X.

Многие прикладные комбинаторные задачи эффективно моделируются на языке ограничений и решаются с использованием специализированных постоянно развивающихся программных средств решения CSP. К таким задачам относятся планирование, обработка изображений, тестирование сверхбольших интегральных схем, анализ естественных языков и языков программирования. Обзор на русском языке по удовлетворению ограничений, программированию в ограничениях, алгоритмам решения и практическому применению приведен в [14].

Вкратце опишем методы решения задач выполнимости и удовлетворения ограничений, а также возможности существующих программных средств решения SAT и CSP.

1.1.1. Методы решения задач выполнимости и удовлетворения ограничений

Несмотря на указанную теоретическую сложность задач выполнимости и удовлетворения ограничений, существует множество эффективных на практике алгоритмов решения. Так, например, применяются алгоритмы *локального поиска* и *метаэвристические алгоритмы*, однако данные подходы являются *неточными* и не гарантируют точности решения: если решение не было найдено, такие алгоритмы не свидетельствуют о невозможности его существования (в общем случае ими не гарантируется нахождение искомого ответа за конечное время). С подробным обзором работ по применению данных

алгоритмов для решения SAT и CSP можно ознакомиться в книге [35], из работ отечественных исследователей можно выделить работы [5, 16]. В настоящей работе решаются задачи точной, а не приближенной генерации конечных автоматов, поэтому в основе решения не могут быть использованы упомянутые приближенные методы. Известные методы и программные средства для точного решения задач выполнимости и удовлетворения ограничений основаны на алгоритме *DPLL*.

Алгоритм Дэвиса-Патнема-Логемана-Лавленда (*DPLL*) [32] был предложен в 1962 году как улучшение алгоритма Дэвиса-Патнема (*DP*) [31]. Спустя более 50 лет алгоритм используется в качестве основы для современных средств решения задач выполнимости и удовлетворения ограничений, а также для систем автоматического доказательства теорем.

Оригинальный алгоритм *DPLL* представляет собой поиск с возвратом для решения задачи CNF-SAT. Рекурсивной функции *DPLL* подается формула ϕ. Если она является тривиальной (логической константой), то функция возвращает, выполнима (TRUE) или невыполнима (FALSE) данная формула ϕ. Если формула ϕ не является тривиальной, то некоторым образом выбирается переменная v и рекурсивно запускается функция *DPLL* с допущением $v = 1$ и соответствующим *упрощением* ϕ. Данное упрощение заключается в том, что из ϕ удаляются все дизъюнкты, в которые v входит без отрицания, а из оставшихся дизъюнктов удаляется вхождение переменной v с отрицанием. Если при данном допущении формула ϕ имеет выполняющую подстановку, то возвращаем TRUE – формула выполнима.

В противном случае повторяем процесс с допущением $v = 0$. Если ни одно из допущений не привело к нахождению выполняющей подстановки, то возвращаем FALSE – формула невыполнима (такая ситуация называется *конфликтом*). Заметим, что описанный алгоритм при

допущениях разбивает задачу на две более простые подзадачи, таким образом поддерживая простую схему параллелизма работы.

Указанные шаги также использовались и алгоритмом *DP*. Алгоритм *DPLL* дополнительно использует правила *распространения переменных* и *исключения «чистых» переменных*. Правило распространения переменных (unit propagation), или правило единичного дизъюнкта, заключается в выделении дизъюнктов, состоящих из одной переменной, и присвоении данным переменным соответствующих для выполнимости формулы значений. На большинстве практических задач после выбора значения для переменной v следует каскад однозначных присваиваний по правилу распространения.

Переменная называется *чистой*, если во все дизъюнкты она входит с одинаковым знаком (либо во всех дизъюнктах с отрицанием, либо во всех без отрицания). При нахождении такой переменной в формуле применяется правило исключения чистых переменных (pure literal elimination) – из формулы удаляются все дизъюнкты, содержащие чистую переменную, после присвоения ей соответствующего выполнимости значения.

Выполнимость формулы ϕ определяется истинностью всех входящих в нее дизъюнктов. Невыполнимость определяется исходя из невозможности найти при сделанных присваиваниях выполняющую подстановку хотя бы для одного дизъюнкта – ситуации его «пустоты» (из дизъюнкта были удалены все переменные, а сам дизъюнкт не был удален из ϕ). Невыполнимость исходно заданной формулы устанавливается после окончания исчерпывающего перебора.

Иллюстрация (заимствована из [84]) работы данного алгоритма приведена на рисунке 1, на котором изображен пример графа состояний алгоритма *DPLL*. Каждой вершине графа соответствует уникальный набор значений переменных булевой формулы, ребра соответствуют добавлению

в набор значения переменной – в каждой вершине выбирается переменная, для которой не было ранее задано значение. Начальное состояние алгоритма с пустым набором значений помечено как «Start». В приведенном примере после последовательности предположений («Guess until conflict») алгоритм переходит в ситуацию невыполнимости формулы («First conflict»). После этого алгоритм откатывается («Backtrack»), отменяя сделанные ранее неверные для выполнимости формулы присвоения значений переменным. Наконец, после ряда предположений и возвратов, алгоритм переходит в состояние выполнимости формулы («Solution found»).

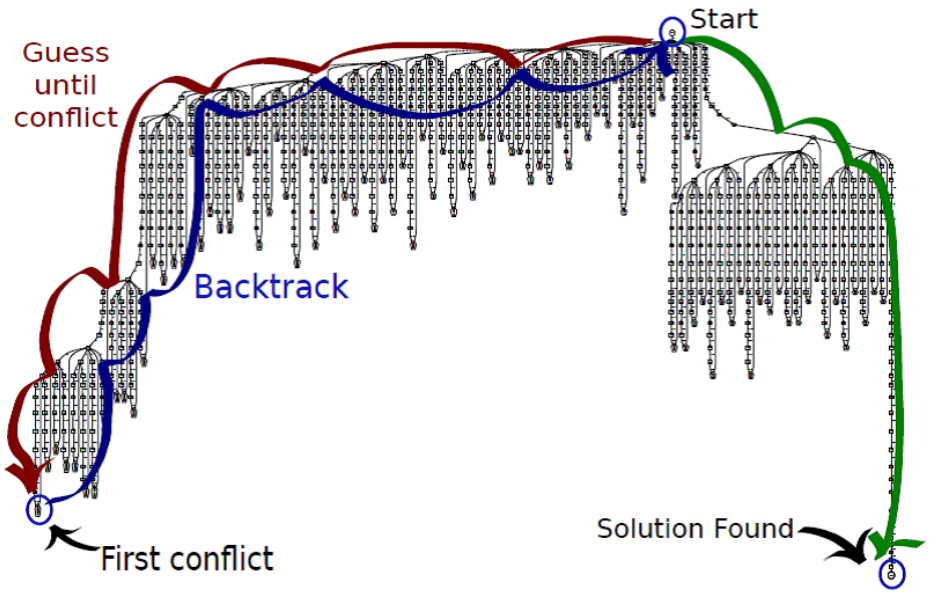

Рисунок 1 – Иллюстрация М. Суса работы алгоритма *DPLL*

В современных программных средствах для решения SAT применяется модификация *DPLL* – стратегия *управляемого конфликтами обучения дизъюнктов* (*conflict-driven clause learning – CDCL*). Алгоритм *CDCL* был независимо предложен в работах [19] и [57]. С подробным обзором работ по *CDCL* можно ознакомиться в книге [22]. Этот алгоритм, помимо использования общей схемы работы *DPLL*, обладает следующими особенностями:

- *обучение дизъюнктов* – создание и периодическое удаление *новых дизъюнктов* в ходе анализа структуры возникающих конфликтов во время поиска с возвратом;
- использование ленивых структур данных для представления формул;
- малое время работы дополнительных эвристик выбора ветвления и использование ими знаний, полученных проделанными ветками поиска;
- периодический перезапуск поиска с возвратом в целом.

Стоит упомянуть ряд зарубежных [29, 63] и отечественных [45, 47, 59] исследований нижних оценок времени работы некоторого класса *DPLL*-алгоритмов. Данные исследования посвящены вопросу времени работы алгоритмов в «худшем случае» на специальным образом полученных формулах. Настоящая диссертация направлена на решение конкретных практических задач, не имеющих отношения к данного рода формулам.

Упомянутые техники, используемые совместно с общей схемой *DPLL*, используются и для решения задачи удовлетворения ограничений. Первая работа по применению поиска с возвратом для удовлетворения ограничений опубликована в 1975 году [23], подробный обзор подходов к решению приведен в работах [14, 51].

Помимо решения задачи CSP «напрямую», существуют методы (и соответствующие программные средства, о которых будет упомянуто позже) сведения CSP к SAT. Задача выполнимости является частным случаем задачи удовлетворения ограничений, однако задачу удовлетворения ограничений также можно (для некоторого определенного вида ограничений) эффективно решать при помощи уже разработанных методов решения задачи выполнимости [70].

1.1.2. Программные средства решения SAT и CSP

С развитием алгоритмов развивались и программные средства для решения задач SAT и CSP. Актуальность развития программных средств повышается с каждым годом – все больше задач оптимизации формулируются на языке ограничений на булевы или целочисленные переменные. В свою очередь, верно и обратное – сведение может быть, как в настоящей диссертации, мотивировано высокой производительностью рассматриваемых средств. С увеличением числа программных реализаций для решения задачи (SAT или CSP) встает вопрос о выборе конкретного программного средства ее решения.

В настоящее время выбор программного средства можно произвести, проанализировав результаты последних *соревнований* между ними. В последнее время соревнования по решению задачи SAT проходят ежегодно [86], при этом публикуются описания участвующих программных средств и задачи для их сравнения [18, 20]. Сравнения проходят как на задачах из практики, так и на случайным образом сгенерированных. На рисунке 2 представлен график производительности программных средств, участвующих в категории «Sequential, Application SAT+UNSAT track» соревнования *SAT Competition* 2014 [86]. Средства перечислены в порядке числа решенных за отведенное время (ось абсцисс «CPU Time (s)», максимум – 5000 секунд на каждый экземпляр) формул из подготовленного набора (ось ординат «number of solved instances», потенциальный максимум – 276 решенных формул).

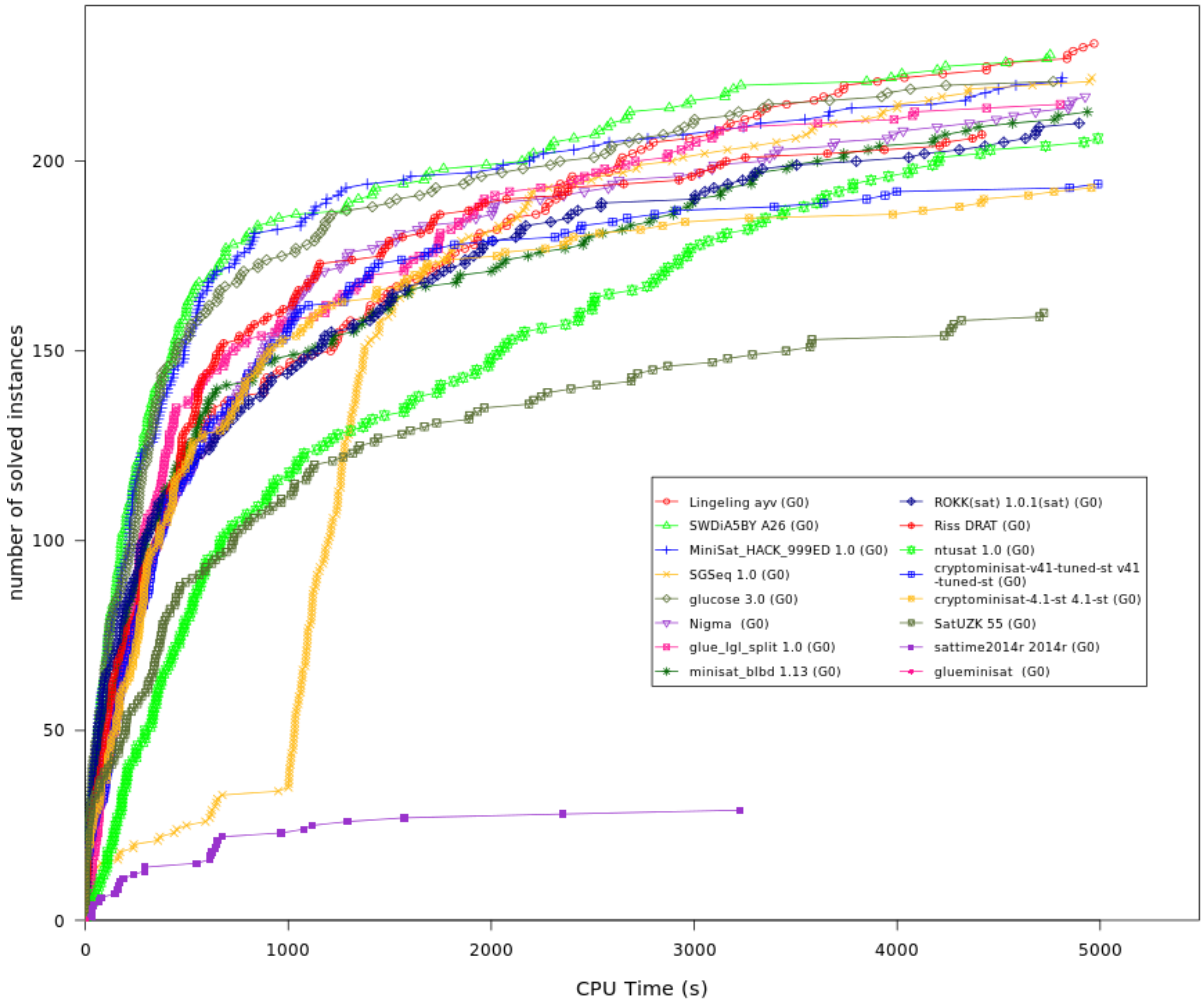

Рисунок 2 – Результаты соревнования SAT Competition 2014

Булева формула подается в конъюнктивной нормальной форме, которой соответствует принятый формат *DIMACS* [83]. Пример записи для формулы $(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)$ в данном формате приведен в листинге 1. Объем формул в таком формате, используемых для соревнований, может достигать сотен мегабайт.

Листинг 1 – Пример булевой формулы в формате *DIMACS*

```
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

Выделим несколько программных средств решения SAT, регулярно принимающих участие и завоевывающих призовые места на указанных

соревнованиях. Некоторые из этих программных средств будут применяться для эффективного решения задач в настоящей диссертации.

MiniSat [38] – одно из самых популярных программных средств для решения задачи выполнимости булевых формул. *MiniSat* популярен за счет того, что авторы выполнили поставленную перед собой задачу – создать средство с малым числом строк исходного кода, который будет как эффективен за счет использования алгоритма *CDCL*, так и понятен начинающим исследователям и разработчикам. Так, в последнем соревновании [20] проходила секция «MiniSAT Hack Track» по наиболее производительным улучшениям рассматриваемой программы. Особенностью данного программного средства является использование препроцессора *SatELite* [37], упрощающего исходную задачу за счет удаления некоторых переменных и ограничений на них.

Одним из самых производительных программных средств, основанных на *MiniSat*, является средство *CryptoMiniSat* [85] (стоит заметить, что *MiniSat* содержит примерно 1500 строк исходного кода, в то время как число строк исходного кода *CryptoMiniSat* достигает 13000). Данное средство победило в главной категории (main track) соревнования *SAT-Race* 2010, завоевывало призовые места на последующих соревнованиях. Изначальной целью автора (М. Суса) являлась разработка программного средства, подбирающего код для некоторого шифрования. Цель была достигнута – за два дня работы *CryptoMiniSat* был подобран ключ для программатора *HiTag2*, используемого в современных системах безопасности автомобилей.

Также в последнее время первые места на соревнованиях регулярно занимает программное средство *lingeling* [21]. Это средство, также основанное на алгоритме *CDCL*, развивается командой А. Биере и включает достижения последних лет в области разработки программных средств решения SAT. В настоящей диссертации не стоит задачи

достижения максимальной производительности за счет выбора программного средства, однако при проведении экспериментов будут использоваться *lingeling* и *CryptoMiniSat*.

Рассмотрим текущее состояние развития программных средств решения CSP. Существуют библиотеки для *программирования в ограничениях*, предоставляющие пользовательские интерфейсы только заданным языкам программирования. Наиболее распространены библиотеки *Choco* [48] и *JaCoP* [79] для языка *Java*, а также *GeCode* [76] и *Z*3 [34] для *C++*. Каждая библиотека предоставляет пользователю уникальный интерфейс, который, во-первых, требуется изучить перед решением поставленной задачи, а, во-вторых, несовместим с интерфейсами аналогичных библиотек. Разумеется, пользователю может быть удобно ограниченное интерфейсом использование, однако для сравнения производительности программных средств и независимости от конкретной реализации библиотеки рационально использовать единый язык записи ограничений.

Таким языком является *MiniZinc* [58], который поддерживают как некоторые указанные библиотеки, так и отдельные программы решения CSP, принимающие на вход файл с заданными ограничениями на языке *MiniZinc*. Приведем пример использования данного языка – в листинге 2 приведен пример записи NP-полной задачи о рюкзаке [80]. Используются целочисленные константы (значения указаны в конце) и переменные «`var`», а также их массивы «`array`»; операторы суммы по индексу «`sum`», произведения; квантор всеобщности «`forall`» и вызов приведенного предиката «`knapsack`». Ограничениям предшествует ключевое слово «`constraint`», предикатам – «`predicate`», оптимизируемому параметру – «`solve`», формату вывода – «`output`».

Листинг 2 – Задача о рюкзаке, записанная на языке *MiniZinc*

```
predicate knapsack (
    array[int] of var int: Weights,
    array[int] of var int: Take,
    var int: Wtmax) =
  sum(i in index_set(Weights)) (Weights[i]*Take[i])
     <= Wtmax;

int: n;
int: weight_max;
array [1..n] of int: values;
array [1..n] of int: weights;
array [1..n] of var int: take;

var int: profit = sum(i in 1..n) (take[i]*values[i]);
solve maximize profit;

constraint
   forall(i in 1..n) (take[i] >= 0)
   /\
   knapsack(weights, take, weight_max);

% data
n = 3;
weight_max = 10;
values = [23, 9, 20];
weights = [4, 3, 2];

output [show(take), "\n"];
```

На данном языке можно наглядно записать большинство задач класса CSP, поэтому он получил широкое распространение, и с его использованием проходят ежегодные соревнования среди программ решения CSP – *MiniZinc Challenge* [65]. По итогам анализа последнего соревнования *MiniZinc Challenge* 2015 среди программных средств

решения CSP можно выделить *OR-tools* [77] и *Opturion CPX* [82], которые будем использовать в дальнейшем.

1.2. Конечные автоматы

Приведем описание основных типов детерминированных автоматных моделей в соответствии с классическими определениями [8, 11]. Отметим, что в настоящей диссертации не рассматриваются вероятностные и недетерминированные модели, вопросы их генерации. Принято использовать следующие термины для описания абстрактных конечных автоматов.

Задано конечное непустое множество символов Σ, которое называется *алфавитом*. Множество всех возможных строк (последовательностей, цепочек, слов), составленных из символов алфавита Σ, обозначается как Σ^*. Пустая последовательность символов обозначается как ε, ее принято рассматривать как единственное слово нулевой длины над любым алфавитом: $\Sigma^0 = \{\varepsilon\}$.

Подмножество L множества всех слов над алфавитом Σ, $L \subset \Sigma^*$, называется *языком*. Говорят, что абстрактный вычислитель *распознает* язык L, если он способен для произвольного слова $l \in \Sigma^*$ определить его принадлежность языку: $l \in L$. Абстрактные автоматы описывают в терминах распознаваемых ими языков – различным автоматным моделям соответствуют разные классы языков. При этом, чем шире класс таких языков, тем большей *вычислительной мощностью* обладает данная автоматная модель.

Детерминированный конечный автомат (*ДКА*) – это пятерка $\langle Q, \Sigma, \delta, q_s, F \rangle$, где Q – конечное множество состояний, Σ – алфавит входных символов, $\delta: Q \times \Sigma \to Q$ – функция переходов, $q_s \in Q$ – начальное (стартовое) состояние, $F \subset Q$ – множество допускающих состояний. ДКА

также называют *конечным автоматом-распознавателем* или просто *автоматом-распознавателем*.

ДКА *допускает* строку $a_1a_2 \dots a_n$, если, начиная работу в состоянии q_s и обработав последовательно символы строки, он оказывается в допускающем состоянии $q_f \in F$ (помеченном *допускающей меткой*). Для более формального определения введем расширенную с символов на последовательности функцию переходов $\hat{\delta}$. Для любого состояния q переход по пустой последовательности не осуществляется: $\hat{\delta}(q, \varepsilon) = q$. Индуктивно определим с использованием δ расширенную функцию для остальных строк $a_1a_2 \dots a_n \in \Sigma^*$ и любого состояния q: $\hat{\delta}(q, a_1a_2 \dots a_n) = \delta\left(\hat{\delta}(q, a_1a_2 \dots a_{n-1}), a_n\right)$. Таким образом, ДКА допускает строку $a_1a_2 \dots a_n$ тогда и только тогда, когда $\hat{\delta}(q_s, a_1a_2 \dots a_n) \in F$, а распознаваемый ДКА язык определяется как

$$L = \left\{l \in \Sigma^* \middle| \hat{\delta}(q_s, l) \in F\right\}.$$

Класс таких языков называют *регулярными* языками. Данный класс совпадает с классом языков регулярных выражений [11]. ДКА стали широко известны после применения их при разработке компиляторов и анализаторов больших текстовых массивов, для проверки протоколов связи, при разработке и проверке цифровых схем.

Приведем пример ДКА из практики проектирования протоколов связи. На рисунке 3 приведен пример автомата, моделирующего работу модуля для входящих звонков [74]. В данном случае используется алфавит команд {ANSWERING_CALL, CALL_CONNECTED, CALL_RELEASED, DC, INCOMING_CALL, REJECTING_CALL}; допускающие состояния (изображены двойной окружностью) соответствуют режимам «1. Бездействие» и «6. Разговор». Автомат изображен при помощи диаграммы состояний; на диаграмме обычно не указываются незначимые переходы – в состоянии игнорируются несоответствующие ему команды.

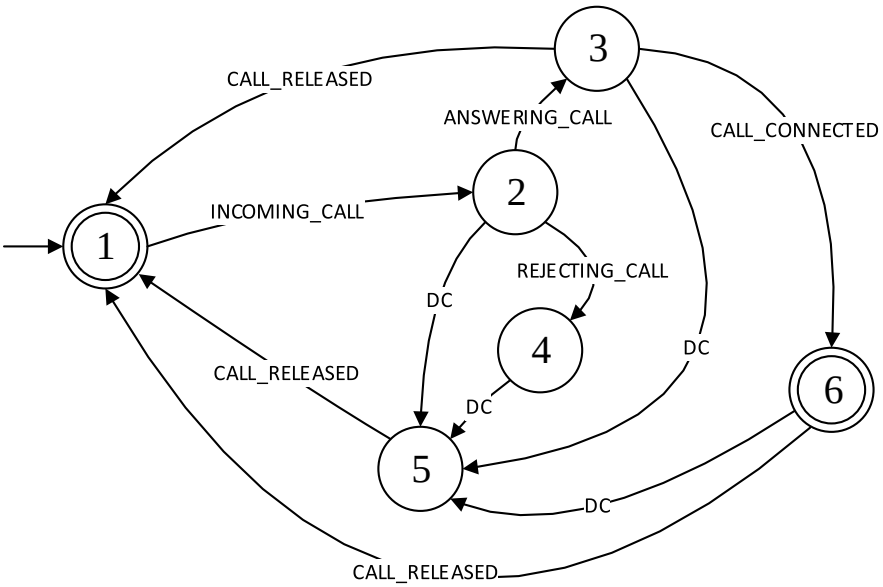

Рисунок 3 – Пример ДКА для протокола входящих звонков

Приведенный детерминированный конечный автомат моделирует протокол связи с высоким уровнем абстракции. Рассмотрим модели автоматов, наделенные возможностью ставить в соответствие входной последовательности символов не решение о допуске или недопуске, а выходную последовательность символов.

Автоматом Мили (Mealy machine) называют шестерку $\langle Q, \Sigma, \mathrm{Z}, \delta, \lambda, q_0 \rangle$, где в дополнение к обозначенным для ДКА сущностям вводятся алфавит выходных символов Z и соответствующая функция выходов $\lambda: Q \times \Sigma \to \mathrm{Z}$. *Автомат Мура* (Moore machine) отличается от автомата Мили тем, что выходными символами помечены не переходы, а состояния: функция выходов имеет вид $\lambda: Q \to \mathrm{Z}$. Также для автоматных моделей с выходными воздействиями алфавит Σ называют *множеством событий*.

Расширим модель автомата Мили для возможности работы с входными переменными. *Управляющим конечным автоматом*, или просто *управляющим автоматом*, называется автомат Мили, каждый переход которого помечен событием, последовательностью выходных воздействий, а также *охранным условием*, представляющим собой булеву формулу от

входных переменных [8]. Формально, управляющим автоматом называется семерка $\langle Q, \Sigma, \mathrm{Z}, \mathrm{X}, \delta, \lambda, q_s \rangle$, где X – множество булевых входных переменных, а функции переходов и действий модифицируются следующим образом: $\delta: Q \times \Sigma \times 2^{\mathrm{X}} \to Q$, $\lambda: Q \times \Sigma \times 2^{\mathrm{X}} \to \mathrm{Z}^*$. Отдельно отметим, что в отличие от автоматов Мили, переходы которых помечены ровно одним выходным символом, управляющие автоматы могут совершить переход без выходных воздействий или с последовательностью выходных воздействий.

В терминах *автоматного программирования* [8], или программирования с *явным выделением состояний*, автомат получает события от так называемых *поставщиков событий* (в их роли могут выступать внешняя среда, интерфейс пользователя и т. д.) и генерирует выходные воздействия для *объекта управления*. При поступлении события автомат выполняет тот соответствующий ему переход, для которого охранное условие оказывается истинным. При выполнении перехода генерируются выходные воздействия, которыми он помечен, и автомат переходит в соответствующее состояние.

В англоязычных работах управляющим автоматам соответствует термин *Extended Finite State Machines* (*EFSM*), под которым разные исследователи подразумевают формально различные модели автоматов. Например, в работе [66] рассматриваются целочисленные входные переменные, а переходы содержат ровно одно выходное воздействие. В работах [15, 49] рассматривается автоматная модель, содержащая также конечное множество внутренних переменных, охранные условия на переходах представляются в виде ограничений на входные и внутренние переменные, выходные воздействия могут вызывать функции среды и изменять внутренние переменные. Похожие модели лежат в основе стандарта проектирования систем распределенного управления и автоматизации *IEC* 61499 [69]. Однако модель, применяемая в стандарте,

соответствует более широкому классу языков за счет наличия в них, не только действий, но и *алгоритмов*, выполняемых в состояниях. Рассматриваемая в настоящей диссертации модель управляющего автомата обладает меньшей выразительной силой, но способна в большом числе случаев моделировать указанные операции над внутренними и входными целочисленными переменными. Приведем и будем рассматривать в дальнейшем два примера управляющих конечных автоматов, соответствующих различным моделям *EFSM*.

На рисунке 4 приведен пример управляющего автомата для торгового аппарата, соответствующего автомату из работы [66]. Автомату на вход поступают события {CHOC, COIN, START, TOFFEE}, соответствующие нажатиям на одну из трех кнопок аппарата и подаче монеты; событие COIN может сопровождаться охранным условием, задающим номинал поданной монеты (x_1 для монеты достоинством 1, $\neg x_1$ для монеты достоинством 2); каждому переходу соответствует выходное воздействие из множества {CHOC, TOFFEE, OK, NO}. Для переходов на диаграммах состояний будем использовать принятую нотацию «событие [охранное условие] / выходные воздействия»; не будем приводить охранное условие, если оно является тавтологией. Также переходы с одними и теми же начальными и конечными состояниями допускается приводить на диаграммах с использованием одной дуги (каждая строка описания дуги соответствует отдельному переходу).

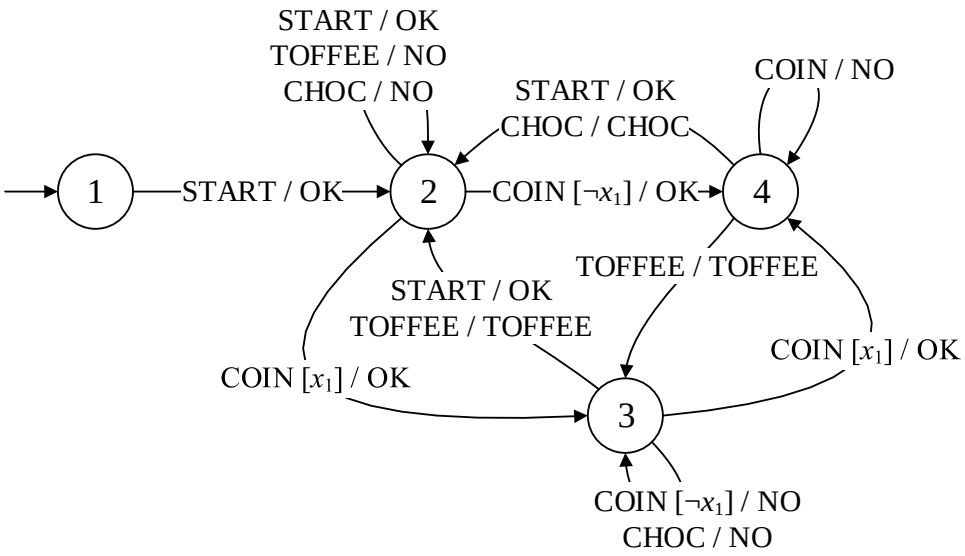

Рисунок 4 – Пример управляющего автомата для торгового аппарата

Рассмотрим еще один пример управляющего автомата, соответствующего модели очереди с приоритетами (priority queue) из стандартной библиотеки *Java Class Library* [71]. Модель *EFSM* была сгенерирована в работе [25], на рисунке 5 приведена аналогичная модель управляющего автомата.

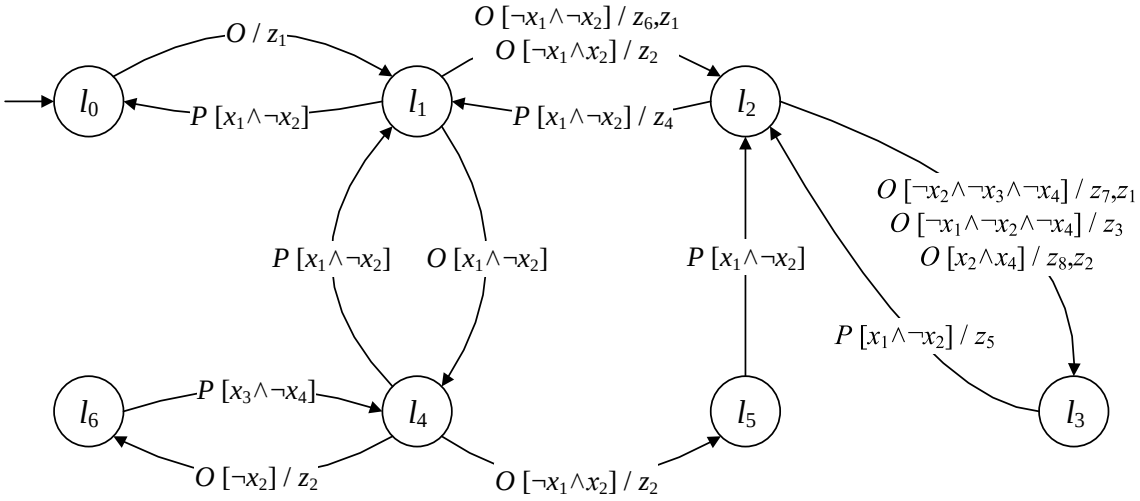

Рисунок 5 – Управляющий автомат, соответствующий очереди с приоритетами из стандартной библиотеки *Java*

В приведенном на рисунке примере, в отличие от приведенного в [25], в соответствие с принципами автоматного программирования [8]

вызовам функций и операциям над переменными для наглядности даны отдельные именования:

- события O и P соответствуют вызовам функций «offer(p)» и «poll(p)»;
- охранные условия на переходах зависят от переменных, соответствующих предикатам от параметра p и внутренних переменных v_1 и v_2: $\{x_1 = (p = v_1),\ x_2 = (p > v_1),\ x_3 = (p = v_2), x_4 = (p > v_2)\}$;
- выходные воздействия соответствуют операциям над внутренними переменными $\{z_1 = (v_1 := p),\ z_2 = (v_2 := p),\ z_3 = (v_3 := p),\ z_4 = (v_1 := v_2),\ z_5 = (v_1 := v_3),\ z_6 = (v_2 := v_1),\ z_7 = (v_3 := v_1),\ z_8 = (v_3 := v_2)\}$.

Для конечных управляющих автоматов примерами поведения, аналогичными последовательностям символов для ДКА, являются сценарии работы. *Сценарием работы* называется последовательность $T_1, \dots, T_n$ троек $T_i = \langle e_i; f_i; A_i \rangle$, где e_i – входное событие, f_i – булева формула от входных переменных, задающая охранное условие, A_i – последовательность выходных воздействий. Данные тройки T_i называют *элементами сценария*.

Автомат, находясь в состоянии q, *удовлетворяет элементу сценария* T_i, если из q исходит переход, помеченный событием e_i, последовательностью выходных воздействий A_i и охранным условием, тождественно равным f_i как булева формула. Автомат *удовлетворяет сценарию работы* $T_1, \dots, T_n$, если он удовлетворяет каждому элементу данного сценария, находясь при этом в состояниях пути, образованного соответствующими переходами. Проще говоря, управляющий автомат удовлетворяет сценарию работы, если в графе переходов автомата существует путь, полностью совпадающий со сценарием $T_1, \dots, T_n$. В

работе [66] приводятся примеры сценариев работы торгового аппарата. Автомат (рисунок 4) удовлетворяет следующим сценариям:

- $\langle \mathrm{START}; 1; \mathrm{OK} \rangle$, $\langle \mathrm{COIN}; x_1; \mathrm{OK} \rangle$, $\langle \mathrm{COIN}; \neg x_1; \mathrm{NO} \rangle$, $\langle \mathrm{COIN}; x_1; \mathrm{OK} \rangle$;
- $\langle \mathrm{START}; 1; \mathrm{OK} \rangle$, $\langle \mathrm{START}; 1; \mathrm{OK} \rangle$, $\langle \mathrm{TOFFEE}; 1; \mathrm{NO} \rangle$, $\langle \mathrm{CHOC}; 1; \mathrm{NO} \rangle$;
- $\langle \mathrm{START}; 1; \mathrm{OK} \rangle$, $\langle \mathrm{COIN}; \neg x_1; \mathrm{OK} \rangle$, $\langle \mathrm{TOFFEE}; 1; \mathrm{TOFFEE} \rangle$.

Также данный автомат, например, не удовлетворяет сценарию $\langle \mathrm{START}; 1; \mathrm{OK} \rangle$, $\langle \mathrm{COIN}; \neg x_1; \mathrm{OK} \rangle$, $\langle \mathrm{TOFFEE}; 1; \mathrm{NO} \rangle$.

1.3. Методы генерации конечных автоматов

Рассмотрим известные в настоящее время методы и алгоритмы генерации конечных автоматов. Настоящая диссертация направлена на развитие этих методов. Особое внимание уделяется методам генерации ДКА и управляющих конечных автоматов по примерам поведения. Задача генерации ДКА по обучающим словарям является одной из центральных задач грамматического вывода. В данном разделе обозреваются основные подходы к ее решению: эвристические и метаэвристические методы, использование программных средств решения NP-полных задач. Некоторые из подходов затем были модифицированы (на кафедре «Компьютерных технологий» Университета ИТМО) для построения управляющих автоматов по примерам поведения. Часть приведенных результатов была использована при выполнении диссертантом магистерской диссертации по теме «Построение управляющих конечных автоматов по сценариям работы на основе решения задачи удовлетворения ограничений».

Помимо задач генерации и анализа программ обозреваемые методы нашли свое применение, например, в биоинформатике, лингвистике, распознавании речи, робототехнике [33, 62]. Примеры поведения для обозреваемых методов при этом могут быть получены как автоматизировано (протоколы работы существующей программы, биологические последовательности, обработанные аудиозаписи речи,

примеры поведения робота, и т. д.), так и построены вручную специалистом.

1.3.1. Эвристические методы генерации конечных автоматов

Опишем эвристические методы генерации конечных автоматов, которым в англоязычных источниках соответствует словосочетание «state merging». Основной идеей работы этих методов, основанных на объединении (*слиянии*) состояний, является построение *префиксного дерева с пометками* (*префиксное дерево*, *augmented prefix tree acceptor – APTA*) и последовательное отождествление его состояний, в результате чего получается конечный автомат. Входными данными для построения префиксного дерева являются набор допускаемых слов S_+ и набор недопускаемых слов S_-, при этом наборы не пересекаются. Такие наборы будем называть *обучающими словарями*.

Состояния префиксного дерева могут быть *помеченными* (допускающими и недопускающими) или *непомеченными*. Для каждого слова из наборов S_+ и S_- в соответствующем ему последнем состоянии дерева присутствует пометка о допуске или недопуске. Пример префиксного дерева для $S_+ = \{ab, b, ba, bbb\}$ и $S_- = \{abbb, baba\}$ приведен на рисунке 6.

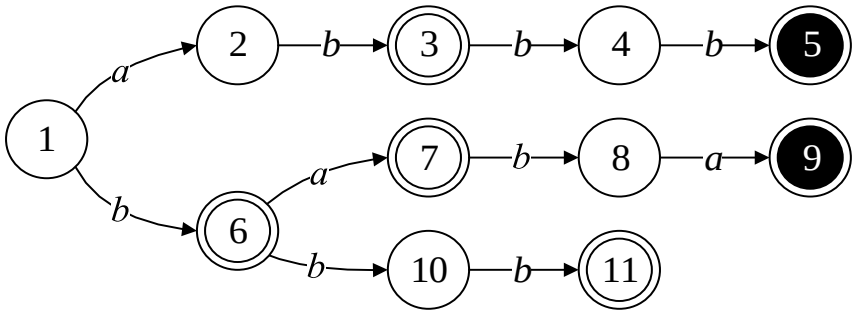

Рисунок 6 – Пример префиксного дерева

Хронологически первым алгоритмом для решения задачи построения минимального автомата, соответствующего входному набору слов, стал *ТВ-алгоритм*, предложенный Б. А. Трахтенбротом и

Я. М. Барздином в 1973 году [68]. *TB*-алгоритм решает частный случай задачи генерации ДКА: все слова над Σ длиной до выбранного натурального числа должны содержаться в одном из наборов S_+ или S_-. Отметим, что ранее описанный набор шести слов не соответствует этому требованию.

TB-алгоритм строит минимальный автомат, совместимый с S_+ и S_-. Алгоритм начинает свою работу с построения префиксного дерева, и, поскольку S_+ и S_- содержат все слова ограниченной выбранным числом длины, все состояния дерева будут помеченными. Алгоритм представляет собой два вложенных цикла, перебирающих состояния дерева в порядке обхода в ширину. Каждая пара различных состояний u и v (для определенности будем считать, что v не дальше от корня, чем u) проверяется на эквивалентность: переходы из обоих состояний по всем строкам длины k (где k – минимум из высот поддеревьев с корнями в u и v) должны приводить в состояния с одинаковыми пометками. Если состояния эквивалентны, то поддерево с корнем в u удаляется, а переход из родителя u «переставляется» на v. В противном случае слияние состояний невозможно. Алгоритм работает за время $\mathcal{O}(PC^2)$, где P – размер префиксного дерева, а C – число состояний в минимальном автомате.

К. Лангом была предложена улучшенная версия *TB*-алгоритма, которая получила название *Traxbar* [53]. Отличие алгоритма заключается в отсутствии требования к обучающим словарям S_+ и S_-. Пары состояний для слияния перебираются в порядке обхода в ширину, как и в *TB*-алгоритме, однако само слияние состояний производится более сложным способом. Для проверки двух состояний на совместимость делается попытка осуществить их слияние, сохранив информацию, с помощью которой можно вернуться назад. При слиянии состояний u и v, где u дальше от корня в порядке обхода в ширину, метки из поддерева с корнем

в u должны быть скопированы в подграф, получившийся из поддерева с корнем в v. Алгоритм *Traxbar* не дает гарантий минимальности результирующего автомата, однако порядок обхода состояний для слияния повышает вероятность того, что первые слияния, которые осуществляет алгоритм, ведут к построению минимального автомата.

Алгоритм Голда [41] – еще один эвристический алгоритм построения минимального автомата по тестам. Для его работы не требуется полноты набора тестов, однако набор должен содержать некоторое особое *характеристическое множество* регулярного языка, иначе алгоритм построит несовместимый с S_+ или S_- автомат.

Алгоритм, названный *RPNI* (*regular positive and negative inference*), был предложен в 1992 году [60]. В начале работы алгоритма строится префиксное дерево по словарю S_+ (оно не будет допускать слова из S_-). После этого производится упорядоченный поиск среди разбиений множества вершин префиксного дерева на классы эквивалентности: в двойном цикле, перебирающем пары различных вершин (u, v) префиксного дерева, осуществляется попытка слить классы эквивалентности состояний, к которым принадлежат u и v. В результате слияния может получиться недетерминированный автомат, для устранения этого проводятся дополнительные слияния. В случае, если получившийся детерминированный автомат не допускает слова из S_-, этот автомат и соответствующее ему разбиение заменяют текущие (автомат и разбиение). В противном случае, текущий автомат и текущее разбиение не меняются. Алгоритм работает за время $O((m+n)n^2)$, где n – суммарная длина слов из S_+, m – суммарная длина слов из S_-.

В 1998 году было проведено соревнование *Abbadingo One* [54], целью которого являлось развитие алгоритмов построения автоматов-распознавателей по обучающим примерам. Каждая задача соревнования представляла собой два набора словарей. Первый набор являлся

обучающим для алгоритма и генерировался по случайному ДКА $\mathcal{A}$ заданного размера. Другой набор словарей также генерировался по автомату $\mathcal{A}$, но использовался для сравнения $\mathcal{A}$ с автоматом $\mathcal{A}'$, полученным в результате работы алгоритма участника соревнования. Задача считалась решенной, если 99 % слов из второго набора словарей верно распознавалась автоматом участника.

Одним из двух победителей соревнования стал алгоритм *объединения состояний на основе свидетельств*, или *EDSM* (*evidence-driven state merging*). Алгоритм хранит текущую структуру искомого автомата, которая инициализируется префиксным деревом обучающих словарей. Далее на каждом шаге алгоритма происходит слияние некоторых состояний текущего автомата. Отличительной особенностью алгоритма является наличие функции $f(u, v)$, оценивающей на текущем шаге все возможные слияния, основываясь на поддеревьях u и v. После вычисления f для всех пар возможных слияний осуществляется слияние с наибольшим значением f. Таким образом, алгоритм является *жадным*. Алгоритм завершается при отсутствии допустимого слияния пары состояний текущего автомата.

Модификацией алгоритма *EDSM* является алгоритм *Blue-Fringe*, предложенный после подведения итогов соревнования *Abbadingo One* [54]. Данный алгоритм накладывает дополнительные ограничения на порядок слияний вершин префиксного дерева. На каждом шаге алгоритма состояния автомата разделены на три множества. Каждому множеству условно соответствует цвет. Множество красных состояний (помечены «R») – попарно не сливаемые состояния, которые являются частью результирующего автомата. Все состояния, не являющиеся красными, в которые ведут переходы из красных состояний, являются синими (помечены «B»). Множеству необработанных состояний соответствует

белый цвет. Пример раскраски состояний после нескольких шагов алгоритма приведен на рисунке 7.

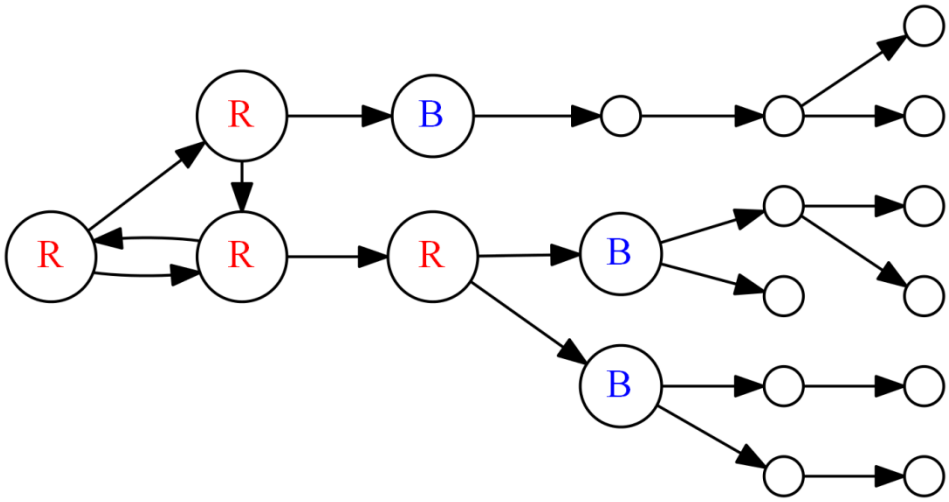

Рисунок 7 – Пример автомата после нескольких шагов алгоритма *Blue-Fringe*

На каждом шаге алгоритма самое близкое к корню синие состояние, которое нельзя слить ни с одним из красных, перекрашивается в красный цвет. Затем, если такого синего состояния не существует, выбирается пара состояний красного и синего цвета с наибольшим значением функции, оценивающей слияния, и производится слияние выбранной пары состояний. В процессе работы алгоритма производится перекраска в синий цвет белых состояний, которые стали потомками красных. Время работы *Blue-Fringe* было оценено сверху как $\mathcal{O}(PC^3)$, где P – размер префиксного дерева, C – число состояний в результирующем автомате.

Другой модификацией алгоритма *EDSM* является *Windowed-EDSM* (*W-EDSM*) [28]. В предложенном подходе рассматриваются только слияния состояний, находящихся в некотором *окне*. Окно представляет собой множество состояний, находящихся в пределах некоторого числа шагов работы обхода в ширину, запущенного от корня дерева. Если в результате слияния размер окна уменьшается, в него добавляются новые вершины, также в порядке обхода в ширину. В случае отсутствия возможных слияний внутри окна его размер удваивается. Как и в классической версии

EDSM, алгоритм завершается, когда не остается допустимой для слияния пары состояний.

За счет уменьшения числа рассматриваемых слияний время работы *W-EDSM* меньше, чем у обычного *EDSM*. Это означает, что *W-EDSM* может быть применен к задачам с большей размерностью. Тем не менее, уменьшение времени работы может компенсироваться ухудшением качества результирующего автомата вследствие более высокой вероятности совершить неправильное слияние.

1.3.2. Методы генерации, основанные на метаэвристических алгоритмах

Другим распространенным подходом к генерации конечных автоматов является применение *метаэвристических алгоритмов* (*метаэвристик*) [24]. Такие методы генерации подробно рассмотрены в диссертационных работах Ф. Н. Царева [12] и К. В. Егорова [3]. Среди метаэвристик, успешно применяемых для генерации конечных автоматов по примерам поведения, можно выделить эволюционные стратегии, генетические алгоритмы и муравьиные алгоритмы.

Эволюционные алгоритмы [17] поддерживают в процессе своей работы популяцию (поколение) *особей*, являющихся кандидатами на решение поставленной задачи оптимизации. Название обусловлено тем, что работа данного семейства алгоритмов соответствует эволюционной теории Дарвина. Первое поколение формируется обычно случайным образом, а каждое следующее поколение генерируется из предыдущего при помощи операторов *мутации* и *скрещивания* (*кроссовера*).

Оператор мутации $f_m(x)$ применяется к одной особи и вносит в нее обычно небольшие изменения (изменяет небольшое число *генов* особи). Оператор скрещивания $f_c(x, y)$ принимает две особи, обменивает их гены, и возвращает получившуюся пару «дочерних» особей. Выбор особей предыдущего поколения, для которых применяются операторы f_m и f_c,

производит *оператор селекции*; качество особей определяет *функция приспособленности* (*фитнесс-функция*).

К эволюционным алгоритмам можно отнести эволюционные стратегии, использующие только оператор мутации, и генетические алгоритмы, использующие операторы скрещивания и, в большинстве случаев, мутации. Приведем краткий обзор работ по применению обозначенных алгоритмов к задачам генерации конечных автоматов по примерам поведения.

Одно из первых применений эволюционных стратегий к задаче генерации ДКА по обучающим словарям изложено в работе [67]. Особью эволюционного алгоритма является детерминированный конечный автомат, который алгоритм изменяет с целью соответствовать обучающим словарям S_+ и S_-. Мутация ДКА случайно применяет одно из изменений к ДКА: добавление и удаление перехода, изменение конечного состояния перехода, изменение пометки состояния. Функция приспособленности вычисляется как разность между правильно и неправильно помеченными словами из обучающих словарей. Таким образом, метаэвристические алгоритмы поддерживают возможность генерации конечных автоматов не только по безошибочным примерам поведения, но и по зашумленным (при уменьшенном целевом значении функции приспособленности).

На данный момент лучшую модификацию эволюционного алгоритма для генерации ДКА предложили С. Лукас и Т. Рейнольдс [55]. Основной особенностью работы является отсутствие информации о метках допуска в состояниях особи эволюционного алгоритма. Таким образом происходит существенное сокращение пространства поиска, которое составлено из всевозможных «скелетов» ДКА с C состояниями. Метки состояниям присваиваются при вычислении функции приспособленности в соответствии с обучающими словарями при помощи *алгоритма расстановки пометок*. Сравнение с алгоритмом слияния состояний,

описанным ранее, показало преимущество предложенного алгоритма генерации ДКА в скорости работы при небольших значениях C. В последующей работе авторы модифицировали эволюционный алгоритм с расстановкой пометок для автоматов Мили [56].

В работе [42] предложена модификация описанного алгоритма генерации ДКА, использующего расстановку пометок. Модификация заключается в применении постепенного обучения: словари S_+ и S_- перед началом работы разбиваются на *блоки* (в блок с меньшим номером попадают слова меньшей длины). На каждом шаге i алгоритма эволюционный алгоритм генерации ДКА добавляет блок i в текущий обучающий набор и генерирует автомат, полностью ему удовлетворяющий. Постепенное обучение позволило ускорить время работы метода [55], хотя качество генерируемых автоматов уменьшилось. Метод занял первое место на соревновании «Learning DFA from Noisy Samples», проходившем в рамках конференции GECCO 2004 [81].

Описанные алгоритмы были модифицированы и применены для задач генерации управляющих конечных автоматов по примерам поведения. Эти алгоритмы разрабатываются, в основном, на кафедре «Компьютерные технологии» Университета ИТМО. Реферативно приведем основные результаты исследований, описанных в [3, 12].

Цель исследования [3] заключалась в разработке метода совместного применения генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением. Исходными данными являлись *тесты* (каждый тест состоит из входной последовательности событий и соответствующей ей последовательности выходных действий, которую должен вырабатывать автомат) для системы со сложным поведением и утверждения на языке логики линейного времени (*Linear Time Logic – LTL*). В предыдущей работе [13] построение автоматов осуществлялось только на основе тестов.

Функция приспособленности, используемая в алгоритме генетического программирования [12], учитывает успешность прохождения тестов и истинность *LTL*-формул. В случае прохождения всех тестов и выполнения всех *LTL*-формул можно считать, что автомат с заранее заданным поведением построен. На одной из задач применение верификации позволяет построить удовлетворяющий спецификации автомат управления дверьми лифта, который не получается построить только на основе тестов, а на другой применение верификации замедляет построение автомата.

В работе [26] был разработан и реализован метод *MuACO* генерации конечных автоматов по заданной функции приспособленности, основанный на *муравьином алгоритме*. Эффективность разработанного метода была оценена путем сравнения с генетическими алгоритмами для задачи об «Умном муравье» и задачи генерации автоматов на основе тестовых примеров. Эксперименты показали, что среднее время работы предлагаемого в [26] алгоритма для построения целевых автоматов в этих задачах в несколько раз меньше времени работы генетического алгоритма.

В работах [1, 2] предлагается применять метаэвристики для генерации автоматов управления объектами со сложным поведением на примере модели беспилотного самолета. При этом вместо подхода [7], в котором для оценки качества управляющего автомата используется моделирование, занимающее обычно большое время, применяется подход, в котором выполняется сравнение поведения автоматов с поведением, обеспечиваемым за счет управления человеком. Особенность рассматриваемого подхода состоит в том, что он позволяет использовать объекты управления не только с дискретными, но и с непрерывными параметрами. Применение подхода иллюстрируется на примере создания автомата, управляющего моделью самолета при выполнении фигур высшего пилотажа. Отметим, что в настоящей диссертации будут

рассматриваться только дискретные параметры обучающих примеров поведения.

1.3.3. Методы генерации, основанные на сведении к задачам из класса трудных в NP

Как уже отмечалось во введении, в последнее время все чаще применяется подход к решению NP-трудных задач, в основе которого используются производительные программные средства решения таких задач, как SAT и CSP. Подход применяют и отечественные исследователи. К примеру, в институте динамики систем и теории управления Сибирского отделения РАН средства решения SAT применяются для *анализа* дискретных автоматов [39] и мультиагентных систем [50], а также при разработке комплекса *Transalg* [9], используемого для решения задач криптографического анализа. В Уральском федеральном университете средства решения SAT применяют для нахождения кратчайших синхронизирующих слов ДКА [64].

Подход с применением производительных программных средств применим и для генерации конечных автоматов по безошибочным обучающим словарям. В работе [44] представлен алгоритм построения детерминированных конечных автоматов по тестовым наборам *DFASAT*, принципиально отличающийся от предлагаемых ранее алгоритмов для решения поставленной задачи. Отличие заключается в том, что данный алгоритм основан на сведении поставленной задачи к задаче SAT. Алгоритм состоит из следующих пяти этапов:

1. Построение префиксного дерева.
2. Построение графа совместимости вершин префиксного дерева.
3. Построение булевой КНФ-формулы, в случае выполнимости которой можно построить автомат с числом состояний, равным C (которое увеличивается в случае невыполнимости формулы).

4. Запуск стороннего программного средства решения задачи SAT для нахождения выполняющей подстановки построенной КНФ-формулы.
5. Построение искомого автомата по подстановке, в случае ее нахождения.

Первым шагом решения задачи, как и в описанных эвристических алгоритмах слияния состояний (state merging), является построение префиксного дерева. В дальнейшем будем обозначать вершины дерева как V, допускающие вершины как V_+, недопускающие – как V_-. Для решения поставленной задачи необходимо найти соответствие между каждой вершиной префиксного дерева и одним из C состояний искомого автомата.

Вторым шагом алгоритма является построение графа совместимости префиксного дерева. Множество вершин данного графа совпадает с множеством вершин префиксного дерева, а две вершины соединены ребром, если они не могут соответствовать одному состоянию искомого ДКА: при их слиянии нарушается свойство детерминированности. На рисунке 8 приведен пример графа совместимости, построенного для префиксного дерева, приведенного на рисунке 6.

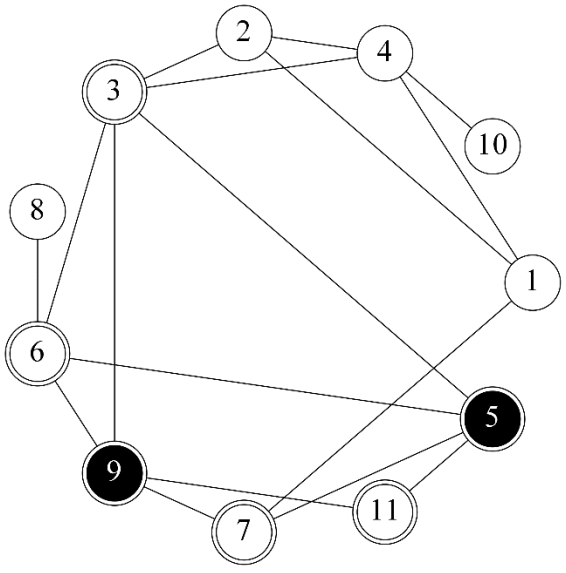

Рисунок 8 – Пример графа совместимости

К примеру, ребро на рисунке 8 соединяет вершины 1 и 7 потому, что их слияние приведет к последующему слиянию (по строке ba) вершин 7 и 9 с разными пометками, что недопустимо. Граф совместимости строится методом динамического программирования: «ленивым» образом вычисляются несовместимые с v вершины исходя из вычисленных вершин, несовместимых с детьми v. Так как для решения задачи нам необходимо (но не достаточно) найти раскраску графа совместимости, будем говорить, что каждая вершина префиксного дерева *раскрашена* в один из C цветов.

Третьим, наиболее интересным шагом, является построение булевой формулы по удовлетворяющей подстановке которой можно построить решение задачи (если такой существует). Авторы [44] предложили два способа построения данной формулы, ограничимся подробным описанием эффективного способа (*compact encoding*).

Опишем булевы переменные, которые используются для построения дизъюнктов формулы.

1. Основными переменными являются *переменные цвета* (*color variables*) $x_{v,i}$, определенные для каждой вершины префиксного дерева v и каждого «цвета» $i \in 1..C$. Истинность переменной $x_{v,i}$ соответствует тому, что вершина v покрашена в цвет i (ей соответствует состояние i искомого автомата). Можно построить КНФ-формулу лишь с использованием данных переменных (*direct encoding*), ее размер составит $O(|C|^2|V|^2)$.
2. *Переменные переходов* $y_{a,i,j}$ (*parent relation variables*) заданы для каждого символа алфавита a, каждой пары цветов $i,j \in 1..C$. Истинность переменной $y_{a,i,j}$ соответствует тому, что из каждой вершины цвета i переход по символу a ведет в вершину с цветом j. Данные переменные задают таблицу переходов генерируемого

автомата. С использованием данных переменных размер КНФ-формулы можно сократить до $\mathcal{O}(|C|^2|V|)$.

3. Для большего сокращения размера формулы вводятся *переменные допуска* (*accepting color variables*) z_i, определенные для каждого цвета i. Истинность данной переменной соответствует тому, что вершины префиксного дерева цвета i (состояние i искомого автомата) являются допускающими.

Заметим, что вспомогательные переменные y и z характеризуют лишь искомый автомат, не обращаясь при этом к префиксному дереву. С использованием описанных переменных построим КНФ-формулу, решение которой будет соответствовать корректной раскраске префиксного дерева. Данная формула состоит из следующих дизъюнктов.

1. $x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,|C|}$ $(v \in V)$ – каждой вершине v соответствует хотя бы один цвет.
2. $(\neg x_{v,i} \vee z_i) \wedge (\neg x_{w,i} \vee \neg z_i)$ $(v \in V_+; w \in V_-; i \in 1..C)$ – допускающие и недопускающие вершины не могут быть одного цвета i. Как только фиксируется цвет допускающей или недопускающей вершины, для соответствующего состояния автомата фиксируется значение флага допуска. Дизъюнкты являются преобразованием в КНФ выражения $(x_{v,i} \Rightarrow z_i) \wedge (x_{w,i} \Rightarrow \neg z_i)$.
3. $y_{l(v),i,j} \vee \neg x_{p(v),i} \vee \neg x_{v,j}$ $(v \in V; i,j \in 1..C)$ – переменная переходов по символу $l(v)$ (символ, которым помечен ведущий в v переход) определена, если определены цвета вершины v и его родителя $p(v)$: $(x_{p(v),i} \wedge x_{v,j}) \Rightarrow y_{l(v),i,j}$.
4. $\neg y_{a,i,j} \vee \neg y_{a,i,h}$ $(a \in \Sigma; i,j,h \in 1..C; j < h)$ – существует не более одного перехода из состояния i по символу a.

Также в формулу добавляются дополнительные дизъюнкты, наличие которых упрощает работу программного средства, решающего задачу выполнимости.

5. $\neg x_{v,i} \vee \neg x_{v,j} \quad (v \in V; \quad i,j \in 1..C; \quad i < j)$ – каждой вершине префиксного дерева соответствует не более одного цвета.
6. $y_{a,i,1} \vee y_{a,i,2} \vee \ldots \vee y_{a,i,|C|}$ $(a \in \Sigma; i \in 1..C)$ – существует хотя бы один переход из состояния i по символу a.
7. $\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j}$ $(v \in V; i,j \in 1..C)$ – обратное следствие для утверждения 3. Цвет вершины v определен, если определен цвет ее родителя $p(v)$ и переменная соответствующего перехода автомата $i \xrightarrow{l(v)} j$: $\left(x_{p(v),i} \wedge y_{l(v),i,j}\right) \Rightarrow x_{v,j}$.
8. $\neg x_{v,i} \vee \neg x_{w,i}$ $((v,w) \in E; \ i \in 1..C)$ – цвета вершин, соединенных ребром графа совместимости, различны.

Четвертым шагом алгоритма является запуск программного средства для решения задачи SAT на построенной булевой формуле. Предложенное сведение строит формулу с больш*и*м числом дизъюнктов на некоторых задачах. Так, на ряде задач, предложенных на упомянутом соревновании *Abbadingo One* [54], размер формулы составляет более 10^8 дизъюнктов, в то время как современные программные средства справляются с порядка 10^7 дизъюнктов.

Для упрощения формулы авторы предложили перед построением булевой формулы выполнить на префиксном дереве несколько шагов алгоритма *EDSM* [54]. Каждое слияние значительно сокращает размер префиксного дерева, что приводит к уменьшению формулы. Так как слияния производятся жадно, может быть произведено некорректное слияние. Однако, как утверждают авторы, на рассмотренных задачах несколько первых слияний обычно не являются ошибочными.

Пятым, заключительным шагом алгоритма является построение автомата по найденной выполняющей подстановке. Если такой подстановки не было найдено, то автомат с заданным числом состояний C по заданным обучающим словарям построить нельзя. Если же подстановка была найдена, то искомый ДКА строится, исходя из найденных значений переменных $x_{v,i}$.

Опишем теперь используемую авторами технику *нарушения симметрии* (*symmetry breaking – SB*), идея которой заключается в следующем. Пусть требуется раскрасить некоторый граф в C цветов с помощью средства решения задачи SAT при том, что этого сделать нельзя. В таком случае для того, чтобы убедиться в отсутствии решения формулы, средство переберет до $C!$ вариантов решения – по варианту на каждую перестановку цветов. Для предотвращения подобного на этапе преподсчета (в нашем случае после построения графа совместимости) фиксируются цвета вершин некоторой клики большого размера, таким образом «нарушая» рассмотрение симметричных решений. Так как задача нахождения наибольшей клики NP-полна, авторы закрепляют цвета большой клики, найденной с помощью приведенного в [44] жадного алгоритма.

Также авторы предложили следующий метод генерации *наименьшего* (с наименьшим числом состояний) ДКА по заданному словарю:

1. Построить префиксное дерево и его граф совместимости.
2. Найти клику clique большого размера в графе совместимости.
3. Инициализировать число цветов C размером найденной клики.
4. Построить булеву формулу в соответствии с числом цветов C и зафиксированными цветами вершин клики clique.
5. Запустить программное средство решения SAT для нахождения выполняющей подстановки построенной формулы.

6. Если формула невыполнима, то добавить один цвет ($C := C + 1$) и вернуться к шагу 4.
7. Вернуть ДКА, соответствующий выполняющей подстановке, найденной на шаге 5.

Экспериментальное исследование [44] состояло в том, что сравнивались предложенные авторами методы нахождения минимального автомата (со сведениями *direct encoding*, *compact encoding* и без дополнительных дизъюнктов) с лучшими алгоритмами слияния состояний *exbar* и *ed-beam*. Рассматривались задачи генерации автоматов с числом состояний от 16 до 21; каждому запуску отводилось 200 секунд для решения; авторами использовалось свободно распространяемое средство *picosat*. Результаты показали, что все методы, помимо «наивного» сведения, работают за доли секунды на большинстве предложенных задач, однако на сложных задачах может быть применим только описываемый метод (с дополнительными дизъюнктами). Однако даже предложенный метод неприменим для точной генерации ДКА с числом состояний, превышающим 19.

В работе [40] предлагается алгоритм сведения задачи построения ДКА по словарям S_+ и S_- к задаче раскраски неориентированных графов – другой классической NP-полной задаче. Этапы метода аналогичны описанным ранее этапам метода *DFASAT*. Работа носит скорее теоретический характер – на настоящий момент производительность методов раскраски графов существенно уступает производительности методов решения SAT и CSP.

1.4. Задачи, решаемые в диссертационной работе

Из выполненного обзора следует, что существующие методы генерации конечных автоматов обладают следующими недостатками.

1. Низкая скорость работы существующих точных методов генерации ДКА по безошибочным обучающим словарям при большом числе

состояний искомого автомата. Качественным недостатком этих методов является невозможность их работы при наличии ошибок в метках допуска/недопуска заданных слов.

2. Неточность существующих методов генерации управляющих конечных автоматов по сценариям работы (а также по другим рассмотренным примерам поведения). Существующие методы основаны на метаэвристиках, применение которых позволяет решать практические задачи, но не гарантирует в общем случае того, что автомат будет сгенерирован за конечное время.

Цель диссертационной работы – разработка точных методов генерации ДКА и управляющих автоматов с использованием программных средств решения задач выполнимости и удовлетворения ограничений. Формально поставим задачи, решаемые в диссертационной работе.

1. Задача генерации ДКА по незашумленным примерам. Данная задача является одной из самых популярных задач грамматического вывода. Пусть заданы обучающие словари S_+ и S_- над алфавитом Σ. Необходимо найти ДКА $\mathcal{A}_{\mathrm{DFA}} = \langle Q, \Sigma, \delta, q_s, F \rangle$ такой, что он допускает все строки из S_+, не допускает все строки из S_- и содержит минимальное число состояний $C = |Q|$. Данная задача сводится к задаче построения ДКА с заданным числом состояний (заданного размера) C: будем увеличивать данное число, пока не найдется автомат размера C, он и будет наименьшим.

2. Задача генерации ДКА по зашумленным примерам. Следующее обобщение предыдущей задачи предлагалось участникам соревнования «Learning DFA from Noisy Samples» конференции GECCO 2004 [81]. Пусть заданы размер искомого автомата C, обучающие словари S_+ и S_- над алфавитом Σ и максимальное число ошибок K. Необходимо найти ДКА $\mathcal{A}_{\mathrm{DFA}}$ размера C такой, что он совершает не более K ошибок

при обработке слов: сумма недопущенных слов из S_+ и допущенных из S_- не превосходит K.

3. Задача генерации управляющих автоматов по безошибочным сценариям работы. Пусть задан размер искомого автомата C и конечный обучающий набор сценариев работы S, при этом обозначим множество встречаемых в сценариях входных событий как Σ, входных переменных как X, выходных воздействий как Z. Необходимо найти такой управляющий конечный автомат $\mathcal{A}_{\mathrm{EFSM}} = \langle Q, \Sigma, \mathrm{Z}, \mathrm{X}, \delta, \lambda, q_s \rangle$, что $C = |Q|$ и автомат удовлетворяет всем обучающим сценариям работы S. Заметим, что таким образом функции переходов $\delta: Q \times \Sigma \times 2^{\mathrm{X}} \to Q$ и выходов $\lambda: Q \times \Sigma \times 2^{\mathrm{X}} \to \mathrm{Z}^*$ устроены так, что каждый переход помечен тройкой $\langle e, f, z \rangle$, встречаемой в сценариях работы.

4. Задача генерации управляющих автоматов по сценариям работы с ошибками в выходных воздействиях. Расширим последнюю задачу для учета возможных ошибок в выходных воздействиях сценариев. Пусть задан размер искомого автомата C, набор сценариев S, возможное число K ошибочных выходных последовательностей в сценариях. Необходимо найти такой управляющий автомат $\mathcal{A}_{\mathrm{EFSM}}$ размера C, что он удовлетворяет сценариям работы S, выдавая при их обработке не более чем K ошибочных выходных воздействий. Иначе говоря, требуется найти управляющий автомат и ограниченное множество исправлений сценариев такие, что поведение автомата полностью совпадает с исправленными сценариями.

В дальнейшем будем использовать введенную нумерацию для обращения к задачам. Еще раз отметим, что в известных исследованиях приводится точное решение только первой задачи [44]. В настоящей диссертации проводится разработка точных методов решения остальных задач, совершенствуется метод точного решения первой задачи. Актуальность разработки методов генерации управляющих автоматов,

основанных на программных средствах решения SAT и CSP, возникает только в случае теоретической сложности решаемых задач, доказательство которой также проводится в диссертации.

Выводы по главе 1

1. Описаны основные методы решения задачи SAT, гарантирующие нахождение искомого ответа за конечное время. Эти методы основаны на алгоритме *DPLL* – поиске с возвратом, ускоряющем нахождение решения за счет применения правил распространения переменных и исключения «чистых» переменных. В современных программных средствах решения задачи SAT применяется стратегия управляемого конфликтами обучения дизъюнктов *CDCL*. На указанных алгоритмах основаны и современные средства решения задачи CSP.
2. Проанализированы результаты соревнований *SAT Competition* 2014 и *MiniZinc Challenge* 2015. Среди программных средств решения SAT для использования в диссертации были выбраны *lingeling* и *CryptoMiniSat*, а для решения CSP – *Opturion CPX* и *OR-tools*.
3. Проведен анализ работ, посвященных методам генерации конечных автоматов по примерам поведения. Выделены три группы методов генерации автоматов, основанных на: эвристических алгоритмах слияния состояний (state merging); метаэвристических алгоритмах; сведении задач генерации к классическим NP-полным задачам. Первые два подхода являются неточными, а третий – точным. Точные методы (основанные на сведениях к задачам SAT и раскраски графов) известны только для решения задачи генерации ДКА по незашумленным обучающим словарям.
4. Приведена формальная постановка решаемых в дальнейшем задач генерации ДКА и управляющих конечных автоматов по безошибочным и зашумленным примерам поведения.

ГЛАВА 2. ТЕОРЕТИЧЕСКАЯ ОЦЕНКА СЛОЖНОСТИ ПОСТАВЛЕННЫХ ЗАДАЧ ГЕНЕРАЦИИ УПРАВЛЯЮЩИХ АВТОМАТОВ

В настоящей главе проводится доказательство принадлежности задач генерации управляющих автоматов классу NP-трудных. Приводится условие полноты задач в классе NP.

2.1. Доказательство принадлежности поставленной задачи классу NP-трудных

В работе 1978 года Е. М. Голд доказал [41], что в общем случае задача генерации ДКА с заданным числом состояний по обучающим словарям является NP-полной. В работе 1993 года [61] теоретическая сложность была усилена – было доказано, что задача нахождения ДКА с числом состояний, приближенным к заданному, также является полной в NP. В настоящем разделе докажем NP-трудность третьей (а, следовательно, и четвертой) поставленной в первой главе задачи генерации управляющих автоматов. Доказательство проведем с использованием первого указанного результата [41].

Рассматриваемой задаче соответствует язык L_{EFSM} – множество пар $\langle S, C \rangle$ (здесь S – набор сценариев работы программы, C – натуральное число, записанное в унарной системе счисления), для которых существует управляющий автомат $\mathcal{A}_{\mathrm{EFSM}}$ размера C, удовлетворяющий сценариям из S. Кратко изложим используемые для доказательства трудности L_{EFSM} в классе NP понятия теории сложности [11].

Классом $\mathrm{NTIME}(f(n))$ называется класс языков (задач), для которых существует недетерминированная машина Тьюринга такая, что она всегда останавливается, и время ее работы не превосходит $f(n)$, где n – длина входа. NP – класс языков (задач), разрешимых на недетерминированной машине Тьюринга за полиномиальное время: $\mathrm{NP} = \mathrm{NTIME}(poly(n))$.

Язык L_A *сводится по Карпу* к языку L_B, если существует функция $f(x)$ такая, что x принадлежит L_A тогда и только тогда, когда $f(x)$ принадлежит L_B. При этом сводящая функция должна быть вычислима за полиномиальное время от длины входа. Язык L_B называется NP-трудным (трудным в NP), если для любого языка L_A, принадлежащего NP, L_A сводится по Карпу к L_B. Язык L_B называется NP-полным (полным в NP), если он является NP-трудным и принадлежит классу NP.

Докажем принадлежность L_{EFSM} классу NP-трудных задач. Для этого докажем, что к поставленной задаче можно свести задачу из класса NP-трудных. В качестве такой задачи выберем задачу L_{DFA} построения ДКА с заданным числом состояний по заданным обучающим словарям. Языком L_{DFA} данной задачи является множество троек $\langle S_+, S_-, C\rangle$ таких, что существует конечный ДКА размера C, который принимает слова из словаря S_+, не принимает слова из словаря S_-.

Согласно определению сведения по Карпу, язык L_{DFA} сводится к языку L_{EFSM}, если существует такая функция $f(w)$, что w принадлежит L_{DFA} **тогда и только тогда**, когда элемент $f(w)$ принадлежит L_{EFSM}.

Опишем сводящую функцию f, принимающую на вход элемент $w = \langle S_+, S_-, C\rangle$ языка L_{DFA} и возвращающую элемент $f(w) = \langle S, C\rangle$ языка L_{EFSM} рассматриваемой в работе задачи. Составим множество сценариев S следующим образом. Для каждого слова $a_1 \dots a_n$ из словаря S_+ в множество S добавляется сценарий работы

$$\langle a_1; 1; ()\rangle, \dots, \langle a_n; 1; ()\rangle, \langle \$; 1; (\text{accept})\rangle,$$

где как «$» обозначен символ конца строки, accept – выходной символ допускающей пометки. Аналогично, для каждого слова из словаря S_- добавляется сценарий работы, последний элемент которого равен $\langle \$; 1; (\text{reject})\rangle$.

Докажем прямое следствие: если $w = \langle S_+, S_-, C\rangle$ принадлежит L_{DFA}, то $f(w) = \langle S, C\rangle$ принадлежит языку L_{EFSM}. То есть, если существует ДКА

$\mathcal{A}_{\mathrm{DFA}}$ для элемента w, то можно построить управляющий автомат для $f(w)$. Для этого преобразуем автомат $\mathcal{A}_{\mathrm{DFA}}$ следующим образом. Во-первых, каждый переход автомата по символу a преобразуем в переход управляющего автомата, помеченный тройкой $\langle a; 1; () \rangle$ − событием a, тождественным охранным условием и пустой последовательностью выходных воздействий. Во-вторых, для каждого допускающего состояния добавим помеченный тройкой $\langle \$; 1; (\mathrm{accept}) \rangle$ переход, начинающийся и заканчивающийся в этом состоянии. При этом пометка допуска состояния из состояния удаляется. В-третьих, для остальных состояний автомата аналогичным образом добавим переход, помеченный тройкой $\langle \$; 1; (\mathrm{reject}) \rangle$. По построению управляющий автомат размера C удовлетворяет всем сценариям из S.

Докажем обратное: если $f(w)$ принадлежит языку L_{EFSM}, то w принадлежит языку L_{DFA} – если существует управляющий автомат $\mathcal{A}_{\mathrm{EFSM}}$, удовлетворяющий построенным сценариям из множества S и число его состояний равно C, то существует и автомат-распознаватель для элемента w. Преобразуем автомат $\mathcal{A}_{\mathrm{EFSM}}$ в искомый ДКА. Для этого выполним следующие действия:

- все переходы автомата, помеченные тройкой $\langle a; 1; () \rangle$, преобразуем в переходы по символу a;
- переходы, помеченные $\langle \$; 1; (\mathrm{accept}) \rangle$, удалим, при этом каждое состояние, из которого вел такой переход, сделаем допускающим;
- удалим оставшиеся переходы.

Таким образом, полученный ДКА по построению принимает слова из S_+, не принимает слова из S_- и число его состояний равно C.

Следовательно, доказано, что существует функция $f(w)$ такая, что w принадлежит L_{DFA} тогда и только тогда, когда элемент $f(w)$ принадлежит L_{EFSM}. Следовательно, язык L_{DFA} сводится к языку L_{EFSM}, что **доказывает**

принадлежность задачи построения управляющего автомата классу NP-трудных задач.

2.2. Условие принадлежности рассматриваемой задачи классу NP

Как уже было указано, полнота в классе NP состоит из трудности в данном классе и принадлежности ему. Трудность задачи была доказана в предыдущем разделе, сделаем попытку доказать принадлежность задачи построения управляющего автомата по сценариям работы классу NP и покажем условие, при котором доказательство будет корректно. Воспользуемся альтернативным определением NP через сертификаты.

Говорят, что y является *сертификатом* принадлежности элемента x языку L, если существует полиномиальное отношение (верификатор) R, такое, что $R(x, y) = 1$ тогда и только тогда, когда x принадлежит L. Классу NP соответствует класс языков (задач) L таких, что для каждого из них существует полиномиальный верификатор R, а также полином p. При этом слово l принадлежит языку L тогда и только тогда, когда существует сертификат y (длина которого не превосходит заданного полинома p), и сертификат y удовлетворяет верификатору R.

Для исследуемого языка задачи сертификатом y элемента $x = \langle S, C \rangle$ является управляющий автомат размера C, удовлетворяющий всем сценариям из S. Таким образом, верификатором $R(x, y)$ является программа, которая проверяет два утверждения.

Во-первых, программа R проверяет то, что автомат состоит ровно из C состояний. В начале доказательства было оговорено, что число C задано в унарной системе счисления. Если бы это было иначе, то размер любого элемента x с пустым множеством S составлял бы $\mathcal{O}(\log|y|)$. Таким образом, размер сертификата y зависел бы экспоненциально от размера x, что не позволило бы работать верификатору R полиномиальное время.

Во-вторых, для каждого сценария из множества S программа R проверяет, удовлетворяет ли ему автомат y. Следуя определению, введенному ранее, проверка для каждого элемента сценария, удовлетворяет ли элементу автомат, производится за $\mathcal{O}(|y|)$ – в худшем случае будут рассмотрены все переходы автомата. Следовательно, для всех заданных сценариев время работы данного шага составит $\mathcal{O}(|S| \cdot |y|)$, где как $|S|$ обозначена суммарная длина всех сценариев набора S.

Таким образом, было показано существование полиномиального отношения R, что доказывает принадлежность исследуемой задачи классу NP. Однако для того, чтобы выполнить для элемента сценария проверку, удовлетворяет ли данному элементу автомат, необходимо проверить на равенство булевы формулы, которыми задаются охранные условия. В общем случае, если охранные условия заданы строковыми представлениями, такую проверку невозможно выполнить за полиномиальное время [11]. Если же охранные условия заданы *таблицами истинности*, или же охранные условия зависят от *ограниченного константой* числа переменных, то проверку на равенство можно выполнить за полиномиальное время, что обеспечивает принадлежность поставленной задачи классу NP.

Выводы по главе 2

1. Доказана трудность задач генерации управляющих автоматов в классе NP. Трудность доказана сведением задачи построения ДКА по обучающим словарям, полнота которой в NP доказана в [41]. Этот результат, полученный диссертантом в данной главе, опубликован в работе [94].
2. Показано условие полноты в NP задачи генерации управляющих автоматов – необходимо, чтобы содержащиеся в сценариях работы охранные условия были или заданы таблицами истинности, или имели ограниченную константой длину.

ГЛАВА 3. ГЕНЕРАЦИЯ ДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ ПО ОБУЧАЮЩИМ СЛОВАРЯМ

В настоящей главе описываются разработка, реализация и экспериментальное исследование точных методов генерации детерминированных конечных автоматов по обучающим словарям. В разделе 3.1 проводится усовершенствование предложенного ранее в [44] метода построения ДКА по обучающим словарям с использованием программных средств решения задачи выполнимости булевых формул. В разделе 3.2 впервые решается задача точной генерации ДКА по зашумленным обучающим словарям – известны только неточные методы генерации. В разделе 3.3 описываются реализация и проведенные экспериментальные исследования разработанных методов.

3.1. Метод генерации по безошибочным обучающим словарям

В настоящем разделе проводится разработка метода решения первой поставленной задачи – построения ДКА по незашумленным, или безошибочным, обучающим словарям S_+ и S_-. Описываются структура метода и предлагаемые предикаты нарушения симметрии.

3.1.1. Структура метода

Предлагаемый метод является модификацией предложенного в [44] и подробно описанного в обзоре метода *DFASAT*, основанного на сведении к SAT. В листинге 3 приведен псевдокод модифицированного метода.

Листинг 3 – Псевдокод метода построения ДКА по безошибочным обучающим словарям

```
function NoiselessDFAGeneration(S+, S−)
    S+, S− – непересекающиеся наборы слов
    T ← APTA(S+, S−)
    G ← consistencyGraph(T)
    clique ← largeClique(G)
```

```
  for C ← |clique|..|𝒯| do
    f_SAT ← ℱ_SAT(𝒯, 𝒢, C)
    solution ← solveSAT(f_SAT)
    if solution ≠ {} then
      𝒜 ← DFA(C, 𝒯, solution)
      return 𝒜
    end if
  end for
end function
```

Опишем указанные в листинге функции:

- $\mathrm{APTA}(S_+, S_-)$ возвращает префиксное дерево для заданных словарей S_+ и S_-;
- $\mathrm{consistencyGraph}(\mathcal{T})$ возвращает граф совместимости дерева $\mathcal{T}$;
- $\mathrm{largeClique}(\mathcal{G})$ возвращает клику графа совместимости $\mathcal{G}$, полученную жадным алгоритмом;
- $\mathrm{solveSAT}(f_{\mathrm{SAT}})$ возвращает выполняющую подстановку для формулы f_{SAT}, полученную сторонней программой, или пустое множество, если выполняющая подстановка не найдена;
- $\mathrm{DFA}(C, \mathcal{T}, \mathrm{solution})$ возвращает восстановленный по подстановке solution минимальный ДКА, являющийся ответом на задачу.

Сведение записывается в виде формулы $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}$, которая состоит из обязательной и дополнительной частей: $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}} = \boldsymbol{\mathcal{F}}_1 \wedge \boldsymbol{\mathcal{F}}_2$ [44]. Выразим в виде единой формулы обязательную часть

$$\boldsymbol{\mathcal{F}}_1 = \bigwedge_{v \in V} \mathrm{ALO}\left(\{x_{v,i}\}_{i=1}^{C}\right) \wedge \bigwedge_{v \in V_+, 1 \le i \le C} (x_{v,i} \Rightarrow z_i) \wedge \bigwedge_{v \in V_-, 1 \le i \le C} (x_{v,i} \Rightarrow \neg z_i)$$
$$\wedge \bigwedge_{v \in V, 1 \le i,j \le C} \left(x_{p(v),i} \wedge x_{v,j} \Rightarrow y_{l(v),i,j}\right)$$
$$\wedge \bigwedge_{a \in \Sigma, 1 \le i \le C} \mathrm{AMO}\left(\{y_{a,i,j}\}_{j=1}^{C}\right),$$

и дополнительную часть

$$\boldsymbol{\mathcal{F}}_2 = \bigwedge_{v \in V} \mathrm{AMO}\left(\{x_{v,i}\}_{i=1}^{C}\right) \wedge \bigwedge_{a \in \Sigma, 1 \le i \le C} \mathrm{ALO}\left(\{y_{a,i,j}\}_{j=1}^{C}\right)$$

$$\wedge \bigwedge_{v \in V, 1 \le i,j \le C} \left(x_{p(v),i} \wedge y_{l(v),i,j} \Rightarrow x_{v,j}\right) \wedge \bigwedge_{(v,w) \in G, 1 \le i \le C} \left(x_{v,i} \Rightarrow \neg x_{w,i}\right).$$

Выражение $\mathrm{ALO}(X)$ обозначает ограничение вида *at-least-one*: в заданном наборе переменных X хотя бы одна переменная должна выполняться. Выражение $\mathrm{AMO}(X)$ (*at-most-one*) соответствует требованию истинности не более чем одной переменной из X. Если $\mathrm{ALO}(X)$ задается одной дизъюнкцией $\bigvee_{x \in X} x$ входящих в X переменных, то для задания $\mathrm{AMO}(X)$ существует несколько подходов [46]. В настоящей диссертации используется *попарное кодирование*, не требующее введения дополнительных переменных.

В работе [44] формула упрощается на основании найденной жадным алгоритмом клики clique графа совместимости G. Для данной клики выполняется, что все входящие в нее вершины префиксного дерева соответствуют разным состояниям ДКА (то есть «покрашены» в разные цвета). Чтобы сократить пространство поиска и не рассматривать в худшем случае |clique|! способов раскраски вершин клики в случае невыполнимости формулы, авторы фиксируют цвета вершин значениями от 1 до |clique|.

В настоящем разделе рассматривается иной способ *нарушения симметрии* – предлагаются предикаты $\boldsymbol{\mathcal{F}}_{\mathrm{SB}}$, задающие ограничения на вид искомого автомата таким образом, что среди всех изоморфных автоматов, удовлетворяющих $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}$, лишь один будет удовлетворять $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}} \wedge \boldsymbol{\mathcal{F}}_{\mathrm{SB}}$.

3.1.2. Предикаты нарушения симметрии

Предикаты нарушения симметрии используются для сокращения пространства поиска поставленной задачи [22]. Данные предикаты добавляются к формуле сведения и не должны влиять на ее выполнимость: если формула выполнима, то после добавления предикатов останется

таковой. После выявления «симметрии» между решениями поставленной задачи разрабатывают нарушающие ее предикаты: минимум симметричных решений (а в идеале ровно одно) должно им соответствовать.

В случае задач генерации автоматов с нумерованными состояниями естественной симметрией является *изоморфизм* автоматов. Автоматы $\mathcal{A}_1$ и $\mathcal{A}_2$ называются изоморфными, если число их состояний совпадает и можно перенумеровать состояния $\mathcal{A}_1$ таким образом, что $\mathcal{A}_1$ будет эквивалентен $\mathcal{A}_2$. Изоморфизм разбивает множество автоматов на классы эквивалентности, и если было доказано, что один из представителей класса не является решением задачи генерации по обучающим словарям S_+ и S_-, то не имеет смысла обрабатывать и остальные автоматы из класса. Подчеркнем, что в случае сведения задачи к SAT, указанную обработку производит программное средство решения SAT.

Предложим способ фиксации номеров состояний автомата, позволяющий избежать рассмотрения изоморфных автоматов во время решения задачи SAT. **Основной идеей** предлагаемого подхода к нарушению симметрии является фиксирование номеров состояний в порядке некоторого однозначного обхода автомата. При такой фиксации в булевой формуле изоморфные решения не должны рассматриваться: среди таких автоматов, соответствующих выполняющим подстановкам $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}$, лишь для одного найдется соответствующая выполняющая подстановка формулы $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}} \wedge \boldsymbol{\mathcal{F}}_{\mathrm{SB}}$.

Одним из самых распространенных однозначных обходов автомата является *обход в ширину* (breadth-first search, BFS). В работе [52] порядок обхода в ширину называется «естественным» порядком и применяется при генерации ДКА методом слияния состояний (функция «`NatOrder`»). Похожие идеи были использованы в [43] для сокращения числа рассматриваемых автоматов при генерации ДКА с использованием

генетических алгоритмов (функция «Move to Front»). В настоящей диссертации нумерацию автомата в порядке обхода в ширину будем называть *BFS-нумерацией*.

Порядок нумерации в ширину ДКА $\mathcal{A} = \langle \{q_1 \dots q_C\}, \Sigma, \delta, q_s, F \rangle$ формируется с использованием структуры данных *очередь*. В начале очередь содержит единственное состояние – начальное состояние q_s ДКА. Затем на каждом шаге из начала очереди извлекается состояние q_i, и в ее конец добавляются все состояния q_j, которые до этого в очередь не добавлялись и в которые есть переход $q_i \xrightarrow[l]{} q_j$. При этом переходы из q_i рассматриваются в лексикографическом порядке символов $l \in \Sigma$, а состояние q_i будем в дальнейшем называть *предком* состояния q_j при обходе в ширину. Автомат, состояния которого добавляются в очередь по порядку номеров (от q_1 до q_C), назовем *BFS-пронумерованным*. На рисунке 9 приведены примеры изоморфных автоматов: BFS-пронумерованного (рисунок 9*а*) и не следующего указанной нумерации (рисунок 9*б*). Размер автоматов равен трем, автоматы удовлетворяют префиксному дереву, приведенному на рисунке 6.

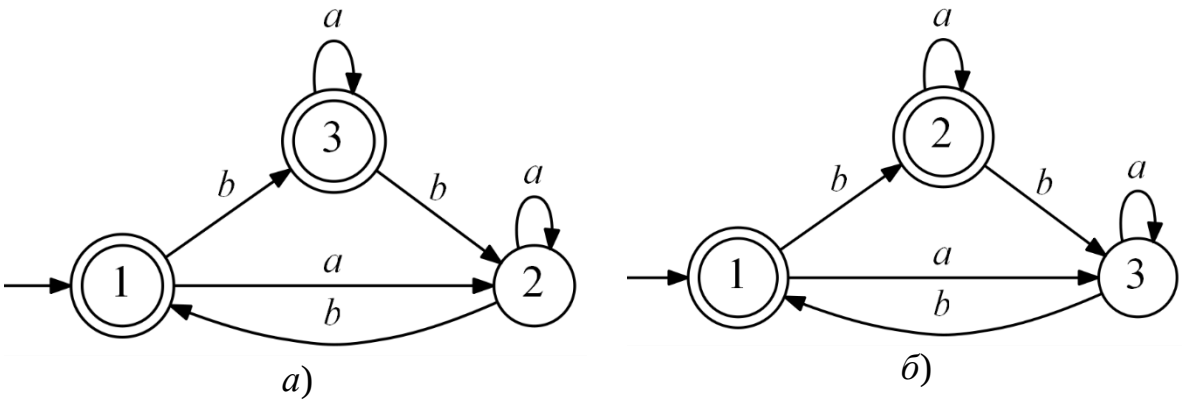

Рисунок 9 – Примеры изоморфных ДКА: *а* – BFS-пронумерованный; *б* – не BFS-пронумерованный

Построим такие предикаты нарушения симметрии $\boldsymbol{\mathcal{F}}_{\mathrm{SB}}$, что им будут удовлетворять только BFS-пронумерованные автоматы. Напомним, что в $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}$ функция переходов автомата, на вид которой и накладывается

ограничение, задается переменными $y_{a,i,j}$ Для построения набора предикатов нарушения симметрии будем дополнительно использовать следующие переменные.

1. **Переменные наличия перехода.** Переменная $t_{i,j}$ верна тогда и только тогда, когда в автомате существует хотя бы один переход из состояния i в состояние j. Переменные определены для каждой пары состояний ДКА: $i, j \in 1..C$.
2. **Переменные предка в обходе.** Переменные $p_{j,i}$ определены для $j \in 2..C,\ i \in 1..j$ и хранят *предка* вершины в дереве обхода искомого автомата в ширину (при его обходе из первого состояния). $p_{j,i} = 1$ тогда и только тогда, когда при обходе ДКА в ширину вершина j была добавлена в очередь при просмотре исходящих ребер из i.
3. **Переменные наименьшего символа.** Будем хранить наименьший символ между каждой парой состояний в булевых переменных $m_{l,i,j}$ для $i, j \in 1..C$, $l \in \Sigma$. Рассмотрим все переходы между состояниями i и j генерируемого автомата. $m_{l,i,j} = 1$ тогда и только тогда, когда l является лексикографически наименьшим символом, который встретился на данных переходах $i \underset{l}{\to} j$.

С использованием данных переменных, а также переменных $y_{a,i,j}$, предложим следующие предикаты BFS-нумерации.

1. Зададим переменные наличия перехода $t_{i,j}$ по их определению:

$$\boldsymbol{\mathcal{F}}_t = \bigwedge_{1 \le i < j \le C} \left(t_{i,j} \Leftrightarrow \bigvee_{l \in \Sigma} y_{l,i,j} \right).$$

2. Зададим переменные предка в обходе $p_{j,i}$ через переменные $t_{i,j}$. Предком состояния j при BFS-нумерации является состояние с наименьшим номером i среди таких, что существует переход $i \to j$:

$$\boldsymbol{\mathcal{F}}_p = \bigwedge_{1 \le i < j \le C} \left(p_{j,i} \Leftrightarrow t_{i,j} \wedge \bigwedge_{1 \le k < i} \neg t_{k,j} \right).$$

3. При этом, так как каждое состояние автомата достижимо, у каждого состояния (кроме начального, то есть первого), должен быть предок с меньшим номером при BFS-нумерации:

$$\boldsymbol{\mathcal{F}}_{\mathrm{ALO}(p)} = \bigwedge_{1<j\leq C} \mathrm{ALO}\left(\{p_{j,i}\}_{i=1}^{j-1}\right).$$

4. Зададим первое ограничение, соответствующее BFS-нумерации, с использованием определенных переменных предка $p_{j,i}$. Рассмотрим пару состояний ДКА q_j и q_{j+1} и их предков при нумерации q_i и q_k: выполняются переменные $p_{j,i}$ и $p_{j+1,k}$ (k может совпадать с i). При BFS-нумерации состояние q_j должно быть добавлено в очередь раньше q_{j+1}, следовательно, q_i извлекается из очереди не позже q_k и k не может быть меньше i:

$$\boldsymbol{\mathcal{F}}_{\mathrm{BFS}(p)} = \bigwedge_{1\leq k<i<j<C} \left(p_{j,i} \Rightarrow \neg p_{j+1,k}\right).$$

5. Для последующего задания порядка нумерации состояний с одинаковым предком определим переменные наименьшего символа $m_{l,i,j}$:

$$\boldsymbol{\mathcal{F}}_{m} = \bigwedge_{1\leq i<j\leq C, l\in\Sigma} \left(m_{l,i,j} \Leftrightarrow y_{l,i,j} \wedge \bigwedge_{l^*\in\Sigma, l^*<l} \neg y_{l^*,i,j}\right).$$

6. С использованием переменных наименьшего символа $m_{l,i,j}$ зададим второе ограничение, соответствующее BFS-нумерации. Рассмотрим такие состояния q_j и q_{j+1}, что они добавлены в очередь при рассмотрении переходов $q_i \xrightarrow[l]{} q_j$ и $q_i \xrightarrow[l^*]{} q_{j+1}$ из одного состояния q_i (иначе говоря, выполняются $p_{j,i}$, $p_{j+1,i}$, $m_{l,i,j}$ и $m_{l^*,i,j+1}$). При BFS-нумерации состояние q_j должно быть добавлено в очередь раньше q_{j+1}, поэтому символ l должен быть лексикографически меньше l^*:

$$\boldsymbol{\mathcal{F}}_{\mathrm{BFS}(m)} = \bigwedge_{1\leq i<j<C} \bigwedge_{l,l^*\in\Sigma, l^*<l} p_{j,i} \wedge p_{j+1,i} \wedge m_{l,i,j} \Rightarrow \neg m_{l^*,i,j+1}.$$

Указанные предикаты шести типов соответствуют BFS-нумерации ДКА и составляют предикаты нарушения симметрии:

$$\boldsymbol{\mathcal{F}}_{\mathrm{SB}} = \boldsymbol{\mathcal{F}}_t \wedge \boldsymbol{\mathcal{F}}_p \wedge \boldsymbol{\mathcal{F}}_{\mathrm{ALO}(p)} \wedge \boldsymbol{\mathcal{F}}_{\mathrm{BFS}(p)} \wedge \boldsymbol{\mathcal{F}}_m \wedge \boldsymbol{\mathcal{F}}_{\mathrm{BFS}(m)}.$$

Упростим предикаты для случая алфавита, состоящего из двух символов: $\Sigma = \{a, b\}$. Такой алфавит рассматривался, например, на соревновании *Abbadingo One* [54]. В данном случае нам не требуется вводить переменные $m_{l,i,j}$: если у двух состояний q_j и q_{j+1} совпадает предок при обходе, то меньшему из номеров соответствует меньший символ a и, автоматически, большему соответствует b:

$$\boldsymbol{\mathcal{F}}^*_{\mathrm{BFS}(m)} = \bigwedge_{1 \le i < j < C} p_{j,i} \wedge p_{j+1,i} \Rightarrow y_{a,i,j}.$$

Окончательный вид предикатов нарушения симметрии в случае $|\Sigma| = 2$ выражается дизъюнкцией пяти составляющих:

$$\boldsymbol{\mathcal{F}}^*_{\mathrm{SB}} = \boldsymbol{\mathcal{F}}_t \wedge \boldsymbol{\mathcal{F}}_p \wedge \boldsymbol{\mathcal{F}}_{\mathrm{ALO}(p)} \wedge \boldsymbol{\mathcal{F}}_{\mathrm{BFS}(p)} \wedge \boldsymbol{\mathcal{F}}^*_{\mathrm{BFS}(m)}.$$

Приведем асимптотические оценки на размер разработанных предикатов нарушения симметрии. Выражение $\boldsymbol{\mathcal{F}}_{\mathrm{SB}}$ состоит из $\mathcal{O}(C^2|\Sigma|^2 + C^3)$ дизъюнктов и использует $\mathcal{O}(C^2|\Sigma|)$ дополнительных переменных. Выражение $\boldsymbol{\mathcal{F}}^*_{\mathrm{SB}}$ для случая двухсимвольного алфавита состоит из $\mathcal{O}(C^3)$ дизъюнктов и использует $\mathcal{O}(C^2)$ дополнительных переменных.

3.2. Метод генерации по зашумленным обучающим словарям

В настоящем разделе проводится разработка точного метода решения второй поставленной задачи – генерации детерминированных конечных автоматов по зашумленным (не более K ошибочных пометок) обучающим словарям S_+ и S_-. Данная задача обобщает первую поставленную задачу и также полна в классе NP.

Задача предлагалась участникам соревнования «Learning DFA from Noisy Samples» конференции GECCO 2004 (при $K = \lfloor(|S_+| + |S_-|) \cdot 0.1\rfloor$) [81], где лучший результат показало решение Дж. Гомеса [42], основанное

на последовательном обучении и эволюционных алгоритмах. На настоящий момент лучшим решением задачи является метод С. Лукаса и Т. Рейнольдса, основанный на эволюционном алгоритме с «умной» расстановкой пометок [55]. Предложим метод генерации ДКА, отличающийся от существующих точностью решения – им гарантируется нахождение искомого ответа за конечное время или сертификата его отсутствия.

3.2.1. Структура метода

Предлагаемый метод является модификацией предложенного ранее (раздел 3.1) метода генерации ДКА, применяющего стороннее программное средство решения задачи выполнимости. В листинге 4 приведен псевдокод предлагаемого метода.

Листинг 4 – Псевдокод метода построения ДКА по зашумленным обучающим словарям

```
function NoisyDFAGeneration(S+, S−, K)
  S+, S− – непересекающиеся наборы слов
  K – допустимое число ошибочных меток слов
  𝒯 ← APTA(S+, S−)
  for C ← 1..|𝒯| do
    f_SAT ← 𝓕_SAT(𝒯, C, K)
    solution ← solveSAT(f_SAT)
    if solution ≠ {} then
      𝒜 ← DFA(C, 𝒯, solution)
      return 𝒜
    end if
  end for
end function
```

Так как метки в вершинах префиксного дерева могут быть ошибочными, граф совместимости $\mathcal{G}$, рассматриваемый ранее, построить невозможно. Вследствие этого невозможно использовать и нарушение симметрии, основанное на клике графа совместимости [44], в то время как

основанные на BFS-нумерации предикаты нарушения симметрии (раздел 3.1.2) не чувствительны к наличию ошибок во входных данных. Описываемое далее сведение $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}(\mathcal{T}, C, K)$ основано на сведении $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}(\mathcal{T}, \mathcal{G}, C)$, описанном в разделе 3.1.

3.2.2. Предикаты обработки ошибочных пометок

Общая идея модификации сведения заключается в следующем. Для каждой вершины префиксного дерева $\mathcal{T}$ с допускающей или недопускающей меткой будем хранить булеву переменную, истинность которой допускает ошибочность метки. По условию задачи ошибок может быть допущено не более K, и предикаты обработки ошибочных пометок ограничивают сверху совокупное число истинных значений данных переменных. Механизм ограничения основан на дополнительном «массиве» длины K, в котором хранятся положения ошибочных меток в дереве.

Сведение $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}(\mathcal{T}, C, K)$ включает в себя модифицированное сведение $\boldsymbol{\mathcal{F}}_{\mathrm{SAT}}(\mathcal{T}, \mathcal{G}, C)$ и дополняется новыми предикатами обработки ошибочных пометок $\boldsymbol{\mathcal{F}}_{\mathrm{NOISY}}(\mathcal{T}, K)$. Будем использовать следующие дополнительные булевы переменные.

1. **Переменные возможных ошибок.** Переменные f_v определены для каждой помеченной вершины $v \in V_{\pm} = V_{+} \cup V_{-} = \{v_j\}_{j=1}^{W}$. Истинность переменной f_v означает, что метка вершины v может быть ошибочна. Иначе говоря, если $f_v = 0$, то заданная метка допуска/недопуска безошибочна, а если $f_v = 1$, то данное утверждение подлежит сомнению.
2. **Переменные положения ошибок.** Для ограничения совокупного числа истинных значений f_v введем переменные $r_{k,v}$ для всех $1 \leq k \leq K$, $v \in V_{\pm}$. Пронумеруем возможные ошибочные метки от 1 до

K. Переменная $r_{k,v} = 1$ тогда и только тогда, когда ошибочная метка с номером k содержится в вершине дерева v.

3. **Переменные порядка номеров ошибок.** Для упорядочивания переменных $r_{k,v}$ и задания ограничений *at-most-one* введем переменные $o_{k,v}$ для всех $1 \le k \le K$, $v \in V_{\pm}$. Смысл переменных $o_{k,v}$ будет раскрыт позже.

Сначала модифицируем приведенное в разделе 3.1 сведение для учета зашумленных меток слов. Изменим часть $\boldsymbol{\mathcal{F}}_1$ следующим образом. Конъюнкции

$$\bigwedge_{v \in V_+, 1 \le i \le C} (x_{v,i} \Rightarrow z_i) \wedge \bigwedge_{v \in V_-, 1 \le i \le C} (x_{v,i} \Rightarrow \neg z_i)$$

дополним информацией о безошибочности меток вершин v:

$$\bigwedge_{v \in V_+, 1 \le i \le C} \neg f_v \Rightarrow (x_{v,i} \Rightarrow z_i) \wedge \bigwedge_{v \in V_-, 1 \le i \le C} \neg f_v \Rightarrow (x_{v,i} \Rightarrow \neg z_i).$$

Так как граф совместимости $\mathcal{G}$, вычисленный по зашумленному дереву сценариев, некорректен, исключим из части $\boldsymbol{\mathcal{F}}_2$ конъюнкцию

$$\bigwedge_{(v,w) \in \mathcal{G}, 1 \le i \le C} (x_{v,i} \Rightarrow \neg x_{w,i}).$$

Теперь осталось ограничить совокупное число истинных переменных f_v. Для этого составим набор ограничений $\boldsymbol{\mathcal{F}}_{\text{NOISY}}(\mathcal{T}, K)$, состоящий из следующих частей.

1. Метка вершины дерева v может быть ошибочной в том и только в том случае, когда на нее «указывает» одна из переменных положения:

$$\boldsymbol{\mathcal{F}}_{fr} = \bigwedge_{v \in V_{\pm}} \left(f_v \Leftrightarrow \bigvee_{1 \le k \le K} r_{k,v} \right).$$

2. Наложим ограничения на вид переменных $o_{k,v}$. Во-первых, значения переменных $o_{k,v_1} \dots o_{k,v_W}$ должны иметь вид $\{11..100..0\}$. Во-вторых,

истинных значений переменных $o_{k+1,v_1} \dots o_{k+1,v_W}$ для $k+1$ должно быть больше, чем в $o_{k,v_1} \dots o_{k,v_W}$ (последняя единица должна стоять правее):

$$\boldsymbol{\mathcal{F}}_o = \bigwedge_{1 \le k \le K, 1 \le j < W} \left(o_{k,v_{j+1}} \Rightarrow o_{k,v_j} \right) \wedge \bigwedge_{1 \le k < K, 1 \le j < W} \left(o_{k,v_j} \Rightarrow o_{k+1,v_{j+1}} \right).$$

3. Именно положение границы между последней выполненной и первой невыполненной переменной $o_{k,v_1} \dots o_{k,v_W}$ будет однозначно определять единственную выполненную переменную среди $r_{k,v_1} \dots r_{k,v_W}$. Также отдельно рассмотрим крайний случай при $k = K, j = W$:

$$\boldsymbol{\mathcal{F}}_{\text{AMO}(r)} = \left(r_{K,v_W} \Leftrightarrow o_{K,v_W} \right) \wedge \bigwedge_{1 \le k \le K, 1 \le j < W} \left(r_{k,v_j} \Leftrightarrow o_{k,v_j} \wedge \neg o_{k,v_{j+1}} \right).$$

Указанные ограничения трех типов составляют набор из $\mathcal{O}(C\,|V_{\pm}| + K\,|V_{\pm}|)$ дополнительных предикатов обработки ошибочных пометок слов, соответствующих тому, что среди $f_{v_1} \dots f_{v_W}$ выполняется ровно K переменных:

$$\boldsymbol{\mathcal{F}}_{\text{NOISY}} = \boldsymbol{\mathcal{F}}_{fr} \wedge \boldsymbol{\mathcal{F}}_o \wedge \boldsymbol{\mathcal{F}}_{\text{AMO}(r)}.$$

На рисунке 10 проиллюстрирована схема взаимодействия переменных f_v, $r_{k,v}$, $o_{k,v}$ на примере выполняющего $\boldsymbol{\mathcal{F}}_{\text{NOISY}}$ набора значений переменных. Префиксное дерево построено для зашумленных обучающих словарей $S_+ = \{ab, ba, baba, bbb\}$ и $S_- = \{abbb, b\}$, содержащих две ошибочных метки: дерево отличается от приведенного на рисунке 6 в метках вершин 6 и 9. Таким образом, для «исправления» указанных ошибок необходимо значение $K = 2$ и выполненные переменные $f_6 = f_9 = 1$, чему соответствуют приведенные на рисунке 10 значения $r_{k,v}$ и $o_{k,v}$.

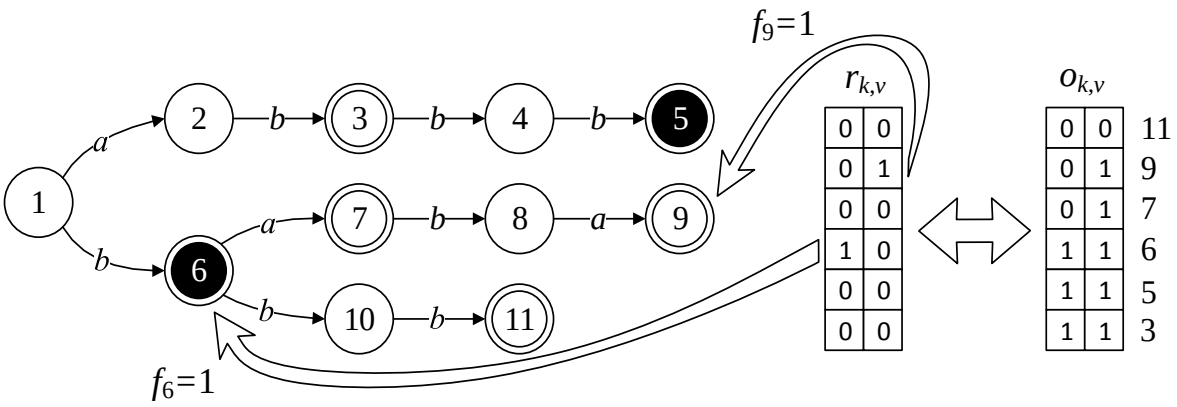

Рисунок 10 – Пример схемы взаимодействия переменных f_v, $r_{k,v}$ и $o_{k,v}$

3.3. Реализация и экспериментальные исследования разработанных методов генерации

В настоящем разделе описываются проведенные реализация и экспериментальные исследования разработанных методов генерации ДКА. Целью исследований является тестирование корректности работы разработанных инструментальных средств, экспериментальное сравнение разработанных и реализованных методов генерации ДКА с существующими, а также выявление статистически значимых положений. Отметим, что аналитическое доказательство корректности разработанных методов в настоящей диссертации (как и в иных работах, посвященных сведению задач к SAT) не проводится. В ходе экспериментов после генерации автоматы будут отдельно проверяться на соответствие обучающим словарям.

3.3.1. Реализация разработанных методов генерации ДКА

Реализация разработанных методов проведена на языке *Java*. Создано инструментальное средство *DFAInducer* с открытым программным кодом [72]. Инструментальное средство поставляется одним кроссплатформенным пакетом «DFAInducer.jar». Структура инструментального средства приведена на рисунке 11.

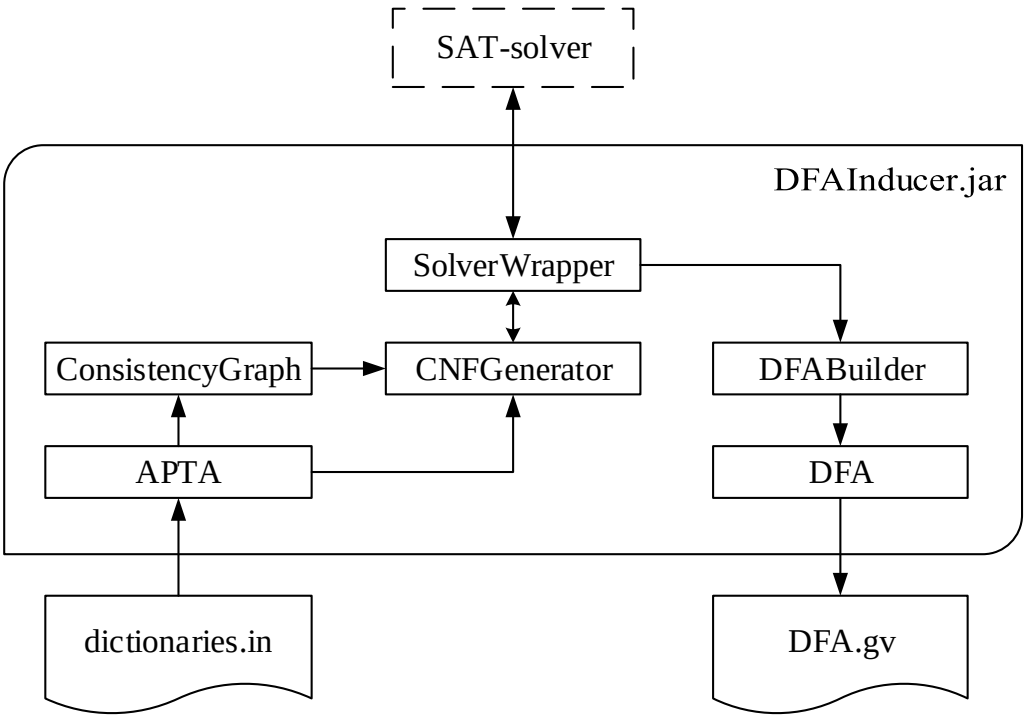

Рисунок 11 – Структура инструментального средства *DFAInducer*

Входными данными для средства являются обучающие словари (файл «`dictionaries.in`») и опциональный набор настроек. В случае допущения ошибок в словарях необходимо через параметр передать число *K* допустимых ошибок в пометках слов. Файл должен соответствовать принятому на соревнованиях по генерации ДКА формату *DIMACS* [73]. В настройках пользователю предоставляется возможность указать путь к стороннему программному средству (блок «SAT-solver»).

В блоке «`DFAInducer.jar`» изображены основные классы реализации и связи между ними, соответствующие передаче данных между классами. Приведены следующие классы:

- `APTA`: структура префиксного дерева и алгоритм его построения;
- `ConsistencyGraph`: структура графа совместимости и алгоритм его построения по экземпляру APTA, а также построение клики в графе;

- `CNFGenerator`: генерация КНФ-формулы по экземплярам классов `APTA` и `ConsistencyGraph`, а также возрастающему числу C состояний генерируемого автомата;
- `SolverWrapper`: «обертка» над сторонним программным средством решения SAT; принимает булеву формулу и возвращает ответ от средства SAT-solver;
- `DFABuilder`: генерация автомата по найденной выполняющей подстановке;
- `DFA`: структура детерминированного конечного автомата и его вывод в формате *GraphViz*.

На выход *DFAInducer* возвращает файл «`DFA.gv`», содержащий текстовое описание ДКА с наименьшим числом состояний и удовлетворяющий заданным словарям. ДКА выводится в распространенном формате *GraphViz* [78]. В листинге 5 приведен пример наименьшего сгенерированного ДКА в формате *GraphViz* для обучающих словарей $S_+ = \{01, 1, 10, 111\}$ и $S_- = \{0111, 1010\}$.

Листинг 5 – Пример выходного файла DFA.gv – текстового описания ДКА в формате *GraphViz*

```
digraph DFA {
    node [shape = circle];
    1 [style = "bold"];
    1 [peripheries = 2]
    1 -> 2 [label = "0"];
    1 -> 2 [label = "1"];
    2 -> 1 [label = "0"];
    2 -> 3 [label = "1"];
    3 [peripheries = 2]
    3 -> 3 [label = "0"];
    3 -> 1 [label = "1"];
}
```

3.3.2. Экспериментальные исследования метода генерации ДКА по безошибочным обучающим словарям

Было проведено статистическое исследование реализованного метода генерации ДКА по безошибочным обучающим словарям. Целью экспериментов являлось сравнение его производительности с методом *DFASAT* [44], являющимся наиболее производительным подходом для решения задачи среди известных.

Эксперименты проведены по схеме, приведенной на рисунке 12. Первым шагом каждого эксперимента являлась генерация случайного ДКА по заданным числу состояний C и алфавиту Σ. Вторым шагом являлась генерация обучающих словарей S_+ и S_- по сгенерированному автомату при помощи случайного обхода. Третьим шагом являлся запуск рассматриваемых методов генерации ДКА по обучающим словарям. Наконец, сгенерированные автоматы были проверены на соответствие заданным обучающим словарям S_+ и S_-.

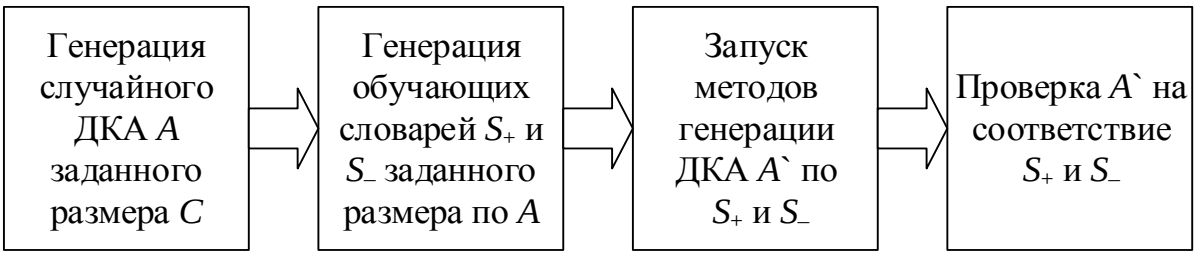

Рисунок 12 – Схема проведения экспериментального исследования

Заметим, что при этом генерируемый автомат A` может быть не изоморфен исходному автомату A, и, более того, наименьший автомат, соответствующий S_+ и S_-, может содержать менее, чем C состояний. В экспериментах такие случаи не рассматривались: если сгенерированный одним из методов автомат содержал менее, чем C состояний, эксперимент начинался сначала с нового ДКА A.

По указанной схеме было проведено 1000 запусков экспериментов для каждого из чисел состояний $C \in [10, 25]$ – каждый из методов

генерации ДКА был запущен 16000 раз. На втором этапе эксперимента словари S_+ и S_- составлялись из $50 \cdot C$ слов. Запуски производились на серверном компьютере с процессором *AMD Opteron* 6378 (тактовая частота 2,4 ГГц), операционной системой *Ubuntu* 14.04. Суммарное процессорное время, использованное для параллельного проведения запусков, эквивалентно 139,4 дням. Напомним, что для решения задачи SAT использовалось одно из лучших на настоящий момент программных средств *lingeling* (в операционной системе *Windows* рекомендуется использовать средство *CryptoMiniSat* [85]).

Каждый запуск использовал одно ядро процессора, вопрос поддержки параллелизма в настоящей диссертации не рассматривается (отметим, однако, что для ускорения работы можно предоставить несколько ядер программному средству решения SAT). Время работы метода построения ДКА на каждом запуске было ограничено одним часом (3600 с), при этом более 95 % времени работы методов занимала работа средства *lingeling*.

В таблице 1 приведены результаты проведенных экспериментов. Столбцы таблицы соответствуют следующим величинам:

- число C сосотояний генерируемого автомата A;
- медианное число $|\mathcal{T}|$ вершин префиксного дерева, построенного по S_+ и S_-;
- медианный размер $|\mathrm{clique}|$ найденной в методах клики;
- медианное время T_1 работы предложенного метода генерации ДКА;
- медианное время T_2 работы метода *DFASAT*;
- P-значение (P-value), вычисленное для времен работ методов при помощи Т-критерия Уилкоксона. Полученные P-значения нормализованы методом Бонферрони-Холма.

Таблица 1. Результаты экспериментального сравнения методов генерации ДКА по незашумленным обучающим словарям

C	$\lvert\mathcal{T}\rvert$	|clique|	T_1, с	T_2, с	P-значение
10	1351	5	71,4	100,7	$2{,}4 \cdot 10^{-53}$
11	1495	5	96,4	166,3	$1{,}1 \cdot 10^{-56}$
12	1644	5	136,8	751,2	$3{,}4 \cdot 10^{-57}$
13	1789	6	174,4	3600*	$9{,}2 \cdot 10^{-58}$
14	1928	6	219,2	3600*	$3{,}1 \cdot 10^{-58}$
15	2067	6	273,3	3600*	$2{,}5 \cdot 10^{-58}$
16	2200	6	333,4	3600*	$2{,}5 \cdot 10^{-58}$
17	2329	6	410,1	3600*	$2{,}4 \cdot 10^{-58}$
18	2455	6	500,4	3600*	$2{,}4 \cdot 10^{-58}$
19	2581	7	598,2	3600*	$2{,}4 \cdot 10^{-58}$
20	2703	7	708,8	3600*	$2{,}3 \cdot 10^{-58}$
21	2841	7	858,3	3600*	$2{,}3 \cdot 10^{-58}$
22	2992	7	1029,2	3600*	$2{,}3 \cdot 10^{-58}$
23	3142	7	1295,2	3600*	$2{,}2 \cdot 10^{-58}$
24	3291	7	1517,6	3600*	$2{,}2 \cdot 10^{-58}$
25	3435	7	1763,3	3600*	$2{,}4 \cdot 10^{-58}$

Значения для T_2, отмеченные звездочкой, соответствуют тому, что медианное время работы для $C > 12$ превысило 3600 секунд – для более, чем половины экспериментов выполнение метода *DFASAT* было остановлено. Полученные значения позволяют сделать вывод о существенном выигрыше в скорости работы предложенного метода точной генерации ДКА по сравнению с *DFASAT*. Данный вывод обладает статистической достоверностью: полученные P-значения не превосходят 0.05. Соответствие сгенерированных автоматов обучающим словарям было дополнительно подтверждено валидатором.

На рисунке 13 приведены ящичные диаграммы для времен работы предложенного в диссертации метода. Напомним, что диаграмма для каждого $C \in [10, 20]$ построена по 100 значениям времен работы.

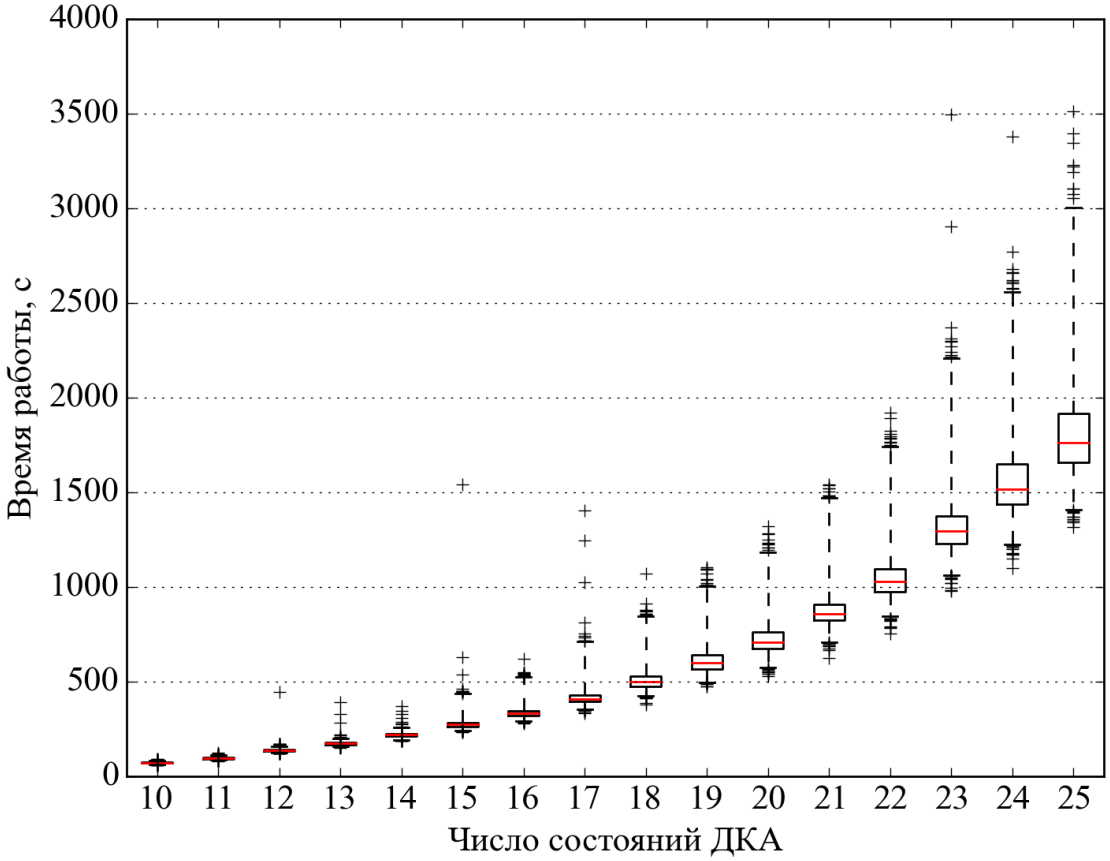

Рисунок 13 – Ящичные диаграммы для времен работы предложенного метода генерации ДКА

Приведенные диаграммы показывают, что предложенный метод решает каждую из задач генерации при $C \leq 14$ менее чем за 10 минут работы персонального компьютера (с тактовой частотой процессора не менее 2,4 ГГц). Метод применим при $C \leq 25$ и доступном часе процессорного времени. Быстрая генерация автоматов актуальна при частом обновлении обучающих словарей.

3.3.3. Экспериментальные исследования метода генерации ДКА по зашумленным обучающим словарям

Экспериментальное сравнение предложенного метода решения задачи точной генерации ДКА по зашумленным словарям (раздел 3.2) с существующими методами было затруднено тем, что среди последних не известно методов точной генерации ДКА. Сравнение проводилось с методом неточной генерации, основанным на эволюционном алгоритме [55], который является лучшим среди известных на сегодняшний день. Будем использовать описанную схему (рисунок 12). При этом, генерируя слова по автомату, внесем K ошибок в их метки. Заметим, что попытка провести сравнение на данных соревнования «Learning DFA from Noisy Samples», проходившем в рамках конференции GECCO 2004 [81], не увенчалась успехом – высокий уровень шума (10 %) не позволил применить предложенный метод на практике. В дальнейшем, после развития методов решения SAT, метод может быть успешно применен для решения задач с высоким уровнем шума за конечное время процессорного работы.

Было проведено 100 запусков экспериментов для каждого из чисел состояний $C \in [5, 8]$ и для $K \in [1, 2]$ – каждый из методов генерации ДКА был запущен 800 раз. Число генерируемых слов равно $10 \cdot C$. На рисунке 14 в виде ящичных диаграмм изображены времена работы сравниваемых методов на едином наборе сгенерированных данных: темные ящики соответствуют предложенному методу, светлые ящики соответствуют методу, основанному на эволюционном алгоритме. Для удобства восприятия на рисунке ординаты (времена работы) отложены в логарифмической шкале.

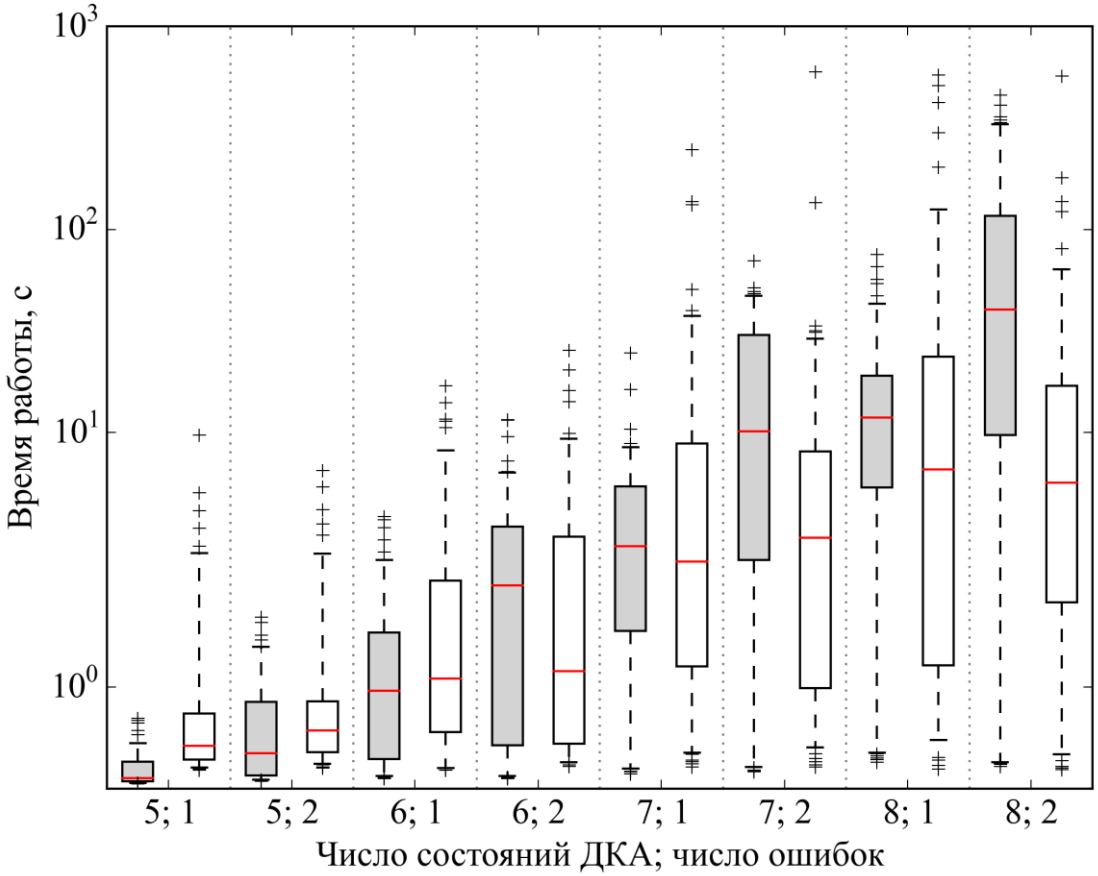

Рисунок 14 – Ящичные диаграммы времен работы предложенного метода (темные ящики) и метода, основанного на эволюционном алгоритме (светлые ящики)

Описанные результаты позволяют сделать следующие выводы:

- наблюдается большой разброс времен работы при одинаковых параметрах задачи (число состояний, размер словарей, число ошибочных меток) как у предложенного метода, так и у метода, основанного на эволюционном алгоритме;
- с ростом параметров задачи повышается медианное время работы методов генерации.

Выводы по главе 3

1. Разработан **метод точной генерации** детерминированных конечных автоматов (ДКА) по безошибочным обучающим словарям. Метод основан на сведении задачи к задаче выполнимости булевых формул (SAT). Основным отличием предложенного метода от лучшего известного метода *DFASAT* [44] решения задачи является подход к нарушению симметрии, сокращающий пространство поиска. Предложены предикаты нарушения симметрии, задающие BFS-нумерацию ДКА: нумерация состояний автомата должна соответствовать порядку обхода автомата в ширину. В булеву формулу добавляется $\mathcal{O}(C^2|\Sigma|^2 + C^3)$ дизъюнктов, использующих $\mathcal{O}(C^2|\Sigma|)$ дополнительных переменных.
2. На основе предложенного метода разработан метод точной генерации ДКА по **зашумленным обучающим словарям.** Наличие ошибок не позволяет использовать граф совместимости и, соответственно, метод нарушения симметрии, основанный на клике данного графа [44]. Предложенные же ранее предикаты BFS-нумерации не чувствительны к наличию ошибок в пометках слов и используются при разработке метода. Предложены предикаты обработки ошибочных пометок, общее число которых составляет $\mathcal{O}(C\ |V_{\pm}|\ +\ K\ |V_{\pm}|)$.
3. Предложенные **методы реализованы** на языке *Java*. Создано инструментальное средство *DFAInducer* с **открытым программным кодом** [72]. Средство в качестве параметра принимает путь к сторонней программе решения задачи SAT. Время работы *DFAInducer* напрямую зависит от времени работы данной сторонней программы на булевой формуле, генерируемой при помощи разработанного сведения. Таким образом, с развитием методов решения SAT будет уменьшаться время работы созданного средства.

4. Проведены **экспериментальные исследования** разработанных и реализованных методов генерации. Получен один из центральных результатов диссертации – время работы предложенного метода точной генерации ДКА по незашумленным данным значительно меньше времени работы метода *DFASAT*, являющегося лучшим известным методом решения задачи.
5. Все результаты главы диссертации были доложены на международной конференции по языкам и автоматам *LATA*-2015 и изложены в работе [91], индексируемой базой *Scopus*.

ГЛАВА 4. ГЕНЕРАЦИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ

В настоящей главе решаются третья и четвертая задачи, поставленные формально в первой главе. Проводится разработка методов точной генерации управляющих конечных автоматов по безошибочным (раздел 4.1) или зашумленным (раздел 4.2) сценариям работы. Проводятся реализация и экспериментальные исследования предлагаемых методов генерации (раздел 4.3). Разработка проводится на основе сведения задач генерации к задаче удовлетворения ограничений. Для управляющих автоматов не известно методов точной генерации: существующие методы, основанные на метаэвристических алгоритмах, в общем случае не гарантируют нахождение решения за конечное время.

4.1. Метод генерации управляющих конечных автоматов по безошибочным сценариям работы

В настоящем разделе проводится разработка метода точной генерации конечных управляющих автоматов по безошибочным сценариям работы. Описывается структура метода, приводится его псевдокод. Разрабатываются алгоритмы построения дерева сценариев и его графа совместимости. В основе метода лежит оригинальное сведение к задаче удовлетворения ограничений, включающее модифицированные предикаты нарушения симметрии. Часть приведенных в разделе результатов была получена при выполнении диссертантом магистерской диссертации по теме «Построение управляющих конечных автоматов по сценариям работы на основе решения задачи удовлетворения ограничений».

4.1.1. Структура метода

Событийные системы зачастую проектируются не только непротиворечивыми (последовательности входных данных однозначно соответствует последовательность выходных воздействий), но и *полными* (для любой последовательности входных данных система поставит в соответствие единственную выходную последовательность). Для управляющих автоматов требование полноты выполняется, если для каждого управляющего состояния q выполняется полнота системы охранных условий исходящих из q переходов [8].

Вопрос генерации полных конечных автоматов рассматривался, например, для построения взаимодействующих ДКА по сценариям взаимодействия (*message sequence charts*) [15]; для построения конечных автоматов-преобразователей по тестам [27]. Известные методы генерации управляющих автоматов, в отличие от предлагаемого в настоящем разделе, вопрос полноты не рассматривают. Однако такая задача может возникать, например, в случае наличия покрывающего набора сценариев работы для неизвестной системы. Таким образом, может быть поставлена и решена задача реверс-инжиниринга (обратной разработки) полных управляющих автоматов, моделирующих неизвестную систему, с полным покрытием переходов тестами.

В листинге 6 приведен псевдокод предлагаемого метода. Структура метода в целом совпадает со структурой метода построения ДКА по незашумленным обучающим словарям (листинг 3).

Листинг 6 – Псевдокод метода точной генерации конечных управляющих автоматов по безошибочным сценариям работы

```
function NoiselessEFSMGeneration(S)
  S – набор непротиворечивых сценариев работы
  𝒯 ← scenariosTree(S)
  𝒢 ← consistencyGraph(𝒯)
  for C ← 1..|𝒯| do
```

```
      f_CSP ← F_CSP(T, G, C)
      solution ← solveCSP(f_CSP)
      if solution ≠ {} then
        A ← EFSM(C, T, solution)
        return A
      end if
    end for
  end function
```

Далее опишем алгоритмы построения дерева сценариев (scenariosTree) и его графа совместимости (consistencyGraph), а также сведение задачи к задаче CSP ($\boldsymbol{\mathcal{F}}_{\mathrm{CSP}}$).

4.1.2. Построение дерева сценариев и графа совместимости

Деревом сценариев назовем дерево, каждый переход которого помечен событием, охранным условием и последовательностью выходных воздействий – тройкой $\langle e; f; A\rangle$. Опишем алгоритм $\mathrm{scenariosTree}(S)$ построения дерева сценариев по заданному множеству сценариев S, аналогичный алгоритму построения префиксного дерева $\mathrm{APTA}(S_+, S_-)$. Изначально дерево сценариев состоит из единственной вершины – корня дерева. Затем по очереди добавим в дерево все сценарии работы из S. Для каждого из сценариев будем добавлять его элементы $T_1 \dots T_n$ в дерево в порядке от T_1 до T_n. При этом будем хранить текущую вершину дерева v и номер i первого необработанного элемента сценария.

В начале процесса добавления v является корнем дерева сценариев, а $i = 1$. На каждом шаге проверяется существование исходящего из вершины v ребра, помеченного событием e_i и логической формулой, задающей ту же булеву функцию, что и f_i. Если такое ребро не существует, то создается новая вершина дерева u, и в нее направляется ребро из v, помеченное тройкой $\langle e_i; f_i; A_i\rangle$. После этого u становится текущей вершиной, а значение i увеличивается на единицу.

Если такое ребро существует, то производится сравнение последовательности A_i и последовательности выходных воздействий A_v, которой помечено рассматриваемое ребро. Если $A_i = A_v$, то текущей становится вершина, в которую ведет рассматриваемое ребро дерева, а значение i увеличивается на единицу. Если же указанные последовательности не совпадают, то заданное множество сценариев S является некорректным и работа алгоритма прерывается.

После завершения добавления всех сценариев в дерево производится проверка охранных условий. Для каждой вершины перебираются все пары исходящих из нее ребер. Если существует такая пара ребер, что они помечены одним и тем же событием, а их охранные условия имеют общий выполняющий набор значений входных переменных, то множество сценариев предполагает недетерминированное поведение. В таком случае работа алгоритма прерывается и пользователю выводится соответствующее сообщение.

Для наглядности описания будем сопровождать настоящий раздел несложным примером задачи генерации управляющего автомата. Рассмотрим пример работы алгоритма построения дерева сценариев. Пусть набор сценариев состоит из следующих последовательностей:

- $\langle \mathrm{T};\ x;\ z_1\rangle, \langle \mathrm{T};\ \neg x;\ z_2\rangle, \langle \mathrm{T};\ \neg x;\ z_3\rangle, \langle \mathrm{T};\ x;\ z_1\rangle$;
- $\langle \mathrm{T};\ \neg x;\ z_2\rangle, \langle \mathrm{T};\ \neg x;\ z_3\rangle, \langle \mathrm{T};\ x;\ z_1\rangle, \langle \mathrm{T};\ x;\ z_1\rangle$;
- $\langle \mathrm{T};\ x;\ z_1\rangle, \langle \mathrm{T};\ x;\ z_1\rangle, \langle \mathrm{T};\ \neg x;\ z_2\rangle$;
- $\langle \mathrm{T};\ \neg x;\ z_2\rangle, \langle \mathrm{T};\ \neg x;\ z_3\rangle, \langle \mathrm{T};\ x;\ z_1\rangle, \langle \mathrm{T};\ \neg x;\ z_2\rangle$.

Данные сценарии работы содержат единственное входное событие T (пусть оно соответствует тику часов), единственную переменную x, два различных охранных условия x и $\neg x$ и три различных последовательности выходных воздействий (z_1), (z_2), (z_3). Построенное описанным алгоритмом дерево сценариев приведено на рисунке 15.

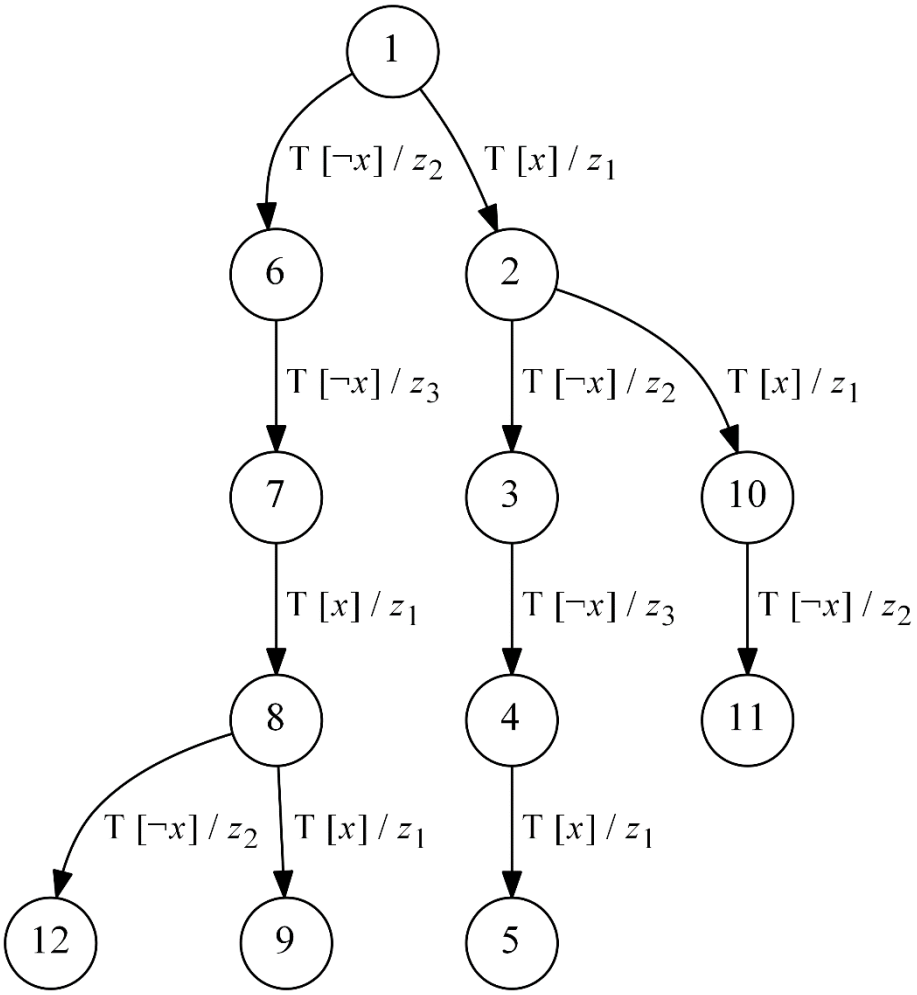

Рисунок 15 – Пример дерева сценариев

Для построения управляющего автомата необходимо «раскрасить» вершины дерева сценариев в заданное число цветов, равное текущему числу состояний C. При этом вершины одного цвета будут объединены в одно состояние результирующего автомата, а множество исходящих из состояния переходов будет строиться из объединения множеств ребер, исходящих из вершин заданного цвета.

Перейдем к описанию функции $\mathrm{consistencyGraph}(\mathcal{T})$ – алгоритму построения графа совместимости $\mathcal{G}$ по дереву сценариев $\mathcal{T}$. Граф аналогичен графу совместимости для префиксного дерева, однако алгоритмы их построения значительно отличаются. Так, например, время построения графа для префиксного дерева оценивается как $\mathcal{O}(|\mathcal{T}|^3)$, а

приведенный далее алгоритм строит граф совместимости дерева сценариев за $\mathcal{O}(|\mathcal{T}|^2)$ (где $|\mathcal{T}|$ – размер дерева сценариев).

Множество вершин $\mathcal{G}$ совпадает с множеством вершин $\mathcal{T}$, поэтому в дальнейшем не будем различать вершины графа и дерева. Вершины графа совместимости u и v соединены ребром (далее такие вершины будем называть *несовместимыми*), если существует последовательность пар $\langle e_1, \text{values}_1 \rangle \dots \langle e_k, \text{values}_k \rangle$ событий и наборов значений входных переменных, которая различает соответствующие вершины дерева. Будем говорить, что указанная последовательность *различает вершины* u и v, если выполняется совокупность следующих условий:

- из вершины u существует путь P_u, ребра которого помечены событиями $e_1 \dots e_k$ и такими охранными условиями $f_1 \dots f_k$, что наборы значений входных переменных $\text{values}_1 \dots \text{values}_k$ являются, соответственно, их выполняющими подстановками;
- аналогичный путь P_v существует из вершины v;
- для последних ребер путей P_u и P_v верно хотя бы одно из двух условий: пометки этих ребер различаются в части выходных воздействий, или у охранных условий этих ребер есть общий выполняющий набор значений входных переменных, но они не эквивалентны как булевы функции.

Опишем алгоритм построения графа совместимости $\mathcal{G}$. Основной идеей данного алгоритма является применение метода динамического программирования [1, 10]. Для каждой вершины дерева сценариев v найдем все несовместимые с ней вершины. Обозначим как $\mathcal{G}(v)$ множество несовместимых с v вершин. Будем вычислять $\mathcal{G}(v)$ начиная с листьев дерева сценариев. Для каждого из листьев v множество $\mathcal{G}(v)$ пусто по введенному определению несовместимых вершин дерева сценариев.

Покажем, как вычислить значение $\mathcal{G}(v)$, если оно уже вычислено для всех детей вершины v дерева. Переберем все вершины дерева – включим

вершину u в множество $\mathcal{G}(v)$, если существует пара ребер $u \to x$ (помечено событием e, формулой f_1 и последовательностью действий A_1) и $v \to y$ (помечено также событием e, формулой f_2 и последовательностью действий A_2) такая, что выполняется хотя бы одно из трех условий:

- формулы f_1 и f_2 имеют общий выполняющий набор значений входных переменных, но не эквивалентны как булевы функции. Тогда $\langle e; \text{values} \rangle$ является последовательностью единичной длины, различающей вершины u и v (как values обозначена выполняющая подстановка f_1);
- формулы f_1 и f_2 эквивалентны как булевы функции, а последовательности A_1 и A_2 не совпадают. Тогда вершины u и v различает такая же последовательность $\langle e; \text{values} \rangle$;
- формулы f_1 и f_2 эквивалентны как булевы функции, и вершина x входит в множество $\mathcal{G}(y)$, вычисленное заранее. Тогда существует последовательность $\langle e_1; \text{values}_1 \rangle \dots \langle e_k; \text{values}_k \rangle$, различающая вершины x и y, из чего следует, что последовательность $\langle e; \text{values} \rangle \langle e_1; \text{values}_1 \rangle \dots \langle e_k; \text{values}_k \rangle$ различает вершины u и v.

Время работы представленного алгоритма составляет $\mathcal{O}(|\mathcal{T}|^2)$, так как каждая пара вершин (и каждая пара ребер) дерева сценариев в процессе работы алгоритма рассматривается не более одного раза. При этом такая асимптотика времени работы верна, если заранее для каждой пары формул вычислено, равны ли они как булевы функции и имеют ли общий выполняющий набор значений входных переменных. В худшем случае время работы этапа обработки формул составляет $\mathcal{O}(2^{2m}|\mathcal{T}|^2)$, где за m обозначено максимальное число входных переменных, использующихся в одном охранном условии. На практике число m не превышает четырех.

Для дерева сценариев, приведенного на рисунке 15, описанный алгоритм построит граф совместимости, приведенный на рисунке 16. На рисунке сплошными линиями выделены ребра графа совместимости, а штриховыми – исходные ребра дерева сценариев. Вершины с номерами 3 и 6 соединены с вершинами 1, 2, 8 и 10, так как их различает последовательность $\langle \mathrm{T}; x = 0 \rangle$ – пометки соответствующих ребер дерева сценариев различаются в выходных воздействиях. Других различающих последовательностей в приведенном дереве сценариев не существует.

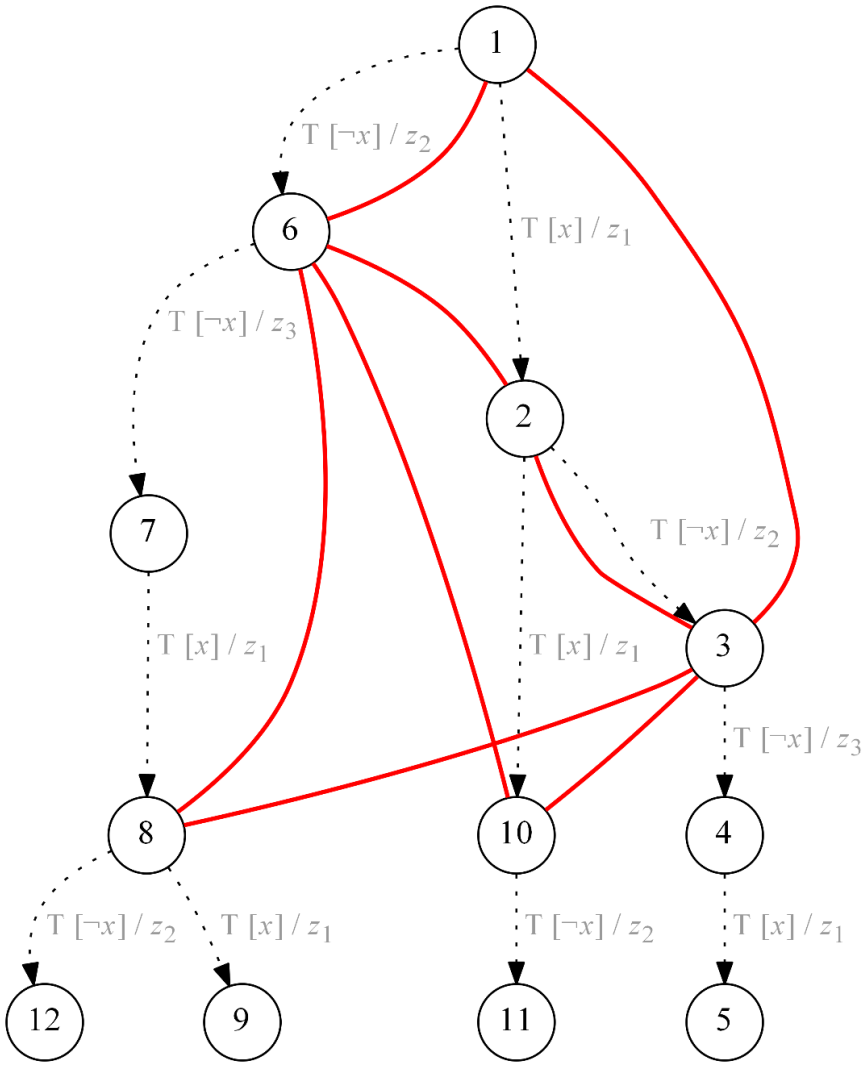

Рисунок 16 – Пример графа совместимости дерева сценариев

Таким образом, были разработаны алгоритмы построения дерева сценариев ($\mathrm{scenariosTree}(S)$) и соответствующего ему графа

совместимости ($\mathrm{consistencyGraph}(\mathcal{T})$). Алгоритм построения графа совместимости основан на методе динамического программирования.

4.1.3. Ограничения на целочисленные переменные

Опишем следующий этап работы алгоритма построения управляющих конечных автоматов по сценариям работы программы – построение набора ограничений $\boldsymbol{\mathcal{F}}_{\mathrm{CSP}}(\mathcal{T}, \mathcal{G}, C)$ на целочисленные переменные. Данные ограничения задают требования к «раскраске» графа совместимости и выражают непротиворечивость и полноту системы переходов искомого автомата.

Так как охранные условия на переходах дерева уже были учтены при построении графа совместимости, можно рассматривать охранные условия f как части событий e и оперировать с событиями вида «$e[f]$». Будем называть такие события *расширенными*. Приведем все охранные условия, встречающиеся в заданных сценариях S, к единому виду, что позволит в дальнейшем сравнивать расширенные события как строки. Составим набор Σ_{X} всех расширенных событий, упорядочим получившиеся строки и пронумеруем их от 1 до $|\Sigma_{\mathrm{X}}|$. Охранное условие f расширенного события e обозначим через $f(e)$.

Для построения набора ограничений будем использовать следующие целочисленные переменные.

1. Переменные x_v соответствуют цвету каждой вершины дерева сценариев $v \in V$ и принимают значения от 1 до C. Напомним, что каждому состоянию результирующего автомата соответствует уникальный «цвет».
2. Переменные $y_{i,e}$ (для $i \in 1..C$, $e \in \Sigma_{\mathrm{X}}$) являются вспомогательными для построения ограничений, задающих непротиворечивость (детерминированность) искомого управляющего автомата, и хранят в себе информацию о его переходах. Данные переменные являются

вспомогательными, так как они не определяют наличие в результирующем автомате переходов (в функции $\mathrm{EFSM}(C, \mathcal{T}, \mathrm{solution})$ используются переменные x_v). Каждая переменная принимает значение от 1 до C и соответствует номеру состояния, в которое ведет переход искомого автомата из состояния i по расширенному событию e.

3. Переменные $u_{i,e}$ (для всех $i \in 1..C,\ e \in \Sigma_{\mathrm{X}}$) используются для задания требования полноты искомого автомата и принимают значения 0 или 1, то есть по своей сути являются логическими. Переменная $u_{i,e}$ равна 1, если существует вершина v в дереве сценариев, цвет которой равен i ($x_v = i$), и из нее ведет ребро, помеченное расширенным событием e. В противном случае значение переменной равно 0. Таким образом, данные переменные хранят информацию о структуре переходов результирующего автомата, получающегося в результате объединения вершин дерева сценариев.

Составим набор ограничений на указанные переменные, задающий требования полноты и непротиворечивости искомого автомата.

1. $x_1 = 1$ – ограничение, задающее соответствие корня дерева сценариев начальному состоянию искомого автомата. В настоящем методе начальным состоянием автомата всегда является состояние с номером 1.
2. $x_v \neq x_u$ (для каждой несовместимой пары вершин дерева сценариев v и u: $(v, u) \in \mathcal{G}$) – ограничения, задающие непротиворечивость искомого автомата. Они гарантируют отсутствие различающих последовательностей, ведущих из одного состояния автомата. Число ограничений данного вида равно числу ребер графа совместимости, то есть в худшем случае таких ограничений может быть $\mathcal{O}(|\mathcal{T}|^2)$.
3. $(x_v = i) \Rightarrow (x_u = y_{i,e})$ (для $i \in 1..C$ и каждого ребра $v \xrightarrow[e]{} u$ дерева сценариев $\mathcal{T}$) – ограничения, задающие детерминированность

искомого автомата. Если вершине v присвоен цвет i, то цвет дочерней вершины u должен совпадать со значением переменной $y_{i,e}$, хранящей номер состояния автомата, в которое ведет переход из состояния i, помеченный расширенным событием e. Число данных ограничений составляет $\mathcal{O}(C \cdot |V|)$.

4. $(u_{i,e} = 1) \Leftrightarrow \bigvee_{v \in V(e)}(x_v = i)$ (для $i \in 1..C$, $e \in \Sigma_X$; как $V(e)$ обозначены все вершины дерева сценариев, из которых ведет ребро, помеченное e) – ограничения, определяющие переменные $u_{i,e}$. Число ограничений данного типа оценивается как $\mathcal{O}(C \cdot |\Sigma_X|)$, однако суммарная длина ограничений составляет $\mathcal{O}(C \cdot |\mathcal{T}|)$.
5. Воспользуемся определенными переменными для задания требования полноты генерируемого автомата. Для $i \in 1..C$, $e_o \in \Sigma$:

$$\sum_{e \in \Sigma_X : \mathrm{event}(e) = e_o} \left(u_{i,e} \cdot c(e)\right) \in \{0, 2^m\},$$

где функция $\mathrm{event}(e)$ принимает на вход расширенное событие $e \in \Sigma_X$ и возвращает его префикс – нерасширенное событие $e_o \in \Sigma$, $c(e)$ – число выполняющих подстановок для охранного условия, включенного в расширенное событие e. При подсчете $c(e)$ считается, что булева формула зависит от всех m переменных, содержащихся в сценариях (например, $c(\mathrm{A}[1]) = 2^m$, а $c(\mathrm{A}[x_1 \vee \neg x_2]) = 2^{m-2}$). Сумма $\sum_{e \in \Sigma_X : \mathrm{event}(e) = e_o} \left(u_{i,e} \cdot c(e)\right)$ равна числу комбинаций значений входных переменных values, для которых существует переход из состояния i, помеченный событием e_o и условием перехода f таким, что выполняется $f|_{\mathrm{values}} = 1$. Условие полноты (в «слабом» смысле) искомого автомата выражается тем, что или для любого значения входных переменных найдется переход, или ни для одного из значений переменных перехода не существует. Заметим, что условие того, что для любого

значения входных переменных найдется переход, можно выразить как $\sum_{e\in\Sigma_X:\mathrm{event}(e)=e_o}\left(u_{i,e}\cdot c(e)\right)=2^m$, так как считается, что требование непротиворечивости уже выполнено – все расширенные события, для которых выполняется $u_{i,e}=1$, попарно не имеют общих выполняющих подстановок. Для удовлетворения требованию «сильной» полноты можно изменить множество $\{0,2^m\}$ на $\{2^m\}$.

Приведенные ограничения пяти типов составляют набор, задающий требования $\boldsymbol{\mathcal{F}}_{\mathrm{CSP}}(\mathcal{T},\mathcal{G},C)$ непротиворечивости и полноты искомого автомата, удовлетворяющего сценариям S и содержащего C управляющих состояний. Отдельно отметим, что ограничения не обращаются к выходным последовательностям сценариев в явном виде, так как они были обработаны при подсчете графа совместимости $\mathcal{G}$.

4.1.4. Предикаты нарушения симметрии

Переведем предикаты нарушения симметрии, разработанные в разделе 3.1.2, с языка булевых формул SAT на язык ограничений CSP. Предикаты нарушения симметрии универсальны и используются во всех методах, разрабатываемых в настоящей диссертации. Напомним, что основной идеей предлагаемого подхода к нарушению симметрии является BFS-нумерация: фиксирование номеров состояний в порядке обхода в ширину.

Предикаты дополняют набор ограничений $\boldsymbol{\mathcal{F}}_{\mathrm{CSP}}(\mathcal{T},\mathcal{G},C)$**.** Для построения набора дополнительных предикатов будем использовать следующие целочисленные переменные, аналогичные ранее предложенным булевым.

1. Переменные $t_{i,j}$, по сути, остаются булевыми (принимают значения 0 или 1). Переменная $t_{i,j}=1$ тогда и только тогда, когда в автомате существует хотя бы один переход из состояния i в состояние j. Переменные определены для каждой пары состояний: $i,j\in 1..C$.

2. Переменные p_j определены для каждого $j \in 2..C$ и хранят предка вершины в дереве обхода искомого автомата в ширину (при его обходе из первого состояния). Таким образом, переменная p_j принимает значение в диапазоне $1..j-1$, и $p_j = i$ тогда и только тогда, когда при обходе ДКА в ширину вершина j была добавлена в очередь при просмотре исходящих ребер из i.
3. Будем хранить лексикографически наименьшее расширенное событие между каждой парой управляющих состояний в переменных $m_{i,j}$ для всех $i, j \in 1..C$ ($1 \le m_{i,j} \le |\Sigma_X|$). Рассмотрим все переходы $i \xrightarrow[e]{} j$ между состояниями i и j генерируемого автомата. $m_{i,j} = e$ тогда и только тогда, когда e является лексикографически наименьшим расширенным событием, которое встретилось на данных переходах.

Составим набор ограничений, задающий определения переменных и «вынуждающий» искомый автомат, заданный переменными y, быть пронумерованным в порядке обхода в ширину.

1. В соответствии с указанным ранее смыслом определим переменные $t_{i,j}$ ограничениями:

$$t_{i,j} = \exists e \in \Sigma_X : (y_{i,e} = j).$$

В тех ситуациях, когда использование оператора существования затруднительно, определения можно переписать как

$$t_{i,j} = \bigvee_{e \in \Sigma_X} (y_{i,e} = j).$$

2. Определим переменные p_j через $t_{i,j}$. Предком состояния j при BFS-нумерации управляющего автомата является состояние i с наименьшим номером среди инцидентных j:

$$(p_j = i) \Leftrightarrow t_{i,j} \wedge \bigwedge_{1 \le k < i} \neg t_{k,j}.$$

3. Зададим первое ограничение, соответствующее BFS-нумерации, с использованием определенных переменных предка p_j. Состояние с б*о*льшим номером попадает в очередь позже состояния с меньшим номером. Чтобы это требование выполнялось, вводятся ограничения на номера предков для состояний с подряд идущими номерами:
$$p_j \leq p_{j+1}$$
для $1 < j < C$.
4. Определим переменные $m_{i,j}$ через определенные ранее $t_{i,j}$ и $y_{i,e}$. Если перехода от состояния i к состоянию j не существует, то зафиксируем значения $m_{i,j}$:
$$\left(t_{i,j} = 0\right) \Rightarrow \left(m_{i,j} = 0\right).$$
Если же хотя бы один переход существует, то выберем лексикографически наименьшее событие, которым помечен переход из i в j:
$$(t_{i,j} = 1) \Rightarrow \left[(m_{i,j} = e) \Leftrightarrow (y_{i,e} = j) \wedge \bigwedge_{e^* \in \Sigma_X, e^* < e} (y_{i,e^*} \neq j)\right].$$
5. Зададим второе ограничение, соответствующее BFS-нумерации, с использованием определенных переменных $m_{i,j}$. Упорядочим при помощи $m_{i,j}$ состояния q_j и q_{j+1} с равным предком q_i – в очередь ранее будет добавлено состояние с переходом по меньшему лексикографически расширенному событию:
$$\left(p_j = i \wedge p_{j+1} = i\right) \Rightarrow \left(m_{i,j} < m_{i,j+1}\right)$$
для всех $1 \leq i < j < C$.

Данный набор ограничений пяти типов задает предикаты нарушения симметрии на языке CSP и дополняет сведение $\boldsymbol{\mathcal{F}}_{\mathrm{CSP}}(\mathcal{T}, \mathcal{G}, C)$. Суммарное число ограничений сведения оценивается как $\mathcal{O}(C(C + |\Sigma_X|) + |\mathcal{T}|^2)$. При этом используется $\mathcal{O}(C(C + |\Sigma_X|) + |\mathcal{T}|)$ целочисленных переменных (размерности не более C).

4.2. Метод генерации по зашумленным сценариям работы

В настоящем разделе проводится разработка последнего предлагаемого в диссертации метода точной генерации конечных автоматов. Решается задача построения управляющих конечных автоматов по сценариям работы, которые могут содержать зашумленные последовательности выходных воздействий (при безошибочных входных событиях и охранных условиях). Описывается структура метода, не использующего дерево сценариев и его граф совместимости. Предлагаются новые ограничения на целочисленные переменные.

4.2.1. Структура метода

Сценарии с зашумленными последовательностями выходных воздействий могут быть получены, к примеру, при передаче данных по каналам, допускающим ошибки, или при ошибках пользователя, вводящего сценарии поведения системы. С другой стороны, такая задача может возникнуть, если пользователь желает сгенерировать автомат с меньшим числом управляющих состояний, чем есть в исследуемой системе, который *приблизительно* соответствует заданным примерам поведения.

Ранее в разделе 3.2 был разработан метод точной генерации ДКА по зашумленным обучающим словарям. Данный метод не использовал граф совместимости (раздел 3.1), но, несмотря на ошибки в пометках префиксного дерева, использовал дерево при генерации ДКА. Покажем, что в случае зашумленных сценариев работы структура соответствующего дерева изменяется (в отличие от префиксного дерева), и использование дерева сценариев нецелесообразно.

На рисунке 17*а* приведен пример дерева сценариев, построенного по трем безошибочным сценариям работы программы:

- $\langle \mathrm{B}; 1; z_1 \rangle$, $\langle \mathrm{A}; 1; z_2 \rangle$, $\langle \mathrm{A}; 1; z_3 \rangle$, $\langle \mathrm{B}; 1; z_1 \rangle$;
- $\langle \mathrm{A}; 1; z_2 \rangle$, $\langle \mathrm{A}; 1; z_3 \rangle$, $\langle \mathrm{B}; 1; z_1 \rangle$, $\langle \mathrm{B}; 1; z_1 \rangle$;

– $\langle A; 1; z_2 \rangle$, $\langle A; 1; z_3 \rangle$, $\langle B; 1; z_1 \rangle$, $\langle A; 1; z_2 \rangle$.

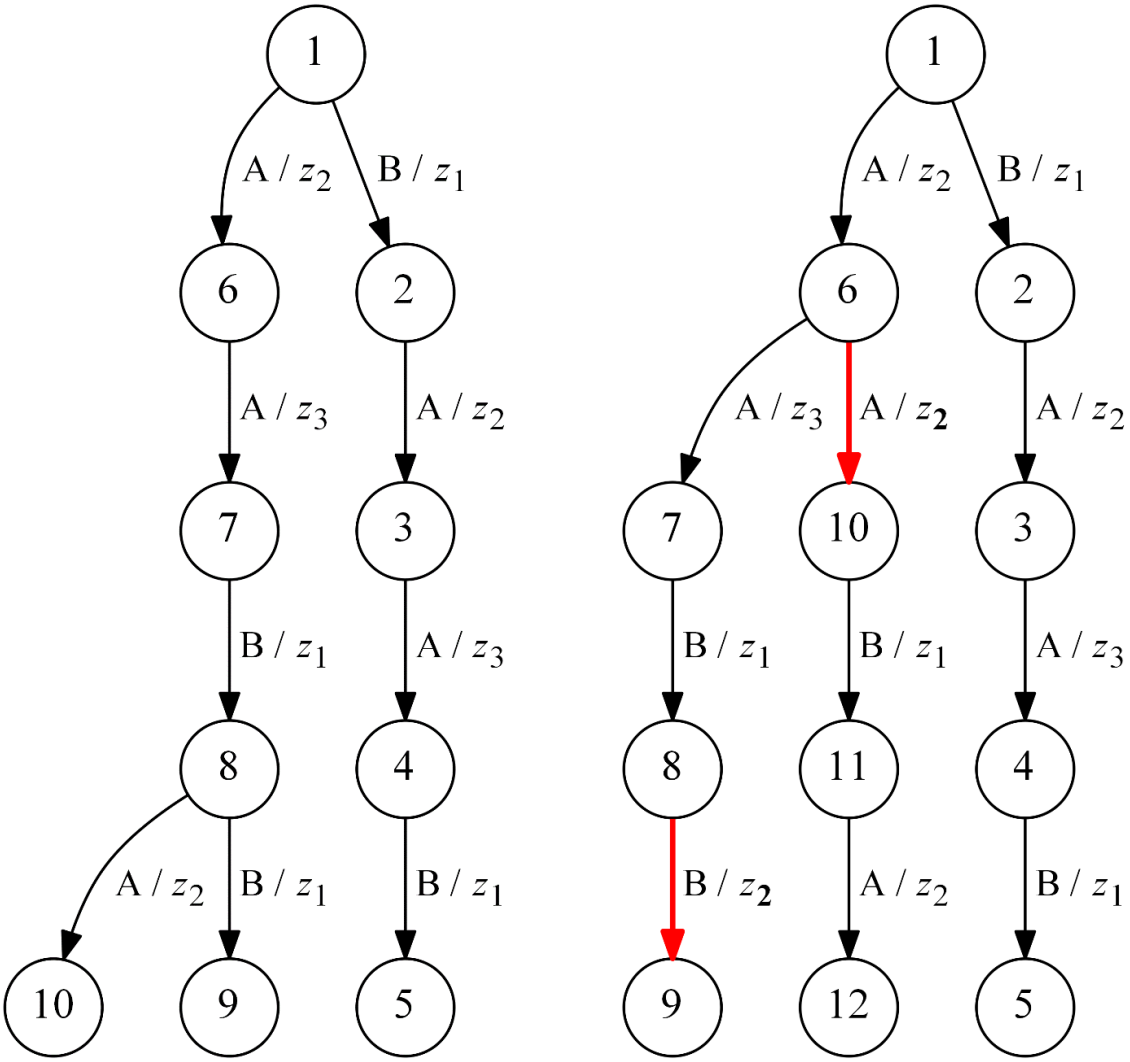

Рисунок 17 – Примеры деревьев сценариев, построенных по: *а* – безошибочным сценариям; *б* – зашумленным сценариям

Пусть в результате считывания или обработки примеров работы системы (или же в случае влияния человеческого фактора) были допущены две ошибки выходных воздействий: во втором элементе второго сценария и четвертом элементе третьего. На рисунке 17*б* приведено дерево сценариев для ошибочных сценариев работы (полужирным выделены допущенные ошибки). Таким образом, при ошибках в сценариях дерево может содержать недетерминированные вершины (вершина 6 на рисунке 17*б*) и большее число вершин. Еще раз отметим, что и построение

графа совместимости при наличии ошибок невозможно. Предложим метод генерации, не использующий дерево сценариев. Псевдокод метода приведен в листинге 7.

Листинг 7 – Псевдокод метода точной генерации конечных управляющих автоматов по зашумленным сценариям работы

```
function NoisyEFSMGeneration(S, K)
  S – набор непротиворечивых сценариев работы
  K – допустимое число ошибочных выходных последовательностей
  for C ← 1..Σ(S) do
    f_CSP ← F_CSP(S, C, K)
    solution ← solveCSP(f_CSP)
    if solution ≠ {} then
      A ← EFSM(C, S, solution)
      return A
    end if
  end for
  return {}
end function
```

В листинге как $\Sigma(S)$ обозначена суммарное число элементов в сценариях S. Функция solveCSP запускает стороннее средство для нахождения удовлетворяющей подстановки генерируемой на каждом шаге формулы f_{CSP}. В случае нахождения подстановки solution генерируется искомый автомат при помощи функции EFSM. Отметим, что в отличие от трех методов генерации, разработанных ранее, может не существовать ни одного автомата, являющегося ответом на задачу. Опишем предлагаемое сведение $\boldsymbol{\mathcal{F}}_{\text{CSP}}(S, C, K)$ задачи генерации управляющих автоматов по зашумленным сценариям работы к задаче CSP.

4.2.2. Ограничения на целочисленные переменные

Как уже было обозначено, метод не использует дерево сценариев. Представим каждый сценарий $T_1, \dots, T_n$ в виде *линейного* графа из $n + 1$ вершины, в котором из вершины i исходит ребро в $i + 1$, помеченное

тройкой $T_i = \langle e_i; f_i; A_i \rangle$. Множество вершин линейных графов всех сценариев, или просто множество вершин сценариев, обозначим как V; введем на данном множестве сплошную нумерацию от 1 до $|V|$. Дополнительно обозначим подмножество стартовых вершин сценариев как V_b, подмножество конечных вершин – как V_e. Будем рассматривать набор всех расширенных событий Σ_X, встречающихся в S. Также будем рассматривать множество Z_S – набор встречающихся в S последовательностей выходных воздействий.

Разработаем сведение, «раскрашивающее» непосредственно вершины сценариев. Для построения набора ограничений $\boldsymbol{\mathcal{F}}_{\mathrm{CSP}}$ создадим и будем использовать следующие целочисленные переменные.

1. Переменные x_v ставят в соответствие каждой вершине сценариев $v \in V$ ее цвет – номер соответствующего состояния генерируемого автомата. Переменные аналогичны введенным в уже разработанных методах и принимают значения от 1 до C.
2. Также, как и в разработанных ранее методах вводятся переменные $y_{i,e}$ (для $i \in 1..C,\ e \in \Sigma_X$) для задания непротиворечивости генерируемого управляющего автомата, хранящие в себе информацию о функции переходов автомата. Переменные принимают значения от 1 до C и соответствуют номеру состояния, в которое ведет переход автомата из состояния i по расширенному событию e.
3. По аналогии с методами генерации ДКА (глава 2) будем хранить функцию выходов управляющего автомата в переменных $z_{i,e}$ (для $i \in 1..C,\ e \in \Sigma_X$). Переменные $z_{i,e}$ принимают значения от 1 до $|Z_S|$ и хранят номер последовательности выходных воздействий, выполняемой на переходе из состояния i по расширенному событию e.

4. Для каждого перехода в сценариях работы будем хранить, допустимо ли в нем наличие ошибок выходных воздействий. Заведем переменную f_v для каждой вершины $v \in V \backslash V_e$, которая принимает значение 0 (нет ошибки) или 1 (наличие ошибки).

Составим набор ограничений на указанные целочисленные переменные, задающий требования детерминированности генерируемого автомата и допущения им на сценариях S не более K ошибок.

1. $x_v = 1$ (для каждой вершины $v \in V_b$) – ограничение, задающее соответствие первых вершин сценариев начальному состоянию автомата.
2. $x_{v+1} = y_{x_v, e(v)}$ (для каждой вершины $v \in V \backslash V_e$) – ограничения, задающие детерминированность искомого автомата (однозначность функции переходов, заданной переменными $y_{i,e}$). Если вершине v присвоен цвет i ($x_v = i$), то цвет вершины $v+1$ совпадает по определению со значением переменной $y_{i,e(v)}$. Как $e(v)$ обозначено расширенное событие на переходе, исходящем из v.
3. $(f_v = 0) \Rightarrow \left(z_{x_v, e(v)} = z(v)\right)$ (для каждой вершины $v \in V \backslash V_e$) – ограничения, задающие, с точностью до ошибок, непротиворечивость функции выходных воздействий генерируемого автомата. Как $z(v)$ обозначен номер последовательности выходных воздействий на переходе, исходящем из вершины v.
4. $\sum_{v \in V \backslash V_e} f_v \leq K$ – ограничение суммарного числа ошибок f_v сверху заданным методу числом K.

Приведенные $\mathcal{O}(|V|)$ ограничений четырех типов составляют набор, задающий требования непротиворечивости генерируемого управляющего конечного автомата, удовлетворяющего сценариям S, содержащего C управляющих состояний и допускающего не более K ошибочных последовательностей выходных воздействиях. Помимо указанных

ограничений, сведение $\boldsymbol{\mathcal{F}}_{\mathrm{CSP}}(S, C, K)$ включает в себя ограничения BFS-нумерации (раздел 4.1.4). Суммарное число ограничений сведения оценивается как $\mathcal{O}(C^2 + |V|)$. При этом используется $\mathcal{O}(C(C + |\Sigma_{\mathrm{X}}|) + |V|)$ целочисленных переменных.

4.3. Реализация и экспериментальные исследования методов

В настоящем разделе проводятся реализация и экспериментальные исследования разработанных методов генерации конечных управляющих автоматов по примерам поведения.

4.3.1. Программное средство генерации конечных управляющих автоматов

Реализация предложенных методов генерации конечных управляющих автоматов проведена на языках программирования *Java* и *Python*, ограничения на целочисленные переменные записаны на языке *MiniZinc*. Создано инструментальное средство *EFSMTools* с открытым программным кодом [75]. Схема работы средства приведена на рисунке 18.

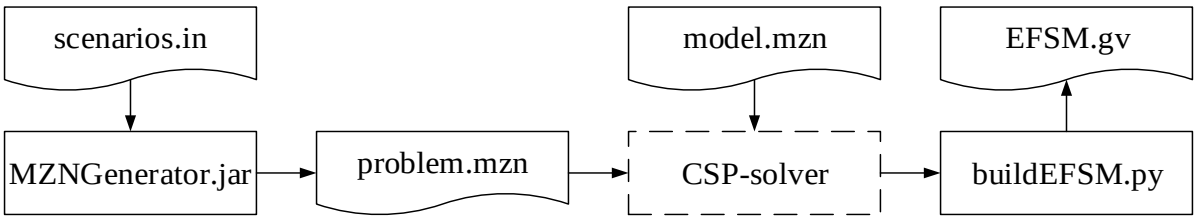

Рисунок 18 – Схема работы *EFSMTools*

На вход *EFSMTools* поступают сценарии работы (файл «`scenarios.in`») в текстовом формате. Сценарии обрабатываются программой `MZNGenerator.jar`, которая по сценариям генерирует файл «`problem.mzn`». Если решается задача генерации по незашумленным сценариям, то файл «`problem.mzn`» будет содержать информацию о дереве сценариев и его графе совместимости на языке *MiniZinc*. Если сценарии зашумлены, то «`problem.mzn`» содержит заданное число

ошибок *K*. Файл «`model.mzn`» содержит разработанные в настоящей главе ограничения на целочисленные переменные.

Ограничения совместно со сгенерированными данными поступают на вход стороннему программному средству решения задачи удовлетворения ограничений (в дальнейших экспериментах используется *Opturion CPX* [82], в случае невозможности его использования рекомендуется использовать *OR-tools* [77]). После нахождения удовлетворяющего набора значений целочисленных переменных, выход стороннего средства поступает на вход скрипту `buildEFSM.py`, который генерирует файл «`EFSM.gv`» с найденным управляющим конечным автоматом в формате *GraphViz* [78]. Пример управляющего автомата для торгового аппарата (рисунок 4), записанного в данном формате, приведен в листинге 8.

Листинг 8 – Пример текстового описания управляющего конечного автомата в формате *GraphViz*

```
digraph EFSM {
  node [shape = circle];

  0 -> 1 [label = "START [1] (OK)"];
  1 -> 1 [label = "START [1] (OK)"];
  1 -> 1 [label = "TOFFEE [1] (NO)"];
  1 -> 1 [label = "CHOC [1] (NO)"];
  1 -> 2 [label = "COIN [x1] (OK)"];
  1 -> 3 [label = "COIN [~x1] (OK)"];
  2 -> 2 [label = "CHOC [1] (NO)"];
  2 -> 2 [label = "COIN [~x1] (NO)"];
  2 -> 1 [label = "START [1] (OK)"];
  2 -> 1 [label = "TOFFEE [1] (TOFFEE)"];
  2 -> 3 [label = "COIN [x1] (OK)"];
  3 -> 1 [label = "START [1] (OK)"];
  3 -> 1 [label = "CHOC [1] (CHOC)"];
  3 -> 2 [label = "TOFFEE [1] (TOFFEE)"];
  3 -> 3 [label = "COIN [1] (NO)"];
```

```
}
```

Данный управляющий автомат был построен следующим образом. Файл «`scenarios.in`» содержал 25 сценариев, некоторые из которых приведены в разделе 1.2. Для них на первом шаге работы `MZNGenerator.jar` сгенерировано дерево сценариев $\mathcal{T}$, состоящее из 340 вершин. На втором шаге для $\mathcal{T}$ сгенерирован граф совместимости $\mathcal{G}$, содержащий 12884 ребра. На третьем шаге работы `MZNGenerator.jar` при $C = 4$ по $\mathcal{T}$ и $\mathcal{G}$ сгенерировано 15240 ограничений на 1870 целочисленных переменных. После нахождения программным средством значений переменных (время работы средства *Opturion CPX* составило 8,7 с) по ним был построен приведенный управляющий автомат.

4.3.2. Экспериментальные исследования метода генерации управляющих автоматов по безошибочным сценариям работы

Проведены экспериментальные исследования разработанного и реализованного метода генерации управляющих автоматов по безошибочным сценариям работы. Эксперименты проведены по схеме, приведенной на рисунке 19. Данная схема аналогична ранее использованной для ДКА (рисунок 12).

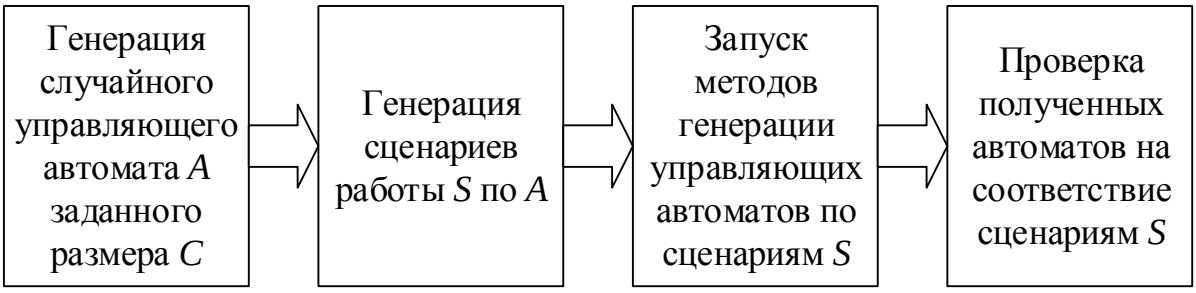

Рисунок 19 – Схема проведения экспериментального исследования методов генерации управляющих автоматов

На первом шаге каждого эксперимента генерировался случайный управляющий автомат A с заданными числом состояний C, множеством событий Σ, множеством переменных X, множеством выходных воздействий Z. На втором шаге генерировался набор сценариев работы S

по управляющему автомату A. При этом сценарии генерировались по случайным путям в графе переходов автомата, покрывающим все ребра графа. На третьем шаге были произведены запуски сравниваемых методов генерации автоматов по сценариям S. В конце произведена валидация: полученные автоматы проверяются на удовлетворение сценариям.

Для проведения корректного сравнения был реализован алгоритм *поиска с возвратом* (*переборный алгоритм*), программный код которого также открыт [75]. Данный алгоритм обладает точностью генерации: в случае отсутствия решения перебираются (в худшем случае) все управляющие автоматы с заданными параметрами. По описанной схеме было проведено 100 запусков экспериментов для каждого из чисел состояний $C \in [4, 13]$ – каждый из двух методов генерации управляющих автоматов был запущен 1000 раз.

Эксперименты были проведены при следующих параметрах. На первом шаге каждого эксперимента генерировался управляющий автомат A размера C с $|\Sigma| = 4$, $|\mathrm{Z}| = 2$, $|\mathrm{X}| = 1$. Число переходов автомата выбиралось равным $4 \cdot C$; на каждом переходе содержится одно или два выходных воздействия. На втором шаге генерировался набор S сценариев работы, покрывающий все переходы A. Суммарное число элементов в сценариях равно $50 \cdot C$. Запуски производились на компьютере с процессором *AMD Phenom II X6 1090T* (тактовая частота 3,2 ГГц). Суммарное процессорное время, использованное для проведения запусков, эквивалентно 11,7 дням. Для решения задачи CSP использовалось одно из лучших на настоящий момент программных средств *Opturion CPX*.

В таблице 2 приведены результаты проведенных экспериментов. Столбцы таблицы соответствуют следующим величинам:

- число C состояний генерируемого управляющего автомата A;
- медианное время T_1 работы предложенного метода генерации (включая время работы *Opturion CPX*);

- медианное время T_2 работы алгоритма поиска с возвратом. При этом время работы было ограничено 10 минутами для $C \leq 10$ и 30 для $C > 10$;
- P-значение (P-value), вычисленное для времен работ методов при помощи Т-критерия Уилкоксона. Полученные P-значения нормализованы методом Бонферрони-Холма.

Таблица 2. Результаты экспериментального сравнения предложенного метода генерации управляющих автоматов с переборным алгоритмом

C	T_1, с	T_2, с	P-значение
4	3,2	0,3	$8{,}7 \cdot 10^{-7}$
5	7,0	3,2	0,2
6	13,3	38,3	$2{,}1 \cdot 10^{-6}$
7	22,1	600*	$3{,}1 \cdot 10^{-15}$
8	34,1	600*	$4{,}4 \cdot 10^{-17}$
9	54,5	600*	$4{,}1 \cdot 10^{-17}$
10	92,8	600*	$3{,}6 \cdot 10^{-17}$
11	160,0	1800*	$3{,}9 \cdot 10^{-17}$
12	351,3	1800*	$1{,}9 \cdot 10^{-13}$
13	632,5	1800*	$4{,}9 \cdot 10^{-9}$

Значения для T_2, отмеченные звездочкой, соответствуют тому, что медианное время работы для $C \geq 7$ превысило допустимое – в таких случаях для более, чем половины экспериментов выполнение поиска с возвратом было остановлено. Результаты экспериментов показывают, что переборный алгоритм работает быстрее для $C \in \{4, 5\}$, а при $C \geq 6$ метод, основанный на сведении к задаче удовлетворения ограничений, показывает статистически значимое (P-значения не превосходят 0.05) преимущество по времени работы. Соответствие сгенерированных автоматов со сценариями было дополнительно подтверждено валидатором.

На рисунке 20 приведены ящичные диаграммы для времен работы предложенного в диссертации метода. Напомним, что диаграмма для каждого $C \in [4, 13]$ построена по 100 значениям времен работы. Для удобства восприятия на рисунке ординаты (времена работы) отложены в логарифмической шкале.

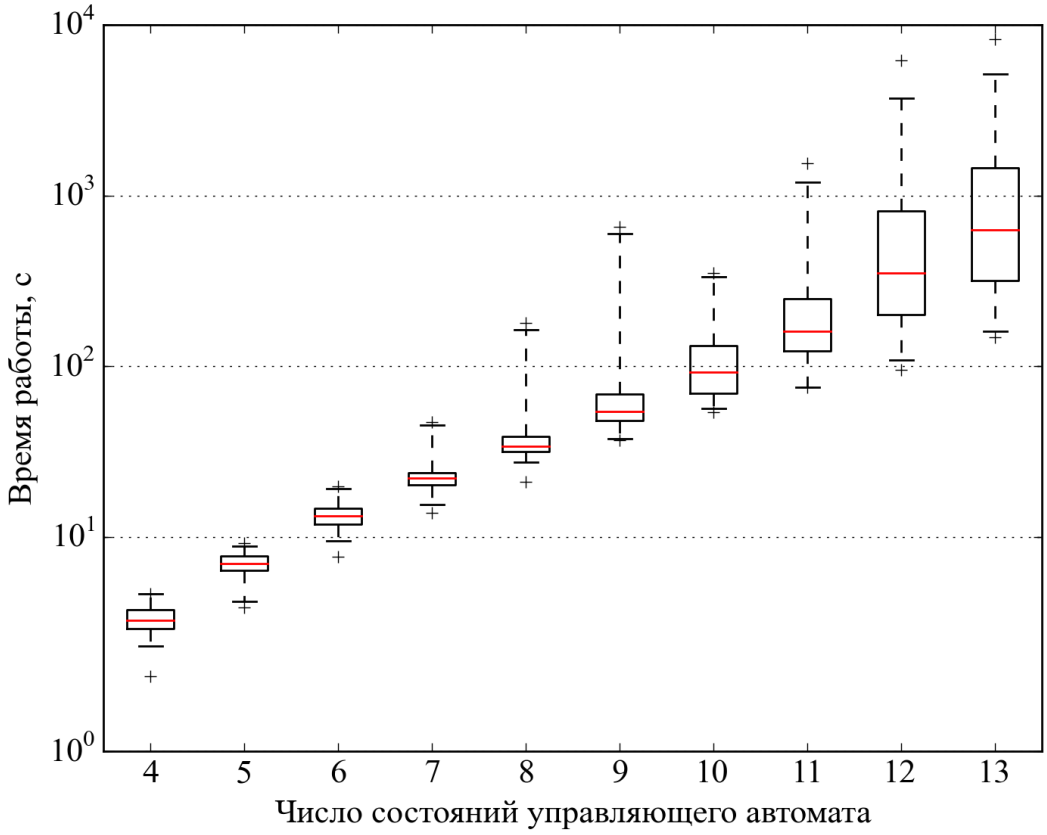

Рисунок 20 – Ящичные диаграммы времен работы предложенного метода точной генерации управляющих автоматов по безошибочным сценариям

Диаграммы позволяют судить об экспоненциальной зависимости времени работы метода от числа состояний искомого управляющего автомата и, соответственно, размера входных данных. Такая зависимость характерна для трудных в NP задач.

4.3.3. Экспериментальные исследования метода генерации управляющих автоматов по зашумленным сценариям работы

По аналогичной схеме были проведены эксперименты с разработанным и реализованным методом генерации управляющих автоматов по зашумленным сценариям работы. На втором шаге каждого эксперимента (рисунок 19) вносились ошибки в указанное число последовательностей выходных воздействий.

Управляющие автоматы генерировались со следующими параметрами: $C \in [5, 9]$, $|\Sigma| = 2$, $|Z| = 3$. Для каждого $C \in [5, 9]$, $K \in \{1, 2\}$, $L \in \{1000, 1500, 2000\}$ было сгенерировано 100 автоматов размера C (для каждого C и K – 300 автоматов). По каждому из них были сгенерированы сценарии работы суммарной длины L с внесенными ошибками в K % выходных последовательностей. При указанных параметрах число ошибок в выходных воздействиях сценариев достигало 40.

На сгенерированных данных предложенный метод был сравнен с методом *MuACO*, основанным на муравьиных алгоритмах [26], и точным алгоритмом поиска с возвратом. Еще раз отметим, что метод *MuACO* не гарантирует точность решения и полноту генерируемого автомата, а алгоритм поиска с возвратом гарантирует.

В таблице 3 приведены результаты экспериментов. Столбцы таблицы соответствуют следующим величинам:

- число C состояний генерируемого управляющего автомата A;
- процент K ошибочных выходных последовательностей в сценариях работы, являющийся параметром сравниваемых методов генерации;
- медианное время T_1 работы предложенного метода генерации (включая время работы *Opturion CPX*);
- медианное время T_2 работы метода *MuACO*;
- медианное время T_3 работы поиска с возвратом. Время работы алгоритма было ограничено пятью минутами;

- P-значение P_1, вычисленное для времен работ первого и второго методов при помощи Т-критерия Уилкоксона. Полученные P-значения нормализованы методом Бонферрони-Холма;
- P-значение P_2, вычисленное аналогичным образом для времен работ первого и третьего методов.

Таблица 3. Результаты экспериментального сравнения методов генерации управляющих автоматов по зашумленным сценариям работы

C	K, %	T_1, с	T_2, с	T_3, с	P_1	P_2
5	1	9,7	3,8	1,2	$8{,}6 \cdot 10^{-15}$	$6{,}1 \cdot 10^{-50}$
5	2	19,6	4,1	3,3	$5{,}8 \cdot 10^{-37}$	$5{,}5 \cdot 10^{-50}$
6	1	11,0	10,6	2,1	0,14	$4{,}3 \cdot 10^{-43}$
6	2	21,2	11,5	13,9	$1{,}9 \cdot 10^{-8}$	0,5
7	1	12,9	16,5	7,5	$8{,}2 \cdot 10^{-8}$	0,78
7	2	26,5	15,2	125,9	0,45	$4{,}5 \cdot 10^{-37}$
8	1	15,5	24,4	48,9	$8{,}3 \cdot 10^{-21}$	$2{,}2 \cdot 10^{-31}$
8	2	28,2	25,4	300*	$4{,}0 \cdot 10^{-5}$	$9{,}3 \cdot 10^{-50}$
9	1	19,6	39,4	300*	$1{,}4 \cdot 10^{-31}$	$8{,}6 \cdot 10^{-47}$
9	2	30,0	52,8	300*	$2{,}8 \cdot 10^{-14}$	$6{,}4 \cdot 10^{-50}$

Приведенные результаты позволяют сделать следующие выводы:

- предложенный в диссертации метод уступает *MuACO* и поиску с возвратом при малых $C \in [5, 6]$, но опережает при $C \in [8, 9]$;
- метод поиска с возвратом при $C \in [8, 9]$ подвержен «экспоненциальному взрыву», в то время как рост времен работы предложенного метода не носит экспоненциальный характер при указанных размерностях задачи.

Выводы по главе 4

1. Разработан **метод точной генерации** управляющих конечных автоматов по безошибочным сценариям работы. Метод основан на сведении к задаче удовлетворения ограничений (CSP). Разработаны алгоритмы построения дерева сценариев и его графа совместимости. Предложено сведение, состоящее из $\mathcal{O}(C(C + |\Sigma_X|) + |\mathcal{T}|^2)$ ограничений на шесть типов целочисленных переменных. Сведение включает в себя модифицированные предикаты нарушения симметрии, предложенные ранее.
2. Разработан метод точной генерации управляющих конечных автоматов по сценариям работы, которые могут содержать **зашумленные последовательности выходных воздействий.** Метод, в отличие от ранее предложенных, не использует древовидное представление входных данных. Основой метода является сведение к задаче CSP: суммарное число ограничений на семь типов переменных составляет $\mathcal{O}(C^2 + |V|)$.
3. Разработанные методы реализованы в **инструментальном средстве *EFSMTools*** с открытым программным кодом [75]. Реализация проведена на языках программирования *Java* и *Python*, ограничения на целочисленные переменные записаны на языке *MiniZinc*. Для решения задачи удовлетворения ограничений используется средство *Opturion CPX* [82], являющееся одним из лучших в настоящее время.
4. Экспериментальные исследования метода генерации управляющих автоматов по безошибочным сценариям работы показали **меньшее время работы** предложенного метода по сравнению с точным алгоритмом поиска с возвратом при числе состояний автомата $C \in [6, 13]$. Для сравнения было проведено 1000 экспериментальных исследований.

5. Метод точной генерации управляющих автоматов по зашумленным сценариям работы был экспериментально сравнен с точным алгоритмом поиска с возвратом и метаэвристическим методом *MuACO* [26]. Для сравнения каждый метод был запущен 3000 раз. Результаты экспериментов показали, что предложенный метод работает значительно быстрее переборного алгоритма при росте размерности задачи и не уступает при этом *MuACO*.
6. Результаты, полученные в настоящей главе, были доложены на международных конференциях Information Control Problems in Manufacturing – INCOM (2012), International Conference on Machine Learning and Applications – ICMLA (2011, 2014), а также на Всероссийском совещании по проблемам управления (2014), Всероссийской научной конференции по проблемам информатики СПИСОК (2011-2014), Всероссийском конгрессе молодых ученых (2011-2013). Были опубликованы статьи в рецензируемых изданиях [87–90]. В работе [89] также описывается алгоритм совместного применения предложенного метода генерации конечных управляющих автоматов и метода *MuACO*.

ГЛАВА 5. ВНЕДРЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ

Настоящая глава посвящена внедрению результатов диссертационной работы. Описывается внедрение разработанных методов генерации в программное средство *DFASAT*.

5.1. Внедрение разработанных предикатов нарушения симметрии в средство *DFASAT*

Опишем первое внедрение результатов диссертационной работы. Как уже отмечалось ранее, лучшим известным методом решения задачи построения детерминированных конечных автоматов по незашумленным обучающим словарям являлся *DFASAT* [44] – метод, разработанный М. Хеуле (Техасский университет в Остине, США) и С. Вервером (Делфтский технический университет, Нидерланды) и реализованный в одноименном программном средстве. После выступления диссертанта на конференции *LATA*-2015 (Ницца, Франция) и публикации работы [91] был установлен контакт с упомянутыми исследователями.

С. Вервер высоко оценил предложенные методы и реализовал в коде *DFASAT* (на языке *C++*) предлагаемые предикаты нарушения симметрии. Также он провел эксперимент по времени работы средства до и после внедрения предикатов на примере, взятом из данных соревнования *STAMINA*. Запуск до внедрения осуществлен командой

```
./dfasat staminadata/8_training.txt.dat -t=50 -y=20
-h=4 -d=80 -b=1000 -e=10 -x=0 "./lingeling",
```

выполнение которой заняло 885 с. После внедрения запуск осуществлен командой

```
./dfasat staminadata/8_training.txt.dat -t=50 -y=20
-h=4 -d=80 -b=1000 -e=10 "./lingeling",
```

выполнение которой заняло 25 с.

В приложении 2 приведен акт, подтверждающий внедрение и использование результатов диссертационной работы в *DFASAT*.

5.2. Внедрение в образовательный процесс

Внедрение результатов диссертации в образовательный процесс произведено на кафедре «Компьютерные технологии» Университета ИТМО. Результаты использовались в рамках курса «Теория автоматов и программирование», а также при руководстве двумя бакалаврскими и выполнении двух магистерских выпускных квалификационных работ студентов кафедры:

1. Закирзянов И. Т. Разработка предикатов нарушения симметрии для построения детерминированных конечных автоматов с помощью сведения к задаче о выполнимости. Бакалаврская работа, 2015.
2. Мельник М. В. Применение методов решения задачи о выполнимости булевой формулы для построения минимальной филогенетической сети. Бакалаврская работа, 2015.
3. Агапова А. И. Поиск оптимальной конфигурации гидрофобно-полярной модели протеина при помощи методов решения задачи выполнимости булевой формулы. Магистерская диссертация, 2014.
4. Панченко Е. В. Построение управляющих конечных автоматов по сценариям работы и темпоральным свойствам на основе методов решения задачи о выполнимости булевой формулы. Магистерская диссертация, 2013.

Выводы по главе 5

1. Осуществлено внедрение разработанных предикатов нарушения симметрии в средство *DFASAT* [44], получен акт внедрения.
2. Результаты диссертации использовались в учебном процессе кафедры «Компьютерные технологии» Университета ИТМО.

ЗАКЛЮЧЕНИЕ

В результате диссертационного исследования получены следующие результаты:

1. Теоретическое доказательство NP-трудности задачи генерации управляющих конечных автоматов по сценариям работы.
2. Точные методы генерации детерминированных конечных автоматов по безошибочным и зашумленным обучающим словарям. Методы основаны на сведении к задаче выполнимости булевой формулы. Создано программное средство *DFAInducer* с открытым кодом, реализующее разработанные методы. В случае безошибочных данных время работы предложенного метода меньше времени работы существующих. Для генерации автоматов по зашумленным данным ранее точных методов не предлагалось.
3. Точные методы генерации управляющих конечных автоматов по безошибочным и зашумленным сценариям работы. В основе методов лежит сведение к задаче удовлетворения ограничений. Методы реализованы в программном средстве *EFSMTools* с открытым кодом. Экспериментальные исследования показали преимущество по сравнению с поиском с возвратом.
4. Внедрение разработанных методов, осуществленное при разработке программного средства *DFASAT* и выполненное после публикации работы диссертанта [91]. Имеется акт внедрения из Делфтского технического университета. Также результаты диссертации использовались в учебном процессе кафедры «Компьютерные технологии» Университета ИТМО.

Полученные результаты рекомендуется использовать для решения практических задач генерации конечных автоматов. Перспективы дальнейшей разработки темы включают в себя развитие предложенных методов, их модификацию для применения к другим автоматным моделям.

СПИСОК ИСТОЧНИКОВ

ПЕЧАТНЫЕ ИЗДАНИЯ НА РУССКОМ ЯЗЫКЕ

1. Применение эволюционного программирования на основе обучающих примеров для генерации конечных автоматов, управляющих объектами со сложным поведением / *Александров А. В., Казаков С. В., Сергушичев А. А., Царев Ф. Н., Шалыто А. А.* // Известия РАН. Теория и системы управления. – 2013. – № 3. – С. 85-100.
2. Генерация управляющих конечных автоматов по обучающим примерам на основе муравьиного алгоритма / *Бужинский И. П.*, *Ульянцев В. И.*, *Чивилихин Д. С.*, *Шалыто А. А.* // Известия РАН. Теория и системы управления. – 2014. – № 2. – С. 111-121.
3. *Егоров К. В.* Генерация управляющих автоматов на основе генетического программирования и верификации. – 2013. – Диссертация на соискание ученой степени кандидата технических наук. НИУ ИТМО.
4. Алгоритмы. Построение и анализ / *Кормен Т.*, *Лейзерсон Ч.*, *Ривест Р.*, *Штайн К.* – М.: Вильямс, 2013. – 1328 с.
5. *Кочетов Ю. А.* Вероятностные методы локального поиска для задач дискретной оптимизации // Дискретная математика и ее приложения. Сборник лекций молодежных и научных школ по дискретной математике и ее приложениям. – М: МГУ, 2001. – С. 87-117.
6. *Левин Л. А.* Универсальные задачи перебора // Проблемы передачи информации. – 1973. – Т. 9, вып. 3. – С. 115, 116.
7. *Поликарпова Н.И., Точилин В.Н., Шалыто А.А.* Метод сокращенных таблиц для генерации автоматов с большим числом входных переменных на основе генетического программирования // Известия РАН. Теория и системы управления. – 2010. – № 2. – С. 100–117.

8. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. – СПб: Питер, 2010. – 176 с.
9. *Отпущенников И. В., Семенов А. А.* Технология трансляции комбинаторных проблем в булевы уравнения // Прикладная дискретная математика. – 2011. – № 1. – С. 96–115.
10. *Скиена С.* Алгоритмы. Руководство по разработке. – СПб: БХВ-Петербург, 2011. – 720 с.
11. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. – М.: Вильямс, 2008. – 528 с.
12. *Царев Ф. Н.* Методы построения конечных автоматов на основе эволюционных алгоритмов. – 2012. – Диссертация на соискание ученой степени кандидата технических наук. НИУ ИТМО.
13. *Царев Ф. Н., Егоров К. В., Шалыто А. А.* Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе обучающих примеров и спецификации // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. – 2010. – № 5. – С. 81 – 86.
14. *Щербина О. А.* Удовлетворение ограничений и программирование в ограничениях // Интеллектуальные системы. – 2011. – Т. 15, № 1-4. – С. 53-170.

Печатные издания на английском языке

15. Synthesizing Finite-state Protocols from Scenarios and Requirements / *Alur R., Martin M., Raghothaman M., Stergiou C., Tripakis S., Udupa A.* // Hardware and Software: Verification and Testing. – Springer, 2014. – P. 75-91. – (Lecture Notes in Computer Science; 8855).
16. *Amiri E., Skvortsov E.* Pushing Random Walk Beyond Golden Ratio // CSR 2007. – Springer, 2007. – P. 44-55. – (Lecture Notes in Computer Science; 4649).

17. Handbook of Evolutionary Computation / Ed. by *T. Back, D. B. Fogel, Z. Michalewicz*. – Bristol, UK: IOP Publishing Ltd., 1997. – 1130 p.
18. Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions / Ed. by *Balint A., Belov A., Heule M. J. H., Järvisalo M.* – University of Helsinki, Helsinki, 2013. – (Department of Computer Science Series of Publications B; vol. B-2013-1).
19. *Bayardo R. Jr., Schrag R.* Using CSP Look-Back Techniques to Solve Real-World SAT Instances // Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence. – AAAI Press, 1997 – P. 203-208.
20. Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions / Ed. by *Belov A., Diepold D., Heule M. J. H., Järvisalo M.* – University of Helsinki, Helsinki, 2014. – (Department of Computer Science Series of Publications B; vol. B-2014-2).
21. *Biere A.* Yet another Local Search Solver and Lingeling and Friends Entering the SAT Competition 2014 // Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions. – University of Helsinki, Helsinki, 2014. – P. 39-40. – (Department of Computer Science Series of Publications B; vol. B-2014-2).
22. Handbook of Satisfiability / Ed. by *Biere A., Heule V., van Maaren H., Walsh T.* – IOS Press, 2009. – 980 p.
23. *Bitner J. R., Reingold E. M.* Backtrack Programming Techniques // Communications of the ACM. – 1975. – Vol. 18, no. 11. – P. 651-656.
24. *Blum C., Roli A.* Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison // ACM Computing Surveys (CSUR). – 2003. – Vol. 35, no. 3. – P. 268-308.
25. Learning Extended Finite State Machines / *Cassel S., Howar F., Jonsson B., Steffen B.* // Software Engineering and Formal Methods.

– Springer, 2014. – P. 250-264. – (Lecture Notes in Computer Science; 8702).

26. *Chivilikhin D., Ulyantsev V.* Learning Finite-State Machines with Ant Colony Optimization // Swarm Intelligence. – Springer, 2012. – P. 268-275. – (Lecture Notes in Computer Science; 7461).
27. *Chongstitvatana P., Aporntewan C.* Improving Correctness of Finite-State Machine Synthesis from Multiple Partial input/output Sequences // Proceedings of the First NASA/DoD Workshop on Evolvable Hardware. – IEEE, 1999. – P. 262-266.
28. *Cicchello O., Kremer S. C.* Beyond EDSM // Grammatical Inference: Algorithms and Applications. – Springer, 2002. – P. 37-48. – (Lecture Notes in Computer Science; 2484).
29. Goldreich's One-Way Function Candidate and Myopic Backtracking Algorithms / *Cook J., Etesami O., Miller R., Trevisan L.* // Theory of Cryptography. – Springer, 2009. – P. 521-538. – (Lecture Notes in Computer Science; 5444).
30. *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.
31. *Davis M., Putnam H.* A Computing Procedure for Quantification Theory. // Journal of the ACM. – 1960. – Vol. 7, no. 3. – P. 201-215.
32. *Davis M., Logemann G., Loveland D.* A Machine Program for Theorem-Proving // Communications of the ACM. – 1962. – Vol. 5, no. 7. – P. 394-397.
33. *De la Higuera C.* A Bibliographical Study of Grammatical Inference // Pattern recognition. – 2005. – Vol. 38, no. 9. – P. 1332-1348.
34. *De Moura L., Bjørner N.* Z3: An Efficient SMT Solver // Tools and Algorithms for the Construction and Analysis of Systems. – Springer, 2008. – P. 337-340. – (Lecture Notes in Computer Science; 4963).

35. *Dechter R.* Constraint Processing. – Morgan Kaufmann, 2003. – 450 p.
36. Estimating the Feasibility of Transition Paths in Extended Finite State Machines / *Derderian K., Hierons R. M., Harman M., Guo Q.* // Automated Software Engineering. – 2010. – Vol. 17, no. 1. – P. 33-56.
37. *Eén N., Biere A.* Effective Preprocessing in SAT Through Variable and Clause Elimination // Theory and Applications of Satisfiability Testing. – Springer, 2005. – P. 61-75. – (Lecture Notes in Computer Science; 3569).
38. *Eén N., Sörensson N.* An Extensible SAT-solver // Theory and Applications of Satisfiability Testing. – Springer, 2004. – P. 502-518. – (Lecture Notes in Computer Science; 2919).
39. Study of Discrete Automaton Models of Gene Networks of Nonregular Structure Using Symbolic Calculations / *Evdokimov A. A., Kochemazov S. E., Otpushchennikov I. V., Semenov A. A.* // Journal of Applied and Industrial Mathematics. – 2014. – Vol. 8, no. 3. – P. 307–316.
40. *Florêncio C., Verwer S.* Regular Inference as Vertex Coloring // Theoretical Computer Science. – 2014. – Vol. 558. – P. 18-34.
41. *Gold E. M.* Complexity of Automaton Identification from Given Data // Information and Control. – 1978. – Vol. 37. – P. 302-320.
42. *Gomez J.* An Incremental-Evolutionary Approach For Learning Finite Automata // Proceedings of the 2006 IEEE Congress on Evolutionary Computation. – 2006. – P. 362-369.
43. *Hammerman N., Goldbeg R.* Algorithms to Improve the Convergence of a Genetic Algorithm with a Finite State Machine Genome // Practical Handbook of Genetic Algorithms: Complex Coding Systems, Vol. 3. – CRC press, 2010. – P. 119-238.
44. *Heule M., Verwer S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications.

– Springer, 2010. – P. 66-79. – (Lecture Notes in Computer Science; 6339).

45. *Hirsch E. A.* SAT Local Search Algorithms: Worst-Case Study // Journal of Automated Reasoning. – 2000. – Vol. 24, no. 1 – P. 127–143.
46. *Hölldobler S, Nguyen V.* An Efficient Encoding of the At-Most-One Constraint. – Technische Universität Dresden, 2013. – 17 p. – (Technical report; KRR Group 2013-04).
47. *Itsykson D., Sokolov D.* Lower bounds for myopic DPLL algorithms with a cut heuristic // Algorithms and Computation. – Springer, 2011. – P. 464-473. – (Lecture Notes in Computer Science; 7074).
48. *Jussien N., Rochart G., Lorca X.* Choco: an Open Source Java Constraint Programming Library // CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08). – 2008. – P. 1-10.
49. *Kalaji A. S., Hierons R. M., Swift S.* An Integrated Search-Based Approach for Automatic Testing from Extended Finite State Machine (EFSM) Models // Information and Software Technology. – 2011. – Vol. 53, no. 12. – P. 1297-1318.
50. *Kochemazov S., Semenov A.* Using Synchronous Boolean Networks to Model Several Phenomena of Collective Behavior. // PLoS ONE. – 2014. – Vol. 9, no. 12. – e115156.
51. *Kumar V.* Algorithms for Constraint-Satisfaction Problems: a Survey // AI magazine. – AAAI, 1992. – Vol. 13, no. 1. – P. 32-44.
52. *Lambeau B., Damas C., Dupont P.* State-Merging DFA Induction Algorithms with Mandatory Merge Constraints // Grammatical Inference: Algorithms and Applications. – Springer, 2008. – P. 139-153. – (Lecture Notes in Computer Science; 5278).
53. *Lang K.* Random DFA's can be Approximately Learned from Sparse Uniform Examples // Proceedings of the Fifth ACM Workshop on Computational Learning Theory. – ACM Press, 1992. – P. 45-52.

54. *Lang K., Pearlmutter B., Price R.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm // Grammatical Inference. – Springer, 1998. – P. 1-12. – (Lecture Notes in Computer Science; 1433).
55. *Lucas S., Reynolds J.* Learning Deterministic Finite Automata with a Smart State Labeling Algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. – IEEE, 2005. – Vol. 27, no. 7. – P. 1063-1074.
56. *Lucas S., Reynolds J.* Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. – 2007. – Vol. 11, no. 3. – P. 308-325.
57. *Marques-Silva J. P., Sakallah K. A.* GRASP: A New Search Algorithm for Satisfiability // Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design. – IEEE, 1996. – P. 220-227.
58. Minizinc: Towards a Standard CP Modelling Language / *Nethercote N., Stuckey P. J., Becket R., Brand S., Duck G. J., Tack G.* // Principles and Practice of Constraint Programming (CP 2007). – Springer, 2007. – P. 529-543. – (Lecture Notes in Computer Science; 4741).
59. *Nikolenko S. I.* Hard Satisfiable Instances for DPLL-type Algorithms // Journal of Mathematical Sciences. – 2005. – Vol. 126, no. 3. – P. 1205-1209.
60. *Oncina J., Garcia P.* Inferring Regular Languages in Polynomial Update Time // Pattern Recognition and Image Analysys, volume 1 of Series in Machine Perception and Artificial Intelligence. – World Scientific, 1992. – P. 49-61.
61. *Pitt L., Warmuth M. K.* The Minimum Consistent DFA Problem Cannot be Approximated Within any Polynomial // Journal of the ACM. – 1993. – Vol. 40, no. 1. – P. 95-142.

62. *Sakakibara Y.* Grammatical Inference in Bioinformatics // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2005. – Vol. 27, no. 7. – P. 1051-1062.
63. *Seto K.*, *Tamaki S.* A satisfiability algorithm and average-case hardness for formulas over the full binary basis // Proceedings of the 27th Annual Conference on Computational Complexity. – IEEE, 2012. – P. 107-116.
64. *Skvortsov E.*, *Tipikin E.* Experimental study of the shortest reset word of random automata // Implementation and Application of Automata. – Springer, 2011. – P. 290-298. – (Lecture Notes in Computer Science; 6807).
65. The MiniZinc Challenge 2008-2013 / *Stuckey P. J.*, *Feydy T.*, *Schutt A.*, *Tack G.*, *Fischer J.* // AI Magazine. – 2014. – Vol. 35, no. 2. – P. 55-60.
66. Using Behaviour Inference to Optimise Regression Test Sets / *Taylor R.*, *Hall M.*, *Bogdanov K.*, *Derrick J.* // Testing Software and Systems. – Springer, 2012. – P. 184-199. – (Lecture Notes in Computer Science; 7641).
67. *Tomita M.* Dynamic construction of finite automata from examples using hill climbing // Proceedings of the Fourth Annual Cognitive Science Conference. – 1982. – P. 105-108.
68. *Trakhtenbrot B.*, *Barzdin Y.* Finite Automata: Behavior and Synthesis. – North-Holland, 1973. – 328 p.
69. *Vyatkin V.* IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design, Second Edition. – ISA, 2012. – 259 p.
70. *Walsh T.* SAT *v* CSP // Principles and Practice of Constraint Programming – CP 2000. – Springer, 2000. – P. 441-456. – (Lecture Notes in Computer Science; 1894).

РЕСУРСЫ СЕТИ ИНТЕРНЕТ

71. Class PriorityQueue (Java Platform SE 7). – URL: http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html.

72. DFA-Inductor – GitHub Repository. – URL.: https://github.com/ctlab/DFA-Inductor.
73. DFAs: Languages and Learning. – URL.: http://abbadingo.cs.nuim.ie/dfa.html.
74. Digital CAS Protocols Installation and Developer's Manual: Responding to Inbound Calls. – URL: http://www.nmscommunications.com/manuals/6206-14/C-09.htm.
75. Tools for Extended Finite-state Machine Induction and Testing – GitHub Repository. – URL.: https://github.com/ulyantsev/EFSM-tools.
76. GeCode – An open, free, efficient constraint solving toolkit. – URL: http://www.gecode.org/.
77. Google Optimization Tools. – URL: https://developers.google.com/optimization/.
78. Graphviz – Graph Visualization Software. – URL: http://graphviz.org/.
79. JaCoP: Java Constraint Programming solver. – URL: http://jacop.osolpro.com/.
80. *Kjellerstrand H.* My MiniZinc page. – URL: http://www.hakank.org/minizinc/.
81. Learning DFS from Noisy Samples: a Contest for GECCO 2004. – URL: – http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html.
82. Opturion CPX: Discrete Optimizer. – URL. http://www.opturion.com/.
83. Satisfiability Suggested Format. – URL: http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf.
84. *Soos M.* Limits of SAT Solvers in Cryptography. – URL: http://www.msoos.org/wordpress/wp-content/uploads/2011/07/CASED_pres_Soos.pdf.
85. *Soos M.* CryptoMiniSat 2. – URL: http://www.msoos.org/cryptominisat2/.
86. The International SAT Competitions Web Page. – URL: http://www.satcompetition.org/.

Публикации автора

Статьи в рецензируемых изданиях из перечней ВАК или Scopus

87. *Панченко Е. В.*, *Ульянцев В. И.* Применение методов решения задачи о выполнимости квантифицированной булевой функции для построения управляющих конечных автоматов по сценариям работы и темпоральным свойствам // Научно-технический вестник информационных технологий, механики и оптики. – 2013. – № 4(86). – С. 151-153. – 0,188 п. л. / 0,1 п. л.
88. *Ульянцев В. И.*, *Царев Ф. Н.* Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы // Научно-технический вестник информационных технологий, механики и оптики. – 2012. – № 1(77). – С. 96-100. – 0,313 п. л. / 0,16 п. л.
89. *Chivilikhin D.*, *Ulyantsev V.*, *Shalyto A.* Combining Exact And Metaheuristic Techniques For Learning Extended Finite-State Machines From Test Scenarios and Temporal Properties // Proceedings of the 13th International Conference on Machine Learning and Applications (ICMLA'14). – 2014. – P. 350-355. – 0,375 п. л. / 0,15 п. л.
90. *Ulyantsev V.*, *Tsarev F.* Extended Finite-State Machine Induction using SAT-Solver / Proceedings of the Tenth International Conference on Machine Learning and Applications. – IEEE, 2011. – Vol. 2. – P. 346-349. – 0,25 п. л. / 0,125 п. л.
91. *Ulyantsev V.*, *Zakirzyanov I.*, *Shalyto A.* BFS-Based Symmetry Breaking Predicates for DFA Identification / Language and Automata Theory and Applications. – Springer, 2015. – P. 611-622. – (Lecture Notes in Computer Science; 8977). – 0,375 п. л. / 0,2 п. л.
92. *Ulyantsev V.*, *Tsarev F.* Extended Finite-State Machine Induction using SAT-Solver / Proceedings of the 14th IFAC Symposium «Information

Control Problems in Manufacturing – INCOM'12». – IFAC, 2012. – P. 512-517. – 0,375 п. л. / 0,25 п. л.

Другие публикации по теме

93. *Ведерников Н. В.*, *Демьянюк В. Ю.*, *Кротков П. А.*, *Ульянцев В. И.*, *Шалыто А. А.* Применение методов машинного обучения для автоматизированного построения управляющих автоматов в высокоуровневых средствах проектирования систем // Труды XII Всероссийского совещания по проблемам управления ВСПУ-2014. – М.: Институт проблем управления им. В.А. Трапезникова РАН, 2014. – С. 3159-3166. – 0,5 п. л. / 0,2 п. л.
94. *Ульянцев В. И.* Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы // Аннотированный сборник научно-исследовательских выпускных квалификационных работ бакалавров и специалистов НИУ ИТМО. – СПб.: НИУ ИТМО, 2011. – С. 35-38. – 0,25 п. л.
95. *Ульянцев В. И.* Применение методов решения задачи удовлетворения ограничениям для построения управляющих конечных автоматов по сценариям работы // Сборник тезисов докладов конгресса молодых ученых, Выпуск 1. Труды молодых ученых. – СПб.: НИУ ИТМО, 2012. – С. 230-231. – 0,125 п. л.
96. *Ульянцев В. И.* Метод построения управляющих конечных автоматов по сценариям работы на основе решения задачи удовлетворения ограничений // Научные работы участников конкурса «Молодые ученые НИУ ИТМО» 2012 года. – СПб.: НИУ ИТМО, 2013. – С. 256-260. – 0,313 п. л.
97. *Ульянцев В. И.*, *Царев Ф. Н.*, *Шалыто А. А.* Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы // Сборник

докладов XIV Международной конференции по мягким вычислениям и измерениям (SCM'2011). – 2011. – Т. 2. – С. 69-75. – 0,438 п. л. / 0,2 п. л.

98. *Ульянцев В. И.*, *Царев Ф. Н.* Применение методов решения задачи удовлетворения ограничениям для построения управляющих конечных автоматов по сценариям работы // Список-2012: материалы всероссийской научной конференции по проблемам информатики. – СПб.: ВВМ, 2012. – С. 444-445. – 0,125 п. л. / 0,07 п. л.
99. *Ульянцев В. И.*, *Вихарев А. К.*, *Шалыто А. А.* Применение алгоритма EDSM для построения управляющих конечных автоматов по сценариям работы // Сборник докладов XIV Международной конференции по мягким вычислениям и измерениям SCM'2011. – 2011. Т. 2, с. 76-80. – 0,313 п. л. / 0,16 п. л.
100. *Ульянцев В. И.*, *Царев Ф. Н.* Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы // Список-2011: материалы межвузовской научной конференции по проблемам информатики. – СПб.: ВВМ, 2011. – С. 356-358. – 0,188 п. л. / 0,1 п. л.
101. *Ульянцев В. И.* Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы // Сборник тезисов докладов конференции молодых ученых, Выпуск 1. Труды молодых ученых. – СПб.: СПбГУ ИТМО, 2011. – С. 247-248. – 0,125 п. л.

ПРИЛОЖЕНИЕ 1. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2012616462

Программное средство для построения графа совместимости вершин дерева сценариев работы программы

Правообладатель(ли): *федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)*

Автор(ы): *Ульянцев Владимир Игоревич, Царев Федор Николаевич (RU)*

Заявка № 2012614587

Дата поступления 5 июня 2012 г.

Зарегистрировано в Реестре программ для ЭВМ *18 июля 2012 г.*

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2012660438

Программное средство для построения КНФ-формулы по графу совместимости вершин дерева сценариев работы программы

Правообладатель(ли): ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Автор(ы): ***Ульянцев Владимир Игоревич (RU)***

Заявка № **2012618294**

Дата поступления **2 октября 2012 г.**

Зарегистрировано в Реестре программ для ЭВМ

20 ноября 2012 г.

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2013619840

«Программный комплекс для построения и тестирования управляющих конечных автоматов»

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Автор: ***Ульянцев Владимир Игоревич (RU)***

Заявка № **2013614906**
Дата поступления **14 июня 2013 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***17 октября 2013 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2015619224

Программное средство преобразования полученных методами машинного обучения управляющих автоматов в формат MATLAB/Stateflow

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (RU)***

Авторы: ***Русин Никита Сергеевич (RU), Ульянцев Владимир Игоревич (RU), Ведерников Николай Викторович (RU), Демьянюк Виталий Юрьевич (RU), Кротков Павел Андреевич (RU), Шалыто Анатолий Абрамович (RU)***

Заявка № **2015616347**

Дата поступления **03 июля 2015 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***27 августа 2015 г.***

Заместитель руководителя Федеральной службы по интеллектуальной собственности

Л.Л. Кирий

ПРИЛОЖЕНИЕ 2. АКТ, ПОДТВЕРЖДАЮЩИЙ ВНЕДРЕНИЕ И ИСПОЛЬЗОВАНИЕ РЕЗУЛЬТАТОВ ДИССЕРТАЦИОННОЙ РАБОТЫ В ПРОГРАММНОЕ СРЕДСТВО *DFASAT*

TU Delft

Statement of the adoption of results of Vladimir Ulyantsev's candidate dissertation

This statement is to certify that some results of Vladimir Ulyantsev's candidate dissertation on inferring finite-state machines with SAT and CSP solvers have been used in a tool for inferring deterministic finite automata DFASAT. Namely, we used the BFS-based symmetry-breaking predicates, which significantly speeds up the algorithm on many problem instances.

03-09-2015

Dr. Sicco Verwer
Assistant Professor in Cyber Security
Delft University of Technology