Национальный исследовательский университет ИТМО

(Университет ИТМО)

На правах рукописи

Закирзянов Илья Тимурович

Методы генерации детерминированных конечных автоматов с использованием сокращения пространства поиска при решении задачи выполнимости

Диссертация на соискание учёной степени

кандидата технических наук

Санкт-Петербург 2020

Национальный исследовательский университет ИТМО

(Университет ИТМО)

На правах рукописи

Закирзянов Илья Тимурович

Методы генерации детерминированных конечных автоматов с использованием сокращения пространства поиска при решении задачи выполнимости

Специальность 05.13.17

«Теоретические основы информатики»

Диссертация на соискание учёной степени

кандидата технических наук

Научный руководитель:
кандидат технических наук
Ульянцев Владимир Игоревич

Санкт-Петербург 2020

Диссертация подготовлена в: федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО».

Научный руководитель: кандидат технических наук
Ульянцев Владимир Игоревич

Официальные оппоненты: Уткин Лев Владимирович, доктор технических наук, профессор, ФГАОУ ВО «Санкт-Петербургский политехнический университет Петра Великого», заведующий кафедрой кафедра «Телематика» (при ЦНИИ РТК)

Сандип Патил, Кандидат технических наук, Технологический университет Лулео, Доцент, Кафедра информатики, электротехники и космических технологий

Защита состоится 16.12.2020 г. в 16:00 часов на заседании диссертационного совета 02.18.25 Университета ИТМО. Санкт-Петербург, Кронверкский пр. 49, Лит. А, ауд. 365.

С диссертацией можно ознакомиться в библиотеке Университета ИТМО по адресу: 197101, Санкт-Петербург, Кронверкский пр., д.49 и на сайте https://dissovet.itmo.ru.

Ученый секретарь диссертационного совета 02.18.00 Университета ИТМО, кандидат технических наук, доцент, Муромцев Дмитрий Ильич.

ITMO University

As a manuscript

Zakirzyanov Ilya Timurovich

Search space reduction techniques for deterministic finite automata inference with Boolean satisfiability problem solving

Speciality 05.13.17
Theoretical foundations of computer science

Academic dissertation candidate of engineering

Supervisor:
PhD
Ulyantsev Vladimir Igorevich

Saint-Petersburg 2020

The research was carried out at: ITMO University.

Supervisor:	PhD Ulyantsev Vladimir Igorevich
Official opponents:	Utkin Lev V., doctor of engineering, Federal State Autonomous Educational Institution of Higher Education «Peter the Great St. Petersburg Polytechnic University», Head of the Department Department «Telematics»
	Sandeep Patil, PhD, Lulea University of Technology, Assistant Professor, Department of Computer Science, Electrical and Space Engineering

The defense will be held on 16.12.2020 at 16:00 at the meeting of the ITMO University Dissertation Council 02.18.25.

The thesis is available in the Library of ITMO University, 49 Kronversky pr., Saint-Petersburg, Russia and on https://dissovet.itmo.ru website.

Science Secretary of the ITMO University Dissertation Council 02.18.00, PhD of engineering, Mouromtsev Dmitry I.

Содержание

Реферат

Общая характеристика работы

Актуальность темы. Конечные автоматы являются одним из основополагающих концептов в дискретной математике, информатике и программировании. Помимо непосредственной роли в теории формальных языков и при проектировании вычислительных машин, различные конечно-автоматные модели используются в наше время на практике при разработке и анализе программного обеспечения.

Например, конечные автоматы используются при проектировании моделей программного обеспечения контроллеров и ответственных систем[1], для спецификации протоколов взаимодействия[2], для моделирования поведения высокоуровневых систем[3]. Основными преимуществами использования автоматов являются наглядность и относительная простота модели для человеческого восприятия, а также возможность автоматизированной верификации формальных свойств модели (model checking)[4].

Во многих случаях проектирование автоматной модели осуществляется разработчиком вручную, например, в парадигме автоматного программирования[5]. При решении других задач подразумевается автоматизированная генерация конечного автомата — извлечение модели из существующих данных или систем.

---

[1] *Поликарпова*, *Н. И.*, *Шалыто*, *А. А.* Автоматное программирование. СПБ: Питер, 2010. 176 с.; *Patil*, *S.*, *Dubinin*, *V.*, *Vyatkin*, *V.* Formal Modelling and Verification of IEC61499 Function Blocks with Abstract State Machines and SMV — Execution Semantics // SETTA. Vol. 9409. Springer, 2015. P. 300–315. (Lecture Notes in Computer Science).

[2] *Jongmans*, *S.-S. T. Q.*, *Halle*, *S.*, *Arbab*, *F.* Automata-Based Optimization of Interaction Protocols for Scalable Multicore Platforms // COORDINATION. Vol. 8459. Springer, 2014. P. 65–82. (Lecture Notes in Computer Science).

[3] *Heule*, *M.*, *Verwer*, *S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 825–856; *Wagner*, *F.*, *Schmuki*, *R.*, *Wagner*, *T.*, *Wolstenholme*, *P.* Modeling Software with Finite State Machines: A Practical Approach. CRC Press, 2006. 392 p.

[4] *Clarke Jr*, *E. M.*, *Grumberg*, *O.*, *Kroening*, *D.*, *Peled*, *D.*, *Veith*, *H.* Model checking. Second edition. MIT press, 2018. 424 p. (Cyber Physical Systems Series).

[5] *Поликарпова*, *Н. И.*, *Шалыто*, *А. А.* Автоматное программирование. СПБ: Питер, 2010. 176 с.

Среди практических примеров использования методов генерации конечных автоматов можно привести: построение моделей программного обеспечения контроллеров по составленным вручную или снятым автоматизированно примерам поведения[6], построение формальных моделей объектов управления[7], анализ моделей поведения сложных программных систем[8] и сетевых протоколов[9], и другие. Активное изучение и разработка алгоритмов генерации автоматов начиналась ведется с 1970-х годов. В 1978 году было доказано, что задача генерации детерминированного конечного автомата (ДКА) с заданным (минимальным) числом состояний по заданным примерам поведения является NP-полной[10]. Данный теоретический результат подчеркивает сложность задачи генерации автоматов в общем случае, актуализируя разработку применимых на практике алгоритмов.

Впоследствии было предложено достаточно много как эвристических, так и метаэвристических алгоритмов генерации ДКА по заданным примерам поведения, и к настоящему моменту они образуют собой целое семейство алгоритмов в дискретной математике. За последние годы была разработана группа методов, сводящих задачу точной генерации (с минимально возможным числом состояний) искомого автомата к другим NP-полным задачам. При этом наиболее эффективные подходы в настоящий момент основаны на сведении к задаче выполнимости.

Задача выполнимости булевых формул (Boolean Satisfiability — SAT) заключается в определении существования выполняющей подстановки для заданной

[6] *Chivilikhin*, *D.*, *Buzhinsky*, *I.*, *Ulyantsev*, *V.*, *Stankevich*, *A.*, *Shalyto*, *A.*, *Vyatkin*, *V.* Counterexample-guided inference of controller logic from execution traces and temporal formulas // ETFA. IEEE, 2018. P. 91–98.

[7] *Buzhinsky*, *I.*, *Vyatkin*, *V.* Modular plant model synthesis from behavior traces and temporal properties // ETFA. IEEE, 2017. P. 1–7; *Buzhinsky*, *I.*, *Vyatkin*, *V.* Automatic Inference of Finite-State Plant Models From Traces and Temporal Properties // IEEE Transactions on Industrial Informatics. 2017. Vol. 13, no. 4. P. 1521–1530.

[8] *Heule*, *M.*, *Verwer*, *S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 825–856; *Cook*, *J. E.*, *Wolf*, *A. L.* Discovering Models of Software Processes from Event-Based Data // ACM Transactions on Software Engineering Methodology. 1998. Vol. 7, no. 3. P. 215–249; *Bertolino*, *A.*, *Inverardi*, *P.*, *Pelliccione*, *P.*, *Tivoli*, *M.* Automatic synthesis of behavior protocols for composable web-services // ESEC/SIGSOFT FSE. ACM, 2009. P. 141–150.

[9] *Sivakorn*, *S.*, *Argyros*, *G.*, *Pei*, *K.*, *Keromytis*, *A. D.*, *Jana*, *S.* HVLearn: Automated Black-Box Analysis of Hostname Verification in SSL/TLS Implementations // IEEE Symposium on Security and Privacy. IEEE Computer Society, 2017. P. 521–538.

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

булевой формулы, представленной в конъюнктивной нормальной форме — в виде конъюнкции дизъюнктов. Согласно теореме Кука-Левина 1971 года, задача выполнимости является NP-полной Данный факт актуализировал и стимулировал разработку применимых на практике программных средств для решения задачи выполнимости.

Методы решения SAT разрабатывались еще до введения теоретических оценок сложности: в 1962 году был предложен алгоритм Дэвиса-Патнема-Логемана-Лавленда (Davis-Putnam-Logemann-Loveland — DPLL)[11] — полный алгоритм поиска с возвратом выполняющей подстановки, использующий эвристики распространения переменной и исключения «чистых» переменных для ускорения поиска. Данный алгоритм, в общем случае за экспоненциальное время от числа переменных, перебирает все пространство поиска подстановок и останавливается, если была найдена выполняющая подстановка или если перебор завершил работу — выполняющей подстановки не существует.

В середине 1990-х на основе алгоритма DPLL был разработан алгоритм CDCL[12] (conflict-driven clause learning — «управляемое конфликтами обучение дизъюнктов»), сохраняющий и использующий в дальнейшем выведенные дизъюнкты в ходе анализа конфликтов, возникающих при работе DPLL. Такие дизъюнкты в дальнейшем позволяют намного раньше принимать решение о невыполнимости формулы с текущими допущениями и переходить к рассмотрению следующих.

Именно на алгоритме CDCL основаны все современные программные средства для решения SAT. Каждый год проводятся соревнования программных средств решения SAT[13], что отличает задачу выполнимости от всех остальных NP-трудных задач. За счет этого для других задач актуально разрабатывать методы,

[11] *Davis*, *M.*, *Logemann*, *G.*, *Loveland*, *D. W.* A machine program for theorem-proving // Communications of the ACM. 1962. Vol. 5, no. 7. P. 394–397.

[12] *Marques-Silva*, *J. P.*, *Sakallah*, *K. A.* GRASP — a new search algorithm for satisfiability // ICCAD. IEEE, 1996. P. 220–227.

[13] The International SAT Competition Web Page. URL: `http://www.satcompetition.org/` (дата обр. 15.10.2020); SAT Competition 2020. URL: `https://satcompetition.github.io/2020` (дата обр. 15.10.2020).

основанные на сведении к SAT: ускорения метода можно добиться без изменения исходного кода, будет достаточно лишь заменить программное средство для решения SAT, выдающее найденную подстановку или доказывающее ее отсутствие.

Однако, в методологии сведения поставленной задачи к SAT фундаментальный характер носит не столько используемое программное средство, сколько способ трансляции экземпляра задачи в булеву формулу, для которой в дальнейшем будет запущен поиск подстановки. Так, в последние годы для задачи точной генерации ДКА по заданным примерам поведения было предложено базовое сведение[14] и несколько его модификаций[15]. Данные модификации предлагали использование предикатов нарушения симметрии — дополнительно введенных в исходную формулу дизъюнктов, задающих специфичное для задачи сокращение пространства поиска.

Данные техники сокращения пространства поиска оказались очень эффективны на практике, позволив решать задачи генерации автоматов большего размера. Однако, последующий анализ показывает, что данные техники не оптимальны, а область применимости существующих методов точной генерации ДКА по примерам поведения все еще весьма ограничена (что обусловлено, в первую очередь, NP-полной природой задачи). Таким образом, тема настоящей диссертации, продолжающей указанные исследования последних лет и направленной на расширение возможностей методов сокращения пространства поиска и генерации ДКА, **является актуальной**.

Степень разработанности темы. Задача генерации детерминированного конечного автомата по заданным примерам поведения, выраженным в виде двух множеств S_+ и S_-, заключается в поиске ДКА с минимальным числом состояний, такого, что все строки из множества S_+ принимаются автоматом, а все строки из

[14] *Heule*, *M.*, *Verwer*, *S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Vol. 6339. Springer, 2010. P. 66–79. (Lecture Notes in Computer Science).

[15] *Heule*, *M.*, *Verwer*, *S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 825–856; *Ульянцев*, *В. И.* Генерация конечных автоматов с использованием программных средств решения задач выполнимости и удовлетворения ограничений : дис. … канд. техн. наук / Ульянцев Владимир Игоревич. НИУ ИТМО, 2015.

S_- — не принимаются. Впервые данная задача была сформулирована в работе Голда в 1967 году[16].

Первый известный алгоритм для решения данной задачи был предложен в работе Трахтенброта и Барздиня в 1970 году[17] — `TB`-алгоритм. Однако, предложенный алгоритм решает только частный случай задачи генерации ДКА по заданным примерам поведения: для некоторого натурального k все возможные слова длины k над алфавитом Σ — всего $|\Sigma|^k$ слов — должны содержаться во множествах S_+ и S_-. В данном алгоритме, как и в подавляющем большинстве последующих, примеры поведения представляются в виде расширенного префиксного дерева — префиксного дерева, в котором вершины могут быть допускающими, отвергающими или промежуточными. `TB`-алгоритм основан на полном переборе всех возможных пар состояний префиксного дерева и слиянии эквивалентных состояний в одно.

В 1978 году Голд доказал NP-полноту задачи генерации ДКА заданного, а значит и минимального, размера[18]. Ввиду указанной сложности новые алгоритмы генерации минимального ДКА не предлагались более десяти лет. В последующие годы начали активно разрабатываться неточные эвристические алгоритмы — ими не гарантируется минимальность найденного ДКА. Среди таких алгоритмов можно выделить следующие: `traxbar`[19], `RPNI`[20], `EDSM`[21], `exbar`[22], `Windowed-EDSM`[23]. Все перечисленные алгоритмы основаны на слиянии состоя-

[16] *Gold*, *E. M.* Language Identification in the Limit // Information and Control. 1967. Vol. 10, no. 5. P. 447–474.

[17] *Трахтенброт*, *Б. А.*, *Барздинь*, *Я. М.* Конечные автоматы (поведение и синтез). Наука. Гл. ред. физ.-мат. лит., 1970. 400 с. (Математическая логика и основания математики).

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

[19] *Lang*, *K. J.* Random DFA's Can Be Approximately Learned from Sparse Uniform Examples // Proceedings of the Fifth ACM Workshop on Computational Learning Theory. ACM, 1992. P. 45–52.

[20] *Oncina*, *J.*, *García*, *P.* Inferring Regular Languages in Polynomial Update Time // Pattern Recognition and Image Analysis. Vol. 1. 1992. P. 49–61. (Series in Machine Perception and Artificial Intelligence).

[21] *Lang*, *K. J.*, *Pearlmutter*, *B. A.*, *Price*, *R. A.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm // ICGI. Vol. 1433. Springer, 1998. P. 1–12. (Lecture Notes in Computer Science).

[22] *Lang*, *K. J.* Faster algorithms for finding minimal consistent DFAs : tech. rep. / NEC Research Institute. 1999
.

[23] *Cicchello*, *O.*, *Kremer*, *S. C.* Beyond EDSM // ICGI. Vol. 2484. Springer, 2002. P. 37–48. (Lecture Notes in Computer Science).

ний расширенного префиксного дерева. Состояния для слияния выбираются эвристически, чем одновременно объясняется высокая скорость работы и неточность данных алгоритмов.

Другим распространенным подходом к генерации ДКА по заданным примерам поведения является применение метаэвристических алгоритмов. Например, были предложены эволюционные алгоритмы[24], муравьиные алгоритмы[25]. Метаэвристические алгоритмы также являются неточными — ими вообще не гарантируется, что какое-то решение будет найдено за конечное время.

Голландские ученые Хойл и Вервер в 2010 году предложили алгоритм `DFASAT`, способный гарантированно генерировать ДКА минимального размера по произвольным данным[26]. Алгоритм основан на сведении задачи генерации ДКА по примерам поведения к задаче выполнимости булевой формулы (SAT), где в качестве промежуточного шага используется сведение к задаче раскраски графа, которое было предложено еще в 1997 году[27]. Для сокращения пространства поиска при решении задачи выполнимости Хойл и Вервер предложили ряд подходов к нарушению симметрии: граф совместимости, дополнительные дизъюнкты, поиск большой клики. Однако, даже с использованием всех этих техник, `DFASAT` способен за разумное время строить автоматы с не более чем десятью состояниями. Хойл и Вервер для упрощения задачи предложили делать предварительно несколько слияний в префиксном дереве с помощью алгоритма `EDSM`, однако тогда теряется точность — гарантия минимальности найденного ДКА.

Позднее автором диссертации совместно с научным руководителем Ульянцевым В. И. и профессором Шалыто А. А. были предложены предикаты наруше-

[24] *Lucas*, *S. M.*, *Reynolds*, *T. J.* Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005. Vol. 27, no. 7. P. 1063–1074; *Gómez*, *J.* An Incremental-Evolutionary Approach for Learning Deterministic Finite Automata // IEEE Congress on Evolutionary Computation. IEEE, 2006. P. 362–369.

[25] *Chivilikhin*, *D.*, *Ulyantsev*, *V.* Learning Finite-State Machines with Ant Colony Optimization // Swarm Intelligence. Vol. 7461. Springer Berlin / Heidelberg, 2012. P. 268–275. (Lecture Notes in Computer Science).

[26] *Heule*, *M.*, *Verwer*, *S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Vol. 6339. Springer, 2010. P. 66–79. (Lecture Notes in Computer Science).

[27] *Coste*, *F.*, *Nicolas*, *J.* Regular Inference as a graph coloring problem // Workshop on Grammatical Inference, Automata Induction, and Language Acquisition (ICML'97). 1997.

ния симметрии на основе алгоритма обхода графа в ширину (BFS) [1]. Для каждого ДКА с M состояниями существует $\mathcal{O}(M!)$ изоморфных ему автоматов, которые, с точки зрения программного средства для решения SAT, являются различными. Предложенные предикаты нарушения симметрии позволяют значительно сократить пространство поиска при решении SAT — вместо факториала изоморфных автоматов при решении перебирается только единственный представитель класса эквивалентности по изоморфизму — ДКА, пронумерованный в порядке BFS-обхода. Данный метод является наиболее эффективным известным точным методом генерации ДКА по заданным примерам поведения и позволяет строить автоматы с числом состояний до сорока. Однако, анализ показал, что предложенное наивное кодирование BFS-предикатов на языке SAT неоптимально — получаемая булева формула слишком велика. Помимо этого, не рассматривались предикаты нарушения симметрии, основанные на кодировании алгоритма обхода графа в глубину (DFS).

Среди прочих недостатков методов генерации ДКА, основанных на сведении к SAT, можно отметить зависимость размера булевой формулы от числа имеющихся примеров поведения. Зачастую множества примеров поведения избыточны и содержат лишние слова. Анализ литературы показал, что интеллектуальных методов выбора подмножества примеров поведения ранее не предлагалось. Возможным решением может быть применение подхода *уточнения абстракции по контрпримерам* (*counterexample-guided abstraction refinement* — CEGAR) — итеративного алгоритма, изначально разработанного для построения модели программного обеспечения[28].

Обратной ситуацией является генерация ДКА по небольшим множествам примеров поведения. В таком случае полезным может быть генерация всех соответствующих неизоморфных ДКА с минимальным числом состояний для проведения дальнейшего анализа. Также, существование единственного автомата ми-

[28] *Clarke*, *E. M.*, *Grumberg*, *O.*, *Jha*, *S.*, *Lu*, *Y.*, *Veith*, *H.* Counterexample-Guided Abstraction Refinement // CAV. Vol. 1855. Springer, 2000. P. 154–169. (Lecture Notes in Computer Science); *Мандрыкин*, *М.*, *Мутилин*, *В.*, *Хорошилов*, *А.* Введение в метод CEGAR — уточнение абстракции по контрпримерам // Труды Института системного программирования РАН. 2013. Т. 24.

нимального размера говорит о качестве имеющихся обучающих данных. Однако, задача генерации всех неизоморфных ДКА ранее не ставилась, и, соответственно, эффективных методов ее решения предложено не было.

Целью настоящей диссертации является повышение эффективности точных методов генерации детерминированных конечных автоматов по заданным примерам поведения посредством сокращения пространства поиска при решении задачи выполнимости.

Для достижения указанной цели в работе поставлены и решены следующие **задачи**:

а) Разработка предикатов нарушения симметрии, основанных на кодировании алгоритмов обхода графа в ширину и в глубину, для сокращения пространства поиска при решении задачи выполнимости. Разработка и реализация точных методов генерации ДКА по заданным примерам поведения, использующих данные предикаты, проведение экспериментальных исследований разработанных методов.

б) Разработка и реализация точного метода генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам. Проведение экспериментальных исследований разработанного метода.

в) Разработка и реализация метода генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств решения задачи выполнимости. Проведение экспериментальных исследований разработанного метода.

Предмет исследования — методы точной генерации ДКА, использующие программные средства для решения задачи выполнимости.

Соответствие паспорту специальности. Данная диссертация соответствует пункту 10 «Разработка основ математической теории языков и грамматик, теории конечных автоматов и теории графов» паспорта специальности 05.13.17 — «Теоретические основы информатики».

Основные положения, выносимые на защиту:

а) *Подход* к построению предикатов нарушения симметрии, основанных на кодировании алгоритмов обхода графа в ширину и в глубину, для сокращения пространства поиска при решении задачи выполнимости. Точные *методы* генерации ДКА по заданным примерам поведения, использующие данные предикаты.

б) Точный *метод* генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам.

в) *Метод* генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств решения задачи выполнимости.

Научная новизна диссертации состоит в следующем:

а) Предикаты нарушения симметрии, основанные на кодировании алгоритма обхода графа в глубину ранее не предлагались. Предложенные предикаты нарушения симметрии, основанные на алгоритме обхода графа в ширину, выражаются булевой формулой, состоящей из асимптотически меньшего числа дизъюнктов сравнительно с существующим подходом. Помимо этого, впервые предложены предикаты нарушения симметрии, использующие особенности дерева обхода графа в ширину.

б) Точных методов генерации ДКА применимых в случае, когда число примеров поведения излишне велико, ранее не предлагалось. Использование подхода уточнения абстракции по контрпримерам одновременно со сведением к задаче выполнимости позволяет генерировать ДКА по избыточному набору примеров поведения путем итеративного добавления только значимых примеров до тех пор, пока не будет получен ДКА, соответствующий всему избыточному набору.

в) Методов для генерации всех неизоморфных ДКА минимального (или любого фиксированного) размера, удовлетворяющих заданным примерам поведения, ранее не предлагалось.

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, теории графов, теории сложности, дискретной математики, объектно-ориентированное программирование, методы проведения и анализа экспериментальных исследований.

Достоверность полученных результатов подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, а также результатами проведенных экспериментальных исследований по использованию предложенных в диссертации методов.

Теоретическая значимость работы заключается в том, что в ней для точных методов генерации ДКА предложены новые подходы к сокращению пространства поиска при решении задачи выполнимости — предикаты нарушения симметрии на основе кодирования алгоритмов обхода графа в глубину и в ширину:

- предложен способ кодирования свойства DFS-пронумерованности ДКА на языке SAT;
- предложен новый способ кодирования свойства BFS-пронумерованности ДКА на языке SAT, требующий асимптотически меньшего числа дизъюнктов, относительно известных способов;
- предложен способ кодирования различных свойств BFS-дерева на языке SAT.

Помимо этого, предложен способ объединить метод генерации ДКА при помощи сведения к задаче выполнимости с подходом уточнения абстракции по контрпримерам. Также, разработанные предикаты нарушения симметрии позволили разра-

ботать метод для решения задачи генерации всех неизоморфных ДКА минимального размера, которая ранее не имела эффективных методов решения.

Практическая значимость работы состоит в повышении эффективности точных методов генерации ДКА по заданным примерам поведения. Экспериментально показано, что разработанный метод генерации ДКА по заданным примерам поведения с использованием предикатов нарушения симметрии на основе алгоритма BFS является самым эффективным по времени генерации автомата среди известных на настоящий момент точных методов и позволяет генерировать автоматы большего размера относительно методов, предложенных ранее. Разработанный точный метод генерации ДКА по избыточному набору примеров поведения с использованием сведения к SAT и подхода CEGAR позволяет эффективно генерировать автоматы в случае, когда набор примеров поведения слишком объемен и получаемая булева формула слишком велика для современных программных средств для решения SAT. Разработанный метод генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств для решения SAT, является первым известным методом для генерации всех неизоморфных автоматов. Также с его помощью можно оценить полноту имеющихся данных, выраженных в виде примеров поведения, путем доказательства или опровержения существования единственного ДКА минимального размера, описывающего эти данные.

Помимо этого, все разработанные методы и подходы в дальнейшем могут быть адаптированы для задач генерации более сложных конечно-автоматных моделей[29]. Так, например, предложенный метод генерации всех неизоморфных ДКА был адаптирован для генерации конечных автоматов, моделирующих поведение программируемых логических контроллеров[30].

[29] *Ульянцев*, *В. И.* Генерация конечных автоматов с использованием программных средств решения задач выполнимости и удовлетворения ограничений : дис. … канд. техн. наук / Ульянцев Владимир Игоревич. НИУ ИТМО, 2015.

[30] *Chivilikhin*, *D.*, *Patil*, *S.*, *Chukharev*, *K.*, *Cordonnier*, *A.*, *Vyatkin*, *V.* Automatic State Machine Reconstruction From Legacy Programmable Logic Controller Using Data Collection and SAT Solver // IEEE Transactions on Industrial Informatics. 2020. Vol. 16, no. 12. P. 7821–7831.

Внедрение результатов работы. Результаты работы использовались при выполнении проекта SAUNA (“Integrated safety assessment and justification of nuclear power plant automaton”), выполненного исследовательской группой “IT in Industrial Automation” кафедры электротехники и автоматики университета Аалто, Финляндия, в рамках Финской программы исследований безопасности атомных электростанций — SAFIR2018[31]. В частности, одной из задач проекта была разработка метода генерации моделей различных компонентов системы управления атомных электростанций по заданным примерам поведения и спецификации выраженной на языке темпоральной логики. Данная задача была решена с использованием подхода уточнения абстракции по контрпримерам способом, аналогичным предложенному автором диссертации для генерации ДКА, что подтверждается письмом руководителя исследовательской группы “IT in Automation” В. В. Вяткина.

Результаты работы также использовались при выполнении под руководством автора диссертации гранта Российского фонда фундаментальных исследований (проект 18-37-00425 «Разработка эффективных методов машинного обучения для построения детерминированных конечных автоматов на основе решения задачи выполнимости», 2018–2020 гг.).

Полученные результаты также использовались в рамках государственной финансовой поддержки ведущих университетов Российской Федерации, субсидия 074-U01 (НИР «Биоинформатика, машинное обучение, технологии программирования, теория кодирования, проактивные системы», 2013–2017 гг.) и субсидия 08-08 (НИР «Методы, модели и технологии искусственного интеллекта в биоинформатике, социальных медиа, киберфизических, биометрических и речевых системах», 2018–2020 гг.)

Результаты работы также внедрены в учебный процесс факультета информационных технологий и программирования Университета ИТМО в рамках курса «Проектирование автоматных программ» программы бакалавриата «Математиче-

[31] http://safir2018.vtt.fi/

ские модели и алгоритмы в разработке программного обеспечения», что подтверждается актом об использовании.

Апробация результатов работы. Основные результаты работы докладывались на следующих конференциях и семинарах:

а) 9th International Conference on Language and Automata Theory and Applications (LATA 2015). 2015, Ницца, Франция.

б) 6th International Symposium “From Data to Models and Back (DataMod)”. 2017, Тренто, Италия.

в) 16th IEEE International Conference on Industrial Informatics (INDIN 2018). 2018, Порту, Португалия.

г) 13th International Conference on Language and Automata Theory and Applications (LATA 2019). 2019, Санкт-Петербург.

д) IV-VII Всероссийский конгресс молодых ученых. 2015-2018, Санкт-Петербург.

е) IX Конгресс молодых ученых. 2020, Санкт-Петербург.

ж) XLVI Научная и учебно-методическая конференция Университета ИТМО. 2017, Санкт-Петербург.

и) XLVIII Научная и учебно-методическая конференция Университета ИТМО. 2019, Санкт-Петербург.

Личный вклад автора. Идея предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в глубину, идея метода генерации ДКА по заданным примерам поведения, использующего их, а также реализация алгоритма на базе предложенного метода и проведение вычислительных экспериментов принадлежит лично автору. Идея предикатов нарушения симметрии на основе алгоритма обхода графа в ширину, кодирование которых требует асимптотически меньшего числа дизъюнктов, идея кодирования свойств BFS-дерева на языке SAT и идея метода генерации ДКА, использующего данные разработки, принадлежат совместно автору диссертации и Ж. Маркешу-Сильве; реализация алгоритмов на базе предложенных методов принадлежит лично автору, проведение вычислительных экспериментов принадлежит совместно автору диссертации и А.И. Иг-

натьеву. Идея точного метода генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам, реализация алгоритма на базе предложенного метода и проведение вычислительных экспериментов принадлежит лично автору. Идея точного метода генерации всех ДКА минимального размера по заданным примерам поведения с использованием программных средств решения задачи выполнимости принадлежит совместно автору диссертации и научному руководителю В.И. Ульянцеву, реализация алгоритма на базе предложенного метода и проведение вычислительных экспериментов принадлежит лично автору. В работах, выполненных в соавторстве, В.И Ульянцевым осуществлялось общее руководство исследованиями.

Публикации. Основные результаты по теме диссертации изложены в десяти публикациях, из них четыре опубликованы в изданиях, индексируемых в базе цитирования Scopus, одна публикация издана в журнале, рекомендованном ВАК. Также у автора диссертации имеется одна публикация по другой теме из области машинного обучения, опубликованная в издании, индексируемом в базе цитирования Scopus.

Содержание работы

Во **введении** диссертационной работы обоснована актуальность проводимых исследований. Сформулированы цель, задачи и положения, выносимые на защиту. Изложена научная новизна, теоретическая и практическая значимость результатов, полученных в диссертационной работе.

Первая глава диссертации посвящена обзору предметной области и результатов существующих исследований, посвященных генерации детерминированных конечных автоматов. Кроме того, в первой главе приведены терминология, основные определения и известные результаты ряда разделов информатики, необходимых для описания предлагаемых в диссертации методов и алгоритмов.

В **разделе 1.1** приведена формальная постановка задачи выполнимости булевой формулы, даны необходимые определения, и представлено краткое описание основных подходов к ее решению. Также в данном разделе описан подход к решению NP-трудных задач с помощью полиномиального сведения таких задач к задаче выполнимости. Помимо этого, приводится краткий обзор существующих программных средств решения SAT.

Булева формула задана в *конъюнктивно-нормальной форме* (КНФ), если является конъюнкцией дизъюнктов — множества литералов, связанных дизъюнкцией. *Задача выполнимости булевой формулы* (*задача выполнимости*, *boolean satisfiability problem* — SAT) заключается в определении, существует ли для некоторой булевой формулы, заданной в КНФ, выполняющая подстановка — набор значений переменных, при подстановке которых формула принимает истинное значение. Задача выполнимости является исторически первой, для которой была доказана NP-трудность — любую задачу из класса NP можно за полиномиальное время свести к SAT. Данный факт объясняет актуальность разработки все более эффективных программных средств для решения задачи выполнимости. Ежегодно в научном сообществе проходят соревнования по выявлению лучшего программного средства для решения SAT, что также способствует постоянному

развитию данной области. В основе современных программных средств решения SAT лежит стратегия *управляемого конфликтами обучения дизъюнктов* (*conflict-driven clause learning*, CDCL).

Подход к решению задачи из класса NP, когда разрабатывается сведение к SAT и затем используется современное программное средство для поиска выполняющей подстановки, зачастую оказывается заметно эффективнее и проще, чем разработка применимого на практике метода, непосредственно решающего исходную задачу. Еще одним немаловажным достоинством такого подхода является тот факт, что достаточно единожды написать сведение к SAT, а затем без прикладывания каких-либо усилий пользоваться развитием программных средств, выбирая самое эффективное из них.

В **разделе 1.2** приведены базовые понятия о детерминированных конечных автоматах и постановка задачи генерации детерминированных конечных автоматов по примерам поведения. Примерами поведения некоторого ДКА $\mathcal{D}$ называются два множества слов, состоящих из символов алфавита автомата, S_+ и S_- таких, что все слова из S_+ принадлежат языку $\mathcal{L}(\mathcal{D})$ — должны им приниматься, а все слова из S_- не принадлежат языку $\mathcal{L}(\mathcal{D})$ — должны им отвергаться. Задача генерации ДКА по примерам поведения заключается в поиске автомата минимального размера (с минимальным числом состояний), соответствующего имеющимся примерам поведения. Ранее было доказано, что настоящая задача является NP-полной, ровно как и задача генерации ДКА любого фиксированного размера, соответствующего имеющимся примерам поведения. Пример детерминированного конечного автомата, соответствующего наборам примеров поведения $S_+ = \{aba, bb, bba\}$ и $S_- = \{b, ba\}$, представлен на рисунке 1. Можно заметить, что автомата размера два, соответствующего множествам S_+ и S_-, не существует.

В **разделе 1.3** приведен обзор существующих эвристических и метаэвристических методов генерации детерминированных конечных автоматов по примерам поведения.

Среди эвристических алгоритмов можно выделить алгоритм *объединения состояний на основе свидетельств* (*evidence-driven state merging* — EDSM). Ме-

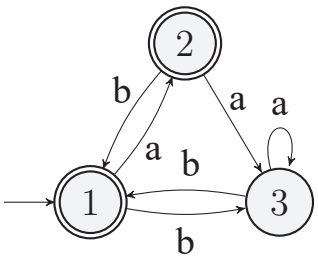


Рисунок 1 – Пример ДКА минимального размера, соответствующего наборам примеров поведения $S_+ = \{aba, bb, bba\}$ и $S_- = \{b, ba\}$

таэвристические методы основаны на эволюционных стратегиях, генетических и муравьиных алгоритмах.

Следует отметить, что данные походы являются неточными — ими не гарантируется, что найденный автомат содержит минимальное возможное число состояний, а иногда вообще не гарантируется, что удовлетворяющий автомат будет найден.

В **разделе 1.4** приведен обзор существующих методов генерации детерминированных конечных автоматов по примерам поведения, основанных на сведении к SAT. В отличие от эвристических и метаэвристических подходов, данные методы являются точными — гарантируется, что автомат, соответствующий примерам поведения, будет построен за конечное время и будет содержать минимальное возможное число состояний.

Первым шагом рассматриваемых методов является построение расширенного префиксного дерева (augmented prefix tree acceptor — APTA) — древовидной структуры данных, основанной на обычном префиксном дереве, в которой каждая вершина либо не помечена, либо помечена как допускающая или отвергающая. Пример расширенного префиксного дерева представлен на рисунке 2.

Далее, начиная с некоторой нижней оценки на размер — в простейшем случае с единицы — происходит поиск автомата текущего размера, соответствующего построенному расширенному префиксному дереву. Данный процесс продолжается до тех пор, пока не будет найден ДКА, удовлетворяющий заданным требованием. Итеративный перебор размера от меньшего к большему гарантирует, что найденный автомат имеет минимальный размер. Как уже было сказа-

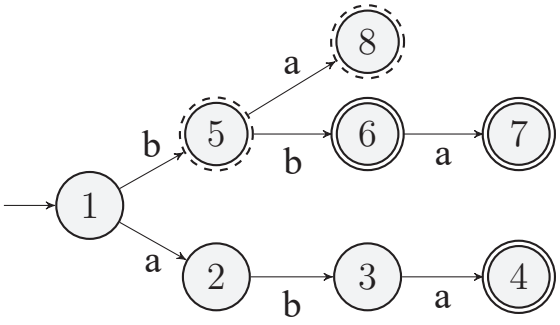


Рисунок 2 – Пример расширенного префиксного дерева, построенного по наборам примеров поведения $S_+ = \{aba, bb, bba\}$ и $S_- = \{b, ba\}$

но, задача поиска ДКА конкретного размера по заданным примерам поведения принадлежит классу NP, а значит, может быть решена путем сведения к некоторой NP-трудной задаче. Самым производительным точным методом до недавнего времени являлся `DFASAT`[32], в котором авторы предложили сначала свести задачу генерации ДКА к задаче раскраски графа — необходимо раскрасить расширенное префиксное дерево в минимальное число цветов так, чтобы все вершины одного цвета объединялись в одно состояние, — которую затем свести к задаче выполнимости. Схема метода `DFASAT` представлена на рисунке 3.

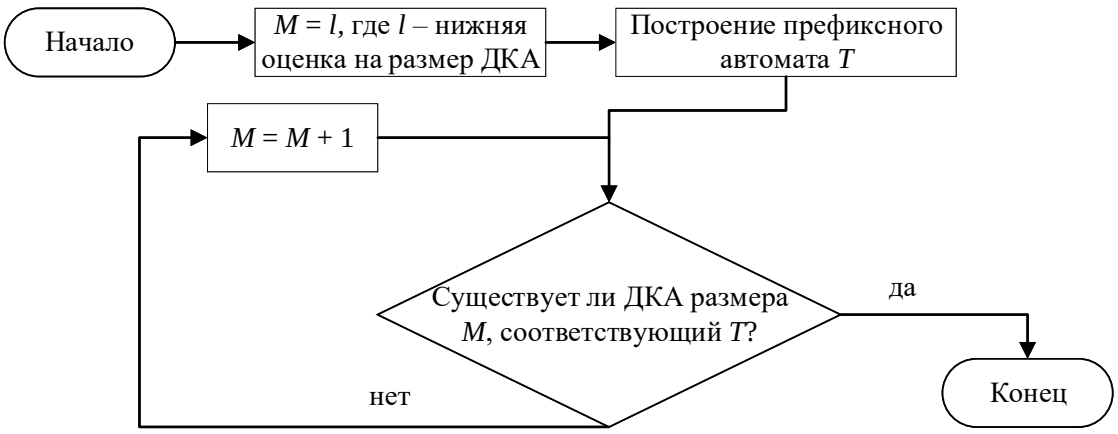


Рисунок 3 – Схема точного метода генерации ДКА по заданным примерам поведения на основе сведения к SAT — `DFASAT`

Авторы `DFASAT` используют несколько различных подходов к сокращению пространства поиска:

[32] *Heule*, *M.*, *Verwer*, *S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Vol. 6339. Springer, 2010. P. 66–79. (Lecture Notes in Computer Science).

а) добавление нескольких видов дополнительных дизъюнктов в булеву формулу, которые не влияют на решение, но добавляют дополнительные ограничения на возможные значения переменных;

б) построение графа совместимости, позволяющего заранее найти некоторые пары вершин префиксного дерева, которые не могут быть объединены в одно состояние автомата;

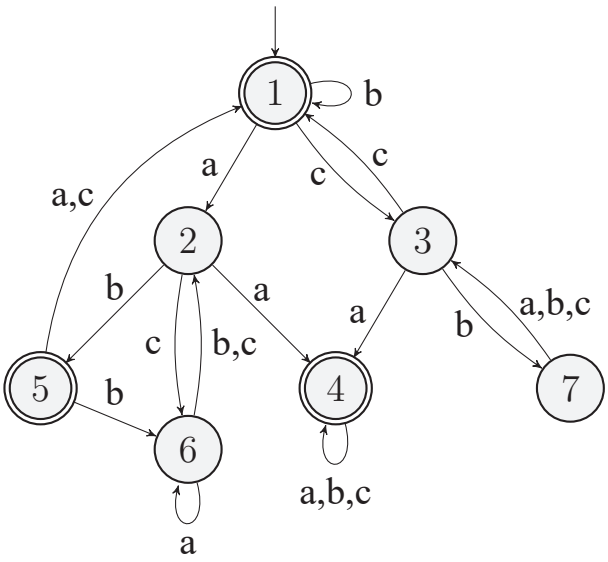
в) нахождение некоторой большой клики в графе совместимости и фиксирование нумерации вершин данной клики.

Использование вышеперечисленных подходов позволяет значительно увеличить производительность оригинального метода, однако они не решают фундаментальной проблемы наличия изоморфных автоматов. Два автомата называются изоморфными, если они различаются только нумерацией состояний. Таким образом, если ДКА содержит N состояний, то существует $\mathcal{O}(N!)$ изоморфных ему автоматов. Несмотря на то, что изоморфные автоматы не отличаются с практической точки зрения, с точки зрения программного средства для решения SAT они различны.

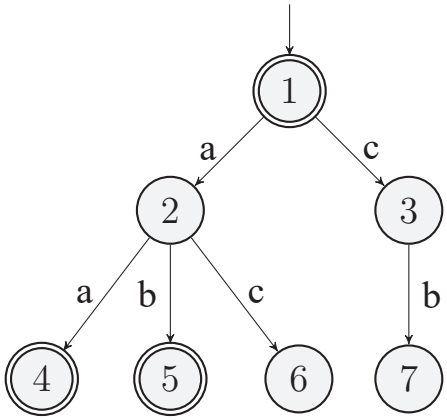
Решением данной проблемы стала разработка предикатов нарушения симметрии на основе *обхода графа в ширину* (*breadth-first search* — BFS) [1]. Добавление таких предикатов в булеву формулу фиксирует нумерацию рассматриваемых автоматов в порядке BFS, что для каждого класса эквивалентности по изоморфизму позволяет оставить для рассмотрения единственного представителя. Для записи предикатов в виде булевой формулы требуется $\mathcal{O}\left(M^3 + M^2 \times L^2\right)$ дизъюнктов, где N — размер префиксного дерева, M — размер искомого ДКА. Был разработан метод генерации ДКА при помощи сведения к SAT, использующий предложенные предикаты нарушения симметрии, который был реализован в виде программного средства `DFA-Inductor`[33]. Метод генерации ДКА при помощи сведения к SAT с использованием BFS-предикатов нарушения симметрии является наиболее производительным точным методом, который лежит в основе всех ме-

[33] *Ilya Zakirzyanov*, *V. U.* DFA-Inductor. URL: `https://github.com/ctlab/DFA-Inductor` (дата обр. 02.10.2020).

тодов, разрабатываемых в данной диссертации. Пример BFS-пронумерованного ДКА представлен на рисунке 4a, а на рисунке 4b представлено его дерево обхода.



(a) Пример BFS-пронумерованного автомата



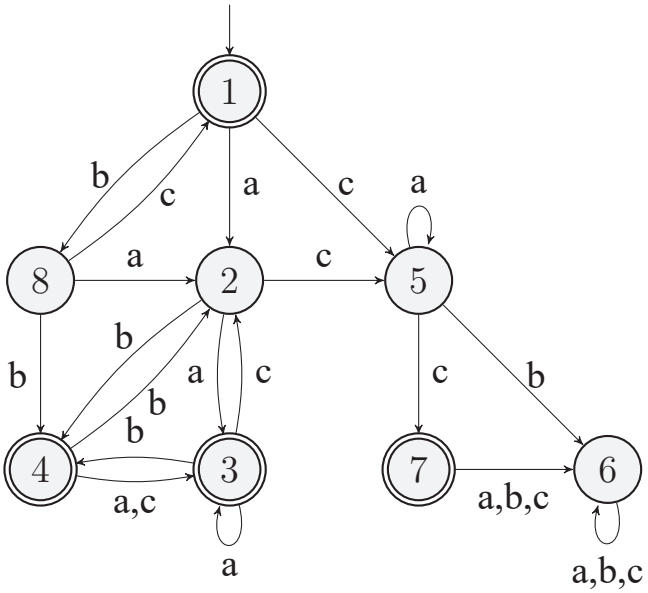
(b) Дерево BFS для автомата, представленного на рисунке 4a

Рисунок 4 – Пример BFS-пронумерованного автомата и соответствующего BFS-дерева

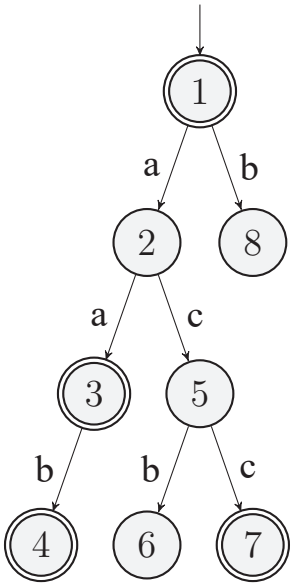
В **разделе 1.5** приведено описание алгоритма уточнения абстракции по контрпримерам (counterexample guided abstraction refinement — CEGAR). Методы, использующие данный алгоритм, применимы в ситуации, когда необходимо построить модель, соответствующую заданным требованиям, имея при этом доступ к некоторой проверяющей системе — оракулу. На начальном шаге строится некоторая, возможно, случайная модель. Затем начинается итеративный процесс уточнения имеющейся модели — на каждом шаге текущая модель проверяется оракулом на соответствие заданным требованиям. Если проверка проходит успешно, то модель найдена. Иначе, оракул сообщает один или несколько контрпримеров, которые затем используются для уточнения модели.

Во **второй главе** настоящей диссертации описываются разработка, реализация и экспериментальные исследования методов генерации детерминированных конечных автоматов с использованием различных подходов к сокращению пространства поиска при решении задачи выполнимости.

В **разделе 2.1** приведено описание разработанных предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в глубину (*depth-first search* — DFS). Использование предикатов нарушения симметрии, задающих BFS нумерацию автомата, ранее позволило значительно улучшить производительность метода `DFASAT`. Логично в качестве следующей задачи научного исследования было разработать предикаты нарушения симметрии на основе алгоритма DFS и метод, использующий их. Пример DFS-пронумерованного ДКА представлен на рисунке 5a, а на рисунке 5b представлено его дерево обхода.



(a) Пример DFS-пронумерованного автомата



(b) Дерево DFS для автомата, представленного на рисунке 5a

Рисунок 5 – Пример DFS-пронумерованного автомата и соответствующего DFS-дерева

Разработанные предикаты выражаются КНФ-формулой, содержащей $\mathcal{O}\left(M^4 + M^3 \times L^2\right)$ дизъюнктов, что в M раз больше чем предикаты, задающие BFS нумерацию ДКА.

В **разделе 2.2** приведено описание разработанных компактных предикатов нарушения симметрии на основе кодирования алгоритма BFS.

Компактность разработанных предикатов заключается в том, что удалось сократить размер формулы, выражающей предикаты нарушения симметрии, зада-

ющие BFS нумерацию автомата, с $\mathcal{O}\left(M^3 + M^2 \times L^2\right)$ дизъюнктов до $\mathcal{O}\left(M^2 \times L\right)$ дизъюнктов. Анализ ограничений исходного сведения, которые выражались через $\mathcal{O}\left(M^3\right)$ и $\mathcal{O}\left(M^2 \times L^2\right)$ дизъюнктов, показал, что часть параметров, задающих размер формулы, являются независимыми, в то время как другие параметры могут быть исключены с помощью добавления новых переменных. Для каждого такого ограничения был разработан способ сделать его более компактным, что и позволило сократить общий размер формулы. Помимо этого, б*о*льшая часть дизъюнктов из оригинального сведения, которые состояли из $\mathcal{O}\left(M\right)$ литералов, были заменены на дизъюнкты, состоящие из двух или трех литералов, что заметно влияет на производительность программного средства для решения SAT, так как такие дизъюнкты обрабатываются за константное время и не хранятся в памяти[34].

В **разделе 2.3** приведено описание разработанных подходов к сокращению пространства поиска при генерации детерминированных конечных автоматов, основанных на особенностях структуры BFS-дерева, а также на связях между расширенным префиксным деревом и искомым ДКА. На рисунке 6 представлено полное BFS-дерево произвольного размера над произвольным алфавитом размера L. Дерево названо полным, так как каждая вершина имеет ровно L детей.

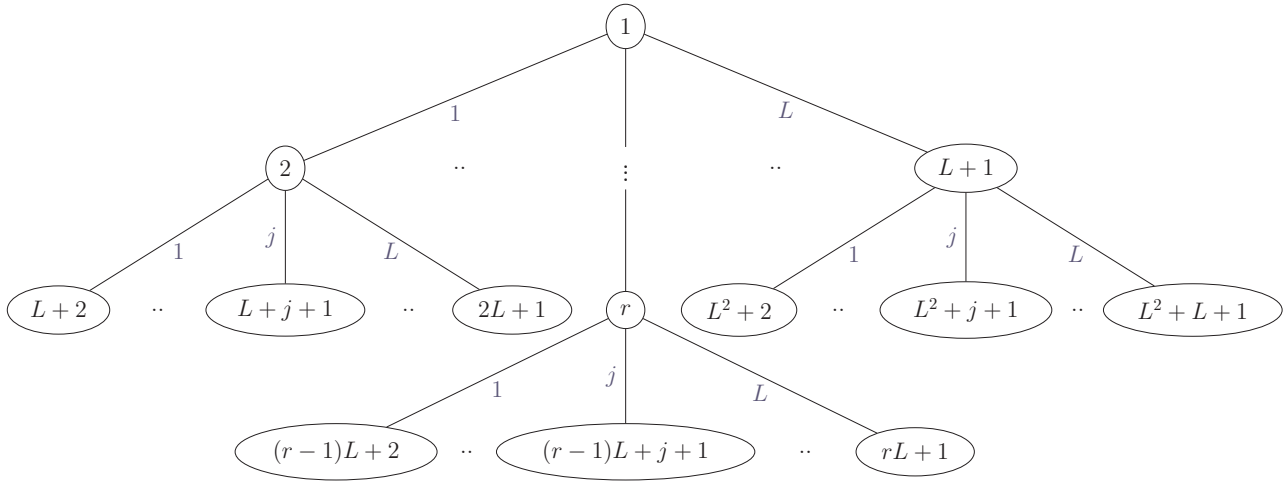


Рисунок 6 – Полное BFS-дерево, где $|\Sigma| = L$

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

После анализа данного дерева были сформулированы некоторые свойства и ограничения, свойственные BFS-дереву. Например, было доказано, что у некоторой вершины с номером r в полном дереве самый правый ребенок имеет номер $rL + 1$. Далее было доказано, что тогда $rL + 1$ является верхней оценкой на номер ребенка вершины r в произвольном BFS-дереве. Тогда, можно утверждать, что в произвольном BFS-дереве у вершины с номером r дети могут иметь номера только в диапазоне от $r + 1$ до $rL + 1$. Введение данных ограничений позволяет сократить число используемых в сведении переменных и сократить размер получающейся формулы.

По определению в любом BFS-дереве дети любой вершины r имеют последовательные номера, а также верно, что их не более чем L. Была проанализирована связь между вершинами расширенного префиксного дерева и состояниями ДКА. Если в префиксном дереве существует путь длины k от корня до некоторой вершины t_i, и вершина t_i соответствует состоянию d_j генерируемого автомата, то в автомате должен существовать путь длины не более чем k от стартового состояния до состояния d_j. Было разработано кодирование данных свойств на языке SAT. Добавление соответствующих ограничений в формулу позволяет дополнительно сократить пространство поиска при решении задачи выполнимости.

В **разделе 2.4** приведены описание разработанного программного средства `DFA-Inductor-py`, предназначенного для генерации детерминированных конечных автоматов, описание реализации разработанных методов как частей данного программного средства, а также результаты экспериментальных исследований всех разработанных методов.

Во время работы над диссертацией на языке *Python* было разработано программное средство `DFA-Inductor-py`[35], предназначенное для генерации ДКА по заданным примерам поведения. В состав средства входят различные модули, позволяющие решать задачу генерации ДКА по заданным примерам поведения, задачу генерации ДКА по избыточному набору примеров поведения и задачу ге-

[35] *Zakirzyanov, I.* DFA-Inductor-py. URL: `https://github.com/ctlab/DFA-Inductor-py` (дата обр. 02.10.2020).

нерации всех неизоморфных ДКА по заданным примерам поведения (данные методы описываются в следующих главах). В средстве реализованы различные предикаты нарушения симметрии — как предложенные ранее другими авторами, так и разработанные в рамках настоящей диссертации.

С использованием реализованного программного средства были проведены две серии экспериментальных исследований. Между собой сравнивались метод генерации ДКА по заданным примерам поведения с использованием оригинальных предикатов нарушения симметрии на основе BFS и аналогичный метод с использованием DFS-предикатов. Поскольку метод `DFASAT`, использующий в качестве предикатов нарушения симметрии фиксирование нумерации некоторой большой клики графа несовместимости, лежит

Имеющиеся в открытом доступе тестовые данные (наборы примеров поведения), например, с соревнований *Abbadingo One DFA Learning Competition*[36] или *StaMinA Competition*[37], разрабатывались под неточные алгоритмы, которые могут строить автоматы с сотнями состояний. Существующие и предлагаемые в рамках настоящей диссертации точные методы пока не способны строить ДКА такого размера, поэтому тестовые данные генерировались случайно. Результаты экспериментов представлены в таблице 1 и позволяют сделать вывод, что использование DFS-предикатов нарушения симметрии нецелесообразно, так как метод, их использующий, значительно проигрывает методу, использующему BFS-предикаты. Однако можно заметить, что метод, использующий DFS-предикаты значительно выигрывает в производительности относительно метода `DFASAT`. Тем не менее, исходя из результатов, было решено не продолжать развитие DFS-предикатов и сосредоточиться на улучшении предикатов, использующих алгоритм обхода графа в ширину.

[36] *Lang*, *K. J.*, *Pearlmutter*, *B. A.*, *Price*, *R. A.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm // ICGI. Vol. 1433. Springer, 1998. P. 1–12. (Lecture Notes in Computer Science).

[37] *Walkinshaw*, *N.*, *Lambeau*, *B.*, *Damas*, *C.*, *Bogdanov*, *K.*, *Dupont*, *P.* STAMINA: a competition to encourage the development and assessment of software model inference techniques // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 791–824.

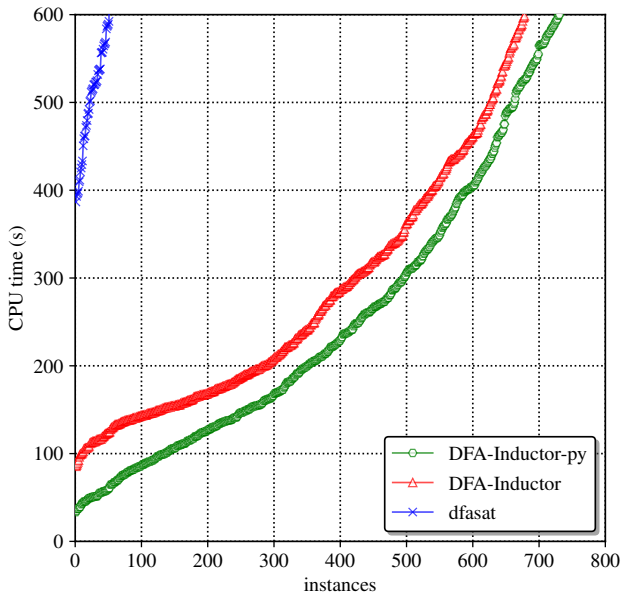
Таблица 1 – Медианное время работы методов генерации ДКА по заданным примерам поведения с использованием BFS-предикатов нарушения симметрии, DFS-предикатов нарушения симметрии и метода DFASAT в секундах, посчитанное на 100 задачах для каждого набора параметров. Время работы методов было ограничено одним часом (TL $= 3600$ секунд)

M	DFS	BFS	DFASAT
10	20,9	20,5	23,3
12	40,4	37,6	240,3
14	82,2	62,4	—
16	205,1	114,1	—
18	601,7	181,9	—
20	2501,6	293,7	—
22	—	453,3	—
24	—	625,1	—
26	—	925,8	—
28	—	1314,4	—
30	—	1635,5	—

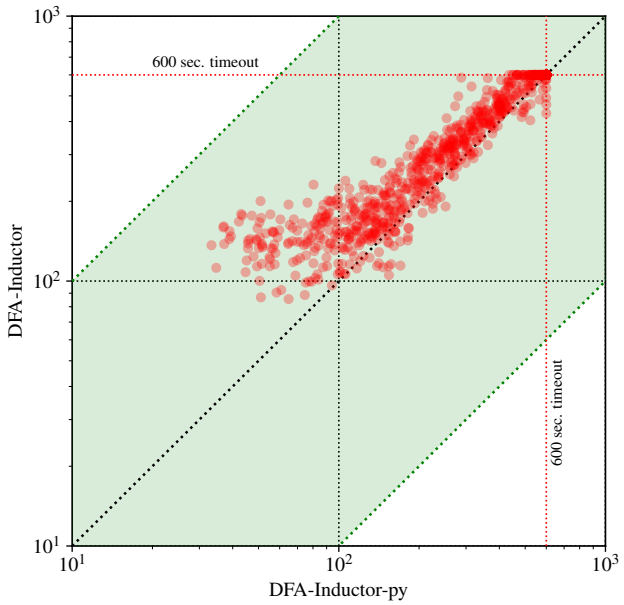
Во втором экспериментальном исследовании сравнивались методы, использующие оригинальные BFS-предикаты (DFA-Inductor) и BFS-предикаты, разработанные в настоящей диссертации (DFA-Inductor-py). Как и ранее, дополнительно в сравнение был включен метод DFASAT, и использовались случайно сгенерированные тестовые данные. Результаты сравнения всех трех методов представлены на рисунке 7a и показывают, что метод DFA-Inductor-py способен за одно и то же время решить большее число экземпляров задачи генерации ДКА по заданным примерам поведения, чем метод DFA-Inductor — 731 решенная задача против 678 за 10 минут. Также можно заметить, что метод DFASAT не способен составить конкуренции двум другим подходам.

На рисунке 7b представлено детальное сравнение методов DFA-Inductor и DFA-Inductor-py, где видно, что при решении подавляющего числа экземпляров задачи генерации ДКА, метод, предложенный в настоящей работе, показывает лучшие результаты.

В **третьей главе** настоящей диссертации описываются разработка, реализация и экспериментальные исследования точного метода генерации детерминированных конечных автоматов по избыточному набору примеров поведения с ис-



(a) Сравнение методов `DFA-Inductor`, `DFA-Inductor-py` и `DFASAT`, показывающее число различных экземпляров задачи, решенных за определенное время



(b) Детальное сравнение методов `DFA-Inductor` и `DFA-Inductor-py`, сравнивающее их производительность в решении каждого экземпляра задачи

Рисунок 7 – Результаты сравнения метода, использующего оригинальные BFS-предикаты (`DFA-Inductor`), метода, использующего новые BFS-предикаты (`DFA-Inductor-py`), и `DFASAT`

пользованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам.

В **разделе 3.1** приведено исследование границ применимости предложенных в предыдущих главах методов в зависимости от размера расширенного префиксного дерева. Размер булевой формулы, кодирующей задачу генерации ДКА, линейно зависит от размера префиксного дерева: $\mathcal{O}\left(N \times M^2\right)$ дизъюнктов, где N — размер префиксного дерева, а M — размер генерируемого ДКА. Число используемых переменных в булевой формуле также линейно зависит от размера префиксного дерева — $\mathcal{O}\left(N \times M + M^2\right)$ переменных. Таким образом, при неизменном генерируемом автомате, размер булевой формулы и число переменных может сильно меняться в зависимости от числа примеров поведения и их длины. Программному средству для решения SAT при увеличении размера префиксного дерева становится все затратнее хранить формулу и работать с ней, а переменных для перебора становится все больше. Это приводит к ситуации, когда один

и тот же автомат может быть получен за секунды работы программного средства по небольшому числу примеров поведения небольшой длины и может быть не найден за часы и дни работы средства в случае *избыточного* числа длинных примеров поведения.

Так как в случае детерминированных конечных автоматов ключевая информация о примере поведения — принимается данное слово автоматом или нет — содержится в последней вершине пути в расширенном префиксном дереве, соответствующего данному слову, то едва ли можно что-то сделать в случае длинных примеров поведения. Однако в случае избыточного числа примеров поведения, можно взять только часть из них и построить тот же автомат быстрее. Сложность состоит в способе выбора примеров поведения, так как можно исключить те примеры, которые являются необходимыми для генерации того ДКА, который является ответом на исходную задачу, и получить совершенно другой автомат. В следующем разделе предлагается метод итеративного выбора только значимых примеров поведения из всего изначального множества, решающий данную проблему.

В **разделе 3.2** приведено описание разработанного точного метода генерации ДКА по избыточному набору примеров поведения на основе сведения к SAT и с использованием подхода CEGAR. Как было указано ранее, обычно алгоритм уточнения абстракции по контрпримерам применяется для решения задач активного обучения. Задача генерации ДКА по заданным примерам поведения, в свою очередь, относится к классу задач пассивного обучения. Однако, в настоящей диссертации предлагается метод, решающий задачу генерации ДКА по заданным примерам, использующий идеи подхода CEGAR.

Как и классический алгоритм CEGAR, предлагаемый метод итеративно уточняет модель, которая в настоящей диссертации является детерминированным конечным автоматом. Изначально расширенное префиксное дерево не содержит вершин, но достраивается на каждом шаге. На каждом шаге работы алгоритма предлагается с помощью сведения к SAT пытаться строить ДКА текущего размера по текущему префиксному дереву. Если такого ДКА не существует, то как

и раньше размер искомого автомата увеличивается на единицу и процесс поиска повторяется. Если же такой автомат найден, он проверяется на соответствие оставшимся примерам поведения. Если ДКА соответствует всем примерам поведения, то задача решена. Иначе, среди тех примеров поведения, которым построенный автомат не соответствует, выбирается один или несколько контрпримеров, по которым достраивается префиксное дерево, строится новая булева формула и поиск продолжается. Схема предложенного метода представлена на рисунке 8.

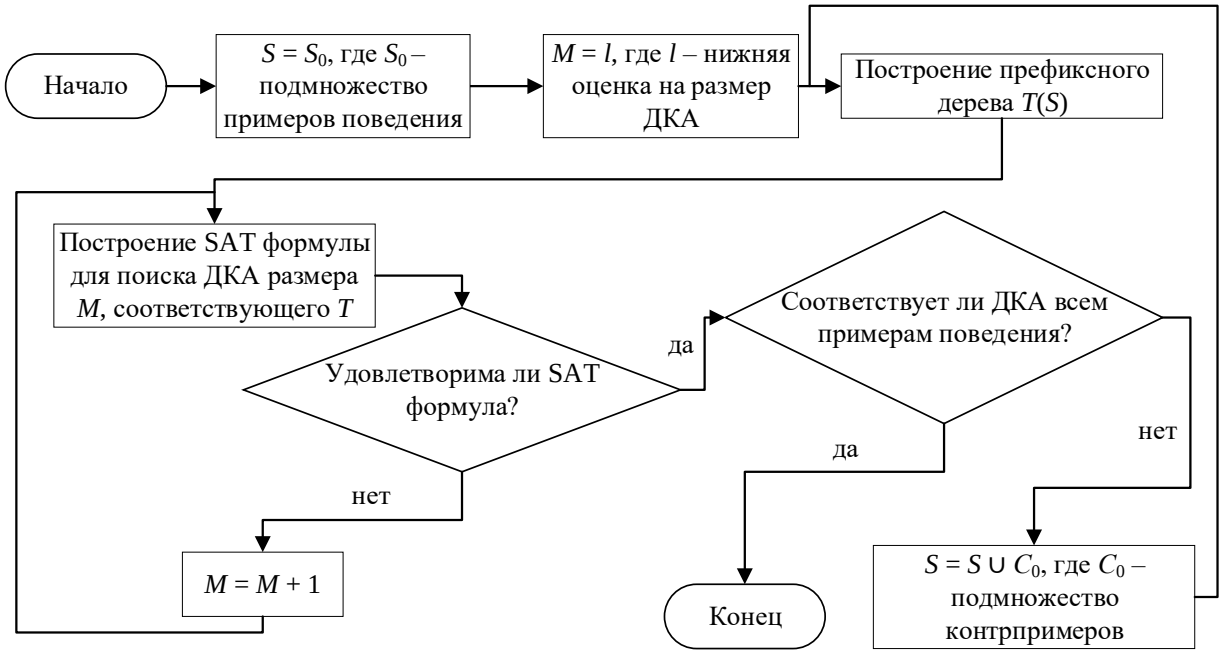


Рисунок 8 – Схема точного метода генерации ДКА по избыточному набору примеров поведения на основе сведения к SAT и с использованием подхода CEGAR

Необходимо заметить, что перезапускать программное средство для решения SAT при каждом достраивании префиксного дерева крайне неэффективно, так как при добавлении новых дизъюнктов в формулу, пространство поиска решения SAT только сужается, а, значит, нет необходимости начинать поиск выполняющей подстановки заново. Можно использовать инкрементальные программные средства для решения SAT, которые после нахождения некоторой выполняющей подстановки переходят в режим ожидания новых дизъюнктов и затем продолжают поиск решения уже для новой уточненной формулы с того места, где остановились в прошлый раз.

В **разделе 3.3** приведены описание реализации разработанного метода как части программного средства `DFA-Inductor-py` и результаты экспериментальных исследований разработанного метода. Для проведения экспериментальных исследований снова использовались случайно сгенерированные тестовые данные. Экспериментальные исследования показали, что при большом числе примеров поведения $S = |S_+| + |S_-| \geqslant 200 \times M$ метод на основе CEGAR работает как минимум в два раза быстрее метода, использующего сразу все примеры поведения. Более того, выигрыш от использования CEGAR увеличивается с ростом числа примеров поведения.

В **четвертой главе** диссертации дается постановка задачи генерации всех неизоморфных детерминированных конечных автоматов, удовлетворяющих заданным примерам поведения, а затем описываются разработка, реализация и экспериментальные исследования двух методов, решающих поставленную задачу.

В **разделе 4.1** приведена формальная постановка задачи генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения. Как уже было показано ранее, ДКА минимального размера является максимально точным обобщением имеющихся данных, выраженных с помощью примеров поведения. Однако в случае, когда примеры поведения недостаточно хорошо описывают искомый автомат, может существовать несколько различных неизоморфных удовлетворяющих автоматов минимального размера. В таком случае рациональным решением может быть построение всех таких автоматов для проведения дальнейшего анализа. На рисунке 9 приведены все неизоморфные ДКА минимального размера, построенные по заданным примерам поведения.

Ранее методы генерации всех различных ДКА с минимальным числом состояний по заданным примерам поведения не предлагались. Более того, без использования предикатов нарушения симметрии на основе BFS или DFS, разрешающих рассматривать для каждого класса эквивалентности по изоморфизму одного представителя вместо факториала, не представляется возможным эффективная генерация всех различных ДКА с помощью сведения к SAT.

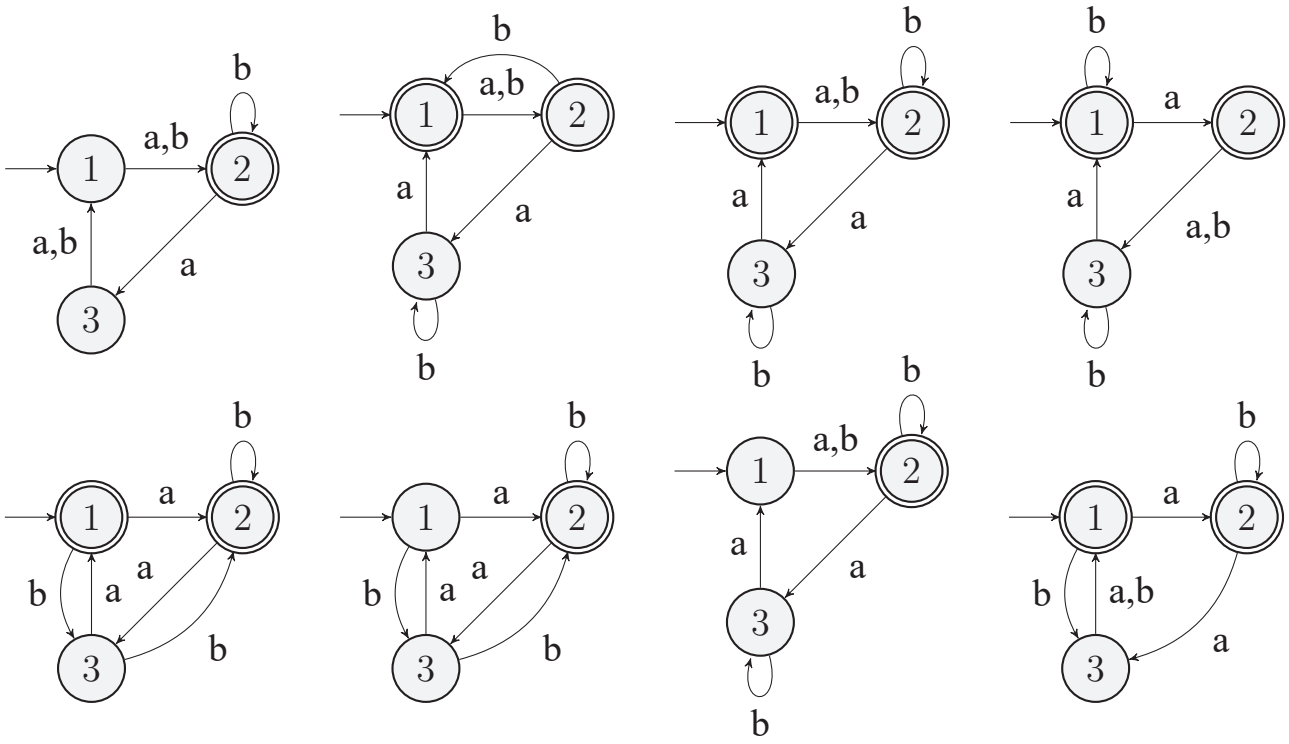


Рисунок 9 – Все неизоморфные ДКА, соответствующие множествам примеров поведения $S_+ = \{a, bb, aaaa\}$ и $S_- = \{aa, bab\}$

В **разделе 4.2** приведено описание метода генерации всех неизоморфных ДКА минимального размера с использованием сведения к SAT и предикатов нарушения симметрии. Так как при использовании предикатов нарушения симметрии, построенных на основе кодирования алгоритма BFS (ровно как и на основе кодирования алгоритма DFS), может быть сгенерирован только ДКА, пронумерованный в порядке BFS (DFS), то достаточно заблокировать уже найденный автомат и исключить его (вместо со всеми изоморфными ему автоматами) из пространства поиска. Из найденной выполняющей постановки построить автомат можно, используя только переменные переходов $y_{i,l,j}$ и переменные допуска z_i. Тогда достаточно из всей найденной выполняющей подстановки запретить только значения этих переменных. Если с помощью $\varphi(q)$ обозначить значение переменной q в найденной подстановке φ и определить множество $\mathcal{Y} = \{y_{i,l,j} | i, j \in [M] \wedge l \in \Sigma \wedge \varphi(y_{i,l,j}) = 1\}$, то *блокирующий дизъюнкт* можно определить следующим об-

разом:

$$\bigwedge_{y \in \mathcal{Y}} \neg y \wedge \bigwedge_{i \in [M]} \neg \varphi\left(z_i\right).$$

Также как и в случае с методом, использующим подход CEGAR, при добавлении блокирующего дизъюнкта пространство поиска только сокращается, поэтому можно не перезапускать программное средство, а использовать его в инкрементальном режиме.

Еще одним важным применением предложенного метода является проверка того, что сгенерированный автомат является единственным существующим автоматом минимального размера, соответствующим заданным примерам поведения. Единственность ДКА в таком случае говорит о том, что имеющиеся данные хорошо описывают найденный автомат.

В **разделе 4.3** приведены описание реализации разработанных методов как частей программного средства `DFA-Inductor-py` и результаты экспериментальных исследований разработанных методов. Так как ранее не существовало эффективных методов решения задачи генерации всех различных ДКА по заданным примерам поведения, то в качестве базового для оценки производительности предложенного метода был разработан переборный алгоритм с возвратами, который ищет все различные ДКА по заданным примерам поведения без использования сторонних программных средств.

Результаты экспериментальных исследований представлены в таблице 2. Эксперименты проводились для трех различных групп экземпляров задачи генерации ДКА по заданным примерам поведения. В первой группе число примеров поведения относительно размера генерируемого автомата имело соотношение $S = 5 \times M$, во второй — $S = 10 \times M$, в третьей $S = 25 \times M$. Столбец «>1» показывает процент экземпляров задачи, где существует больше одного неизоморфного автомата.

Результаты экспериментов позволяют сделать несколько выводов:

а) впервые успешно решена задача генерации всех различных ДКА минимального размера по заданным примерам поведения;

Таблица 2 – Медианное время нахождения всех различных ДКА с помощью метода на основе сведения к SAT с перезапуском программного средства (REST), метода на основе сведения к SAT с использованием инкрементального программного средства (INC) и переборного метода с возвратами (BTR)

M	$S = 5 \times M$				$S = 10 \times M$				$S = 25 \times M$			
	>1	REST	INC	BTR	>1	REST	INC	BTR	>1	REST	INC	BTR
5	53	2,3	2,0	0,8	40	3,6	3,3	1,3	17	4,1	3,4	1,5
6	56	2,8	2,4	2,1	31	4,7	3,9	1,7	27	5,4	4,3	1,7
7	87	3,9	2,5	4,1	27	3,7	3,0	3,1	13	7,4	6,7	2,5
8	80	4,6	3,7	87,2	34	7,0	6,5	41,7	16	10,1	8,9	11,6
9	91	7,6	3,9	475,1	50	7,7	6,4	121,6	10	13,8	13,0	61,4
10	89	15,7	5,3	2756,2	47	8,6	7,0	974,7	11	18,8	16,1	276,8
11	94	19,9	7,3	—	63	18,5	13,8	3108,0	9	24,5	21,9	1158,4
12	90	28,0	9,9	—	49	22,3	16,7	—	8	33,5	27,2	3289,1
13	92	185,5	18,1	—	57	36,9	22,6	—	12	62,0	51,4	—
14	87	408,5	49,0	—	71	85,1	41,8	—	4	67,0	56,2	—
15	95	571,1	174,1	—	69	193,3	95,7	—	6	29,2	26,2	—

б) оба метода, использующие сведение к SAT, значительно превосходят по производительности переборный метод;

в) использование инкрементального программного средства, как и предполагалось, дает заметное преимущество относительно подхода с перезапуском программного средства, что объясняется сохранением промежуточного состояния инкрементальным программным средством после нахождения некоторого ДКА;

г) чем больше примеров поведения дано для генерации ДКА, тем реже случается ситуация, когда существует несколько различных ДКА, соответствующих им.

В **заключении** приведены основные результаты работы, которые заключаются в следующем:

а) Разработаны предикаты нарушения симметрии, основанные на кодировании алгоритмов обхода графа в ширину и в глубину, для сокращения пространства поиска при решении задачи выполнимости и точные методы генерации ДКА по заданным примерам поведения, использующие данные предикаты. Предикаты нарушения симметрии, основанные на кодировании алгоритма обхода графа в глубину имеют скорее теорети-

ческую значимость, так как не продемонстрировали никаких преимуществ, относительно предикатов, основанных на кодировании алгоритма обхода графа в ширину. Новый способ кодирования предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в ширину позволяет использовать асимптотически меньшее число дизъюнктов относительно предложенного ранее. Разработанный точный метод, использующий данные предикаты совместно с предикатами, учитывающими особенности дерева обхода графа в ширину, демонстрирует лучшую производительность относительно существовавших ранее методов.

б) Разработан точный метод генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам. Размер булевой формулы и число используемых переменных линейно возрастает при увеличении числа примеров поведения. Разработанный метод позволяет генерировать ДКА по избыточному набору примеров поведения, итеративно расширяя множество используемых примеров, достраивая расширенное префиксное дерево и используя программное средство для решения SAT в инкрементальном режиме. Так как множество используемых примеров поведения расширяется контрпримерами к построенным промежуточным ДКА, то в итоге для генерации используются только значимые примеры поведения. Таким образом, с помощью данного метода удается точно решать такие задачи, которые раньше не могли быть решены ввиду большого размера булевой формулы.

в) Разработан метод генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств решения задачи выполнимости. Ранее задача генерации всех неизоморфных ДКА минимального размера не имела эффективного решения ввиду того, что изоморфные автоматы, являясь одинаковыми по структуре и по

определяемому языку, программным средством для решения SAT считаются различными, что приводит к рассмотрению $\mathcal{O}(M!)$ изоморфных автоматов с M состояниями. Использование предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в ширину позволяет для каждого класса эквивалентности по изоморфизму оставить для рассмотрения единственного представителя вместо факториала. Таким образом, впервые был предложен метод решения задачи генерации всех неизомофрных ДКА минимального размера. Поиск всех неизоморфных автоматов может быть полезен для дальнейшего их анализа, либо для анализа имеющихся примеров поведения. Другим применением разработанного метода является возможность доказать единственность минимального автомата, соответствующего заданным примерам поведения.

г) Во время работы над диссертацией на языке *Python* было разработано программное средство с открытым исходным кодом `DFA-Inductor-py`, предназначенное для генерации ДКА по заданным примерам поведения. В состав средства входят различные модули, позволяющие решать задачу генерации ДКА по заданным примерам поведения, задачу генерации ДКА по избыточному набору примеров поведения и задачу генерации всех неизоморфных ДКА по заданным примерам поведения. В средстве реализованы различные предикаты нарушения симметрии — как предложенные ранее другими авторами, так и разработанные в рамках настоящей диссертации.

д) Результаты работы были использованы в учебном процессе на факультете информационных технологий и программирования Университета ИТМО в рамках курса «Проектирование автоматных программ» программы бакалавриата «Математические модели и алгоритмы в разработке программного обеспечения», что подтверждается актом об использовании.

е) Часть результатов работы использовалась при выполнении проекта SAUNA (“Integrated safety assessment and justification of nuclear power

plant automaton”), выполненного исследовательской группой “IT in Automation” кафедры электротехники и автоматики университета Аалто, Финляндия, в рамках Финской программы исследований безопасности атомных электростанций — SAFIR2018, что подтверждается письмом руководителя исследовательской группы “IT in Automation” В. В. Вяткина.

ж) Результаты работы также использовались при выполнении под руководством автора диссертации гранта Российского фонда фундаментальных исследований (проект 183700425 «Разработка эффективных методов машинного обучения для построения детерминированных конечных автоматов на основе решения задачи выполнимости», 2018–2020 гг.) и в рамках проектов по программе повышения конкурентоспособности ведущих российских университетов среди ведущих мировых научно-образовательных центров «5-100».

Точный метод генерации ДКА по заданным примерам поведения, использующий предикаты нарушения симметрии на основе алгоритма обхода графа в ширину, демонстрирует лучшую производительность относительно известных ранее методов и позволяет генерировать автоматы большего размера за меньшее время. Метод генерации ДКА по избыточному набору примеров поведения позволяет генерировать автоматы по таким данным, по которым известные ранее методы не могли построить ДКА в принципе ввиду большого размера булевой формулы. Метод генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, является первым известным методом, позволяющим сгенерировать все неизоморфные автоматы — существовавшие ранее методы могут быть адаптированы для поиска всех ДКА, но их использование приводит к возникновению комбинаторного взрыва числа рассматриваемых автоматов ввиду отсутствия эффективных предикатов нарушения симметрии. Целью настоящего диссертационного исследования являлось повышение эффективности точных методов генерации детерминированных конечных автоматов по заданным примерам поведения посредством сокращения пространства поиска при

решении задачи выполнимости. Таким образом, согласно результатам экспериментальных исследований, цель можно считать успешно достигнутой.

Благодарности. Автор выражает благодарность своему научному руководителю, кандидату технических наук, В. И. Ульянцеву за неоценимую помощь в исследовательской деятельности и в написании настоящей работы, профессору, доктору технических наук, А. А. Шалыто за наставничество, коллегам по международной лаборатории «Компьютерные технологии» кандидату технических наук Д. С. Чивилихину за многочисленные консультации по вопросам исследовательской деятельности и за помощь с переводом текста реферата на английский язык, К. И. Чухареву и Д. М. Суворову за помощь в подготовке иллюстративного материала настоящей диссертации, А. И. Бугровскому и Т. Р. Галимжанову за помощь в документообороте во время процедуры подготовки защиты настоящей диссертационной работы, иностранным коллегам кандидату физико-математических наук А. И. Игнатьеву и доктору Ж. Маркешу-Сильве за организацию стажировки и активное в ней участие, в рамках которой проводилась часть исследовательской деятельности автора.

Также автор выражает благодарность своим родителям Тимуру и Валентине за привитую любовь к знаниям и математике, Лазаревой Екатерине за то, что всегда была рядом, своим друзьям Александру Б., Александру С., Анастасии, Валентину, Евгению, Марине, Матвею, Наталье, Талгату и Татьяне за оказанную моральную поддержку.

Публикации автора по теме диссертации

Публикации в зарубежных изданиях, индексируемых в базах цитирования Web of Science или Scopus

1. *Ulyantsev*, *V.*, *Zakirzyanov*, *I.*, *Shalyto*, *A.* BFS-Based Symmetry Breaking Predicates for DFA Identification // Language and Automata Theory and Applications — 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings. Vol. 8977. — Springer, 2015. — P. 611–622. — (Lecture Notes in Computer Science).

2. *Zakirzyanov*, *I.*, *Shalyto*, *A.*, *Ulyantsev*, *V.* Finding All Minimum-Size DFA Consistent with Given Examples: SAT-Based Approach // Software Engineering and Formal Methods — SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers. Vol. 10729. — Springer, 2017. — P. 117–131. — (Lecture Notes in Computer Science).

3. *Zakirzyanov*, *I.*, *Morgado*, *A.*, *Ignatiev*, *A.*, *Ulyantsev*, *V.*, *Marques-Silva*, *J.* Efficient Symmetry Breaking for SAT-Based Minimum DFA Inference // Language and Automata Theory and Applications — 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings. Vol. 11417. — Springer, 2019. — P. 159–173. — (Lecture Notes in Computer Science).

4. *Ovsiannikova*, *P.*, *Chivilikhin*, *D.*, *Ulyantsev*, *V.*, *Stankevich*, *A.*, *Zakirzyanov*, *I.*, *Vyatkin*, *V.*, *Shalyto*, *A.* Active Learning of Formal Plant Models For Cyber-Physical Systems // 16th IEEE International Conference on Industrial Informatics, INDIN 2018, Porto, Portugal, July 18-20, 2018. — IEEE, 2018. — P. 719–724.

Публикации в журналах из перечня ВАК

5. *Закирзянов*, *И. Т.* Построение детерминированных конечных автоматов по примерам поведения с использованием подхода уточнения абстракции по контрпримерам // Научно-технический вестник информационных технологий, механики и оптики. — 2020. — Т. 20, № 3. — С. 394–401.

Прочие публикации

6. *Закирзянов*, *И. Т.* Разработка предикатов нарушения симметрии на основе поиска в ширину для построения детерминированных конечных автоматов // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2015.

7. *Закирзянов*, *И. Т.* Эмпирическая оценка производительности программных средств решения задачи SAT для синтеза ДКА // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2016.

8. *Закирзянов*, *И. Т.*, *Ульянцев*, *В. И.* Сравнительный анализ методов задания ограничений типа at-most-one на примере задачи построения ДКА с использованием программных средств решения SAT // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2017.

9. *Закирзянов*, *И. Т.* Разработка предикатов нарушения симметрии для построения автоматных моделей программного обеспечения по заданной спецификации в виде темпоральных формул // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2018.

10. *Закирзянов*, *И. Т.* Построение минимального ДКА на основе сведения к SAT с использованием предположений // Сборник тезисов докладов конгресса молодых ученых. — 2020.

Публикации автора по другим темам

Публикации в зарубежных изданиях, индексируемых в базах цитирования Web of Science или Scopus

11. *Kachalsky*, *I.*, *Zakirzyanov*, *I.*, *Ulyantsev*, *V.* Applying Reinforcement Learning and Supervised Learning Techniques to Play Hearthstone // 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017. — IEEE, 2017. — P. 1145–1148.

Свидетельства о государственной регистрации программ для ЭВМ

12. *Программный комплекс методов машинного обучения DFA-Inductor для построения детерминированных конечных автоматов*. Программа / И. Т. Закирзянов, В. И. Ульянцев, А. А. Шалыто. — № 2016618241 ; опубл. 20.09.2016.

13. *Программное средство для генерации дискретной формальной модели объекта управления по примерам поведения*. Программа / И. П. Бужинский, И. Т. Закирзянов, В. А. Миронович, С. В. Казаков, М. А. Лукин, А. С. Буздалова, В. И. Ульянцев, А. А. Шалыто. — № 2018619728 ; опубл. 10.08.2018.

Synopsis

Thesis overview

Relevance. Finite-state machines are one of the fundamental concepts in discrete mathematics, informatics, and software engineering. Besides a direct role in the formal languages theory and design of computing devices, various finite-state machine models are practically used nowadays for development and analysis of software.

For example, finite-state machines are used for development of software models for controllers and mission-critical systems[1], protocol specification[2], behavior modeling for complex high-level systems[3]. The main advantages of using finite-state machines include their relative simplicity for human perception and the possibility of automated formal verification of model properties (model checking)[4].

In many cases the finite-state model is created by a developer manually, an example is the automata-based programming paradigm[5]. Other applications assume automated inference of a finite-state machine: extraction of the model from existing data or systems. Among practical examples of finite-state model inference applications are: synthesis of software models for controllers from manually or automatically collected behavior examples[6], synthesis of formal models for plants in control systems[7], analy-

[1] *Polykarpova*, *N.*, *Shalyto*, *A.* Automata-based Programming. St. Petersburg : Piter, 2010. 176 p. In Russian; *Patil*, *S.*, *Dubinin*, *V.*, *Vyatkin*, *V.* Formal Modelling and Verification of IEC61499 Function Blocks with Abstract State Machines and SMV — Execution Semantics // SETTA. vol. 9409. Springer, 2015. P. 300–315. (Lecture Notes in Computer Science).

[2] *Jongmans*, *S.-S. T. Q.*, *Halle*, *S.*, *Arbab*, *F.* Automata-Based Optimization of Interaction Protocols for Scalable Multicore Platforms // COORDINATION. vol. 8459. Springer, 2014. P. 65–82. (Lecture Notes in Computer Science).

[3] *Heule*, *M.*, *Verwer*, *S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 825–856; *Wagner*, *F.*, *Schmuki*, *R.*, *Wagner*, *T.*, *Wolstenholme*, *P.* Modeling Software with Finite State Machines: A Practical Approach. CRC Press, 2006. 392 p.

[4] *Clarke Jr*, *E. M.*, *Grumberg*, *O.*, *Kroening*, *D.*, *Peled*, *D.*, *Veith*, *H.* Model checking. Second edition. MIT press, 2018. 424 p. (Cyber Physical Systems Series).

[5] *Polykarpova*, *N.*, *Shalyto*, *A.* Automata-based Programming. St. Petersburg : Piter, 2010. 176 p. In Russian.

[6] *Chivilikhin*, *D.*, *Buzhinsky*, *I.*, *Ulyantsev*, *V.*, *Stankevich*, *A.*, *Shalyto*, *A.*, *Vyatkin*, *V.* Counterexample-guided inference of controller logic from execution traces and temporal formulas // ETFA. IEEE, 2018. P. 91–98.

[7] *Buzhinsky*, *I.*, *Vyatkin*, *V.* Modular plant model synthesis from behavior traces and temporal properties // ETFA. IEEE, 2017. P. 1–7; *Buzhinsky*, *I.*, *Vyatkin*, *V.* Automatic Inference of Finite-State Plant Models From

sis of interaction models for complex software systems[8] and network protocols[9], etc. Active research and development in finite-state machine inference algorithms began in the 1970's. In 1978 Gold proved that the problem of inferring a deterministic finite automaton (DFA) with the minimal number of states from given behavior examples is NP-complete[10]. This theoretical result emphasizes the complexity of the finite-state machine inference problem in the general case, actualizing the development of practically applicable algorithms.

Since then, a large variety of both heuristic and metaheuristic algorithms for finite-state machine inference from behavior examples have been proposed, now comprising a whole class of algorithms in discrete mathematics. In recent years a group of methods have been developed that are based on reductions of the problem of finding a minimal (in terms of the number of states) finite-state machine to other NP-complete problems. The most efficient approaches are based on reductions to the Boolean satisfiability problem.

The Boolean satisfiability problem (SAT) consists of determining whether a satisfying assignment exists for a given Boolean formula. According to the Cook-Levin theorem from 1971, the Boolean satisfiability problem is NP-complete. This fact stimulated and actualized the development of practically applicable software tools for solving SAT.

SAT solving methods have been developed even before any theoretical complexity bounds were introduced. In 1962 the Davis-Putnam-Logemann-Loveland (DPLL)[11]

Traces and Temporal Properties // IEEE Transactions on Industrial Informatics. 2017. Vol. 13, no. 4. P. 1521–1530.

[8] *Heule*, *M.*, *Verwer*, *S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 825–856; *Cook*, *J. E.*, *Wolf*, *A. L.* Discovering Models of Software Processes from Event-Based Data // ACM Transactions on Software Engineering Methodology. 1998. Vol. 7, no. 3. P. 215–249; *Bertolino*, *A.*, *Inverardi*, *P.*, *Pelliccione*, *P.*, *Tivoli*, *M.* Automatic synthesis of behavior protocols for composable web-services // ESEC/SIGSOFT FSE. ACM, 2009. P. 141–150.

[9] *Sivakorn*, *S.*, *Argyros*, *G.*, *Pei*, *K.*, *Keromytis*, *A. D.*, *Jana*, *S.* HVLearn: Automated Black-Box Analysis of Hostname Verification in SSL/TLS Implementations // IEEE Symposium on Security and Privacy. IEEE Computer Society, 2017. P. 521–538.

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

[11] *Davis*, *M.*, *Logemann*, *G.*, *Loveland*, *D. W.* A machine program for theorem-proving // Communications of the ACM. 1962. Vol. 5, no. 7. P. 394–397.

algorithm has been proposed for solving SAT: it is a complete algorithm with the capability of returning a satisfying assignment for satisfiable formulas, which is based the unit propagation rule and elimination of clean variables for accelerating the search. This algorithm heuristically searches the space of all variable assignments and stops if a satisfying assignment has been found; if the algorithm completed the search and did not find a satisfying assignment, the formula is concluded to be unsatisfiable.

In the 1990's the Conflict Driven Clause Learning (CDCL)[12] algorithm has been proposed based on the DPLL algorithm: CDCL saves clauses derived from analyzing conflicts encountered in DPLL. Such clauses allow the algorithm to much earlier decide the unsatisfiability of the formula with current assumptions, and switch to considering new assumptions.

The CDCL algorithm is the base of all modern software tools for solving SAT (SAT solvers). Unlike the situation for all other NP-hard problems, the research community annually holds SAT solver competitions[13]. This fact actualizes development of problem solving methods based on reductions to SAT: to increase the efficiency of the method one does not need to change its source code, it will be sufficient to simply change the SAT solver to a more modern one.

However, in the methodology of reducing a given problem to SAT the fundamental question is not the used SAT solver, but the way an instance of the problem is translated to a Boolean formula, for which a satisfying assignment will be searched. For example, in recent years for the problem of inferring a minimal DFA from given behavior examples a basic reduction to SAT[14] and several of its extensions[15] have been

[12] *Marques-Silva*, *J. P.*, *Sakallah*, *K. A.* GRASP — a new search algorithm for satisfiability // ICCAD. IEEE, 1996. P. 220–227.

[13] The International SAT Competition Web Page. URL: `http://www.satcompetition.org/` (дата обр. 15.10.2020); SAT Competition 2020. URL: `https://satcompetition.github.io/2020` (дата обр. 15.10.2020).

[14] *Heule*, *M.*, *Verwer*, *S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Vol. 6339. Springer, 2010. P. 66–79. (Lecture Notes in Computer Science).

[15] *Heule*, *M.*, *Verwer*, *S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 825–856; *Ulyantsev*, *V.* Finite-state machine inference using Boolean satisfiability and constraint satisfaction problem solvers : PhD thesis / Ulyantsev V. ITMO University, 2015. In Russian.

developed. The modifications proposed the use of symmetry-breaking predicates: auxiliary clauses that define a problem-dependent way of search space reduction.

These search space reduction techniques proved to be very effective in practice, allowing inference of finite-state machines of larger size than it was possible before. However, consequent analysis shows that these techniques are not optimal, and the area of applicability of state-of-the-art methods for exact DFA synthesis from behavior examples is still quite limited (which is, first of all, explained by the NP-completeness of the problem). Thus, the topic of this thesis, which continues the described research of recent years and aims to extend the capabilities of methods for search space reduction and DFA inference, is **relevant** to this research domain.

State of the art. The problem of inferring a deterministic finite automaton from behavior examples given by two sets S_+ and S_- consists in finding a DFA with the minimal number of states, such that all strings from the set S_+ are accepted by the automaton, and all strings from S_- are rejected. This problem was first formulated in Gold's paper in 1967[16].

The first known algorithm for solving this problem has been proposed in the work by Trakhtenbrot and Barzdin in 1970[17]: the `TB`-algorithm. However, this algorithm only covers one special case of the problem of DFA inference from behavior examples: sets S_+ and S_- must contain all words of length k over the alphabet Σ (a total of $|\Sigma|^k$ words). In this algorithm, as in the majority of the following ones, behavior examples are represented as an augmented prefix tree: a prefix tree, in which vertices can be accepting, rejecting, or intermediate. The `TB`-algorithm performs brute-force search of all possible pairs of states of the prefix tree, and merges equivalent states.

In 1978 Gold proved that the problem of finding a DFA of a given size (and thus of minimal size too) is NP-complete[18]. Due to this complexity result, development of new minimal DFA inference algorithm seized for a decade. In the following

[16] *Gold*, *E. M.* Language Identification in the Limit // Information and Control. 1967. Vol. 10, no. 5. P. 447–474.

[17] *Trakhtenbrot*, *B. A.*, *Barzdin*, *Y. M.* Finite Automata: Behavior and Synthesis. Amsterdam : North-Holland Publishing Company, 1973. 321 p. (Fundamental studies in computer science).

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

years researchers started developing inexact heuristic algorithms, which do not guarantee the minimality of the found DFA. Among these algorithms are: `traxbar`[19], `RPNI`[20], `EDSM`[21], `exbar`[22], `Windowed-EDSM`[23]. All mentioned algorithms are based on merging the states of the augmented prefix tree. The states for merging are selected heuristically: this explains the high efficiency of these algorithms and their inexactness.

Another common approach to DFA inference from behavior examples is the application of metaheuristic algorithms. For example, evolutionary[24] and ant colony optimization[25] algorithms have been proposed. Metaheuristic algorithms are also inexact: they do not even guarantee that any solution will be found in finite time.

Scientists from Netherlands Heule and Verwer proposed in 2010 an algorithm `DFASAT` that can infer a DFA of guaranteed minimal size from arbitrary behavior examples[26]. The algorithm is based on a reduction of the problem of DFA inference from behavior examples to SAT, where an arbitrary step is a reduction to graph coloring proposed in 1997[27]. For reducing the size of the search space during SAT solving Heule and Verwer proposed a number of symmetry breaking approaches: consistency graph, auxiliary clauses, search for a large clique. However, even with all these techniques, in reasonable time `DFASAT` can infer automata with no more than ten states. In order to

[19] *Lang*, *K. J.* Random DFA's Can Be Approximately Learned from Sparse Uniform Examples // Proceedings of the Fifth ACM Workshop on Computational Learning Theory. ACM, 1992. P. 45–52.

[20] *Oncina*, *J.*, *García*, *P.* Inferring Regular Languages in Polynomial Update Time // Pattern Recognition and Image Analysis. Vol. 1. 1992. P. 49–61. (Series in Machine Perception and Artificial Intelligence).

[21] *Lang*, *K. J.*, *Pearlmutter*, *B. A.*, *Price*, *R. A.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm // ICGI. vol. 1433. Springer, 1998. P. 1–12. (Lecture Notes in Computer Science).

[22] *Lang*, *K. J.* Faster algorithms for finding minimal consistent DFAs : tech. rep. / NEC Research Institute. 1999 .

[23] *Cicchello*, *O.*, *Kremer*, *S. C.* Beyond EDSM // ICGI. vol. 2484. Springer, 2002. P. 37–48. (Lecture Notes in Computer Science).

[24] *Lucas*, *S. M.*, *Reynolds*, *T. J.* Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005. Vol. 27, no. 7. P. 1063–1074; *Gómez*, *J.* An Incremental-Evolutionary Approach for Learning Deterministic Finite Automata // IEEE Congress on Evolutionary Computation. IEEE, 2006. P. 362–369.

[25] *Chivilikhin*, *D.*, *Ulyantsev*, *V.* Learning Finite-State Machines with Ant Colony Optimization // Swarm Intelligence. Vol. 7461. Springer Berlin / Heidelberg, 2012. P. 268–275. (Lecture Notes in Computer Science).

[26] *Heule*, *M.*, *Verwer*, *S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Vol. 6339. Springer, 2010. P. 66–79. (Lecture Notes in Computer Science).

[27] *Coste*, *F.*, *Nicolas*, *J.* Regular Inference as a graph coloring problem // Workshop on Grammatical Inference, Automata Induction, and Language Acquisition (ICML'97). 1997.

simplify the problem, Heule and Verwer proposed to use the EDSM algorithm to make several merges in the augmented prefix tree prior to using the SAT approach: however, this leads to the whole algorithm becoming inexact, and the DFA minimality guarantee is lost.

Later, the author of the thesis together with his supervisor Vladimir Ulyantsev and professor Anatoly Shalyto proposed symmetry breaking predicates based on the breadth-first search algorithm (BFS) [1]. For each DFA with M states there exist $\mathcal{O}(M!)$ isomorphic automata, which are distinct from the point of view of a SAT solver. The proposed symmetry breaking predicates yield a considerable reduction of the search space during SAT solving: instead of a factorial of isomorphic automata, only one representative of each isomorphism equivalence class is considered during search. This one representative is a DFA with states enumerated in the order of BFS traversal. This method is the most efficient among known methods for exact DFA inference from behavior examples, and allows inference of automata with up to 40 states. However, analysis showed that the proposed naïve SAT encoding of BFS predicates is not optimal: the resulting Boolean formula is too large. Furthermore, symmetry breaking predicates based on depth-first search (DFS) have not been considered.

Among other drawbacks of SAT-based DFA inference methods an important one is the dependence of the size of the Boolean formula from the size of the behavior examples. Sets of behavior examples are often excessive and contain unnecessary words. Literature analysis showed that no intellectual methods of choosing a subset of behavior examples have been proposed earlier. A possible solution is the application of *counterexample-guided abstraction refinement* (CEGAR) – an iterative algorithm initially proposed for software model inference[28].

The opposite situation is inference of a DFA from small sets of behavior examples. In this case it may be beneficial to infer all corresponding minimal non-isomorphic DFA for further analysis. Also, the existence of a single unique DFA of minimal size

[28] *Clarke*, *E. M.*, *Grumberg*, *O.*, *Jha*, *S.*, *Lu*, *Y.*, *Veith*, *H.* Counterexample-Guided Abstraction Refinement // CAV. vol. 1855. Springer, 2000. P. 154–169. (Lecture Notes in Computer Science); *Mandrykin*, *M.*, *Mutilin*, *V.*, *Khoroshilov*, *A.* Introduction to CEGAR — Counter-Example Guided Abstraction Refinement // Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2013. Vol. 24. P. 219–292. In Russian.

for a given set of training data illustrates the quality of the latter. However, the problem of inferring all non-isomorphic DFA has not been considered before by the research community, and thus no efficient methods have been proposed.

The **aim** of this thesis is to increase the efficiency of exact methods for deterministic finite automata inference from given behavior examples by means of reducing the search space during Boolean satisfiability problem solving. In order to achieve this aim, the following **tasks** have been defined and completed:

a) Development of symmetry breaking predicates based on SAT encodings of the breadth-first and depth-first graph search algorithms for reducing the search space during SAT solving. Development and implementation of exact methods for DFA inference from behavior examples using said predicates, and experimental evaluation of developed methods.
b) Development and implementation of an exact method for DFA inference from an excessive set of behavior examples using a reduction to SAT and counterexample-guided abstraction refinement. Experimental evaluation of the developed method.
c) Development and implementation of a method for inferring all non-isomorphic DFA of minimal size from given behavior examples using symmetry breaking predicates and SAT solvers. Experimental evaluation of the developed method.

The **object of the study** is the problem of DFA inference from given behavior examples.

The **subject of the study** are exact DFA inference methods that use Boolean satisfiability solvers.

Compliance with specialty requirements. The thesis complies with §10 “Development of foundations of mathematical theory of formal languages and grammars, finite automata theory, and graph theory”.

Principal statements of the thesis:

a) An *approach* for constructing symmetry breaking predicates based on SAT encodings of breadth-first and depth-first graph search algorithms for the pur-

pose of reducing the search space during SAT solving. Exact *methods* for DFA inference from given behavior examples that use the said predicates.

b) An exact *method* for DFA inference from an excessive set of behavior examples using a reduction to SAT and counterexample-guided abstraction refinement.

c) A *method* for inferring all non-isomorphic DFA of minimal size from given behavior examples using symmetry breaking predicates and SAT solvers.

The **scientific novelty** of the thesis is as follows:

a) Symmetry breaking predicates based on a SAT encoding of the depth-first search algorithm have not been proposed before. The proposed symmetry breaking predicates based on the breadth-first search algorithm are expressed with a Boolean formula that is comprised of an asymptotically smaller number of clauses in comparison to the existing approach. Moreover, new symmetry breaking predicates that utilize special features of the breath-first search tree are proposed.

b) Previously, no exact DFA inference methods for the case of an excessively large number of behavior examples have been proposed. The use of counterexample-guided abstraction refinement together with a reduction to SAT allows for DFA inference from an excessive set of behavior examples through iteratively adding only meaningful examples until a DFA satisfying the whole excessive set will be inferred.

c) No methods for inferring all non-isomoprhic DFA of minimal (or any fixed) size satisfying given behavior examples have been previously proposed.

Research methodology and methods. The methodological basis of the thesis is formed by the principles of formalization, generalization, deductive and inductive substantiation of statements, experimental evaluation and analysis of experimental results. This thesis uses methods of automata theory, graph theory, probability theory, discrete mathematics, object oriented programming, experiment design and analysis.

Soundness and correctness of scientific results obtained in this thesis is confirmed by correct justification of problem settings, precise formulation of criteria, as well as by the results of experimental evaluation of the proposed methods.

The **theoretical significance** of the thesis is that it proposes new methods for search space reduction during SAT solving for the problem of DFA inference, namely, symmetry breaking predicates based on SAT encodings of breadth-first and depth-first graph search algorithms:

- a SAT encoding of the DFS-enumeration property of a DFA is proposed;
- a new SAT encoding of the BFS-enumeration property of a DFA is proposed that requires an asymptotically smaller number of clauses than before;
- a SAT encoding of various properties of the BFS traversal tree is proposed.

Furthermore, the thesis proposes an approach that combines a SAT-based DFA inference method with counterexample-guided abstraction refinement. Also, the developed symmetry breaking predicates enable the proposal of a method for the problem of inferring all non-isomorphic DFA of minimal size, which previously did not have efficient solving methods.

The **practical significance** of the thesis lies in efficiency improvement of exact methods for DFA inference from given behavior examples. Experiments showed that the proposed method for DFA inference from given behavior examples using BFS-based symmetry breaking methods is the most efficient one among known exact methods and allows inferring automata of larger size than previously proposed methods. The developed exact method for DFA inference from an excessive set of behavior examples using a SAT encoding and CEGAR allows efficient inference of automata in the case when the size of the behavior examples is too large, and the resulting Boolean formula is too large to process for modern SAT solvers. The developed method of inferring all non-isomorphic DFA of minimal size from given behavior examples using symmetry breaking predicates and SAT solvers is the first known method for inferring all non-isomorphic automata. It also allows one to estimate the completeness of the data represented by behavior examples by means of proving the existence or absence of a single minimal-sized DFA that describes the data.

Moreover, all developed methods and approaches may later be adapted for application to inference of more complex finite-state models[29]. For example, the proposed method for inferring all non-isomorphic DFA has been adapted to synthesize state machines that model the behavior of programmable logic controllers[30].

The results of the thesis **were used** in the project SAUNA ("Integrated safety assessment and justification of nuclear power plant automaton") by the "IT in Industrial Automation" research group of the department of electrical engineering and automation in Aalto University, Finland, in the framework of the Finnish research program in safety of nuclear power plants SAFIR2018[31]. In particular, one of the project tasks was to develop a model inference method for various control systems of nuclear power plants components from given behavior examples and linear temporal logic specification. This task was solved using counterexample-guided abstraction refinement in a similar way to the one proposed by the author of the thesis for DFA inference. This is confirmed by a letter from the principal investigator of the "IT in Industrial Automation" group, Valeriy Vyatkin.

Results of this thesis have also been used in the project No. 18-37-00425 "Development of efficient machine learning methods for deterministic finite automata inference based on Boolean satisfiability solving" (2018–2020) funded by the Russian Foundation for Basic Research and led by the author of the thesis.

Results have also been used in the framework of the governmental financial support of leading universities in Russia, grant 074-U01 (project "Bioinformatics, machine learning, programming technologies, coding theory, proactive systems", 2013–2017) and grant 08-08 (project "Methods, models and technologies of artificial intelligence in bioinformatics, social media, cyber-physical, biometric and speech systems", 2018–2020).

---

[29] *Ulyantsev, V.* Finite-state machine inference using Boolean satisfiability and constraint satisfaction problem solvers : PhD thesis / Ulyantsev V. ITMO University, 2015. In Russian.

[30] *Chivilikhin, D., Patil, S., Chukharev, K., Cordonnier, A., Vyatkin, V.* Automatic State Machine Reconstruction From Legacy Programmable Logic Controller Using Data Collection and SAT Solver // IEEE Transactions on Industrial Informatics. 2020. Vol. 16, no. 12. P. 7821–7831.

[31] http://safir2018.vtt.fi/

Results are also used in the “Design of automata-based programs” course of the “Mathematical models and algorithms in software engineering” Bachelor’s program of the Faculty of Information Technologies and Programming (supported by the official act of use).

Dissemination. The main results of the thesis were presented at the following venues:

a) 9th International Conference on Language and Automata Theory and Applications (LATA 2015). 2015, Nice, France.
b) 6th International Symposium “From Data to Models and Back (DataMod)”. 2017, Trento, Italy.
c) 16th IEEE International Conference on Industrial Informatics (INDIN 2018). 2018, Porto, Portugal.
d) 13th International Conference on Language and Automata Theory and Applications (LATA 2019). 2019, Saint Petersburg.
e) IV-VII Russian Young scientists congress. 2015-2018, Saint Petersburg.
f) IX Young scientists congress. 2020, Saint Petersburg.
g) XLVI Scientific and educational-practical conference of ITMO University. 2017, Saint Petersburg.
h) XLVIII Scientific and educational-practical conference of ITMO University. 2019, Saint Petersburg.

Personal contribution. The ideas of symmetry breaking predicates based on depth-first search, of the DFA inference method using these predicates, as well as implementation of an algorithm based on the proposed method and its experimental evaluation belong to the author. The idea of symmetry breaking predicates based on breadth-first search that require an asymptotically smaller number of clauses in the encoding, the idea of SAT encoding of BFS tree properties, and the idea of the DFA inference method that uses these results belong to the author of the thesis and João Marques-Silva; the implementation of algorithms based on proposed methods belong to the author, experimental evaluation belongs to the author and Alexey Ignatiev. The idea of an exact method for DFA inference from behavior examples using a reduction to SAT

and counterexample-guided abstraction refinement, the implementation of an algorithm based on the proposed method and experimental evaluation belong to the author. The idea of the exact method for inferring all minimal-sized DFA from given behavior examples using SAT solvers belongs to the author and the supervisor Vladimir Ulyantsev, implementation of the algorithm based on the proposed method and experimental evaluation belongs to the author. In papers co-authored with the supervisor, Vladimir Ulyantsev, his contribution consists of the general supervision of the work.

Publications. The main results of this thesis are presented in ten publications, four of which are indexed in Scopus, and one is published in a journal included into the List of the Higher Attestation Commission. Also the author has one publication indexed in Scopus on another topic from the machine learning domain.

Thesis contents

The **introduction** of the thesis contains the relevance of the research, a formulation of the aim, objectives, principal statements of the thesis, as well as scientific novelty, theoretical and practical significance of the work.

Chapter 1 presents an analytical review of the research domain and existing research results in the area of deterministic finite automata inference. It also introduces the terminology, main definitions and known results from a number of topics in informatics that are necessary for the description of methods and algorithms proposed in the thesis.

In the **section 1.1** a formal setting of the Boolean satisfiability problem is given along with necessary definitions and a brief description of the main approaches for solving this problem. Also this section describes an approach to solving NP-hard problems through their polynomial-time reduction to the Boolean satisfiability problem. Moreover, a brief overview of existing software tools for solving the Boolean satisfiability problem (SAT solvers) is given.

A Boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, where each clause is a disjunction of literals. The *Boolean satisfiability problem* (*satisfiability problem*, SAT) consists of determining, whether for a given Boolean formula in CNF there exists a satisfying assignment: a set of values of its variables for which the formula is true. SAT is the historically first problem for which NP-completenss has been proved: any problem from NP can be reduced to SAT in polynomial time. This fact explains the relevance of the development of more and more efficient SAT solvers. The research community annually holds competitions to determine the best SAT solver; this fact also contributes to continuous development of this area. The basis of all modern SAT solvers is the *conflict driven clause learning* (CDCL) strategy.

The approach to solving problems from NP, in which a reduction to SAT is devised and then a SAT solver is used to find a satisfying assignment of the constructed Boolean formula, often turns out to be much more efficient and simpler than the development of a practically applicable method that directly solves the original problem. Another important advantage of this approach is the fact that it is enough to write a reduction to the SAT once, and then, without applying any effort, use the progress of SAT solvers, choosing the most efficient one.

In the **section 1.2** basic concepts of deterministic finite automata and the formulation of the problem of inferring deterministic finite automata from behavior examples are given. Behavior examples for some DFA $\mathcal{D}$ are defined as two sets of words S_+ and S_- over the symbols of the DFA's alphabet, such that all words from S_+ belong the the language $\mathcal{L}(\mathcal{D})$ and must be accepted by the DFA, and all words from S_- do not belong to the language $\mathcal{L}(\mathcal{D})$ and must be rejected by the DFA. The problem of DFA inference from behavior examples consists in finding a DFA of minimal size (with the minimal number of states) that satisfies given behavior examples. Previously it has been shown that this problem is NP-complete, as well as the problem of finding a DFA of any given size that satisfies given behavior examples. An example of a DFA that satisfies behavior examples $S_+ = \{aba, bb, bba\}$ and $S_- = \{b, ba\}$ is shown in Figure 1. One can notice that an automaton with two states satisfying sets S_+ and S_- does not exist.

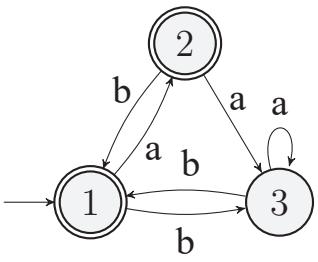


Figure 1 – An example of a minimal DFA satisfying behavior examples $S_+ = \{aba, bb, bba\}$ and $S_- = \{b, ba\}$

Section 1.3 presents an overview of existing heuristic and metaheurstic methods and algorithms for DFA inference from behavior examples. Heuristic algorithms include the *evidence-driven state merging* (EDSM) algorithm. Metaheuristic algorithms include evolutionary strategies, genetic algorithms, and ant colony optimization algorithms.

Note that the mentioned approaches are inexact: they do not guarantee that the found DFA contains the minimal number of states, and, in some cases, they do not even guarantee that any DFA satisfying the behavior examples will be found.

In the **section 1.4** an overview of existing methods for inferring DFA from behavior examples based on reduction to SAT is given. Unlike heuristic and metaheuristic approaches, these methods are exact: it is guaranteed that the automaton corresponding to the behavior examples will be found in finite time and will contain the minimum possible number of states.

The first step of considered methods is the construction of the *augmented prefix tree acceptor* (APTA): a tree-like data structure based on the ordinary prefix tree, in which each vertex is either not marked, or marked as accepting or rejecting. An example of an augmented prefix tree is shown in 2.

Further, starting from a certain lower bound on the size (in the simplest case, from one) the search for an automaton of the current size corresponding to the constructed APTA is performed. This process continues until a DFA is found that satisfies the given requirement. Iterative increase of the automaton size guarantees that the found automaton has the minimum size. As already mentioned, the problem of finding a DFA of a specific size for given behavior belongs to the NP class, and therefore can be solved

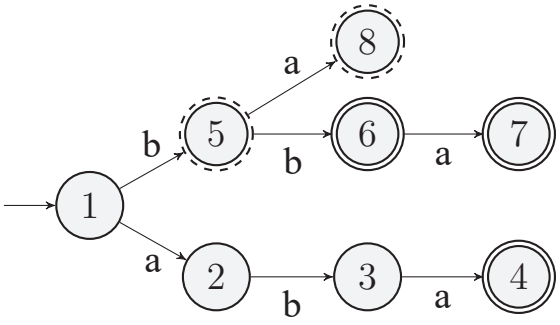


Figure 2 – An example of a prefix tree acceptor constructed from behavior examples $S_+ = \{aba, bb, bba\}$ and $S_- = \{b, ba\}$

by reducing it to some NP-hard problem. Until recently, the most efficient method was considered to be DFASAT[32], in which the authors proposed to first reduce the DFA inference problem to graph coloring (color the APTA in the minimal number of colors in such a way that vertices of one color would be merged into one state), and then reduce graph coloring to SAT. The scheme of the DFASAT method is shown in Figure 3.

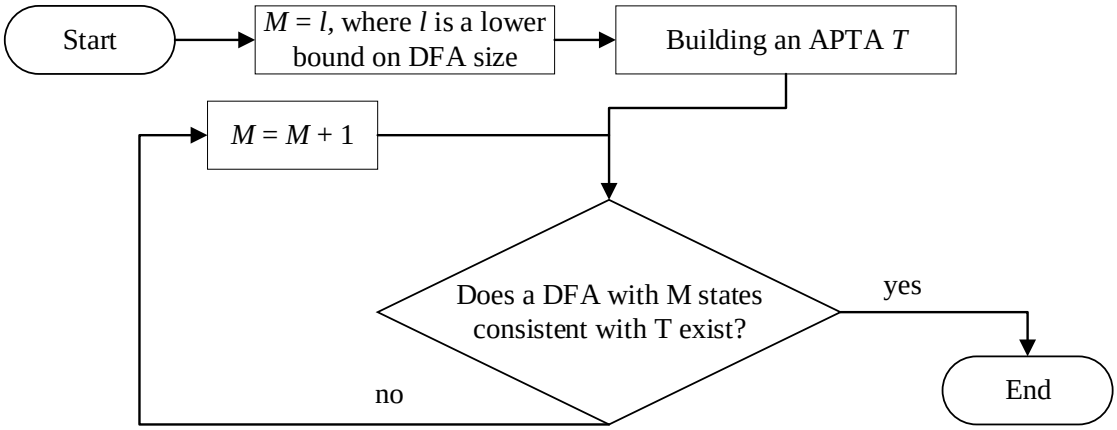


Figure 3 – Scheme of DFASAT, an exact SAT-based method for DFA inference from given behavior examples

Authors of DFASAT use several approaches for search space reduction:

a) adding several types of auxiliary clauses that do not influence the solution, but introduce additional constraints on possible variable values;
b) construction of the consistency graph allowing finding in advance some pairs of vertices of the prefix tree that cannot be combined into one state of the automaton;

[32] *Heule*, *M.*, *Verwer*, *S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Vol. 6339. Springer, 2010. P. 66–79. (Lecture Notes in Computer Science).

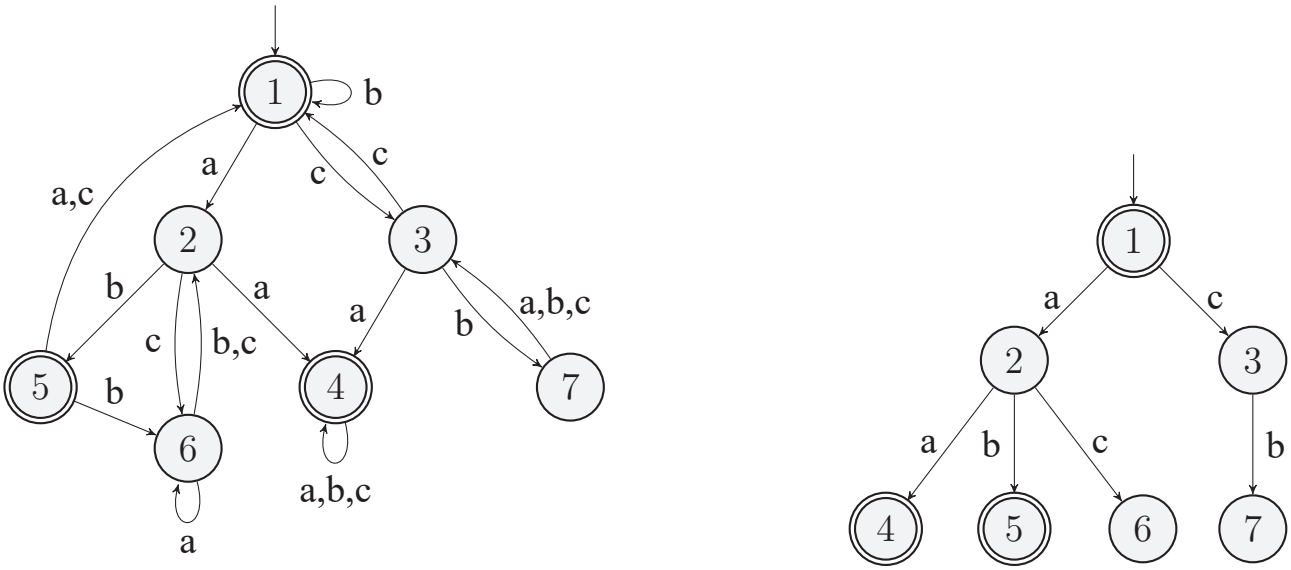
c) construction of some large clique in the consistency graph and fixing the enumeration of this clique's vertices.

Application of the aforementioned approaches yields a considerable increase of the efficiency of the original method, but they do not solve the fundamental problem of the existence of isomorphic automata. Two automata are called isomorphic if they differ only in the enumeration of states. Thus, if a DFA contains N states, then there exist $\mathcal{O}(N!)$ automata isomorphic to it. Despite that isomorphic automata do not differ from a practical point of view, for a SAT solver they are distinct.

The solution for this problem was found by means of development of symmetry breaking predicates based on *breadth-first search* (BFS) [1]. Inclusion of such predicates in the Boolean formula fixes the enumeration of considered automata in the BFS order, allowing to consider only on representative of each isomorphism equivalence class. The SAT encoding of these predicates requires $\mathcal{O}\left(M^3 + M^2 \times L^2\right)$ clauses, where N is the size of the prefix tree and M is the size of the sought DFA. The proposed symmetry breaking predicates were used to develop a SAT-based DFA inference method that has been implemented in a software tool `DFA-Inductor`[33]. This SAT-based DFA inference method that uses BFS symmetry breaking predicates is the most efficient exact DFA inference method, and it is the basis of all methods developed in this thesis. An example of a BFS-enumerated DFA is shown in Figure 4a, and Figure 4b shows the corresponding BFS traversal tree.

In the **section 1.5** the algorithm based on counterexample-guided abstraction refinement (CEGAR) is described. Methods based on CEGAR are applicable in a situation when one needs to infer a model that corresponds to some given requirements, and a checking system (oracle) is available. On the first step some model (possibly, random) is constructed. Then, an iterative process of refining the current model is initiated: on each step the compliance of the current model to the requirements is checked using the oracle. If the oracle check is successfully passed, then the sought model is found. Oth-

[33] *Zakirzyanov*, *I.*, *Ulyantsev*, *V.* DFA-Inductor. URL: `https://github.com/ctlab/DFA-Inductor` (visited on 10/02/2020).



(a) An example of a BFS-enumerated automaton

(b) BFS traversal tree for the automaton shown in Figure 4a

Figure 4 – An example of a BFS-enumerated automaton and the corresponding BFS traversal tree

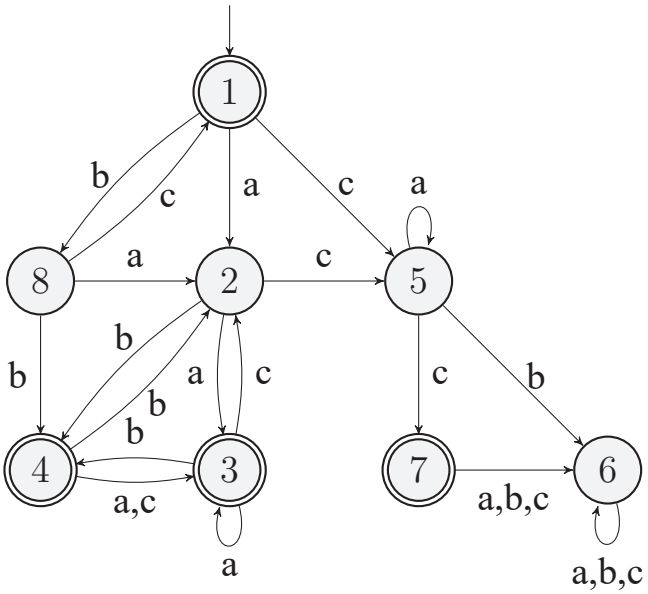
erwise, the oracle returns one or several counterexamples, which are consequently used to refine the model.

Chapter 2 describes the development, implementation, and experimental evaluation of DFA inference methods with different approaches to reducing the search space during SAT solving.

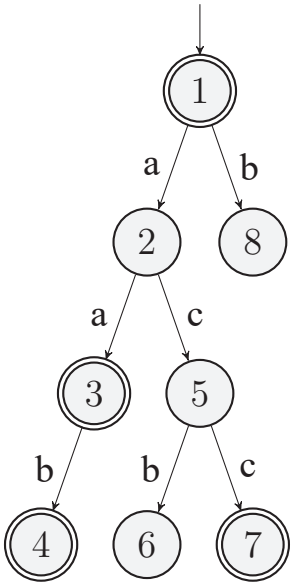
In the **section 2.1** a description of developed symmetry breaking predicates based on encoding the *depth-first search* (DFS) algorithm is provided. The use of BFS-based symmetry breaking predicates has previously allowed increasing the efficiency of the `DFASAT` method considerably. It was logical to consider as the next research task the development of symmetry breaking predicates based on the DFS algorithm and the method that uses them. An example of a DFS-enumerated DFA is shown in Figure 5a, and 5b shows the corresponding traversal tree.

The developed predicates are expressed as a CNF formula with $\mathcal{O}\left(M^4 + M^3 \times L^2\right)$ clauses, which is M times greater than the number of clauses for the BFS-based DFA symmetry breaking predicates.

In the **section 2.2** a description of the developed compact BFS-based symmetry breaking predicates is given.



(a) An example of a DFS-enumerated automaton



(b) The DFS traversal tree for the automaton in Figure 5a

Figure 5 – An example of a DFS-enumerated automaton and the corresponding DFS-traversal tree

The compactness of the developed predicates consists in reduction of the size of the formula that encodes the BFS-based symmetry breaking predicates from $\mathcal{O}\left(M^3 + M^2 \times L^2\right)$ to $\mathcal{O}\left(M^2 \times L\right)$ clauses. Analysis of constraints from the original encoding, which were expressed using $\mathcal{O}\left(M^3\right)$ и $\mathcal{O}\left(M^2 \times L^2\right)$ clauses, showed that some parameters that define the size of the formula are independent, and other parameters may be eliminated by introduction of new variables. For each such constraint a way of making it more compact was developed, leading to an overall decrease of formula size. Apart from this, the majority of clauses for the original encoding, which consisted of $\mathcal{O}\left(M\right)$ literals, were replaced by clauses comprised of two or three literals: this considerably influences the performance of SAT solvers, since such clauses are processed in constant time and are not stored in memory[34].

Section 2.3 describes developed search space reduction approaches for DFA inference, that are based on BFS tree structure properties, also on connections between the augmented prefix tree acceptor and the sought DFA. Figure 6 shows a complete BFS

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

tree of an arbitrary size on an arbitrary L-sized alphabet. The tree is called complete since each vertex has exactly L children.

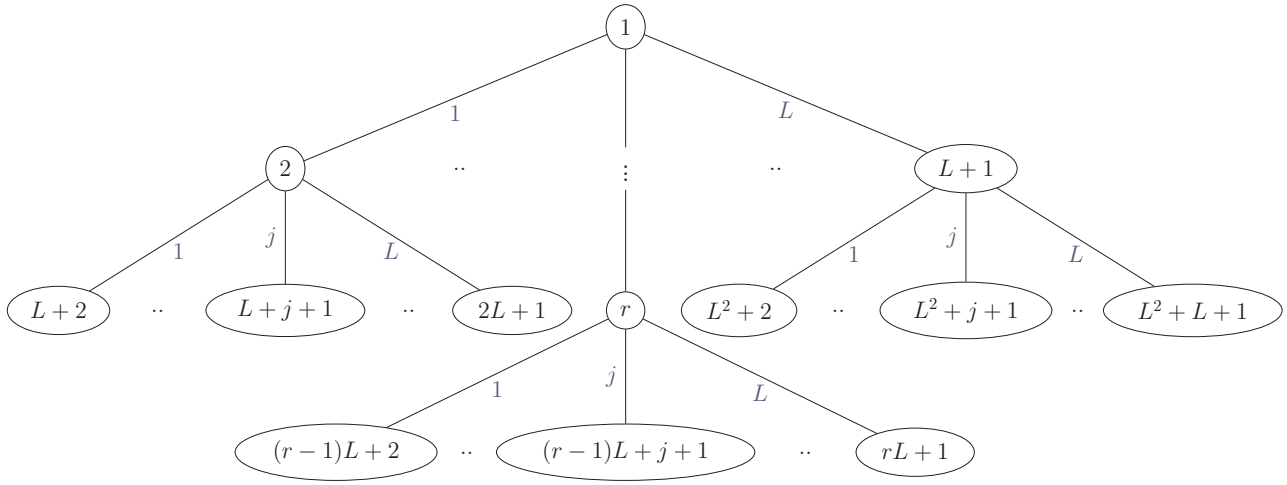


Figure 6 – A complete BFS tree with $|\Sigma| = L$

Following an analysis of this tree some properties and constraints that are characteristic for a BFS tree were formulated. For example, it has been proved that for a vertex with index r in the complete tree, its rightmost child has index $rL + 1$. Furthermore, we proved that then $rL + 1$ is the upper bound for the index of vertex r in an arbitrary BFS tree. Then we can state that in an arbitrary BFS tree a vertex with index n can only have children with indices from the range from $r + 1$ to $rL + 1$. Introduction of these constraints allows reducing the number variables used in the encoding and thus reduce this size of the resulting formula.

By definition, in any BFS tree the children of any vertex r have consecutive indices, and there are no more than L children. We analyzed the connection between vertices of the APTA vertices and the DFA states. If the prefix tree contains a path of length k from the root to some vertex t_i, and vertex t_i corresponds to state d_j of the sought automaton, then the automaton must contain a path of length no more than k from the initial state to state d_j. A SAT encoding of these properties was developed. Augmentation of the formula with corresponding constraints allows for additional reduction of the search space during SAT solving.

Section2.4 describes the developed software tool `DFA-Inductor-py` for DFA inference, implementation of developed methods as parts of this software tool, and results of experimental evaluation of all developed methods.

In the process of preparing this thesis a software tool `DFA-Inductor-py`[35] for DFA inference from behavior examples has been developed using the *Python* programming language. The tool is comprised of various modules for solving the problem of DFA inference from behavior examples, the problem of DFA inference from an excessive set of behavior examples, and the problem of inferring all non-isomorphic DFA from given behavior examples (these methods are described in the following chapters). The tool implements different symmetry breaking predicates, including ones previously proposed by other authors, and the ones developed in this thesis.

The developed software tool was used to perform two series of experimental evaluations. First we compared two methods for DFA inference from behavior examples that use symmetry breaking predicates based on BFS and DFS, correspondingly. Since these two methods are based on the method `DFASAT`, in which symmetry breaking constraints are based on fixing the enumeration of some large clique in the consistency graph, we also added `DFASAT` to this experimental evaluation.

Publicly open tests (sets of behavior examples), for example, from *Abbadingo One DFA Learning Competition*[36] and *StaMinA Competition*[37], were developed for inexact algorithms which can generate automata with hundreds of states. Exact methods, both existing and proposed in this thesis, are currently not capable of inferring DFA of such size; for this reason, tests for this experimental evaluation were generated randomly. Experimental results are shown in Table 1 and allow drawing the conclusion that the use of DFS-based symmetry breaking predicates is impractical, since the method that uses them is significantly inferior to the method that uses BFS-predicates.

[35] *Zakirzyanov*, *I.* DFA-Inductor-py. URL: `https://github.com/ctlab/DFA-Inductor-py` (visited on 10/02/2020).

[36] *Lang*, *K. J.*, *Pearlmutter*, *B. A.*, *Price*, *R. A.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm // ICGI. vol. 1433. Springer, 1998. P. 1–12. (Lecture Notes in Computer Science).

[37] *Walkinshaw*, *N.*, *Lambeau*, *B.*, *Damas*, *C.*, *Bogdanov*, *K.*, *Dupont*, *P.* STAMINA: a competition to encourage the development and assessment of software model inference techniques // Empirical Software Engineering. 2013. Vol. 18, no. 4. P. 791–824.

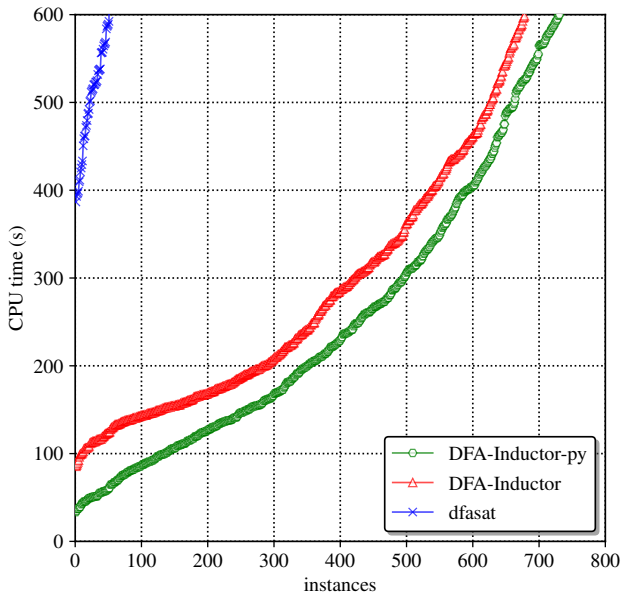
However, note that the method that uses DFS-predicates still considerably outperforms DFASAT. Nevertheless, based on experimental results, it was decided not to continue the development of DFS-predicates and concentrate on improving predicates based on the breadth-first search algorithm.

Table 1 – Median execution time in seconds for methods of DFA inference from behavior examples: method based on BFS symmetry breaking predicates, based on DFS symmetry breaking predicates, and the DFASAT method. Each value was calculated as a result of 100 independent runs on 100 problems. The execution of the methods was limited to one hour (TL $= 3600$ seconds)

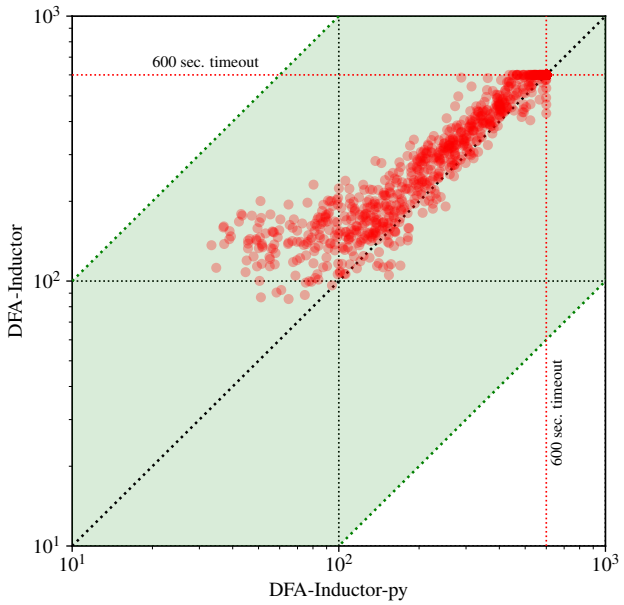
M	DFS	BFS	DFASAT
10	20.9	20.5	23.3
12	40.4	37.6	240.3
14	82.2	62.4	—
16	205.1	114.1	—
18	601.7	181.9	—
20	2501.6	293.7	—
22	—	453.3	—
24	—	625.1	—
26	—	925.8	—
28	—	1314.4	—
30	—	1635.5	—

In the second experimental evaluation we compared methods that use the original BFS-predicates (DFA-Inductor) and the BFS-predicates proposed in this thesis (DFA-Inductor-py). As before, we also considered DFASAT and used randomly generated tests. Comparison results of all three methods are depicted in Figure 7a and show that within a fixed time budget DFA-Inductor-py can solve more instances of the problem of DFA inference from behavior examples than DFA-Inductor: in ten minutes the new method solved 731 solved problems against 678 for the old one. Also note that DFASAT cannot compete with two other methods.

Figure 7b depicts a more detailed compaison of DFA-Inductor и DFA-Inductor-py: one may notice that the method proposed in this thesis shows better results for the vast majority of instances.



(a) Comparison of methods `DFA-Inductor`, `DFA-Inductor-py`, and `DFASAT`, showing the number of problem instances solved within a fixed time budget



(b) Detailed comparison of methods `DFA-Inductor` and `DFA-Inductor-py`, showing their performance for each problem instance

Figure 7 – Comparison results of the method that uses the original BFS-predicates (`DFA-Inductor`), the method that uses new BFS-predicates (`DFA-Inductor-py`), and `DFASAT`

Chapter 3 describes the development, implementation, and experimental evaluation of the exact method for DFA inference from an excessive set of behavior examples using a reduction to SAT and counterexample-guided abstraction refinement.

In the **section 3.1** we provide the study of the limits of applicability of methods proposed in previous chapters in dependence on the size of the augmented prefix tree acceptor. The size of the Boolean formula that encodes the DFA inference problem depends linearly on the size of the prefix tree: $\mathcal{O}\left(N \times M^2\right)$ clauses, where N is the size of the prefix tree and M is the size of the DFA. The number of variables in the Boolean formula also linearly depends on the size of the prefix tree: $\mathcal{O}\left(N \times M + M^2\right)$ variables. Thus, when the sought automaton is fixed, the size of the Boolean formula and the number of its variables may vary considerably depending on the number of behavior examples and their length. As the size of the prefix tree increases, the SAT solver has to use more and more resources to store the formula and handle it, and the number of variables also increases. This leads to a situation when the same automaton

may be constructed by a SAT solver in seconds from a small set of behavior examples, and may not be found in hours and days in the case of an excessive number of long behavior examples.

Since in the case of DFA the key information about each behavior example (whether this word is accepted by the DFA) is contained in the last vertex of a path in the APTA corresponding to this word, it is doubtful that anything can be done in the case of long behavior examples. However, in the case of an excessive number of behavior examples we can take only a subset of them and infer the same automaton faster. The issue is in the way the behavior examples are selected: it is possible to accidently eliminate the examples that are necessary to generate the DFA that is the answer to the original problem, and thus end up with a completely different DFA. The next section proposes a method that solves this problem: it iteratively enumerates meaningful behavior examples from the original set.

Section 3.2 describes the proposed exact method for DFA inference from an excessive set of behavior examples, that is based on a reduction to SAT and the CEGAR approach. As noted before, counterexample-guided abstraction refinement is commonly used in active learning problems. The considered problem of DFA inference is, on the contrary, a passive learning problem. However, in this thesis a method is proposed that solves the problem of DFA inference from behavior examples using ideas of the CEGAR approach.

Similar to the classical CEGAR, the proposed approach iteratively refines the model, which in this thesis is a deterministic finite automaton. The initial prefix tree does not contain any vertices, but is augmented on each step. On each step it is proposed to try inferring a DFA of the current size for the current prefix tree using a reduction to SAT. If such a DFA does not exist, then, as before, the size of the sought DFA is increased by one, and the search process is repeated. And if such an automaton is found, its compliance with the remaining behavior examples is checked. If the DFA complies with all behavior examples, then the problem is solved. If not, then we select one or several counterexamples from the behavior examples which are not safisfied by

the DFA, and use them to augment the prefix tree, build a new Boolean formula, and continue the search. The scheme of the proposed method is shown in Figure 8.

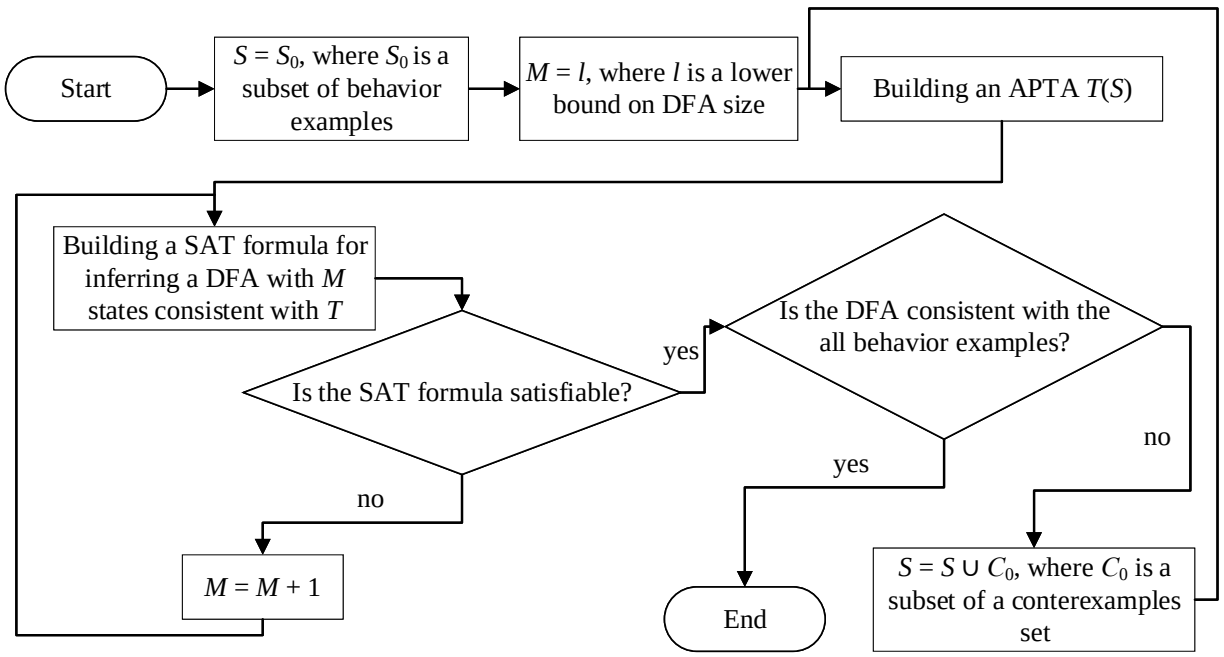


Figure 8 – Scheme of the exact method for DFA inference from an excessive set of behavior examples based on a reduction to SAT and the CEGAR approach

Note that restarting the SAT solver after each augmentation of the prefix tree is extremely inefficient, since when new clauses are added to the formula, the search space of SAT is only reduced, and thus, there is no fundamental need to start searching for a satisfying assignment from scratch. Incremental SAT solvers may be used to cope with this: after some satisfying assignment is found they switch to a state in which they wait for new clauses to be supplied, and continue the search with a new refined formula from the same place where the search was paused before.

Section 3.3 presents a description of the implementation of the developed method as a part of the tool `DFA-Inductor-py`, as well as results of experimental evaluation of the developed method. In experimental evaluation we again used random tests. Results of experimental evaluation showed that when the size of behavior examples $S = |S_+| + |S_-|$ exceeds $200 \times M$, the CEGAR-based method works at least two times faster than the method that uses all behavior examples at once. Moreover, the benefit of using CEGAR increases with the growth of the number of behavior examples.

In **Chapter 4** the problem of inferring all non-isomorphic DFA from given behavior examples is formulated, and the development, implementation, and experimental evaluation of two methods that solve this problem is described.

Section 4.1 provides a formal statement of the problem of inferring all non-isomorphic DFA of minimal size that satisfy given behavior examples. As shown before, a DFA of minimal size yields a maximally exact generalization of given data expressed with behavior examples. However, in the case when the behavior examples do not describe the sought automaton to a satisfactory degree, several different and minimal non-isomorphic automata may exist. In this case a rational solution may be to infer all such automata for further analysis. Figure 9 shows all non-isomorphic minimal-sized DFA inferred for given behavior examples.

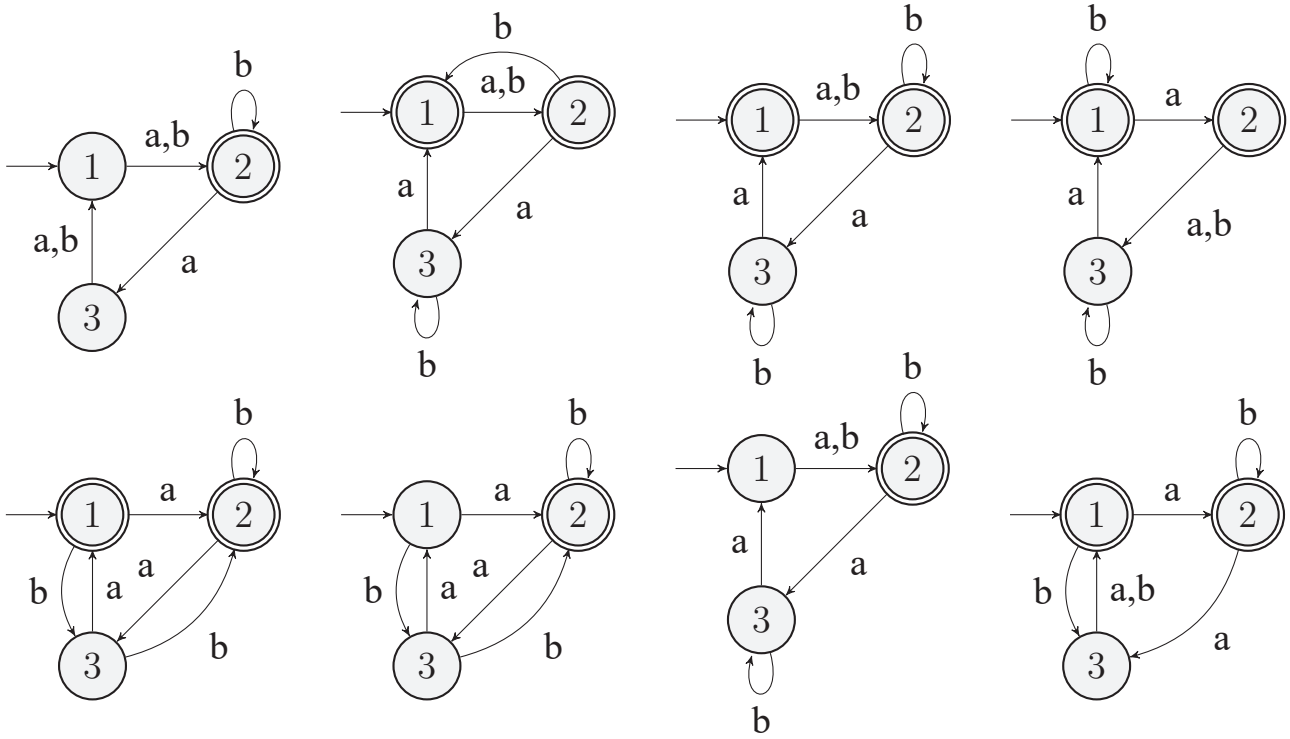


Figure 9 – All non-isomorphic DFA corresponding to sets of behavior examples $S_+ = \{a, bb, aaaa\}$ and $S_- = \{aa, bab\}$

Previously no methods for inferring all distinct minimal-sized DFA from given behavior examples have been proposed. Moreover, without the use of symmetry breaking predicates based on BFS or DFS, that allow consideration of only one automaton for

each isomorphism equivalence class instead of a factorial number of automata, efficient SAT-based inference of all distinct DFA does not seem possible.

Section 4.2 describes the method for inferring all non-isomorphic DFA of minimal size that uses a reduction to SAT and symmetry breaking. When symmetry breaking predicates based on BFS or DFS are used, only a BFS-enumerated (DFS-enumerated) automaton can be inferred thus, it is sufficient to block the inferred automaton and exclude it (along with all automata isomorphic to it) from the search space. Note that in order to construct a DFA from the satisfying assignment, we only need to use values of transition variables $y_{i,l,j}$ and accepting state variables z_i. This means that it suffices to forbid only the current values of these variables. Denote $\varphi(q)$ the value of variable q in the found satisfying assignment φ, and define a set $\mathcal{Y} = \{y_{i,l,j} | i, j \in [M] \wedge l \in \Sigma \wedge \varphi(y_{i,l,j}) = 1\}$. Then, a *blocking clause* can be defined as follows:

$$\bigwedge_{y \in \mathcal{Y}} \neg y \wedge \bigwedge_{i \in [M]} \neg \varphi(z_i).$$

As in the CEGAR-based method, the addition of a blocking clause only reduces the size of the search space, so we do not have to restart the SAT solver, and may use it in incremental mode.

One more important property of the proposed method is the validation of the fact that the inferred automaton is the only existing minimal-sized automaton that satisfies given behavior examples. The uniqueness of a DFA in this case suggests that the given data sufficiently describes the found automaton.

Section 4.3 describes the implementation of the developed methods as parts of the tool `DFA-Inductor-py` and experimental evaluation. Since previously there were no known efficient methods for inferring all distinct DFA from given behavior examples, in order to have a baseline we developed an enumerative backtracking algorithm which searches for all distinct DFA for given behavior examples without the use of external software tools.

Experimental results are summarized in Table 2. Experiments were run for three different groups of instances of the problem of DFA inference from given behavior ex-

Table 2 – Median execution time for inferring all distinct DFA with the SAT-based method that restarts the SAT solver (REST), the SAT-based method that uses an incremental SAT solver (INC), and the enumerative backtracking algorithm (BTR)

M	$S = 5 \times M$				$S = 10 \times M$				$S = 25 \times M$			
	>1	REST	INC	BTR	>1	REST	INC	BTR	>1	REST	INC	BTR
5	53	2.3	2.0	0.8	40	3.6	3.3	1.3	17	4.1	3.4	1.5
6	56	2.8	2.4	2.1	31	4.7	3.9	1.7	27	5.4	4.3	1.7
7	87	3.9	2.5	4.1	27	3.7	3.0	3.1	13	7.4	6.7	2.5
8	80	4.6	3.7	87.2	34	7.0	6.5	41.7	16	10.1	8.9	11.6
9	91	7.6	3.9	475.1	50	7.7	6.4	121.6	10	13.8	13.0	61.4
10	89	15.7	5.3	2756.2	47	8.6	7.0	974.7	11	18.8	16.1	276.8
11	94	19.9	7.3	—	63	18.5	13.8	3108.0	9	24.5	21.9	1158.4
12	90	28.0	9.9	—	49	22.3	16.7	—	8	33.5	27.2	3289.1
13	92	185.5	18.1	—	57	36.9	22.6	—	12	62.0	51.4	—
14	87	408.5	49.0	—	71	85.1	41.8	—	4	67.0	56.2	—
15	95	571.1	174.1	—	69	193.3	95.7	—	6	29.2	26.2	—

amples. In these three groups we used a different ratio between the number of behavior examples and the size of the inferred DFA: for the first group the ratio was $S = 5 \times M$, for the second group $S = 10 \times M$, and the third group had $S = 25 \times M$. The column ">1" shows the percentage of problem instances for which more than one non-isomorphic DFA exists.

Experimental results from Table 2 allow making the following conclusions:

a) a first successful solution for the problem of inferring all distinct minimal-sized DFA from given behavior examples has been proposed;
b) both SAT-based methods greatly outperfrom the backtracking method;
c) as expected, the use of an incremental SAT solver gives a considerable advantage over the approach in which the solver is restarted: this is because after some DFA is found the intermediate state is preserved by the incremental SAT solver;
d) the more behavior examples are given for inferring a DFA, the less often a situation occurs when there are several different DFA corresponding to them.

To **conclude**, the results of this study are:

a) Symmetry breaking predicates based on SAT encodings of the breadth-first search and depth-first search algorithms for reducing the search space reduction during SAT solving, and exact methods for DFA inference from behav-

ior examples that use these predicates, have been developed. The symmetry breaking predicates based on the encoding of the depth-first search algorithm have a rather theoretical significance, since they have not demonstrated any advantage over predicates based on the breadth-first search algorithm encoding. The encoding of the symmetry breaking predicates based on breadth-first search is expressed with an asymptotically smaller number of clauses in comparison to the previously proposed encoding. The developed exact method that uses these predicates together with predicates based on characteristics of the BFS traversal tree demonstrates better performance in comparison to previous methods.

b) A new exact method for DFA inference from an excessive set of behavior examples that uses a reduction to SAT and counterexample-guided abstraction refinement has been developed. The size of the Boolean formula and the number of used variables linearly increase with the growth of the number of behavior examples. The developed method allows inferring a DFA from an excessive set of behavior examples by iteratively augmenting the set of used behavior examples, completing the augmented prefix tree and using the SAT solver in incremental mode. Since the set of used behavior examples is augmented with counterexamples to inferred intermediate DFA, as a result only meaningful behavior examples are used for inference. Thus, this method allows for an exact solution of problems which previously could not be solved due to a large size of the Boolean formula.

c) A method for inferring all non-isomorphic DFA of minimal size satisfying given behavior examples has been developed that uses symmetry breaking predicates and SAT solvers. Previously, the problem of inferring all minimal non-isomorphic DFA did not have an efficient solution, since isomorphic automata, being equivalent in structure and defined language, are deemed to be distinct by a SAT solver, leading to considering $\mathcal{O}(M!)$ isomorphic automata with M states. The use of symmetry breaking predicates based on an encoding of the breadth-first search algorithm allows the SAT solver to con-

sider only one representative of each isomorphism equivalence class instead of a factorial number of automata. Thus, a method for solving the problem of inferring all minimal-sized non-isomorphic DFA has been first proposed. The search for all non-isomorphic automata may be useful for their further analysis, and also for analysis of existing behavior examples. Another application of the developed method is the possibility to prove the uniqueness of the minimal automaton that satisfies given behavior examples.

d) During thesis preparation an open source software tool `DFA-Inductor-py` has been developed in *Python* for the purpose of DFA inference from given behavior examples. The software tool is comprised of different modules for solving the problem of DFA inference from given behavior examples, the problem of inferring a DFA from an excessive set of behavior examples, and the problem of inferring all non-isomorphic DFA from given behavior examples. The tool implements different symmetry breaking predicates, including the ones developed by other authors and the ones developed in this thesis.

e) The results of this thesis were used in the educational process of the Faculty of Information Technologies and Programming in the course “Design of automata-based programs” of the Bachelors’s program “Mathematical models and algorithms in software engineering” (supported by the official act of use).

f) A part of results have been used in the project SAUNA (“Integrated safety assessment and justification of nuclear power plant automaton”) by the “IT in Industrial Automation” research group of the department of electrical engineering and automation in Aalto University, Finland, in the framework of the Finnish research program in safety of nuclear power plants SAFIR2018. This is confirmed by a letter from the principal investigator of the “IT in Industrial Automation” group, Valeriy Vyatkin.

g) Results of this thesis have also been used in the project No. 18-37-00425 “Development of efficient machine learning methods for deterministic finite automata inference based on Boolean satisfiability solving” (2018–2020)

funded by the Russian Foundation for Basic Research and led by the author of the thesis, and in research projects in the framework of the Russian academic excellence project «5-100».

The exact method for DFA inference from given behavior examples that uses symmetry breaking predicates based on the breadth-first first algorithm demonstrates better performance the previously known methods and allows inferring larger automata in less time. The method for inferring a DFA from an excessive set of behavior examples allows inferring automata from data, for which the previously known methods could not infer any DFA in principle due to a large size of the Boolean formula. The method for inferring all minimal-sized non-isomorphic DFA satisfying given behavior examples is the first known method that allows inferring all non-isomorphic DFA: previously known methods may adapted to finding all DFA, but their application leads to a combinatorial explosion of the size of considered DFA due to lack of efficient symmetry breaking predicates. The aim of this thesis was to increase the efficiency of exact methods for DFA inference from given behavior examples by reducing the search space during Boolean satisfiability problem solving. Thus, according to experimental results, the aim is achieved.

Acknowledgement. The author expresses gratitude to his supervisor, V. Ulyantsev, Ph.D., for invaluable assistance in research and in preparation of this thesis, Prof. A. Shalyto for mentoring, colleagues from the International Laboratory “Computer Technologies”, D. Chivilikhin, Ph.D., for numerous consultations on research and for help with the English translation of the synopsis, K. Chukharev and D. Suvorov for help in preparing illustrative material for this thesis, A. Bugrovsky and T. Galimzhanov for their help with documents for preparing the defense of this thesis, and to foreign colleagues A. Ignatiev, Ph.D., and Prof. J. Marques-Silva, Dr. habil., for organizing and active participation in the internship, within the framework of which part of the author’s research was carried out.

The author is also grateful to his parents, Timur and Valentina, for the fostered love of knowledge and mathematics, to Lazareva Ekaterina for always being there, to

his friends Alexander B., Alexander S., Anastasia, Valentin, Eugene, Marina, Matvey, Natalya, Talgat and Tatiana for the moral support provided.

Author's publications on the topic of the thesis

Publications indexed in Web of Science or Scopus

1. *Ulyantsev*, *V.*, *Zakirzyanov*, *I.*, *Shalyto*, *A.* BFS-Based Symmetry Breaking Predicates for DFA Identification // Language and Automata Theory and Applications — 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings. Vol. 8977. — Springer, 2015. — P. 611–622. — (Lecture Notes in Computer Science).

2. *Zakirzyanov*, *I.*, *Shalyto*, *A.*, *Ulyantsev*, *V.* Finding All Minimum-Size DFA Consistent with Given Examples: SAT-Based Approach // Software Engineering and Formal Methods — SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers. Vol. 10729. — Springer, 2017. — P. 117–131. — (Lecture Notes in Computer Science).

3. *Zakirzyanov*, *I.*, *Morgado*, *A.*, *Ignatiev*, *A.*, *Ulyantsev*, *V.*, *Marques-Silva*, *J.* Efficient Symmetry Breaking for SAT-Based Minimum DFA Inference // Language and Automata Theory and Applications — 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings. Vol. 11417. — Springer, 2019. — P. 159–173. — (Lecture Notes in Computer Science).

4. *Ovsiannikova*, *P.*, *Chivilikhin*, *D.*, *Ulyantsev*, *V.*, *Stankevich*, *A.*, *Zakirzyanov*, *I.*, *Vyatkin*, *V.*, *Shalyto*, *A.* Active Learning of Formal Plant Models For Cyber-Physical Systems // 16th IEEE International Conference on Industrial Informatics, INDIN 2018, Porto, Portugal, July 18-20, 2018. — IEEE, 2018. — P. 719–724.

Publications indexed in Russian journal included in the List of the Higher Attestation Commission

5. *Закирзянов*, *И. Т.* Построение детерминированных конечных автоматов по примерам поведения с использованием подхода уточнения абстракции по контрпримерам // Научно-технический вестник информационных технологий, механики и оптики. — 2020. — Т. 20, № 3. — С. 394–401.

Other publications (in Russian)

6. *Закирзянов*, *И. Т.* Разработка предикатов нарушения симметрии на основе поиска в ширину для построения детерминированных конечных автоматов // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2015.
7. *Закирзянов*, *И. Т.* Эмпирическая оценка производительности программных средств решения задачи SAT для синтеза ДКА // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2016.
8. *Закирзянов*, *И. Т.*, *Ульянцев*, *В. И.* Сравнительный анализ методов задания ограничений типа at-most-one на примере задачи построения ДКА с использованием программных средств решения SAT // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2017.
9. *Закирзянов*, *И. Т.* Разработка предикатов нарушения симметрии для построения автоматных моделей программного обеспечения по заданной спецификации в виде темпоральных формул // Сборник тезисов докладов Всероссийского конгресса молодых ученых. — 2018.
10. *Закирзянов*, *И. Т.* Построение минимального ДКА на основе сведения к SAT с использованием предположений // Сборник тезисов докладов конгресса молодых ученых. — 2020.

Author's publications on other topics

Publications indexed in Web of Science or Scopus

11. *Kachalsky*, *I.*, *Zakirzyanov*, *I.*, *Ulyantsev*, *V.* Applying Reinforcement Learning and Supervised Learning Techniques to Play Hearthstone // 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017. — IEEE, 2017. — P. 1145–1148.

Registered computer programs

12. *Программный комплекс методов машинного обучения DFA-Inductor для построения детерминированных конечных автоматов*. Программа / И. Т. Закирзянов, В. И. Ульянцев, А. А. Шалыто. — № 2016618241 ; опубл. 20.09.2016.

13. *Программное средство для генерации дискретной формальной модели объекта управления по примерам поведения*. Программа / И. П. Бужинский, И. Т. Закирзянов, В. А. Миронович, С. В. Казаков, М. А. Лукин, А. С. Буздалова, В. И. Ульянцев, А. А. Шалыто. — № 2018619728 ; опубл. 10.08.2018.

Введение

Актуальность темы. Конечные автоматы являются одним из основополагающих концептов в дискретной математике, информатике и программировании. Помимо непосредственной роли в теории формальных языков и при проектировании вычислительных машин, различные конечно-автоматные модели используются в наше время на практике при разработке и анализе программного обеспечения.

Например, конечные автоматы используются при проектировании моделей программного обеспечения контроллеров и ответственных систем [1; 2], для спецификации протоколов взаимодействия [3], для моделирования поведения высокоуровневых систем [4; 5]. Основными преимуществами использования автоматов являются наглядность и относительная простота модели для человеческого восприятия, а также возможность автоматизированной верификации формальных свойств модели (model checking) [6].

Во многих случаях проектирование автоматной модели осуществляется разработчиком вручную, например, в парадигме автоматного программирования [1]. При решении других задач подразумевается автоматизированная генерация конечного автомата — извлечение модели из существующих данных или систем. Среди практических примеров использования методов генерации конечных автоматов можно привести: построение моделей программного обеспечения контроллеров по составленным вручную или снятым автоматизированно примерам поведения [7], построение формальных моделей объектов управления [8; 9], анализ моделей поведения сложных программных систем [4; 10; 11] и сетевых протоколов [12], и другие. Активное изучение и разработка алгоритмов генерации автоматов начиналась ведется с 1970-х годов. В 1978 году было доказано, что задача генерации детерминированного конечного автомата (ДКА) с заданным (минимальным) числом состояний по заданным примерам поведения является NP-полной [13]. Данный теоретический результат подчеркивает сложность

задачи генерации автоматов в общем случае, актуализируя разработку применимых на практике алгоритмов.

Впоследствии было предложено достаточно много как эвристических, так и метаэвристических алгоритмов генерации ДКА по заданным примерам поведения, и к настоящему моменту они образуют собой целое семейство алгоритмов в дискретной математике. За последние годы была разработана группа методов, сводящих задачу точной генерации (с минимально возможным числом состояний) искомого автомата к другим NP-полным задачам. При этом наиболее эффективные подходы в настоящий момент основаны на сведении к задаче выполнимости.

Задача выполнимости булевых формул (Boolean Satisfiability — SAT) заключается в определении существования выполняющей подстановки для заданной булевой формулы, представленной в конъюнктивной нормальной форме — в виде конъюнкции дизъюнктов. Согласно теореме Кука-Левина 1971 года, задача выполнимости является NP-полной Данный факт актуализировал и стимулировал разработку применимых на практике программных средств для решения задачи выполнимости.

Методы решения SAT разрабатывались еще до введения теоретических оценок сложности: в 1962 году был предложен алгоритм Дэвиса-Патнема-Логемана-Лавленда (Davis-Putnam-Logemann-Loveland — DPLL) [14] — полный алгоритм поиска с возвратом выполняющей подстановки, использующий эвристики распространения переменной и исключения «чистых» переменных для ускорения поиска. Данный алгоритм, в общем случае за экспоненциальное время от числа переменных, перебирает все пространство поиска подстановок и останавливается, если была найдена выполняющая подстановка или если перебор завершил работу — выполняющей подстановки не существует.

В середине 1990-х на основе алгоритма DPLL был разработан алгоритм CDCL [15] (conflict-driven clause learning — «управляемое конфликтами обучение дизъюнктов»), сохраняющий и использующий в дальнейшем выведенные дизъюнкты в ходе анализа конфликтов, возникающих при работе DPLL. Такие дизъюнкты в дальнейшем позволяют намного раньше принимать решение о невыпол-

нимости формулы с текущими допущениями и переходить к рассмотрению следующих.

Именно на алгоритме CDCL основаны все современные программные средства для решения SAT. Каждый год проводятся соревнования программных средств решения SAT [16; 17], что отличает задачу выполнимости от всех остальных NP-трудных задач. За счет этого для других задач актуально разрабатывать методы, основанные на сведении к SAT: ускорения метода можно добиться без изменения исходного кода, будет достаточно лишь заменить программное средство для решения SAT, выдающее найденную подстановку или доказывающее ее отсутствие.

Однако, в методологии сведения поставленной задачи к SAT фундаментальный характер носит не столько используемое программное средство, сколько способ трансляции экземпляра задачи в булеву формулу, для которой в дальнейшем будет запущен поиск подстановки. Так, в последние годы для задачи точной генерации ДКА по заданным примерам поведения было предложено базовое сведение [18] и несколько его модификаций [4; 19]. Данные модификации предлагали использование предикатов нарушения симметрии — дополнительно введенных в исходную формулу дизъюнктов, задающих специфичное для задачи сокращение пространства поиска.

Данные техники сокращения пространства поиска оказались очень эффективны на практике, позволив решать задачи генерации автоматов большего размера. Однако, последующий анализ показывает, что данные техники не оптимальны, а область применимости существующих методов точной генерации ДКА по примерам поведения все еще весьма ограничена (что обусловлено, в первую очередь, NP-полной природой задачи). Таким образом, тема настоящей диссертации, продолжающей указанные исследования последних лет и направленной на расширение возможностей методов сокращения пространства поиска и генерации ДКА, **является актуальной**.

Степень разработанности темы. Задача генерации детерминированного конечного автомата по заданным примерам поведения, выраженным в виде двух

множеств S_+ и S_-, заключается в поиске ДКА с минимальным числом состояний, такого, что все строки из множества S_+ принимаются автоматом, а все строки из S_- — не принимаются. Впервые данная задача была сформулирована в работе Голда в 1967 году [20].

Первый известный алгоритм для решения данной задачи был предложен в работе Трахтенброта и Барздиня в 1970 году [21] — `TB`-алгоритм. Однако, предложенный алгоритм решает только частный случай задачи генерации ДКА по заданным примерам поведения: для некоторого натурального k все возможные слова длины k над алфавитом Σ — всего $|\Sigma|^k$ слов — должны содержаться во множествах S_+ и S_-. В данном алгоритме, как и в подавляющем большинстве последующих, примеры поведения представляются в виде расширенного префиксного дерева — префиксного дерева, в котором вершины могут быть допускающими, отвергающими или промежуточными. `TB`-алгоритм основан на полном переборе всех возможных пар состояний префиксного дерева и слиянии эквивалентных состояний в одно.

В 1978 году Голд доказал NP-полноту задачи генерации ДКА заданного, а значит и минимального, размера [13]. Ввиду указанной сложности новые алгоритмы генерации минимального ДКА не предлагались более десяти лет. В последующие годы начали активно разрабатываться неточные эвристические алгоритмы — ими не гарантируется минимальность найденного ДКА. Среди таких алгоритмов можно выделить следующие: `traxbar` [22], `RPNI` [23], `EDSM` [24], `exbar` [25], `Windowed-EDSM` [26]. Все перечисленные алгоритмы основаны на слиянии состояний расширенного префиксного дерева. Состояния для слияния выбираются эвристически, чем одновременно объясняется высокая скорость работы и неточность данных алгоритмов.

Другим распространенным подходом к генерации ДКА по заданным примерам поведения является применение метаэвристических алгоритмов. Например, были предложены эволюционные алгоритмы [27; 28], муравьиные алгоритмы [29]. Метаэвристические алгоритмы также являются неточными — ими вообще не гарантируется, что какое-то решение будет найдено за конечное время.

Голландские ученые Хойл и Вервер в 2010 году предложили алгоритм DFASAT, способный гарантированно генерировать ДКА минимального размера по произвольным данным [18]. Алгоритм основан на сведении задачи генерации ДКА по примерам поведения к задаче выполнимости булевой формулы (SAT), где в качестве промежуточного шага используется сведение к задаче раскраски графа, которое было предложено еще в 1997 году [30]. Для сокращения пространства поиска при решении задачи выполнимости Хойл и Вервер предложили ряд подходов к нарушению симметрии: граф совместимости, дополнительные дизъюнкты, поиск большой клики. Однако, даже с использованием всех этих техник, DFASAT способен за разумное время строить автоматы с не более чем десятью состояниями. Хойл и Вервер для упрощения задачи предложили делать предварительно несколько слияний в префиксном дереве с помощью алгоритма EDSM, однако тогда теряется точность — гарантия минимальности найденного ДКА.

Позднее автором диссертации совместно с научным руководителем Ульянцевым В. И. и профессором Шалыто А. А. были предложены предикаты нарушения симметрии на основе алгоритма обхода графа в ширину (BFS) [31]. Для каждого ДКА с M состояниями существует $\mathcal{O}\left(M!\right)$ изоморфных ему автоматов, которые, с точки зрения программного средства для решения SAT, являются различными. Предложенные предикаты нарушения симметрии позволяют значительно сократить пространство поиска при решении SAT — вместо факториала изоморфных автоматов при решении перебирается только единственный представитель класса эквивалентности по изоморфизму — ДКА, пронумерованный в порядке BFS-обхода. Данный метод является наиболее эффективным известным точным методом генерации ДКА по заданным примерам поведения и позволяет строить автоматы с числом состояний до сорока. Однако, анализ показал, что предложенное наивное кодирование BFS-предикатов на языке SAT неоптимально — получаемая булева формула слишком велика. Помимо этого, не рассматривались предикаты нарушения симметрии, основанные на кодировании алгоритма обхода графа в глубину (DFS).

Среди прочих недостатков методов генерации ДКА, основанных на сведении к SAT, можно отметить зависимость размера булевой формулы от числа имеющихся примеров поведения. Зачастую множества примеров поведения избыточны и содержат лишние слова. Анализ литературы показал, что интеллектуальных методов выбора подмножества примеров поведения ранее не предлагалось. Возможным решением может быть применение подхода *уточнения абстракции по контрпримерам* (*counterexample-guided abstraction refinement* — CEGAR) — итеративного алгоритма, изначально разработанного для построения модели программного обеспечения [32; 33].

Обратной ситуацией является генерация ДКА по небольшим множествам примеров поведения. В таком случае полезным может быть генерация всех соответствующих неизоморфных ДКА с минимальным числом состояний для проведения дальнейшего анализа. Также, существование единственного автомата минимального размера говорит о качестве имеющихся обучающих данных. Однако, задача генерации всех неизоморфных ДКА ранее не ставилась, и, соответственно, эффективных методов ее решения предложено не было.

Целью настоящей диссертации является повышение эффективности точных методов генерации детерминированных конечных автоматов по заданным примерам поведения посредством сокращения пространства поиска при решении задачи выполнимости.

Для достижения указанной цели в работе поставлены и решены следующие **задачи**:

а) Разработка предикатов нарушения симметрии, основанных на кодировании алгоритмов обхода графа в ширину и в глубину, для сокращения пространства поиска при решении задачи выполнимости. Разработка и реализация точных методов генерации ДКА по заданным примерам поведения, использующих данные предикаты, проведение экспериментальных исследований разработанных методов.

б) Разработка и реализация точного метода генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче вы-

полнимости и подхода уточнения абстракции по контрпримерам. Проведение экспериментальных исследований разработанного метода.

в) Разработка и реализация метода генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств решения задачи выполнимости. Проведение экспериментальных исследований разработанного метода.

Предмет исследования — методы точной генерации ДКА, использующие программные средства для решения задачи выполнимости.

Соответствие паспорту специальности. Данная диссертация соответствует пункту 10 «Разработка основ математической теории языков и грамматик, теории конечных автоматов и теории графов» паспорта специальности 05.13.17 — «Теоретические основы информатики».

Основные положения, выносимые на защиту:

а) *Подход* к построению предикатов нарушения симметрии, основанных на кодировании алгоритмов обхода графа в ширину и в глубину, для сокращения пространства поиска при решении задачи выполнимости. Точные *методы* генерации ДКА по заданным примерам поведения, использующие данные предикаты.

б) Точный *метод* генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам.

в) *Метод* генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств решения задачи выполнимости.

Научная новизна диссертации состоит в следующем:

а) Предикаты нарушения симметрии, основанные на кодировании алгоритма обхода графа в глубину ранее не предлагались. Предложенные предикаты нарушения симметрии, основанные на алгоритме обхода графа в

ширину, выражаются булевой формулой, состоящей из асимптотически меньшего числа дизъюнктов сравнительно с существующим подходом. Помимо этого, впервые предложены предикаты нарушения симметрии, использующие особенности дерева обхода графа в ширину.

б) Точных методов генерации ДКА применимых в случае, когда число примеров поведения излишне велико, ранее не предлагалось. Использование подхода уточнения абстракции по контрпримерам одновременно со сведением к задаче выполнимости позволяет генерировать ДКА по избыточному набору примеров поведения путем итеративного добавления только значимых примеров до тех пор, пока не будет получен ДКА, соответствующий всему избыточному набору.

в) Методов для генерации всех неизоморфных ДКА минимального (или любого фиксированного) размера, удовлетворяющих заданным примерам поведения, ранее не предлагалось.

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, теории графов, теории сложности, дискретной математики, объектно-ориентированное программирование, методы проведения и анализа экспериментальных исследований.

Достоверность полученных результатов подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, а также результатами проведенных экспериментальных исследований по использованию предложенных в диссертации методов.

Теоретическая значимость работы заключается в том, что в ней для точных методов генерации ДКА предложены новые подходы к сокращению пространства поиска при решении задачи выполнимости — предикаты нарушения симметрии на основе кодирования алгоритмов обхода графа в глубину и в ширину:

– предложен способ кодирования свойства DFS-пронумерованности ДКА на языке SAT;
– предложен новый способ кодирования свойства BFS-пронумерованности ДКА на языке SAT, требующий асимптотически меньшего числа дизъюнктов, относительно известных способов;
– предложен способ кодирования различных свойств BFS-дерева на языке SAT.

Помимо этого, предложен способ объединить метод генерации ДКА при помощи сведения к задаче выполнимости с подходом уточнения абстракции по контрпримерам. Также, разработанные предикаты нарушения симметрии позволили разработать метод для решения задачи генерации всех неизоморфных ДКА минимального размера, которая ранее не имела эффективных методов решения.

Практическая значимость работы состоит в повышении эффективности точных методов генерации ДКА по заданным примерам поведения. Экспериментально показано, что разработанный метод генерации ДКА по заданным примерам поведения с использованием предикатов нарушения симметрии на основе алгоритма BFS является самым эффективным по времени генерации автомата среди известных на настоящий момент точных методов и позволяет генерировать автоматы большего размера относительно методов, предложенных ранее. Разработанный точный метод генерации ДКА по избыточному набору примеров поведения с использованием сведения к SAT и подхода CEGAR позволяет эффективно генерировать автоматы в случае, когда набор примеров поведения слишком объемен и получаемая булева формула слишком велика для современных программных средств для решения SAT. Разработанный метод генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств для решения SAT, является первым известным методом для генерации всех неизоморфных автоматов. Также с его помощью можно оценить полноту имеющихся данных, выраженных в виде примеров поведения, путем доказательства или опро-

вержения существования единственного ДКА минимального размера, описывающего эти данные.

Помимо этого, все разработанные методы и подходы в дальнейшем могут быть адаптированы для задач генерации более сложных конечно-автоматных моделей [19]. Так, например, предложенный метод генерации всех неизоморфных ДКА был адаптирован для генерации конечных автоматов, моделирующих поведение программируемых логических контроллеров [34].

Внедрение результатов работы. Результаты работы использовались при выполнении проекта SAUNA (“Integrated safety assessment and justification of nuclear power plant automaton”), выполненного исследовательской группой “IT in Industrial Automation” кафедры электротехники и автоматики университета Аалто, Финляндия, в рамках Финской программы исследований безопасности атомных электростанций — SAFIR2018[1]. В частности, одной из задач проекта была разработка метода генерации моделей различных компонентов системы управления атомных электростанций по заданным примерам поведения и спецификации выраженной на языке темпоральной логики. Данная задача была решена с использованием подхода уточнения абстракции по контрпримерам способом, аналогичным предложенному автором диссертации для генерации ДКА, что подтверждается письмом руководителя исследовательской группы “IT in Automation” В. В. Вяткина.

Результаты работы также использовались при выполнении под руководством автора диссертации гранта Российского фонда фундаментальных исследований (проект 18-37-00425 «Разработка эффективных методов машинного обучения для построения детерминированных конечных автоматов на основе решения задачи выполнимости», 2018–2020 гг.).

Полученные результаты также использовались в рамках государственной финансовой поддержки ведущих университетов Российской Федерации, субсидия 074-U01 (НИР «Биоинформатика, машинное обучение, технологии программирования, теория кодирования, проактивные системы», 2013–2017 гг.) и субси-

[1] http://safir2018.vtt.fi/

дия 08-08 (НИР «Методы, модели и технологии искусственного интеллекта в биоинформатике, социальных медиа, киберфизических, биометрических и речевых системах», 2018–2020 гг.)

Результаты работы также внедрены в учебный процесс факультета информационных технологий и программирования Университета ИТМО в рамках курса «Проектирование автоматных программ» программы бакалавриата «Математические модели и алгоритмы в разработке программного обеспечения», что подтверждается актом об использовании.

Апробация результатов работы. Основные результаты работы докладывались на следующих конференциях и семинарах:

а) 9th International Conference on Language and Automata Theory and Applications (LATA 2015). 2015, Ницца, Франция.

б) 6th International Symposium “From Data to Models and Back (DataMod)”. 2017, Тренто, Италия.

в) 16th IEEE International Conference on Industrial Informatics (INDIN 2018). 2018, Порту, Португалия.

г) 13th International Conference on Language and Automata Theory and Applications (LATA 2019). 2019, Санкт-Петербург.

д) IV-VII Всероссийский конгресс молодых ученых. 2015-2018, Санкт-Петербург.

е) IX Конгресс молодых ученых. 2020, Санкт-Петербург.

ж) XLVI Научная и учебно-методическая конференция Университета ИТМО. 2017, Санкт-Петербург.

и) XLVIII Научная и учебно-методическая конференция Университета ИТМО. 2019, Санкт-Петербург.

Личный вклад автора. Идея предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в глубину, идея метода генерации ДКА по заданным примерам поведения, использующего их, а также реализация алгоритма на базе предложенного метода и проведение вычислительных экспериментов принадлежит лично автору. Идея предикатов нарушения симметрии на основе ал-

горитма обхода графа в ширину, кодирование которых требует асимптотически меньшего числа дизъюнктов, идея кодирования свойств BFS-дерева на языке SAT и идея метода генерации ДКА, использующего данные разработки, принадлежат совместно автору диссертации и Ж. Маркешу-Сильве; реализация алгоритмов на базе предложенных методов принадлежит лично автору, проведение вычислительных экспериментов принадлежит совместно автору диссертации и А.И. Игнатьеву. Идея точного метода генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам, реализация алгоритма на базе предложенного метода и проведение вычислительных экспериментов принадлежит лично автору. Идея точного метода генерации всех ДКА минимального размера по заданным примерам поведения с использованием программных средств решения задачи выполнимости принадлежит совместно автору диссертации и научному руководителю В.И. Ульянцеву, реализация алгоритма на базе предложенного метода и проведение вычислительных экспериментов принадлежит лично автору. В работах, выполненных в соавторстве, В.И Ульянцевым осуществлялось общее руководство исследованиями.

Публикации. Основные результаты по теме диссертации изложены в десяти публикациях, из них четыре опубликованы в изданиях, индексируемых в базе цитирования Scopus, одна публикация издана в журнале, рекомендованном ВАК. Также у автора диссертации имеется одна публикация по другой теме из области машинного обучения, опубликованная в издании, индексируемом в базе цитирования Scopus.

Глава 1 Обзор предметной области

В настоящей главе приводятся результаты обзора работ, посвященных задаче выполнимости булевых формул и методам ее решения, подходу уточнения абстракции по контрпримерам, а также детерминированным конечным автоматам и методам их генерации.

1.1 Задача выполнимости булевой формулы

В данном разделе приводятся основные необходимые определения, связанные с задачей выполнимости булевой формулы, а также ее формальная постановка. Рассматриваются классические и лучшие на данный момент подходы к решению данной задачи.

1.1.1 Постановка задачи

Пусть $X = \{x_1, x_2, \ldots, x_n\}$ — множество булевых (*пропозициональных*) переменных. Булева (*пропозициональная*) формула индуктивно определяется над множеством X, с помощью стандартных булевых операций: отрицание $\neg$, дизъюнкция $\vee$ и конъюнкция $\wedge$ — следующим образом:

- $\mathcal{F} = x$, где $x \in X$, является пропозициональной формулой.
- Если $\mathcal{F}$ является пропозициональной формулой, то $(\neg\mathcal{F})$ является пропозициональной формулой.
- Если $\mathcal{F}$ и $\mathcal{G}$ являются пропозициональными формулами, то $(\mathcal{F} \vee \mathcal{G})$.
- Если $\mathcal{F}$ и $\mathcal{G}$ являются пропозициональными формулами, то $(\mathcal{F} \wedge \mathcal{G})$ является пропозициональной формулой.

Данное определение может быть дополнено другими логическими операциями, однако, по теореме Поста набор операций $\{\neg, \vee, \wedge\}$ является полным [35] — любая булева функция может быть выражена с помощью данных операций.

Литералом называется либо переменная x_i, либо ее отрицание $\neg x_i$ (для сокращения записи, иногда обозначается как $\bar{x_i}$). *Дизъюнктом* (*clause*) называется дизъюнкция конечного числа литералов, либо одиночный литерал, например, $(x_1 \vee \neg x_2 \vee \neg x_5 \vee x_7)$. Также, дизъюнкт можно представить в виде множества литералов, подразумевая дизъюнкцию неявно, например, $\{x_1, \neg x_2, \neg x_4, x_6\}$. Булевой формулой, представленной в *конъюнктивно-нормальной форме* (КНФ), называется конъюнкция нескольких дизъюнктов, либо одиночный дизъюнкт, например, $(x_1 \vee x_3) \wedge (\bar{x_2} \vee \bar{x_3} \vee x_5) \wedge x_6$. Булеву формулу в КНФ можно также представить в виде множества дизъюнктов, которые выражены как множество литералов, например, $\{\{x_1, x_3\}, \{\bar{x_2}, \bar{x_3}, x_5\}, \{x_6\}\}$. В настоящей диссертации любая булева формула считается представленной в КНФ, если не явно не указано иначе.

Каждой пропозициональной переменной может быть присвоено одно из булевых значений $\{0, 1\}$. *Выполняющей подстановкой* $\nu = (v_1, v_2, \ldots, v_n)$, где $v_i \in \{0, 1\}$, для некоторой булевой формулы φ называется такой булев вектор, что формула φ принимает истинное значение при подстановке значений v_i вместо переменных x_i что для всех i. Тогда, задача *выполнимости булевой формулы* (*задача выполнимости*, *Boolean satisfiability* — SAT) заключается в определении, существует ли для некоторой булевой формулы φ выполняющая подстановка [36]. Если такая подстановка существует, то формула φ является *выполнимой* (*satisfiable*), иначе — *невыполнимой* (*unsatisfiable*). Так, например, формула

$$(x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_2 \vee x_4) \wedge x_3 \wedge \bar{x_4}$$

выполнима — при подстановке $x_1 = 1, x_2 = 1, x_3 = 1$ и $x_4 = 0$ формула принимает истинное значение, — а формула

$$(x_1 \vee x_2) \wedge (\bar{x_1} \vee x_3) \wedge (\bar{x_2} \vee x_3) \wedge \bar{x_3}$$

невыполнима.

Задача выполнимости является исторически первой задачей, для которой была доказана NP-полнота, то есть что она принадлежит классу NP и любая другая задача из класса NP может быть сведена за полиномиальное время к SAT [37]. Данный результат был получен независимо двумя учеными: американцем Стивеном Куком [38] и русским Леонидом Левиным [39]. Важным следствием данной теоремы является то, что если существует детерминированный полиномиальный алгоритм для решения задачи выполнимости, то любая задача из класса NP может быть решения с помощью детерминированного полиномиального алгоритма. Вопрос существования такого алгоритма для решения SAT эквивалентен вопросу равенства классов P и NP, который является одной из семи задач тысячелетия [40] и считается одной из самых важных нерешенных задач в теоретической информатике.

1.1.2 Методы решения задачи выполнимости

Так как задача выполнимости является NP-полной, то все известные методы для ее решения в худшем случае работают экспоненциально от размера формулы время. Однако, на протяжении последний десятилетий был разработан ряд эвристических методов, способных определять выполнимость действительно больших формул с миллионами дизъюнктов и сотнями тысяч переменных, кодирующих различные практические задачи.

Все известные эффективные современные методы основаны на алгоритме *Дэвиса-Патнема-Логемана-Лавленда* (*Davis-Putnam-Logemann-Loveland* — DPLL) [14], который в свою очередь является усовершенствованной версией алгоритма *Дэвиса-Патнема* (*Davis-Putnam* — DP) [41].

Алгоритм DP был предложен в 1960 году для проверки истинности формулы логики первого порядка с помощью процедуры, основанной на правиле резолюций для исчисления высказываний.

В 1962 был предложен алгоритм DPLL для решения выполнимости булевой формулы, представленной в КНФ, который является усовершенствованием алгоритма DP. DPLL является полным алгоритмом поиска с возвратом. Несмотря на более чем пятидесятилетнюю историю, модификация алгоритма DPLL до сих пор лежит в основе современных программных средств для решения SAT.

В следующие годы предлагались различные модификации алгоритма DPLL, но поворотным моментом в истории стала разработка стратегии *управляемого конфликтами обучения дизъюнктов* (*conflict-driven clause learning* — CDCL) [15; 42]. Алгоритм CDCL лежит в основе всех современных программных средств для решения SAT. Множество различных дополнительных эвристик было предложено в последние два десятилетия после изобретения алгоритма CDCL, однако ни одна из них не дала столь значительного прироста эффективности. Среди современных программных средств для решения SAT можно выделить, например, *lingeling* [43], *CaDiCaL* [44], *CryptoMiniSAT* [45], Kissat [46] и другие [17]. В настоящей диссертации при реализации всех методов для работы с программными средствами для решения SAT использовалось программное средство `PySAT` [47], которое предоставляет удобный интерфейс для работы с большинством современных программных средств.

Помимо полных переборных алгоритмов, основанных на алгоритме DPLL, существуют, например, алгоритмы локального поиска, такие как GSAT [48] и WalkSAT [49], или вероятностные алгоритмы [50].

Ежегодно проводятся соревнования для определения лучшего программного средства для решения SAT *SAT Competition*, что способствует их постоянному развитию [16].

1.2 Генерация детерминированных конечных автоматов

В данном разделе приводятся основные понятия, связанные с детерминированными конечными автоматами, необходимые в настоящей диссертации. При-

водится постановка задачи генерации детерминированного конечного автомата минимального размера по заданным примерам поведения.

1.2.1 Базовые понятия

Алфавитом Σ называется некоторое конечное непустое множество символов. *Размером алфавита* Σ называется число его символов — $L = |\Sigma|$. В данной диссертации в основном будет рассматривать бинарный алфавит $\Sigma = \mathbb{B} = \{0, 1\}$. *Словом* (*строкой*, *цепочкой*) ω называется конечная последовательность символов некоторого алфавита. *Длиной слова* называется число символов в нем, обозначается как $|\omega|$. Множество слов длины k над алфавитом Σ обозначается как Σ^k. *Пустой строкой* называется слово, не содержащее ни одного символа. Такая строка, обозначаемая как ε, имеет нулевую длиной и может рассматриваться как слово над любым алфавитом — $\Sigma^0 = \{\varepsilon\}$. Множество всех возможных слов, составленных из символов некоторого алфавита Σ, является его замыканием:

$$\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k.$$

Подмножество множества всех слов над алфавитом Σ называется *языком* — $\mathcal{L} \subset \Sigma^*$.

Детерминированным конечным автоматом (ДКА) называется пятерка $\mathcal{D} = (D, \Sigma, \delta, d_1, D^+)$, где D — конечное множество состояний, Σ — алфавит входных символов, $\delta : D \times \Sigma \to D$ — *функция переходов*, d_1 — *стартовое* (*начальное*) состояние, $D^+ \subset D$ — множество *допускающих* (*принимающих*) состояний [51]. В неявном виде также задано множество *недопускающих* (*отвергающих*) состояний $D^- = D \setminus D^+$. *Размером* детерминированного конечного автомата называют число его состояний — $|D|$.

Индуктивно определим вспомогательную *расширенную функцию переходов* $\hat{\delta} : D \times \Sigma^* \to D$:

а) для любого состояния d_i верно, что переход по пустой строке не осуществляется — $\hat{\delta}(d_i, \varepsilon) = d_i$;

б) для любого состояния d_i верно, что переход по строке $\pi = \pi' c$, где $\pi, \pi' \in \Sigma^*$, $c \in \Sigma$, может быть определен следующим образом $\hat{\delta}(d_i, \pi) = \delta\left(\hat{\delta}(d_i, \pi'), c\right)$.

Говорят, что ДКА $\mathcal{D}$ *допускает* (*принимает*) слово ω, если $\hat{\delta}(d_1, \omega) \in D^+$. Иначе, если $\hat{\delta}(d_1, \omega) \in D^-$, говорят, что ДКА $\mathcal{D}$ *не допускает* (*отвергает*) слово ω. Множество всех слов, допускаемых автоматом $\mathcal{D}$, называется языком автомата $\mathcal{D}$: $\mathcal{L}(D) = \{\omega \mid \hat{\delta}(d_1, \omega)\}$.

Согласно приведенному определению детерминированные конечные автоматы всегда являются *полными* — для каждого состояния существуют исходящие переходы по всем символам алфавита. Данное определение общепринято, однако некоторые авторы (например, [52]) отходят от него и говорят, что для каждого состояния существует не более одного исходящего перехода по каждому из символов, то есть, что функция переходов является частично заданной. В настоящей диссертации используется классическое определение.

1.2.2 Изоморфные автоматы

При решении многих задач комбинаторной оптимизации, коей фактически является задача генерации ДКА минимального размера, зачастую существуют симметричные решения (симметрии) [53]. Симметрии увеличивают пространство поиска и, таким образом, значительная часть времени тратится на проверку новых решений, которые симметричны уже просмотренным. В задаче генерации ДКА по примерам поведения такой симметрией являются изоморфные автоматы. Автоматы называются *изоморфными* если существует биекция между их состояниями такая, что сохраняются все переходы, принимающие состояния соответствуют принимающим, а начальное — начальному. Иными словами, автоматы являются изоморфными, если они различаются только нумерацией состояний, но не своей

структурой. Тогда можно сделать вывод, что число изоморфных автоматов можно оценить с помощью числа перестановок. Таким образом, для некоторого ДКА $\mathcal{D}$ с M состояниями существует $\mathcal{O}(M!)$ изоморфных ему ДКА. На рисунке 1 представлены два изоморфных автомата, отличающиеся лишь нумерацией состояний.

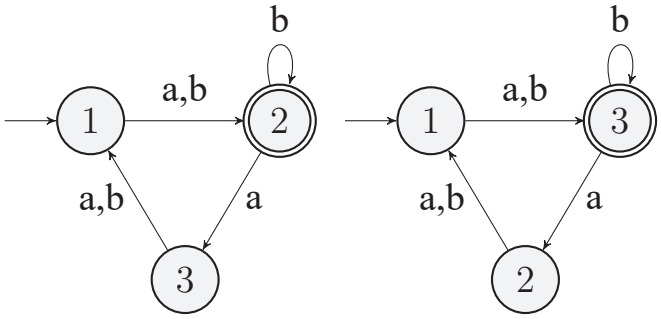


Рисунок 1 – Пример изомофрных ДКА

1.2.3 Задача генерации детерминированных конечных автоматов по заданным примерам поведения

В настоящей диссертации рассматривается задача пассивного построения ДКА по имеющимся примерам поведения [54; 55] — обучающему множеству строк, которые были приняты или отвергнуты некоторым неизвестным ДКА $\mathcal{U} = (U, \Sigma, \mu, u_1, U^+, U^-)$. Альтернативой является задача активного построения минимального ДКА [56–58], когда алгоритм построения может делать некоторые запросы к оракулу, который имеет информацию о неизвестном ДКА.

Для простоты будем считать, что примеры поведения представлены двумя множествами слов над алфавитом Σ: S_+ — множество слов, допущенных неизвестным автоматом $\mathcal{U}$; S_- — множество слов, отвергнутых неизвестным автоматом $\mathcal{U}$. Тогда задача генерации детерминированного конечного автомата заключается в поиске такого автомата, который будет принимать все слова из S_+ и отвергать все слова из S_-. Несмотря на то, что генерация любого ДКА по заданным примерам поведения может быть решена за линейное время, задача генерации

ДКА минимального размера (с минимальным числом состояний) является NP-полной [13]. Задача генерации ДКА является одной из многих задач обобщения знаний [59] — по конечному набору примеров поведения строится автомат, определяющий язык с бесконечным (хоть и счетным) числом слов. Тогда, чем меньше автомат — тем лучше он описывает имеющиеся данные. Итак, *задача генерации детерминированного конечного автомата по заданным примерам поведения* заключается в поиске ДКА минимального размера, соответствующего примерам поведения. На рисунке 2 приведен пример ДКА минимального размера, построенного по множества примеров поведения $S_+ = \{aba, bb, bba\}$ и $S_- = \{b, ba\}$.

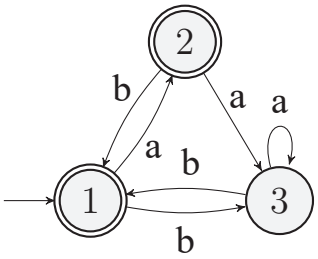


Рисунок 2 – Пример ДКА минимального размера, соответствующего наборам примеров поведения $S_+ = \{aba, bb, bba\}$ и $S_- = \{b, ba\}$

1.2.4 Расширенное префиксное дерево

Первым шагом большинства существующих подходов является построение *расширенного префиксного дерева* (augmented prefix tree acceptor — APTA) по имеющимся множествам примеров поведения S_+ и S_-. Расширенное префиксное дерево — это древовидная структура данных, основанная на обычном префиксном дереве (бор, нагруженное дерево, prefix tree, trie), отличающаяся от нее метками вершин. Более формально, префиксным деревом называется шестерка $\mathcal{T} = (T, \Sigma, \tau, t_1, T^+, T^-)$, где T — конечное множество вершин, Σ — алфавит входных символов, $\tau : T \times \Sigma \nrightarrow T$ — *функция переходов*, t_1 — *корневая* вершина (*корень*), $T^+ \subset T$ — множество *допускающих* (*принимающих*) вершин, $T^- \subset T$ — множество *не допускающих* (*отвергающих*) вершин. В отличие от функции переходов в ДКА, функция переходов τ является частичной. Также, в префиксном

дереве $T^+ \cup T^- \subset T$, то есть некоторые вершины могут не являться ни принимающими, ни отвергающими — промежуточные вершины.

Расширенное префиксное дерево для множеств примеров поведения S_+ и S_- строится как обычное префиксное дерево, но терминальные вершины для каждого слова помечаются соответствующей меткой. Вершины, в которых не заканчивается ни один из примеров поведения, остаются непомеченными. Пример расширенного префиксного дерева приведен на рисунку 3. Также далее в диссертации с помощью $\Delta(v)$ будет обозначаться расстояния от корневой вершины t_1 до вершины t_v.

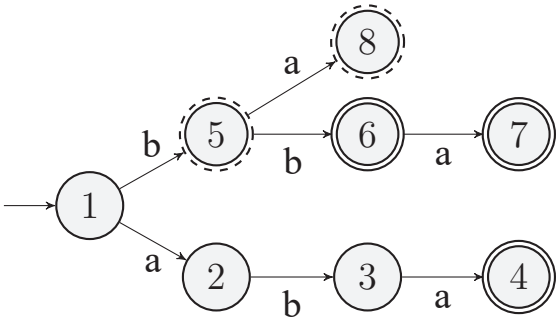


Рисунок 3 – Пример расширенного префиксного дерева, построенного по наборам примеров поведения $S_+ = \{aba, bb, bba\}$ и $S_- = \{b, ba\}$

1.3 Эвристические и метаэвристические методы генерации детерминированных конечных автоматов

В настоящем разделе приводится обзор существующих эвристических и метаэвристических методов генерации ДКА минимального размера по заданным примерам поведения. Отличительной особенностью таких методов является то, что они не являются точными — ими не гарантируется, что какой-то ДКА будет найден в принципе, а если ДКА найден, то не гарантируется его минимальность. Несмотря на то, что настоящая диссертация посвящена расширению границ применимости точных методов генерации ДКА, для построения достаточно больших

автоматов по большому числу примеров поведения неточные эвристические и метаэвристические методы все еще являются единственным возможным вариантом.

Эвристические алгоритмы. В основе эвристических алгоритмов лежит идея *слияния состояний* (*state merging*) — последовательных объединения вершин расширенного префиксного дерева и устранения образующейся недетерминированности. Первый известный алгоритм для решения данной задачи был предложен в работе Трахтенброта и Барздиня в 1970 году [21] — `TB`-алгоритм. Однако, предложенный алгоритм решает только частный случай задачи генерации ДКА по заданным примерам поведения: для некоторого натурального k все возможные слова длины k над алфавитом Σ — всего $|\Sigma|^k$ слов — должны содержаться во множествах S_+ и S_-. `TB`-алгоритм основан на полном переборе всех возможных пар состояний префиксного дерева и слиянии эквивалентных состояний в одно.

В 1978 году Голд доказал NP-полноту задачи генерации ДКА заданного, а значит и минимального, размера [13]. Ввиду указанной сложности новые алгоритмы генерации минимального ДКА не предлагались более десяти лет. В последующие годы разрабатывались исключительно неточные эвристические алгоритмы. Среди таких алгоритмов можно выделить следующие: `traxbar` [22], `RPNI` [23], `EDSM` [24], `exbar` [25], `Windowed-EDSM` [26]. Все перечисленные алгоритмы основаны на слиянии состояний расширенного префиксного дерева. Состояния для слияния выбираются эвристически, чем одновременно объясняется высокая скорость работы и неточность данных алгоритмов.

Самым успешным среди перечисленных эвристических алгоритмов можно считать алгоритм *объединения состояний на основе свидетельств* (*evidence-driven state merging* — EDSM) [24]. На каждой итерации алгоритм EDSM хранит текущую версию частично построенного ДКА. Изначально в качество такого ДКА выступает расширенное префиксное дерево. Затем на каждом шаге эвристически осуществляется выбор двух состояний текущего частично построенного ДКА, которые *объединяются* (*сливаются*) в одно состояние. При слиянии двух состояний может возникнуть недетерминированность — несколько исходящих перехо-

дов из нового состояния, помеченных одним и тем же символом. Недетерминированность устраняется с помощью рекурсивного слияния тех состояний, в которые ведут переходы с одинаковыми метками. Данный процесс повторяется до тех пор, пока существуют пары состояний, пригодные для слияния. Нетривиальным шагом является выбор состояний для объединения. Ситуация, когда происходит объединение принимающего и отвергающего состояния называется *конфликтом*. Для построения корректного ДКА не могут проводиться слияния, приводящие к конфликтам. Состояния, одно из которых помечено как принимающее, а другое — как отвергающее, не могут быть объединены в одно состояние, так как это незамедлительно приводит к конфликту. Если объединение двух состояний не приводит незамедлительно к конфликту (сливаются два принимающих состояния, два отвергающих состояния, либо одно из состояний является неопределенным), то конфликт может возникнуть при дальнейшем устранении недетерминированности. Перебор всех возможных пар состояний занимает экспоненциальное время, поэтому и используются эвристики для выбора подходящих пар. В основе алгоритма EDSM лежит эвристика выбора состояний для слияния, основанная на подсчете различных характеристик на поддеревьях потенциальных кандидатов для объединения. Например, может подсчитываться число исходящих переходов из каждого состояния по одним и тем же символам. Таким образом, жадно выбираются пары состояний имеющие максимальное значение соответствующих характеристик и происходит процесс слияния и устранения недетерминированности. Первые слияния состояний близких к корню расширенного префиксного дерева с большой вероятностью не увеличивают размер итогового автомата в виду большого размера поддеревьев и высоких показателей оценочных характеристик. Однако, чем ближе к листьям дерева, тем меньше поддеревья и тем меньше точность таких слияний. Данный алгоритм демонстрирует высокую скорость работы, однако генерируемый ДКА зачастую значительно больше, чем минимально возможный, соответствующий заданным примерам поведения. Алгоритм EDSM является победителем соревнования *Abbadingo One DFA Learning Competition* [24], посвященному генерации ДКА.

Метаэвристические алгоритмы. Среди метаэвристических алгоритмов, успешно применяемых для генерации конечных автоматов по примерам поведения, можно выделить эволюционные стратегии, генетические алгоритмы и муравьиные алгоритмы.

Эволюционные алгоритмы получили свое название в честь эволюционной теории Дарвина, упрощенная версия которой и лежит в их основе. Эволюционные стратегии и генетические алгоритмы являются разновидностями эволюционных алгоритмов. В эволюционных стратегиях обычно используются только оператор мутации, в то время как в генетических алгоритмах используются как оператор мутации, так и оператор скрещивания. В ходе своей работы такие алгоритмы поддерживают популяцию особей — в данном случае различных детерминированных конечных автоматов, — являющихся кандидатами на решение исходной задачи. Изначально автоматы выбираются случайным образом, а затем при помощи операторов мутации и скрещивания генерируются новые автоматы. На каждой итерации для каждого из полученных автоматов рассчитывается функция приспособленности, оценивающая насколько хорошо текущий автомат описывает заданные примеры поведения. Затем, лучшие из особей остаются в популяции, а худшие исключаются из рассмотрения. С помощью операторов мутации, меняющего незначительно одиночный автомат, и скрещивания, определенным образом объединяющих два автомата, генерируется новые особи на замену исключенным. Такой процесс повторяется, пока не будет найден автомат, достаточно хорошо описывающий примеры поведения, либо пока не выйдет заданное время. Среди эволюционных стратегий для генерации ДКА можно выделить алгоритмы, описанные в работе Лукаса и Рейнольдса [27] и в работе Гомеза [28]. Среди генетических алгоритмов можно выделить алгоритм, предложенный в Университете ИТМО ученым Егоровым, [60].

В работе Чивилихина [61], выполненной в Университете ИТМО, был предложен метод `MuACO` для генерации конечных автоматов по заданной функции приспособленности. В основе данного метода используются идеи муравьиных алгоритмов оптимизации [62]. Муравьиные алгоритмы — это класс метаэвристик,

предназначенных для решения оптимизационных задач. В основе данных алгоритмов лежит биологический процесс поиска пищи у муравьев. В методе MuACO используется такая структура данных, как граф мутаций. Вершинами данного графа являются конечные автоматы, потенциально являющиеся решением задачи генерации ДКА по заданным примерам поведения, а ребрам соответствуют мутации автоматов — небольшие изменения в структуре переходов автомата или во множестве допускающих состояний автомата. Каждому ребру в графе мутации соответствует некоторое значение эвристической информации и феромона. Эвристическая информация и феромон являются некоторыми вспомогательными величинами, используемыми при генерации новых автоматов. Каждый шаг алгоритма представляется в виде движения виртуальных муравьев по графу мутаций. Периодически значения эвристической информации и феромона обновляется в зависимости от того, насколько текущий автомат хорошо описывает заданные примеры поведения. Такой процесс также повторяется, пока не будет найден автомат, достаточно хорошо описывающий примеры поведения, либо пока не выйдет заданное время.

Метаэвристические алгоритмы также являются неточными — ими вообще не гарантируется, что какое-то решение будет найдено за конечное время, поэтому сравнение с ними в настоящей диссертации, направленной на развитие точных методов генерации ДКА, не проводилось.

1.4 Методы генерации детерминированных конечных автоматов, основанные на сведении к другим NP-трудным задачам

В настоящем разделе приводится обзор существующих точных методов генерации ДКА по заданным примерам поведения при помощи сведений к задачам раскраски графа и выполнимости булевой формулы. Также приводится обзор существующих подходов к нарушению симметрии.

Лучшие эвристики выбора пары состояний для слияния в алгоритме EDSM обычно основаны на успешных эвристиках для других гораздо более активно изучаемых задач, например, для задачи выполнимости или задачи раскраски графа. Как уже говорилось, ежегодно проходят соревнования по определению лучшего программного средства для решения SAT [16]. Несмотря на то, что различные эвристические стратегии выбора вершин для слияния повышают производительность метода EDSM и при его реализации используются оптимизированные структуры данных, данные реализации все еще значительно проигрывают по эффективности программным средствам для решения более проработанных задач. Так как задача генерации ДКА является NP-полной [13], то ее можно свести к задачам, которые изучаются дольше и интенсивнее, чтобы использовать максимально оптимизированные стратегии поиска решения.

1.4.1 Метод, основанный на сведении к задаче раскраски графа

В работе французских ученых Косте и Николя [30] предлагается сведение задачи генерации ДКА по заданным примерам поведения к задаче раскраски графа. На первом шаге алгоритма как и ранее строится расширенное префиксное дерево по заданным примерам поведения. Основной идей предложенного сведения является использование различных цветов для каждого из состояний генерируемого ДКА. Раскраска производится таким образом, что, если все вершины расширенного префиксного дерева, соответствующие одному цвету в раскрашиваемом графе, объединить в одно состояние автомата, и проделать данную операцию для всех цветов, то в итоге должен получиться детерминированный конечный автомат.

Ситуация, когда объединяются допускающая и отвергающая вершины графа, называется *конфликтом*, так как состояние ДКА может быть только либо допускающим, либо отвергающим. Очевидно, что тогда две вершины соединены ребром, если одна из них помечена как допускающая, а другая как отвергающая,

так как их объединение приводит к конфликту. Использование расширенного префиксного дерева, в котором существуют вершины, помеченные как промежуточные (то есть не являющиеся ни принимающими, ни отвергающими), позволяет в явном виде находить некоторые конфликты, которые возникают при объединении двух вершин, одна или две из которых являются промежуточными. Объединение двух вершин расширенного префиксного дерева в одну может привести к недетерминированности — для новой вершины будет существовать несколько исходящих переходов, помеченных одним символом. Избавиться от недетерминированности можно с помощью детерминизации — рекурсивного объединив вершины, куда ведут такие переходы. Если в процессе детерминизации возникает конфликт, то можно утверждать, что слияние вершин, объединенных на каждом из предыдущих шагов, приводит к конфликту.

Тогда, отметив в графе все найденные конфликты, можно перейти к решению задачи раскраски графа в k цветов. Так как исходная задача заключается в поиске ДКА с минимальным числом состояний, то требуется найти минимальное k для которого существует правильная раскраска. Добиться этого предлагается путем итеративного перебора числа цветов k, начиная с единицы и до тех пор, пока не будет найдено решение. На рисунке 4 представлен пример расширенного префиксного дерева, раскрашенного в три цвета так, что для каждого цвета объединение всех вершин данного цвета в одно состояние не приводит к несовместимости.

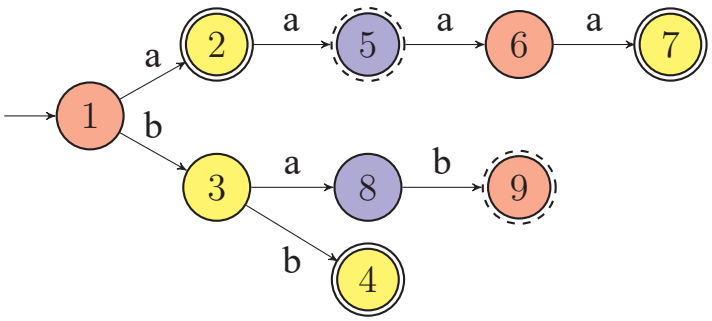


Рисунок 4 – Пример расширенного префиксного дерева, построенного по множествам примеров поведения $S_+ = \{a; aaaa; bb\}$ и $S_- = \{aa; bab\}$ и раскрашенного в три цвета так, что для каждого цвета объединение всех вершин данного цвета в одно состояние не приводит к несовместимости

На рисунке 5 представлен пример ДКА, построенного путем объединения вершин расширенного префиксного дерева, изображенного на рисунке 4, покрашенных в один цвет в одно состояние автомата.

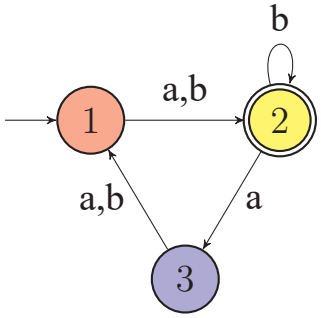


Рисунок 5 – Пример ДКА, получающегося после объединения вершин одного цвета в расширенном префиксном дереве, представленном на рисунке 4

1.4.2 Методы, основанные на сведении к задаче выполнимости

Как было сказано в разделе 1.1.2, программные средства для решения задачи выполнимости в последние несколько десятилетий — фактически с 1996 года, когда был предложен алгоритм CDCL — сильно развились и стали достаточно мощными средствами для решения многих прикладных задач, выраженных на языке SAT. Например, сведение к SAT используется для решения таких задач, как символьная проверка моделей [63], поиск связей между различными контекстами в распределенных системах [64], поиск ошибок в программах на языке программирования C [65], поиск кратчайшего синхронизирующего слова [66], кластеризация в ограничения [67], генерация оптимальных деревьев решений [68] и множества других.

DFASAT. В 2010 году в статье [18] учеными Марейном Хойлом (Marijn Heule) и Сикко Вервером (Sicco Verwer) впервые был предложен метод генерации ДКА по заданным примерам поведения, основанный на сведении к задаче выполнимости, названный DFASAT. Метод DFASAT основан на описанном в предыдущем разделе сведении задачи генерации ДКА к задаче раскраски графа. Метод DFASAT

состоит из следующих этапов, которые будут подробнее описаны далее. Как и ранее, первым шагом метода является построение расширенного префиксного дерева по имеющимся примерам поведения. Затем, выбирается нижняя оценка на размера искомого автомата M. В простейшем случае, в качестве нижней оценки выбирается значение $M = 2$, если имеются как положительные, так и отрицательные примеры поведения, однако, какие-то более сложные оценки могут также использоваться. Далее строится булева КНФ-формула, кодирующая задачу генерации ДКА текущего размера M, соответствующего заданным примерам поведения, представленным в виде расширенного префиксного дерева. Построенная формула передается на вход программному средству для решения SAT, которое решает задачу выполнимости. Если формула выполнима, то программное средство возвращает выполняющую подстановку, по которой, затем, строится ДКА, являющийся решением исходной задачи генерации ДКА минимального размера по примерам поведения. В случае, если формула невыполнима, размер искомого автомата увеличивается на единицу и процесс повторяется заново с построения булевой КНФ-формулы. Такой итеративный процесс повторяется до тех пор, пока для некоторого M не будет найдена выполняющая выполняющая подстановка для соответствующей формулы, что гарантирует минимальность найденного автомата. Схема метода DFASAT представлена на рисунке 6.

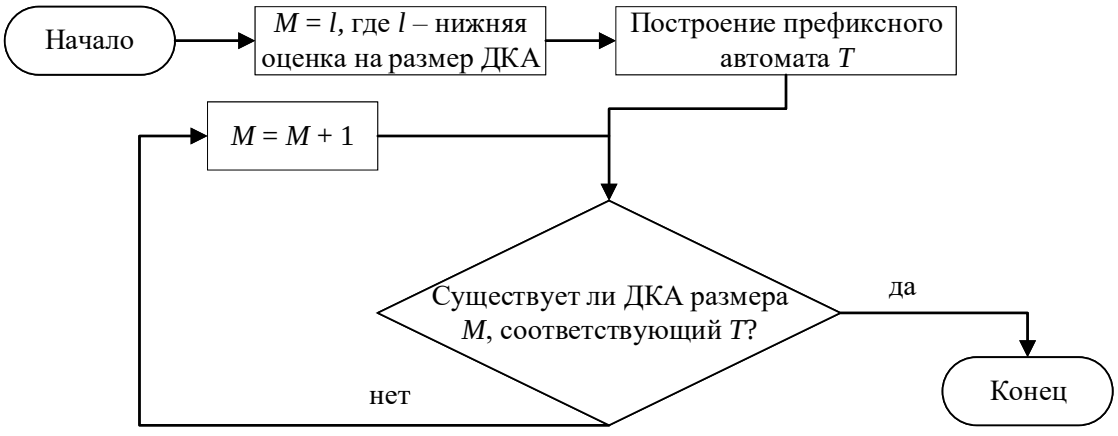


Рисунок 6 – Схема точного метода генерации ДКА по заданным примерам поведения на основе сведения к SAT — DFASAT

Построение префиксного дерева описывалось в разделе 1.2.4. Следующим шагом метода DFASAT является построение булевой формулы на языке SAT. Су-

ществует широко известное сведение задачи раскраски графа к SAT, которое Хойл и Вервер называют *прямым кодированием* (*direct encoding*) [69]. Однако, авторы показывают, что использование прямого кодирования для задачи генерации ДКА приводит к построению булевой формулы, состоящей из $\mathcal{O}\left(N^2 \times M^2\right)$, где N — размер префиксного дерева, M — размер генерируемого автомата, дизъюнктов. Для нетривиальных примеров задачи генерации ДКА формула такого сложности получается слишком большой для современных программных средств для решения SAT, поэтому голландские ученые предложили *компактное сведение* (*compact encoding*).

Здесь, и далее в диссертации, используется следующее обозначение: $[N] = 1, 2, \ldots, N$. Для компактного сведения необходим ввести три типа булевых переменных:

а) переменные соответствия $\{x_{v,i}\}_{v \in T, i \in [M]}$, которые истинны тогда и только тогда, когда вершина t_v в расширенном префиксном дереве $\mathcal{T}$ соответствует состоянию d_i в автомате $\mathcal{D}$;

б) переменные перехода $\{y_{i,l,j}\}_{i,j \in [M], l \in \Sigma}$, которые истинны тогда и только тогда, когда в автомате $\mathcal{D}$ существует переход из состояния d_i в состояние d_j по символу l;

в) переменные допуска $\{z_i\}_{i \in [M]}$, которые истинны тогда и только тогда, когда в автомате $\mathcal{D}$ состояние d_i является допускающим.

Переменные $x_{v,i}$ задают связь между префиксным деревом и генерируемым ДКА. Переменные $y_{i,l,j}$ и z_i полностью определяют генерируемый ДКА. Действительно, алфавит (Σ), множество состояний (всего существует M состояний, и они пронумерованны числами от 1 до M) и стартовое состояние (всегда имеет номер 1) задаются в неявном виде, в то время как функция переходов задается с помощью переменных $y_{i,l,j}$ и множество допускающих состояний с помощью переменных z_i.

Используя вышеопределенные переменные, компактное сведение представляется с помощью следующих множеств дизъюнктов:

а) $(x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,M})$ для $v \in [N]$ — каждой вершине t_v расширенного префиксного дерева $\mathcal{T}$ в соответствие ставится как минимум одно состояние автомата $\mathcal{D}$.

б) $(\neg y_{i,l,j} \vee \neg y_{i,l,h})$ для $i, j, h \in [M]; j < h; l \in \Sigma$ — из каждого состояния d_i автомата $\mathcal{D}$ существует не более одного перехода по каждому символу l алфавита Σ, иными словами, ДКА $\mathcal{D}$ детерминирован.

в) $(\neg x_{v,i} \vee z_i)$ для $t_v \in T^+; i \in [M]$ — если принимающей вершине t_v расширенного префиксного дерева $\mathcal{T}$ в соответствие ставится состояние d_i автомата $\mathcal{D}$, то это состояние также должно быть принимающим.

г) $(\neg x_{v,i} \vee \neg z_i)$ для $t_v \in T^-; i \in [M]$ — если отвергающей вершине t_v расширенного префиксного дерева $\mathcal{T}$ в соответствие ставится состояние d_i автомата $\mathcal{D}$, то это состояние также должно быть отвергающим.

д) $(\neg x_{v,i} \vee \neg x_{w,j} \vee y_{i,l,j})$ для $v, w \in [N]; i, j \in [M]; l \in \Sigma; \tau(t_v, l) = t_w)$ — если вершине t_v расширенного префиксного дерева $\mathcal{T}$ в соответствие ставится состояние d_i автомата $\mathcal{D}$, вершине t_w — состояние d_j и в префиксном дереве $\mathcal{T}$ существует переход из вершины t_v в вершину t_w по символу l, то в автомате $\mathcal{D}$ должен быть переход из состояния d_i в состояние d_j по символу l.

е) $x_{1,1}$ — корню t_1 расширенного префиксного дерева $\mathcal{T}$ в соответствие ставится начальное состояние d_1 автомата $\mathcal{D}$.

Все представленные выше наборы дизъюнктов могут быть объединены с помощью конкатенации в одну большую булеву формулу, которая затем передается программному средству для решения SAT. Перечисленных множеств дизъюнктов достаточно для генерации ДКА размера M, соответствующего примерам поведения, однако Хойл и Вервер в [18] предложили ряд дополнительных ограничений, которые упрощают работу программного средства, сокращая пространство поиска. Так, был предложен ряд дополнительных (*redundant*) дизъюнктов:

а) $(\neg x_{v,i} \vee \neg x_{v,j})$ для $v \in [N]\,; i, j \in [M]\,; i < j$ — каждой вершине t_v расширенного префиксного дерева $\mathcal{T}$ в соответствие ставится не более одного состояния автомата $\mathcal{D}$.

б) $(y_{i,l,1} \vee y_{i,l,2} \vee \ldots \vee y_{i,l,M})$ для $i \in [M]\,; l \in \Sigma$ — из каждого состояния d_i автомата $\mathcal{D}$ существует как минимум один переход по каждому символу l алфавита Σ, иными словами, ДКА $\mathcal{D}$ полон.

в) $(\neg x_{v,i} \vee \neg y_{i,l,j} \vee x_{w,j})$ для $v, w \in [N]\,; i, j \in [M]\,; l \in \Sigma; \tau\,(t_v, l) = t_w)$ — если вершине t_v расширенного префиксного дерева $\mathcal{T}$ в соответствие ставится состояние d_i автомата $\mathcal{D}$, и в префиксном дереве $\mathcal{T}$ существует переход из вершины t_v в вершину t_w по символу l и в автомате $\mathcal{D}$ существует переход из состояния d_i в состояние d_j по символу l, вершине t_w дерева $\mathcal{T}$ должно ставиться в соответствие состояние d_j.

Все представленные выше наборы дизъюнктов могут быть объединены с помощью конкатенации в одну большую булеву формулу. Всего в такой формуле будет $\mathcal{O}(N \times M^2)$ дизъюнктов и для их кодирования будет использовано $\mathcal{O}(M^2 + N \times M)$ переменных.

Помимо дополнительных дизъюнктов, авторы [18] предложили использовать вспомогательную структуру данных, названную ими *графом совместимости* (*consistency graph*). Несмотря на оригинальное название, как будет видно далее, данная структура данных должна скорее называться *графом несовместимости* (*inconsistency graph*). В настоящей диссертации предлагается использовать последнее название.

Две вершины t_v и t_w расширенного префиксного дерева можно объединить в одну вершину $t_{v'}$, объединив множества исходящих из них переходов. Если в результате данной операции в префиксном дереве возникла недетерминированность, то есть из вершины $t_{v'}$ теперь исходят два различных перехода по одному и тому же символу в различные вершины t_q и t_r, то от нее можно избавиться объединив вершины t_q и t_r в одну вершину $t_{q'}$. Рекурсивно продолжая данный процесс, можно избавиться от всех случаев недетерминированности в префиксном

дереве. Важным фактом является то, что если в процессе избавления от недетерминированности в какой-то момент приходится объединить принимающую вершину с отвергающей, то изначальное слияние вершин t_v и t_w невозможно. В таком случае говорят, что вершины t_v и t_w *несовместимы* (*inconsistent*). Данную информацию можно использовать для помощи программному средству для решения задачи SAT.

Более формально, по имеющемуся расширенному префиксному дереву $\mathcal{T} = (T, \Sigma, \tau, t_1, T^+, T^-)$ предлагается построить граф $\mathcal{I} = (V, E)$ такой, что его множество вершин V совпадает со множеством вершин $\mathcal{T}$ префиксного дерева $\mathcal{T}$, а множество ребер E определяется следующим образом. Две вершины в графе $\mathcal{I}$ соединены ребром тогда и только тогда, когда их объединение и последующее избавление от недетерминированности приводит к несовместимости.

Так как искомый ДКА $\mathcal{D}$ соответствует префиксному дереву $\mathcal{T}$, то если две вершины t_v и t_w смежны в графе несовместимости $\mathcal{I}$, то они не могут соответствовать одному и тому же состоянию d_i автомата $\mathcal{D}$. Данное свойство может быть выражено с помощью переменных $x_{v,i}$ в виде следующего множества дизъюнктов — $(\neg x_{v,i} \vee \neg x_{w,i})$ для $v, w \in [N]\,; (v, w) \in E; i \in [M]$. Такие дизъюнкты необязательны, но их добавление к основной булевой формуле помогает значительно сократить пространство поиска программного средства для решения SAT. Однако, надо заметить, что в общем случае таких дизъюнктов будет $\mathcal{O}\left(N^2 \times M\right)$, что на порядок относительно N увеличивает размер формул. Использование графа несовместимости в таком виде для построения больших автоматов по большому множеству примеров поведения не представляется возможным ввиду размера итоговой булевой формулы.

Как было сказано в разделе 1.2.2, при отсутствии каких-либо ограничений на нумерацию состояний автомата, существует $M!$ изоморфных автоматов. Авторы [18] предложили свой способ нарушения симметрии, позволяющий сократить число рассматриваемых изоморфных автоматов. Так как в графе несовместимости смежные вершины по определению не могут соответствовать одному состоянию в искомом ДКА, то все вершины в некоторой клике (клика — полный граф, граф

в котором каждая вершина соединена ребром со всеми остальными) [70] такого графа будут соответствовать различным состояниям автомата. Учитывая, что итоговая нумерация состояний автомата не важна, можно, не уменьшая общности, зафиксировать номера вершин такой клики, что и было предложено в рассматриваемой статье.

Однако, задача поиска клики максимального размера в некотором графе является NP-полной [71]. Поэтому авторами было предложено найти клику большого, но не обязательно максимального размера. Сделать это можно с помощью следующего эвристического подхода. В графе ищется вершина максимальной степени, которая будет входить в искомую большую клику. Затем ищется смежная ей вершина максимальной степени и добавляется в клику. Затем ищется вершина максимальной степени смежная обеим предыдущим и также добавляется в клику. Повторяя данный процесс, пока есть возможность существуют вершины смежные всем уже добавленным в клику.

Таким образом, если размер найденной клики равен C, то вместо $M!$ изоморфных автоматов будут рассмотрены $(M - C)!$ изоморфных автомата. Учитывая скорость роста факториала, разница между $M!$ и $(M - C)!$ может быть значительной, однако в общем случае число рассматриваемых изоморфных автоматов все еще остается факториальным относительно размера автомата. Помимо вышеуказанного недостатка, данный подход требует обязательного построения графа несовместимости, что, как было показано ранее, далеко не всегда является возможным.

Несмотря на то, что DFASAT является первым точным методом генерации ДКА по заданным примерам поведения, способным строить автомат хотя бы с 10 состояниями, автоматы большего размера с его помощью не сгенерировать. Для упрощения задачи, Хойл и Вервер предложили сначала выполнить несколько шагов алгоритма EDSM. Каждое объединение состояний сильно уменьшает размера префиксного дерева и сокращает пространство поиска. Первые слияния в алгоритме EDSM с большой вероятностью не являются ошибочными — не приводят к увеличению размера минимально возможного ДКА, соответствующего

префиксному дереву, — но в общем случае даже после одного слияния состояний не гарантируется, что размер минимального возможного автомата останется тем же, а значит, такой алгоритм не является точным.

Предикаты нарушения симметрии на основе алгоритма обхода графа в ширину. Идеальным нарушением симметрии является ситуация, когда для каждого класса эквивалентности относительно симметрии остается единственный представитель. Для задачи генерации ДКА — когда для каждого класса эквивалентности по изоморфизму остается единственный представитель. Автором настоящей диссертации совместно с научным руководитем в [31] были разработаны предикаты нарушения симметрии на основе алгоритма обхода графа в ширину, которые позволяют добиться идеального нарушения симметрии.

Идея предложенного подхода состоит в том, чтобы зафиксировать нумерацию состояний в порядке *обхода в ширину* (*breadth-first search* — BFS) [72] данного автомата. Для того, чтобы сделать обход в ширину уникальным, необходимо зафиксировать некоторый порядок на входных символах переходов, например, лексикографический. Будем называть ДКА *BFS-пронумерованным*, если нумерация его состояний соответствует порядку обхода его состояний в ширину. Тогда в каждом классе эквивалентности по изоморфизму будет ровно один BFS-пронумерованный представитель.

Другими словами, если рассмотреть дерево BFS, построенное для некоторого ДКА, расположив при этом детей в соответствии с выбранным порядком на символах переходов, тогда номера состояний:

- должны быть различными числами от 1 до M;
- на одной глубине должны увеличиваться слева направо (*порядок по уровню*);
- должны увеличиваться сверху вниз (*порядок по глубине*).

Пример BFS-пронумерованного автомата представлен на рисунке 7. На рисунке 8 представлено соответствующее ему BFS-дерево.

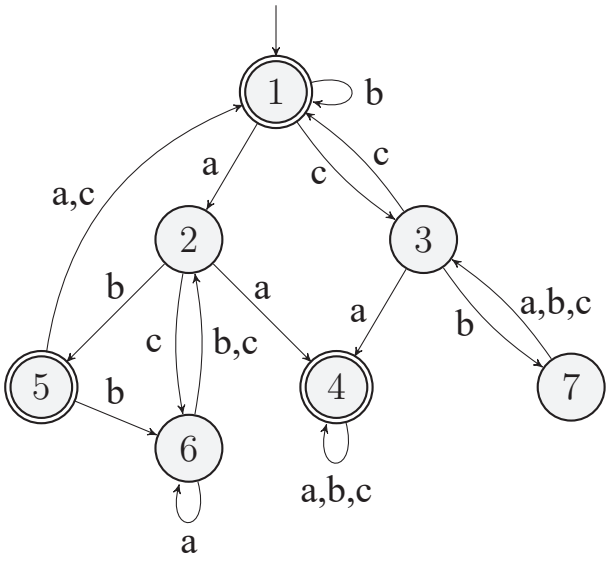


Рисунок 7 – Пример BFS-пронумерованного автомата

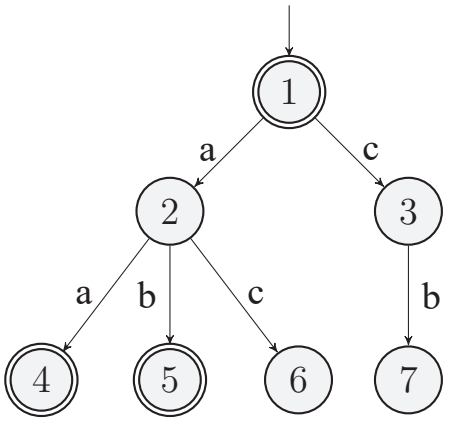


Рисунок 8 – BFS-дерево для автомата, представленного на рисунке 7

Если закодировать требование к автомату, чтобы он был BFS-пронумерованным, в виде булевых предикатов и добавить к имеющейся формуле, предложенной в [18], то программное средство для решения SAT будет искать автоматы, удовлетворяющие заданным примерам поведения, пронумерованные в порядке BFS. Для того, чтобы закодировать данное требование, было предложено ввести три новых типа булевых переменных:

а) переменные родителей $\{p_{j,i}\}_{1 \leqslant i < j \leqslant M}$, которые истинны тогда и только тогда, когда состояние d_i является родителем состояния d_j в BFS-дереве автомата $\mathcal{D}$;

б) переменные наличия переходов $\{t_{i,j}\}_{1\leqslant i<j\leqslant M}$, которые истинны тогда и только тогда, когда в автомате $\mathcal{D}$ существует переход из состояния d_i в состояние d_j;

в) переменные минимального символа $\{m_{i,l,j}\}_{1\leqslant i<j\leqslant M;l\in\Sigma}$, которые истинны тогда и только тогда, когда в автомате $\mathcal{D}$ из состояния d_i в состояние d_j существует переход по символу l, но не существует переходов по меньшим (согласно выбранному порядку на символах) символам. Данные переменные используются только в случае небинарного алфавита.

Используя данные переменные, можно закодировать свойство BFS-пронумерованности автомата с помощью следующих множеств дизъюнктов.

а) $(t_{i,j} \leftrightarrow y_{i,l_1,j} \vee y_{i,l_2,j} \vee \ldots \vee y_{i,l_L,j})$ для $1 \leqslant i < j \leqslant M; l_k \in \Sigma$ — в автомате $\mathcal{D}$ переход из состояния d_i в состояние d_j существует тогда и только тогда, когду существует переход из состояния d_i в состояние d_j хотя бы по одному из символов алфавита Σ.

б) $(p_{j,i} \leftrightarrow t_{i,j} \wedge \neg t_{i-1,j} \wedge \neg t_{i-2,j} \wedge \ldots \wedge \neg t_{1,j})$ для $1 \leqslant i < j \leqslant M$ — состояние d_i автомата $\mathcal{D}$ является родителем состояния d_j в BFS-дереве, если из состояния d_i существует переход в состояние d_j, а из состояний с меньшим номером такого перехода не существует.

в) $(p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1})$ для $2 \leqslant j \leqslant M$ — у каждого состояния d_j автомата $\mathcal{D}$, кроме стартового, родитель в BFS-дереве должен иметь меньший номер. Данные дизъюнкты позволяют закодировать порядок по глубине в поддереве.

г) $(p_{j,i} \to \neg p_{j+1,k})$ для $1 \leqslant k < i < j \leqslant M$ — если состояние d_i является родителем в дереве BFS состояния d_j, то у состояния с б*о*льшим номером d_{j+1} родитель не может иметь номер меньший, чем i. Данные дизъюнкты позволяют задать порядок по уровню для детей различных родителей, а также порядок по глубине в разных поддеревьях.

д) $(p_{j,i} \wedge p_{j+1,i} \to y_{i,l_1,j})$ для $1 \leqslant i < j \leqslant M; \Sigma = \{l_1, l_2\}$ — если состояние d_i является родителем в дереве BFS двух состояний d_j и d_{j+1} и алфавит

бинарный, то переход из состояния d_i в d_j должен быть по меньшему символу. В случае бинарного алфавита данного множества дизъюнктов достаточно, чтобы упорядочить двух детей d_j и d_{j+1} одного состояния d_i. Дополнительно, для сокращения пространства поиска, можно добавить множество дизъюнктов $(p_{j,i} \wedge p_{j+1,i} \rightarrow y_{i,l_2,j+1})$ для $1 \leqslant i < j \leqslant M$. Данные дизъюнкты задают порядок по уровню для детей одного родителя в случае бинарного алфавита.

е) $\left(m_{i,l_n,j} \leftrightarrow y_{i,l_n,j} \wedge \neg y_{i,l_{n-1},j} \wedge \neg y_{i,l_{n-2},j} \wedge \ldots \wedge \neg y_{i,l_1,j}\right)$ для $1 \leqslant i < j \leqslant M; 1 \leqslant n \leqslant L; l_n \in \Sigma$ — в автомате $\mathcal{D}$ символ l_n является минимальным символом на переходах из состояния d_i в состояние d_j тогда и только тогда, когда в автомате существует переход из состояния d_i в состояние d_j по символу l_n, но не существует переходов по меньшим (относительно выбранного порядка) символам. Данное множество дизъюнктов используется в случае небинарного алфавита.

ж) $(p_{j,i} \wedge p_{j+1,i} \wedge m_{i,l_n,j} \rightarrow \neg m_{i,l_k,j+1})$ для $1 \leqslant i < j \leqslant M; 1 \leqslant k < n \leqslant L; l_n, l_k \in \Sigma$ — если состояние d_i является родителем в дереве BFS двух состояний d_j и d_{j+1} и алфавит состоит из более чем двух символов, то из состояния d_i переход в состояние d_j должен быть по меньшему (относительно выбранного порядка) символу чем в состояние d_{j+1}. Данные дизъюнкты задают порядок по уровню для детей одного родителя в случае небинарного алфавита.

Все представленные выше наборы дизъюнктов могут быть объединены с помощью конкатенации в одну большую булеву формулу. Всего в такой формуле будет $\mathcal{O}(M^3 + M^2 \times L^2)$ дизъюнктов и для их кодирования будет использовано $\mathcal{O}(M^2 \times L)$ переменных.

1.5 Подход уточнения абстракции по контрпримерам

Программные системы с каждым годом становятся все сложнее, и все сильнее растет их роль в жизни общества — все больше различных ответственных промышленных и повседневных процессов автоматизируются с помощью компьютеров и программных средства. Наличие ошибок в программном коде во многих случаях может привести к серьезным последствиям [73]: значительным финансовым убыткам, утечке важной информации, нанесению вреда здоровью людей и даже к их смерти. Поэтому значительную актуальность в последние десятилетия приобрела разработка применимых на практике методов проверки корректности программ. Такие методы можно разделить на две большие группы: динамические и статические методы проверки. В основе динамических методов лежит методология тестирования, когда проверка за корректностью программы происходит во время ее работы. Однако, с помощью такого тестирования можно обнаружить только те ошибки, которые проявляются во время конкретного воспроизведенного сценария выполнения программы. Даже в не самых больших и важных программах зачастую существует такое множество ситуаций, потенциально приводящих к ошибкам, что их полное покрытие тестами не представляется возможным. Статические же методы проверяют корректность программного кода без его выполнения и запуска программы и потенциально способны обнаружить все возможные ошибки или доказать их отсутствие. Однако, Тьюринг в 1936 году доказал, что проблема останова неразрешима [74], и, тем самым, показал, что невозможность существования алгоритма, способного доказать корректность некоторой программы в общем случае. Впоследствии было предложено множество алгоритмов, способных формально доказывать корректность либо небольших программ, либо каких упрощенных моделей программ.

Среди таких статических методов можно выделить метод *уточнения абстракции по контрпримерам* (counterexample-guided abstraction refinement — CEGAR), который впервые был предложен в 2000 году Эдмундом Кларком и кол-

легами [32]. Данный метод был разработан для автоматизированного итеративного построения абстрактной модели программы, которую необходимо верифицировать — доказать корректность. Подробный обзор метода уточнения абстракции по контрпримерам можно найти в работе российских ученых Мандрыкина, Мутилина и Хорошилова из Института системного программирования им. В.П. Иванникова РАН [33].

В основе данного метода лежит понятие *абстракции*. При использовании абстракции вместо реальных состояний верифицируемой программы рассматриваются абстрактные состояния, которые объединяют в себе множества реальных состояний. При этом, эти множества не обязательно должны быть непересекающимися. На первом шаге алгоритма выбираются какие-то абстрактные состояния (эвристически, случайно или на основе каких-то известных данных). Затем, на основе этих состояний строится *абстрактное дерево достижимости*, в котором абстрактные состояния соединены переходом тогда и только тогда, когда переходом соединены какие-то из соответствующих реальных состояний. Таким образом по построению получается, что все пути, по которым возможно выполнение в реальной программе, также представлены в абстрактном дереве, но не наоборот.

Из вышесказанного следует, что если ошибочные состояния (описывающие нежелательное состояние программы) не достижимы в абстрактном дереве, то они не достижимы и в исходной программе. Если же существует путь в абстрактном дереве в некоторое ошибочное состояние, то такой путь называется контрпримером. Наличие контрпримера говорит либо о том, что в исходной программе есть ошибка, либо о том, что построенная абстрактная модель неточна. Найденный контрпример проверяется на исходной программе и, если соответствующий путь содержится в ней, то в программе найдена ошибка. Иначе, с помощью найденного контрпримера абстракция уточняется — перестраивается абстрактное дерево достижимости. Данный процесс повторяется до тех пор, пока не будет найдена ошибка в исходной программе, либо пока не будет доказано, что в исходной программе ошибки отсутствуют.

Несмотря на то, что данный метод скорее относится к классу методов активного обучения, перспективным выглядит применение схожей идеи для улучшения точных методов генерации ДКА по заданны примерам поведения. Так, всеми методами, основанными на сведении к SAT, генерируется булева формула, размер которой линейно зависит от размера расширенного префиксного дерева — $\mathcal{O}\left(N \times M^2\right)$, где N — размер префиксного дерева, а M — размер генерируемого автомата (см. раздел 1.4.2). Таким образом, при наличии *избыточного* числа примеров поведения и относительно небольшом ДКА, булева формула, кодирующая задачу генерации ДКА, может быть слишком большой для современных программных средств для решения SAT. В настоящей диссертации будет разработан метод, позволяющий совместить в себе сведение к SAT и идеи подхода уточнения абстракции по контрпримерам, для генерации ДКА.

Выводы по главе 1

В первой главе был произведен обзор предметной области, приведена терминология, а также изложены известные результаты ряда разделов информатики, необходимых для описания предлагаемых в диссертации методов:

Дана постановка задачи SAT, приведены необходимые определения, а также описаны основные методы решения данной задачи. Задача выполнимости является одной из самых активно исследуемых NP-трудных задач, что подтверждается проведением ежегодных конференций, посвященных SAT, а также проведением ежегодных соревнований по выявлению лучшего программного средства для ее решения. Поэтому множество других задач из класса NP могут быть эффективно решены при помощи сведения к задаче выполнимости и использования современных программных средств для решения SAT. Однако, несмотря на постоянное повышение эффективности методов решения SAT, в худшем случае для поиска решения требуется экспоненциальное относительно размера входных данных время.

Даны базовые понятия о детерминированных конечных автоматах, об изоморфизме и постановка задачи генерации ДКА по заданным примерам поведения. Также, приведено определение расширенного префиксного дерева — древовидной структуры данных для представления множества примеров поведения.

Проведен анализ работ, посвященных методам генерации ДКА по заданным примерам поведения. Рассмотрены три различных типа методов генерации ДКА, основанных на: эвристических алгоритмах (различные методы слияния состояний); метаэвристических алгоритмах (генетические и муравьиные алгоритмы); сведении к NP-полным задачам раскраски графа и выполнимости булевой формулы. Эвристические и метаэвристические методы являются неточными — первые не гарантируют минимальности сгенерированного ДКА, вторые вообще не гарантируют, что какой-то автомат будет найден за конечное время. Методы, основанные на сведении к NP-полным задачам являются точными — ими гарантируется, что ДКА, удовлетворяющий примерам поведения, будет найден за конечное время и будет состоять из минимального числа состояний.

Также был описан подход уточнения абстракции по контрпримерам, изначально предложенный для итеративного построения модели программного обеспечения для последующей верификации.

Исходя из анализа литературы можно сделать следующие выводы.

– Булева формула, получаемая при генерации ДКА большого размера по большому числу примеров поведения при помощи сведения к SAT, зачастую слишком сложна для современных программных средств для решения SAT, что актуализирует разработку новых предикатов нарушения симметрии, сокращающих пространство поиска.
– Предложенное ранее кодирование предикатов нарушения симметрии на основе BFS, образуют булеву формулу, состоящую из $\mathcal{O}\left(M^3 + M^2 \times L^2\right)$ дизъюнктов (где M — размер генерируемого ДКА, L — размер алфавита), что неприменимо для автоматов большого размера.
– Размер булевой формулы зависит от размера расширенного префиксного дерева, а значит и от числа и длины заданных примеров поведения. В слу-

чае избыточного набора примеров поведения не существует применимых на практике методов генерации ДКА. Потенциальным решением может стать разработка метода, сочетающего в себе сведение к задаче выполнимости и идеи подхода уточнения абстракции по контрпримерам.

– Ранее не предлагалось методов генерации всех неизоморфных ДКА минимального размера, соответствующих заданным примерам поведения. Более того, без использования предикатов нарушения симметрии на основе кодирования алгоритма BFS, разработка эффективных методов генерации всех ДКА не представляется возможной ввиду комбинаторного взрыва числа рассматриваемых автоматов.

Глава 2 Сокращение пространства поиска при генерации детерминированных конечных автоматов с использованием сведения к задаче выполнимости

Настоящая глава посвящена разработке предикатов нарушения симметрии, основанных на алгоритмах обхода графа в ширину и в глубину, для сокращения пространства поиска при решении задачи выполнимости, а также разработке, реализации и экспериментальным исследованиям методов генерации ДКА по заданным примерам поведения, использующих данные предикаты.

2.1 Предикаты нарушения симметрии на основе кодирования алгоритма обхода в глубину

В настоящем разделе описывается разработка предикатов нарушения симметрии на основе алгоритма обхода графа в глубину. Предикаты нарушения симметрии, основанные на алгоритме обхода графа в ширину, описанные в разделе 1.4.2, помогли существенно улучшить производительность существующих точных методов генерации ДКА по заданным примерам поведения. Логичным продолжением данного исследования является разработка предикатов нарушения симметрии на основе алгоритма обхода графа в глубину (depth-first search — DFS). Данные предикаты нарушения симметрии являются модификацией BFS-предикатов и допускают только ДКА, пронумерованные в порядке обхода в глубину.

Во время обхода графа (или автомата) в глубину необходимо находить все смежные непосещенные состояния для каждого состояния ДКА. Обход начинается со стартового состояния автомата. На каждом шаге текущее состояние помечается как посещенное. Затем перебираются все исходящие из рассматриваемого состояния переходы, и, если переход ведет в еще не посещенное состояние, алгоритм запускается рекурсивно из этого состояния. После возвращения в текущее

состояние перебор исходящих переходов продолжается и прекращается, когда не останется смежных непосещенных состояний [75]. Если зафиксировать порядок перебора переходов (например, лексикографически по символам на переходах), то снова для каждого автомата существует единственный способ обойти в порядке DFS. А значит, если закодировать свойство *DFS-пронумерованности* в виде булевой формулы, то получившиеся предикаты нарушения симметрии будут как BFS-предикаты допускать единственного представителя для каждого класса эквивалентности по изоморфизму. Пример DFS-пронумерованного автомата представлен на рисунке 9. На рисунке 10 представлено соответствующее ему DFS-дерево.

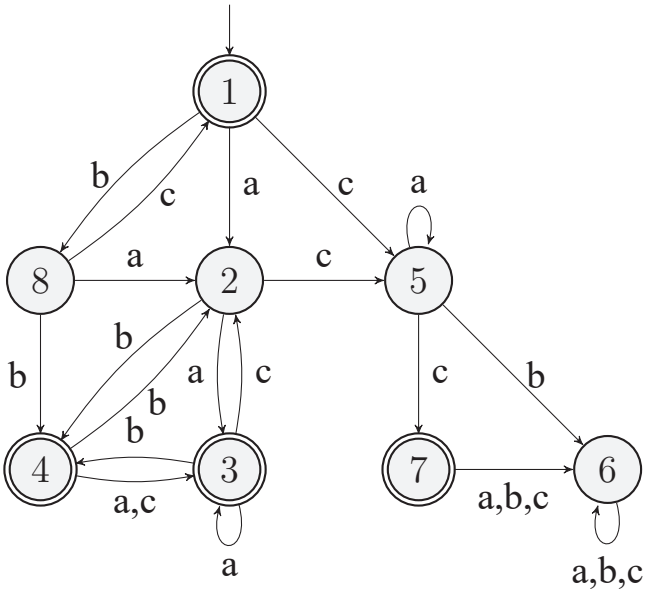


Рисунок 9 – Пример DFS-пронумерованного автомата

Наборы переменных и дизъюнктов, требуемых для кодирования свойства DFS-пронумерованности детерминированного конечного автомата, схожи с наборами для BFS-предикатов, но имеют некоторые отличия. Для DFS-предикатов используются следующие наборы переменных:

а) переменные родителей $\{p_{j,i}\}_{1 \leqslant i < j \leqslant M}$, которые истинны тогда и только тогда, когда состояние d_i является родителем состояния d_j в DFS-дереве автомата $\mathcal{D}$;

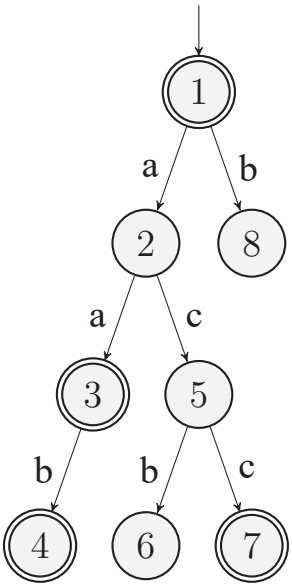


Рисунок 10 – Дерево DFS для автомата, представленного на рисунке 9

б) переменные наличия переходов $\{t_{i,j}\}_{1 \leqslant i < j \leqslant M}$, которые истинны тогда и только тогда, когда в автомате $\mathcal{D}$ существует переход из состояния d_i в состояние d_j;

в) переменные минимального символа $\{m_{i,l,j}\}_{1 \leqslant i < j \leqslant M; l \in \Sigma}$, которые истинны тогда и только тогда, когда в автомате $\mathcal{D}$ из состояния d_i в состояние d_j существует переход по символу l, но не существует переходов по меньшим (согласно выбранному порядку на символах) символам. Данные переменные используются только в случае небинарного алфавита.

Для задания DFS нумерации кодирование определения переменных наличия перехода $t_{i,j}$ не отличается от кодирования при задании BFS нумерации:

$$\bigwedge_{1 \leqslant i < j \leqslant M} \bigwedge_{l_k \in \Sigma} (t_{i,j} \leftrightarrow y_{i,l_1,j} \vee y_{i,l_2,j} \vee \ldots \vee y_{i,l_L,j}) \ —$$

в автомате $\mathcal{D}$ переход из состояния d_i в состояние d_j существует тогда и только тогда, когду существует переход из состояния d_i в состояние d_j хотя бы по одному из символов алфавита Σ.

Кодирование определения родительских переменных $p_{j,i}$, в свою очередь, отличается:

$$\bigwedge_{1 \leqslant i < j \leqslant M} (p_{j,i} \Leftrightarrow t_{i,j} \wedge \neg t_{i+1,j} \wedge \ldots \wedge \neg t_{j-1,j}) \ —$$

в виду жадности алгоритма DFS, состояние d_i автомата $\mathcal{D}$ является родителем состояния d_j в DFS-дереве, если оно имеет максимальный номер среди состояний, которые имеют переход в состояние d_j.

Как и ранее, необходимо задать ограничение на номер родителя для каждого состояния:

$$\bigwedge_{2 \leqslant j \leqslant M} (p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}) \text{ —}$$

у каждого состояния d_j автомата $\mathcal{D}$, кроме стартового, родитель в DFS-дереве должен иметь меньший номер.

Дизъюнкты, задающие порядок по уровню для детей различных родителей, а также порядок по глубине в разных поддеревьях, задаются следующим образом:

$$\bigwedge_{1 \leqslant i < k < j < q \leqslant M} (p_{j,i} \to \neg t_{k,q}) \text{ —}$$

если состояние d_i является родителем в дереве BFS состояния d_j, а состояние d_k имеет номер между номерами состояний d_i и d_j ($i < k < j$), то не может существовать перехода из состояния d_k в состояние d_q, где $q > j$. Действительно, так как $i < k < j$, то состояние d_k должно быть посещено алгоритмом DFS раньше состояния d_j. Тогда, если бы существовал переход из состояния d_k в состояние d_q, то тогда q должно было бы быть меньше, чем j.

Далее надо, для кодирования свойства DFS-пронумерованности автомата необходимо задать некоторый порядок на детях. Для простоты предлагается кодировать алфавитный порядок. Отдельно рассматривается два случая: алфавит Σ, состоящий из двух символов и состоящий из более чем двух символов. В случае бинарного алфавита $\Sigma = \{l_1, l_2\}$ требуются следующие дизъюнкты:

$$\bigwedge_{1 \leqslant i < j < k < M} (p_{j,i} \wedge t_{i,k} \to y_{i,l_1,j}) \text{ —}$$

если состояние d_i автомата $\mathcal{D}$ является родителем в DFS-дереве состояния d_j и есть переход из него в состояние d_k, где $j < k$, то переход из состояния d_i в состояние d_j должен быть помечен меньшим символом. Иначе, состояние d_k должно было бы иметь меньший номер, так как было бы обработано раньше.

В случае небинарного алфавита, необходимо использовать переменные минимального символа $m_{i,l,j}$. Определение переменных минимального символа для DFS-предикатов нарушения симметрии совпадает с определением для BFS-предикатов:

$$\bigwedge_{1 \leqslant i < j \leqslant M} \bigwedge_{1 \leqslant n \leqslant L} \left(m_{i,l_n,j} \leftrightarrow y_{i,l_n,j} \wedge \neg y_{i,l_{n-1},j} \wedge \neg y_{i,l_{n-2},j} \wedge \ldots \wedge \neg y_{i,l_1,j} \right) \text{ —}$$

в автомате $\mathcal{D}$ символ l_n является минимальным символом на переходах из состояния d_i в состояние d_j тогда и только тогда, когда в автомате существует переход из состояния d_i в состояние d_j по символу l_n, но не существует переходов по меньшим (относительно выбранного порядка) символам.

Дизъюнкты, кодирующие порядок детей в данном случае, похожи на дизъюнкты для бинарного алфавита:

$$\bigwedge_{1 \leqslant i < j < k \leqslant M} \bigwedge_{1 \leqslant m < n \leqslant L} \left(p_{j,i} \wedge t_{i,k} \wedge m_{i,l_n,j} \rightarrow \neg m_{i,l_m,k} \right) \text{ —}$$

если состояние d_i автомата $\mathcal{D}$ является родителем в DFS-дереве состояния d_j и есть переход из него в состояние d_k, где $j < k$, то переход из состояния d_i в состояние d_j должен быть помечен меньшим символом.

Таким образом, предлагается новое множество ограничений, допускающих в качестве решения только DFS-пронумерованные ДКА. Предикаты нарушения симметрии для более чем бинарного алфавита выражаются через $\mathcal{O}\left(M^4 + M^3 \times L^2\right)$ дизъюнктов, где M — размер искомого ДКА, а L — мощность алфавита. Множества дизъюнктов, кодирующих свойство DFS-пронумерованности, приведенные к конъюнктивно-нормальной форме, представлены в таблице 1 вместе с BFS-предикатами нарушения симметрии.

2.2 Модернизированное булево кодирование предикатов нарушения симметрии, использующих особенности алгоритма обхода в ширину

Кодирование свойства BFS-пронумерованности автомата, описанное в разделе 1.4.2, состоящее из $\mathcal{O}\left(M^3 + M^2 \times L^2\right)$ дизъюнктов, слабо применимо на

Таблица 1 – Дизъюнкты, кодирующие свойства DFS и BFS-пронумерованности ДКА

	Дизъюнкты	Домены переменных
Общие	$t_{i,j} \to (y_{i,l_1,j} \vee \ldots \vee y_{i,l_L,j})$	$1 \leqslant i < j \leqslant M$
	$y_{i,l,j} \to t_{i,j}$	$1 \leqslant i < j \leqslant M; l \in \Sigma$
	$p_{j,i} \to t_{i,j}$	$1 \leqslant i < j \leqslant M$
	$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$2 \leqslant j \leqslant M$
	$m_{i,l,j} \to y_{i,l,j}$	$1 \leqslant i < j \leqslant M; l \in \Sigma$
	$m_{i,l_n,j} \to \neg y_{i,l_k,j}$	$1 \leqslant i < j \leqslant M; 1 \leqslant k < n \leqslant L$
	$(y_{i,l_n,j} \wedge \neg y_{i,l_{n-1},j} \wedge \ldots \wedge \neg y_{i,l_1,j}) \to m_{i,l_n,j}$	$1 \leqslant i < j \leqslant M; 1 \leqslant n \leqslant L$
DFS	$p_{j,i} \to \neg t_{k,j}$	$1 \leqslant i < k < j \leqslant M$
	$(t_{i,j} \wedge \neg t_{i+1,j} \wedge \ldots \wedge \neg t_{j-1,j}) \to p_{j,i}$	$1 \leqslant i < j \leqslant M$
	$p_{j,i} \to \neg t_{k,q}$	$1 \leqslant i < k < j < q \leqslant M$
	$(p_{j,i} \wedge p_{k,i} \wedge m_{i,l_n,j}) \to \neg m_{i,l_m,k}$	$1 \leqslant i < j < k \leqslant M; 1 \leqslant m < n \leqslant L$
BFS	$p_{j,i} \to \neg t_{k,j}$	$1 \leqslant k < i < j \leqslant M$
	$(t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j}) \to p_{j,i}$	$1 \leqslant i < j \leqslant M$
	$p_{j,i} \to \neg p_{j+1,k}$	$1 \leqslant k < i < j < M$
	$(p_{j,i} \wedge p_{j+1,i} \wedge m_{i,l_n,j}) \to \neg m_{i,l_m,j+1}$	$1 \leqslant i < j < M; 1 \leqslant m < n \leqslant L$

практике для ДКА большого размера, то есть при большом M. В настоящем разделе описывается как модифицировать предикаты нарушения симметрии, рассмотренные ранее, таким образом, что для их булевого кодирования понадобится только $\mathcal{O}\left(M^2 \times L\right)$ дизъюнктов.

Далее повторно кратко приводятся наборы дизъюнктов, с помощью которых кодируются предикатов нарушения симметрии на основе алгоритма BFS, приведенные в разделе 1.4.2.

$$\bigwedge_{1 \leqslant i < j \leqslant M} (t_{i,j} \leftrightarrow y_{i,l_1,j} \vee y_{i,l_2,j} \vee \ldots \vee y_{i,l_L,j}) \tag{1}$$

$$\bigwedge_{1 \leqslant i < j \leqslant M} (p_{j,i} \leftrightarrow t_{i,j} \wedge \neg t_{i-1,j} \wedge \neg t_{i-2,j} \wedge \ldots \wedge \neg t_{1,j}) \tag{2}$$

$$\bigwedge_{2 \leqslant j \leqslant M} (p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}) \tag{3}$$

$$\bigwedge_{1 \leqslant k < i < j \leqslant M} (p_{j,i} \to \neg p_{j+1,k}) \tag{4}$$

$$\bigwedge_{1 \leqslant i < j \leqslant M} \bigwedge_{1 \leqslant n \leqslant L} \left(m_{i,l_n,j} \leftrightarrow y_{i,l_n,j} \wedge \neg y_{i,l_{n-1},j} \wedge \neg y_{i,l_{n-2},j} \wedge \ldots \wedge \neg y_{i,l_1,j}\right) \tag{5}$$

$$\bigwedge_{1 \leqslant i < j \leqslant M} \bigwedge_{1 \leqslant k < n \leqslant L} (p_{j,i} \wedge p_{j+1,i} \wedge m_{i,l_n,j} \rightarrow \neg m_{i,l_k,j+1}) \tag{6}$$

Изучив формулы, приведенные выше, можно заключить, что:

а) формула (1) содержит $\mathcal{O}\left(M^2 \times L\right)$ дизъюнктов;

б) формула (2) содержит $\mathcal{O}\left(M^3\right)$ дизъюнктов;

в) формула (3) содержит $\mathcal{O}\left(M\right)$ дизъюнктов;

г) формула (4) содержит $\mathcal{O}\left(M^3\right)$ дизъюнктов;

д) формула (5) содержит $\mathcal{O}\left(M^2 \times L^2\right)$ дизъюнктов;

е) формула (6) содержит $\mathcal{O}\left(M^2 \times L^2\right)$ дизъюнктов.

Таким образом, будет предложено новое кодирование для свойств, ранее выраженных формулами (2), (3), (5) и (6).

Прежде всего, можно заметить, что формула (3) задает свойство, что у каждого состояния d_j (кроме начального) автомата $\mathcal{D}$ существует как минимум один родитель в дереве BFS, при этом с меньшим номером. Однако, надо заметить, что по определению в любом дереве у любой вершины кроме корня существует ровно один родитель. Тогда, можно добавить ограничение, задающее свойство, что у каждого состояния d_j (кроме начального) автомата $\mathcal{D}$ существует не более одного родителя в дереве BFS, при этом с меньшим номером. В совокупности два данных ограничения зададут вышеупомянутое свойство об единственности родителя. Заметим, что ограничение, задающее свойство, что не более чем одна из M переменных истинна, может быть выражено через $\mathcal{O}\left(M^2\right)$ или $\mathcal{O}\left(M \times \log M\right)$ дизъюнктов, что укладывается в целевой размер формулы $\mathcal{O}\left(M^2 \times L\right)$ дизъюнктов.

Фактически, новое ограничение можно записать следующим образом.

$$\bigwedge_{1 < j \leqslant M} \sum_{i=1}^{j-1} p_{j,i} = 1 \tag{7}$$

2.2.1 Определение родительских переменных

Формула

$$\bigwedge_{1 \leqslant i < j \leqslant M} (p_{j,i} \leftrightarrow t_{i,j} \wedge \neg t_{i-1,j} \wedge \neg t_{i-2,j} \wedge \ldots \wedge \neg t_{1,j})$$

при преобразовании в КНФ выражается через $\mathcal{O}\left(M^3\right)$ дизъюнктов, так как обе переменные i и j имеют область допустимых значений размера M, а также правая часть формулы имеет длину $O\left(M\right)$. Так как переменные i и j независимы,то, чтобы сократить количество дизъюнктов, нужно сократить правую часть формулы. Для этого предлагается ввести новые булевы переменные $\{ft_{i,j}\}_{0 \leqslant i < j \leqslant M}$. Переменная $ft_{i,j}$ истинна тогда и только тогда, когда все переменные $t_{k,j}$, где $1 \leqslant k \leqslant i$, ложны (**f**alse ***t*** variables — ложные переменные t). Иными словами, $ft_{i,j} \leftrightarrow \neg t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \neg t_{1,j}$. Для $i = 0$ в явном виде определим, что $ft_{0,j} = 1$ для любого j. Стоит, однако, заметить, что определение переменных $ft_{i,j}$, приведенное выше, требует также $\mathcal{O}\left(M^3\right)$ дизъюнктов, что не решает изначальную проблему.

При этом, можно заметить, что значение переменной $ft_{i,j}$ зависит от значения всех переменных $ft_{k,j}$, где $k < i$. Тогда, можно определить переменные $ft_{i,j}$ рекурсивно:

$$ft_{i,j} \leftrightarrow \begin{cases} 1 & i = 0, 1 \leqslant j \leqslant M \\ ft_{i-1,j} \wedge \neg t_{i,j} & 1 \leqslant i < j \leqslant M \end{cases} \tag{8}$$

При преобразовании в КНФ формула (8) будет состоять из $\mathcal{O}\left(M^2\right)$ дизъюнктов.

Используя новые переменные $ft_{i,j}$ формулу (2) можно переписать следующим образом.

$$\bigwedge_{1 \leqslant i < j \leqslant M} \left(p_{j,i} \leftrightarrow t_{i,j} \wedge ft_{i-1,j}\right) \tag{9}$$

В данной формуле решена проблема длинной правой части и общее число дизъюнктов, требуемых для кодирования переменных $p_{j,i}$, также $\mathcal{O}\left(M^2\right)$.

2.2.2 Задание порядка детей с помощью родительских переменных

Формула

$$\bigwedge_{1 \leqslant k < i < j \leqslant M} (p_{j,i} \to \neg p_{j+1,k})$$

при преобразовании в КНФ выражается через $\mathcal{O}\left(M^3\right)$ дизъюнктов, так как все три переменные i, j и k имеют домен размера M. Переменные i и j являются независимыми, поэтому сократить размер всей формулы можно только избавившись от переменной k.

Фактически, данная формула задает ограничение, что родитель состояния d_{j+1} автомата $\mathcal{D}$ имеет номер не меньший, чем номер родителя состояния d_j. Также можно заметить, что учитывая ограничение, заданное формулой (7), двоичное число $\mathbf{p_j} = \overline{p_{j,1}p_{j,2}\ldots p_{j,j-1}}$ состоит из $j-2$ нулей и одной единицы. Тогда рассматриваемая формула говорит, что в числе $\mathbf{p_{j+1}}$ единственная единица стоит не левее чем в векторе $\mathbf{p_j}$ (и наоборот, что в числе $\mathbf{p_j}$ единственная единица стоит не правее чем в векторе $\mathbf{p_{j+1}}$). Для удобства сравнения данных чисел, рассматривается расширенное число $\tilde{\mathbf{p}}_\mathbf{j} = \overline{p_{j,1}p_{j,2}\ldots p_{j,j-1}0}$. В контексте имеющейся формулы расширенное число не отличается от обычного, так как в нем все еще содержится ровно одна единица на том же месте, что и раньше (считая, слева), но теперь двоичные числа $\tilde{\mathbf{p}}_\mathbf{j}$ и $\mathbf{p_{j+1}}$ имеют одинаковое число цифр. Тогда исходная формула фактически задает ограничение $\tilde{\mathbf{p}}_\mathbf{j} \geqslant \mathbf{p_{j+1}}$. С помощью $\mathbf{p_j}^\mathbf{i}$ далее в данном разделе будет обозначаться двоичное число, являющееся суффиксом числа $\mathbf{p_j}$, начинающимся с i-ой цифры и заканчивая последней: $\mathbf{p_j}^\mathbf{i} = \overline{p_{j,i}p_{j,i+1}\ldots p_{j,j-1}}$. Аналогично, $\tilde{\mathbf{p}}_\mathbf{j}^\mathbf{i} = \overline{p_{j,i}p_{j,i+1}\cdots p_{j,j}}$.

Для сравнения необходимо ввести новое множество булевых переменных $\{geq_{j,i}\}_{1<j\leqslant M, 1\leqslant i\leqslant j+1}$. Переменная $\{geq_{j,i}\}$ истинна тогда и только тогда, когда число $\tilde{\mathbf{p}}_\mathbf{j}^\mathbf{i}$ больше или равно (**g**reater or **eq**ual) чем число $\mathbf{p_{j+1}}^\mathbf{i}$, а значит и единица во втором числе находится не левее чем в первом. Определить переменные $geq_{j,i}$

можно рекурсивно следующим образом:

$$geq_{j,i} \leftrightarrow \begin{cases} 1 & i = j+1, 1 \leqslant j \leqslant M \\ geq_{j,i+1} \wedge (p_{j,i} \leftrightarrow p_{j+1,i}) \vee p_{j,i} \wedge \neg p_{j+1,i} & 1 \leqslant i \leqslant j \leqslant M \end{cases} \tag{10}$$

Ограничение $geq_{j,j+1} = 1$ задает исходное значение для рекурсивного определения переменных $geq_{j,i}$. Далее, число $\tilde{\mathbf{p}}_\mathbf{j}{}^\mathbf{i}$ больше или равно числа $\mathbf{p_{j+1}}{}^\mathbf{i}$, то есть $geq_{j,i} = 1$, если i-ый бит чисел $\tilde{\mathbf{p}}_\mathbf{j}$ и $\mathbf{p_{j+1}}$ совпадает, а для суффиксов $\tilde{\mathbf{p}}_\mathbf{j}{}^\mathbf{i+1}$ и $\mathbf{p_{j+1}}{}^\mathbf{i+1}$ верно, что первый больше либо равен второго, или если i-ый бит числа $\tilde{\mathbf{p}}_\mathbf{j}$ равен единице, а числа $\mathbf{p_{j+1}}$ — нулю. Последнее верно, так как оба числа содержат по одной единице и вне зависимости от того, где находится единица во втором числе, правее или левее, число $\tilde{\mathbf{p}}_\mathbf{j}{}^\mathbf{i}$ строго больше числа $\mathbf{p_{j+1}}{}^\mathbf{i}$.

Для упрощения записи и уменьшения размера дизъюнктов можно ввести еще одни вспомогательные переменные $\{peq_{j,i}\}_{1 \leqslant i \leqslant j \leqslant M}$. Переменная $peq_{j,i}$ истинна тогда и только тогда, когда $p_{j,i} = p_{j+1,i}$.

$$peq_{j,i} \leftrightarrow (p_{j,i} \leftrightarrow p_{j+1,i}) \tag{11}$$

Тогда формула (10) примет окончательный вид:

$$geq_{j,i} \leftrightarrow \begin{cases} 1 & i = j+1, 1 \leqslant j \leqslant M \\ geq_{j,i+1} \wedge peq_{j,i} \vee p_{j,i} \wedge \neg p_{j+1,i} & 1 \leqslant i \leqslant j \leqslant M \end{cases} \tag{12}$$

При преобразовании в КНФ формула (12) будет состоять из $\mathcal{O}\left(M^2\right)$ дизъюнктов.

Используя новые переменные $geq_{i,j}$ формулу (4) можно переписать следующим образом.

$$\bigwedge_{1<j\leqslant M} geq_{j,1} \tag{13}$$

Действительно, $geq_{j,1} = 1 \Leftrightarrow \tilde{\mathbf{p}}_\mathbf{j} = \tilde{\mathbf{p}}_\mathbf{j}{}^\mathbf{1} \geqslant \mathbf{p_{j+1}}{}^\mathbf{1} = \mathbf{p_{j+1}}$.

Таким образом, формула (9) выражается через $\mathcal{O}\left(M\right)$ дизъюнктов, а формулы (11) и (12) — через $\mathcal{O}\left(M^2\right)$ дизъюнктов.

2.2.3 Определение переменных минимального символа

Формула

$$\bigwedge_{1\leqslant i<j\leqslant M}\bigwedge_{1\leqslant n\leqslant L}\left(m_{i,l_n,j}\leftrightarrow y_{i,l_n,j}\wedge\neg y_{i,l_{n-1},j}\wedge\neg y_{i,l_{n-2},j}\wedge\ldots\wedge\neg y_{i,l_1,j}\right)$$

при преобразовании в КНФ выражается через $\mathcal{O}\left(M^2\times L^2\right)$ дизъюнктов, так как обе переменные i и j имеют область допустимых значений размера M, переменная n — размера L, а также правая часть формулы имеет длину $O\left(L\right)$. Так как переменные i, j и n независимы,то, чтобы сократить количество дизъюнктов, нужно сократить правую часть формулы. Можно заметить, что данная формула по своей структуре аналогична той, что рассматривалась в разделе 2.2.1. Тогда аналогично можно ввести новые булевы переменные $\{fy_{i,l_n,j}\}_{0\leqslant i<j\leqslant M,0\leqslant n\leqslant M}$. Переменная $fy_{i,l_n,j}$ истинна тогда и только тогда, когда все переменные $y_{i,l_k,j}$, где $1\leqslant k\leqslant n$, ложны (**false** $\boldsymbol{y}$ variables — ложные переменные y). Иными словами, $fy_{i,l_n,j}\leftrightarrow\neg y_{i,l_n,j}\wedge\neg y_{i,l_{n-1},j}\wedge\ldots\neg y_{i,l_1,j}$. Для $n=0$ в явном виде определим, что $fy_{i,l_0,j}=1$ для любых i,j. Далее, аналогично тому, как это было сделано в разделе 2.2.1, определим переменные $fy_{i,l_n,j}$ рекурсивно.

$$fy_{i,l_n,j}\leftrightarrow\begin{cases}1 & 1\leqslant i<j\leqslant M, n=0\\ fy_{i,l_{n-1},j}\wedge\neg y_{i,l_n,j} & 1\leqslant i<j\leqslant M, 1\leqslant n\leqslant L\end{cases}\tag{14}$$

При преобразовании в КНФ формула (14) будет состоять из $\mathcal{O}\left(M^2\times L\right)$ дизъюнктов.

Используя новые переменные $fy_{i,l_n,j}$ формулу (5) можно переписать следующим образом.

$$\bigwedge_{1\leqslant i<j\leqslant M}\bigwedge_{1\leqslant n\leqslant L}\left(m_{i,l_n,j}\leftrightarrow y_{i,l_n,j}\wedge fy_{i,l_{n-1},j}\right)\tag{15}$$

В данной формуле решена проблема длинной правой части и общее число дизъюнктов, требуемых для кодирования переменных $m_{i,l_n,j}$, также $\mathcal{O}\left(M^2\times L\right)$.

2.2.4 Задание порядка детей одного родителя

Формула

$$\bigwedge_{1\leqslant i<j\leqslant M}\bigwedge_{1\leqslant k<n\leqslant L}(p_{j,i}\wedge p_{j+1,i}\wedge m_{i,l_n,j}\rightarrow\neg m_{i,l_k,j+1})$$

при преобразовании в КНФ выражается также через $\mathcal{O}\left(M^2\times L^2\right)$ дизъюнктов, так как обе переменные i и j имеют область допустимых значений размера M, а обе переменные n и k — размера L. Переменные i,j и n являются независимыми, поэтому сократить размер всей формулы можно только избавившись от переменной k. Можно заметить, что данная формула по своей структуре аналогична той, что рассматривалась в разделе 2.2.2. Тогда аналогично можно ввести новые булевы переменные $\{fm_{i,l_n,j}\}_{0\leqslant i<j\leqslant M,0\leqslant n\leqslant M}$. Переменная $fm_{i,l_n,j}$ истинна тогда и только тогда, когда все переменные $m_{i,l_k,j}$, где $1\leqslant k\leqslant n$, ложны (**f**alse $\boldsymbol{m}$ variables — ложные переменные m). Иными словами, $fm_{i,l_n,j}\leftrightarrow\neg m_{i,l_n,j}\wedge\neg m_{i,l_{n-1},j}\wedge\ldots\neg m_{i,l_1,j}$. Для $n=0$ в явном виде определим, что $fm_{i,l_0,j}=1$ для любых i,j. Далее, аналогично тому, как это было сделано в разделе 2.2.2, определим переменные $fm_{i,l_n,j}$ рекурсивно.

$$fm_{i,l_n,j}\leftrightarrow\begin{cases}1 & 1\leqslant i<j\leqslant M, n=0\\ fm_{i,l_{n-1},j}\wedge\neg m_{i,l_n,j} & 1\leqslant i<j\leqslant M, 1\leqslant n\leqslant L\end{cases}\tag{16}$$

При преобразовании в КНФ формула (16) будет состоять из $\mathcal{O}\left(M^2\times L\right)$ дизъюнктов.

Используя новые переменные $fm_{i,l_n,j}$ формулу (6) можно переписать следующим образом.

$$\bigwedge_{1\leqslant i<j<M}\bigwedge_{1\leqslant n\leqslant L}\left(p_{j,i}\wedge p_{j+1,i}\wedge m_{i,l_n,j}\rightarrow\neg fm_{i,l_{n-1},j+1}\right)\tag{17}$$

Таким образом общее число дизъюнктов, требуемых для ограничения порядка детей одного состояния, равняется $\mathcal{O}\left(M^2\times L\right)$.

2.3 Подходы к сокращению пространства поиска, основанные на особенностях автомата дерева обхода в ширину

В данной главе предлагаются новые методы по сокращению пространства поиска в задаче генерации ДКА минимального размера по заданным словарям. Данные методы не являются необходимыми для нахождения соответствующего автомата, но помогают сделать это быстрее. В основе предлагаемых методов лежит использование структурных особенностей BFS-пронумерованного ДКА, а также связь между расширенным префиксным деревом и ДКА.

2.3.1 Полное дерево обхода в ширину

На рисунке 11 показано полное BFS-дерево, построенное по некоторому автомату. Данное дерево является полным, так как у каждой его внутренней вершины имеется по L детей. Тогда данное дерево показывает максимально возможные номера, которые могут быть у детей некоторого состояния d_i. Действительно, нельзя добавить в данное дерево новые вершины, которые будут иметь номер между i и $i \cdot L + 1$, так как все возможные позиции заняты. В то же время, если удалить какие-то из вершин правее или ниже вершины d_i, то номера детей могут только уменьшиться. Далее будут представлены дополнительные ограничения, которые следуют из рисунка 11.

Сокращение области определения родительских переменных. У некоторого состояния d_i, где $1 \leqslant i < M$, детьми в BFS-дереве могут быть только состояния с номерами от $i+1$ до $\min(i \cdot L + 1, M)$. Так как в BFS-дереве номер ребенка всегда больше номера родителя, то нижняя граница тривиальна. Рисунок 11 иллюстрирует обоснование верхней границы. Действительно, можно доказать по индукции, что состояния на k-ом уровне имеют номера от $\sum_{r=0}^{k-1} L^r + 1$ до $\sum_{r=0}^{k} L^r$. База индукции при $k = 0$, очевидно, верна. Если для некоторого слоя k утверждение

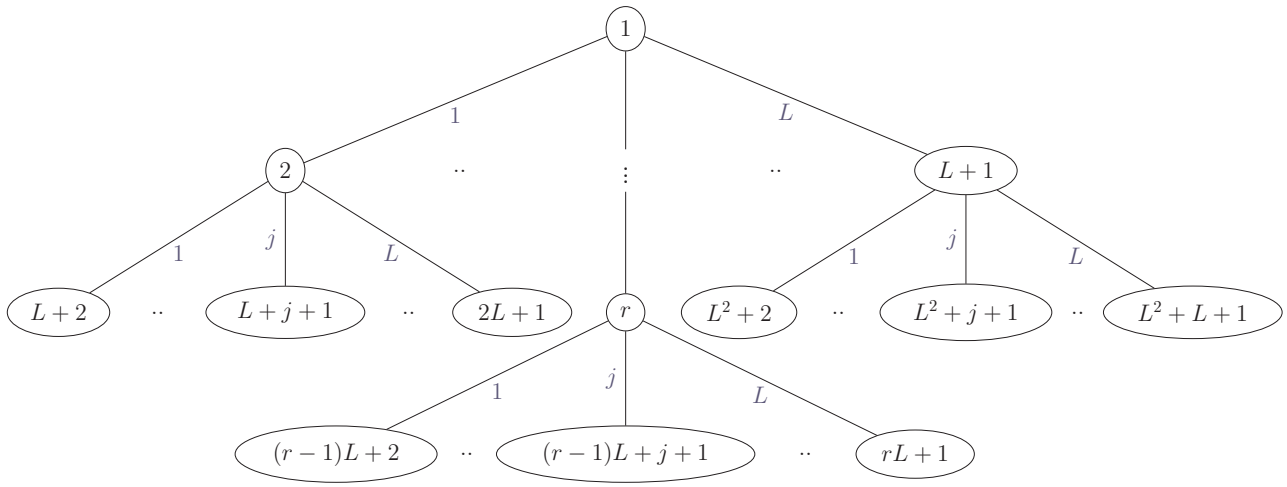


Рисунок 11 – Полное BFS-дерево, где $|\Sigma| = L$

выше верно, то для слоя $k+1$ верно, что нумерация состояний на нем начинается с $\sum_{r=0}^{k-1} L^r + 1$, а всего вершин $\left(\sum_{r=0}^{k} L^r - \left(\sum_{r=0}^{k-1} L^r + 1\right) + 1\right) \cdot L = L^k \times L = L^{k+1}$, из чего следует, что последнее состояние имеет номер $\sum_{r=0}^{k} L^r + L^{k+1} = \sum_{r=0}^{k+1} L^r$. Выразить данное свойство можно, либо определив переменные для соответствующей области определения — $\{p_{j,i}\}_{1 \leqslant i < j \leqslant min(i \cdot L+1, M)}$, либо в явном виде указав, что $p_{j,i} = 0$ при $j > i \cdot L + 1$.

Сокращение области определения переменных перехода и переменных наличия переходов. Помимо закономерностей между номерами родителей и детей в BFS-пронумерованном автомате, можно заметить более общую закономерность относительно переходов. Из состояния d_i в BFS-пронумерованном автомате не может в принципе существовать перехода в состояние d_j если $j > i \cdot L + 1$. Действительно, из доказанного в предыдущей секции следует, что у состояния d_j родителем должно быть состояние d_k, где $k > i$. Но, если существует из состояния d_i существует переход в состояние d_j, то по принципу BFS обхода родителем состояния d_j должно быть состояние d_k, где $k \leqslant i$. Получившееся противоречие доказывает исходное утверждение. Таким образом, можно сделать заключение, что $y_{i,l,j} = 0$ при $j > i \cdot L + 1; l \in \Sigma$.

Как следствие, по определению переменных наличия переходов верно, что $t_{i,j} = 0$ при $j > i \cdot L + 1$.

2.3.2 Зависимости между номерами родительских вершин и детей

Помимо того, что у каждого состояния d_i автомата $\mathcal{D}$ детьми могут быть состояния с номерами от $i+1$ до $i \cdot L + 1$, можно утверждать, что состояние d_i может быть родителем не более чем L состояний, которые при этом пронумерованны последовательно. Количество детей ограничено размером алфавита, так как рассматриваемый автомат является детерминированным. Последовательная нумерация следует из структуры алгоритма BFS — дети некоторого состояния поочередно добавляются в очередь и им присваиваются последовательные номера. Данное свойство можно назвать *свойством непрерывности*. Для булевого кодирования предикатов нарушения симметрии данное свойство означает, что для фиксированного i переменные $p_{j,i}$ ложны для всех j, кроме некоторого отрезка $[j_0, \ldots, j_s]$, где $1 \leqslant j_0 \leqslant j_s \leqslant M, s \leqslant L$.

Можно добавить дополнительные ограничения, задающие данное свойство, которые дополнительно ограничат пространство поиска. Для этого необходимо ввести два дополнительных множества булевых переменных — $\{lnp_{j,i}\}_{1 \leqslant i < j \leqslant M}$ и $\{rnp_{j,i}\}_{1 \leqslant i < j \leqslant M}$.

Переменная $lnp_{j,i}$ истинна тогда, когда переменная $p_{j,i} = 0$ и $j < j_0$. Иными словами, данная переменная истинна в случае, когда j находится левее отрезка истинных родительских переменных (**l**eft **n**o **p**arent). Определить на языке выполнимости булевых формул данные переменные можно следующим образом. Формула

$$\bigwedge_{1 \leqslant i < j \leqslant M} \neg p_{j,i} \wedge p_{j+1,i} \rightarrow lnp_{j,i}$$

задает пограничное истинное значение переменных $lnp_{j,i}$. Далее, необходимо добавить формулу

$$\bigwedge_{1 \leqslant i < M, i+1 < j \leqslant M} lnp_{j,i} \rightarrow lnp_{j-1,i},$$

которая задает значения переменных $lnp_{j,i}$ левее пограничного. Как следствие из определения переменных $lnp_{j,i}$, можно добавить следующую формулу:

$$\bigwedge_{1 \leqslant i < j \leqslant M} lnp_{j,i} \rightarrow \neg p_{j,i}.$$

Таким образом, переменные $lnp_{j,i}$ для каждого i истинны начиная с $j = 1$ и до тех пор, пока $p_{j+1,i}$ не будет истинно. Можно заметить, что начиная с момента, когда $p_{j,i}$ истинно, значение переменных $lnp_{j,i}$ не определено, что, как будет показано далее, не играет никакой роли.

Аналогичным образом определяются переменные $rnp_{j,i}$. Переменная $rnp_{j,i}$ истина тогда, когда переменная $p_{j,i} = 0$ и $j > j_s$, то есть когда j находится правее отрезка истинных родительских переменных (**r**ight **n**o **p**arent). Пограничное истинное значение переменных $rnp_{j,i}$ задается с помощью формулы

$$\bigwedge_{1 \leqslant i < j \leqslant M} p_{j-1,i} \wedge \neg p_{j,i} \rightarrow rnp_{j,i}.$$

Значение переменных правее пограничного задаются аналогично предыдущему случаю:

$$\bigwedge_{1 \leqslant i < j < M} rnp_{j,i} \rightarrow rnp_{j+1,i}.$$

Как и в случае с переменными $lnp_{j,i}$, можно добавить формулу

$$\bigwedge_{1 \leqslant i < j \leqslant M} rnp_{j,i} \rightarrow \neg p_{j,i}.$$

Переменные $rnp_{j,i}$ для каждого i истинны начиная с $j = M$ и в порядке убывания истинны до тех пор, пока $p_{j-1,i}$ не будет истинно. Можно заметить, что, аналогично, начиная с $j = 1$, и до тех пор, пока $p_{j,i}$ не станет ложной после серии истинных значений, значение переменных $rnp_{j,i}$ не определено. Помимо этого, можно добавить следующую формулу:

$$\bigwedge_{1 \leqslant i < j \leqslant M, l \in \Sigma} rnp_{j,i} \rightarrow \neg y_{i,l,j}.$$

Действительно, если состояние d_i имеет детей с номерами $j_0, \ldots, j_s$, то из состояния d_i не может быть переходов состояния с номерами б*о*льшими чем j_s, иначе данные состояния были бы также детьми состояния d_i.

Дополнительно, из того, что d_i может иметь не более чем L детей, следует, что

$$\bigwedge_{1 \leqslant i < M; i+L < j \leqslant M-L} p_{j,i} \to lnp_{j-L,i}$$

и что

$$\bigwedge_{1 \leqslant i < j \leqslant M-L} p_{j,i} \to rnp_{j+L,i}.$$

Переменные $lnp_{j,i}$ и $rnp_{j,i}$ помогают задать некоторым переменным $p_{j,i}$ ложное значение. Однако, исходя из их значения, можно некоторым переменным $p_{j,i}$ задать истинное значение. Так, если для некоторых $j_1 < j_2$ верно, что $lnp_{j_1,i}$ и $rnp_{j_2,i}$ ложны, то для всех j' таких, что $j_1 \leqslant j' \leqslant j_2$ верно, что $p_{j',i}$ истинна. Формально,

$$\bigwedge_{1 \leqslant i < M; i < j_1 \leqslant j' \leqslant j_2 \leqslant \min(j_1+L-1, M)} \neg lnp_{j_1,i} \wedge \neg rnp_{j_2,i} \to p_{j',i}.$$

Также, учитывая, что дети некоторого состояния d_i пронумерованны последовательно, можно добавить следующее ограничение:

$$\bigwedge_{1 \leqslant i < j < k \leqslant \min(j+L-1, M)} p_{j,i} \wedge p_{k,i} \to p_{k-1,i}.$$

2.3.3 Минимальное расстояние в дереве обхода автомата в ширину

Еще одним следствием анализа полного BFS-дерева (см. рисунок 11), является ограничение минимального расстояния от стартового состояния автомата $\mathcal{D}$ до всех других. Не сложно заметить, что в полном BFS-дереве, представленном на рисунку 11, глубина некоторого состояния d_j минимальна. Действительно, в неполном BFS-дереве на каждой глубине состояний не больше чем в полном дереве, а значит состояние может находится или на том же уровне, или глубже. Тогда глубина состояния в полном BFS-дереве будет являться оценкой снизу для глубины состояния в случайном дереве.

Для доказательства можно воспользоваться ранее доказанным фактом, что на уровне k в полном BFS-дереве находятся состояния с номерами от $\left(\sum_{i=0}^{k-1} + 1\right)$

до $\left(\sum_{i=0}^{k}\right)$. Иными словами, номера состояний на уровне k находятся в полуоткрытом интервале $\left(\sum_{i=0}^{k-1} L^i; \sum_{i=0}^{k} L^i\right]$. Если домножить левую и правую границы интервала на $(L-1)$ и прибавить единицу, то получится интервал $\left(L^k; L^{k+1}\right]$. Теперь, если взять логарифм по основанию L от обеих границ и вычесть единицу, то получится, интервал $(k-1; k]$. Из этого можно заключить, что минимальная глубина состояния с номером j, а значит и минимальное расстояние от стартового состояния до него, равняется $D_{\min}(j) = \lceil \log_L (j \cdot (L-1) + 1) - 1 \rceil$.

Таким образом, для любого состояния d_j автомата $\mathcal{D}$ минимальное расстояние от стартового состояния d_1 до d_j не меньше, чем $D_{\min}(j)$. Тогда, если расстояние от корня t_1 префиксного дерева $\mathcal{T}$ до некоторой вершины t_v меньше, чем минимально возможное расстояние до состояния d_j автомата $\mathcal{D}$: $\Delta(v) < D_{\min}(j)$, то можно утверждать, что вершина t_v не может соответствовать состоянию d_j, то есть $x_{v,j} = 0$.

2.4 Реализация и экспериментальные исследования методов, использующих разработанные подходы к сокращению пространства поиска

В настоящем разделе приводятся описание разработанного программного средства для генерации ДКА по примерам поведения и реализации разработанных методов генерации ДКА в рамках данного программного комплекса, а также результаты экспериментальных исследований разработанных методов.

Программное средство для генерации детерминированных конечных автоматов по примерам поведения. Для решения различных задач, связанных с генерацией ДКА по заданным примерам поведения, на языке *python* было разработано программное средство с открытым исходным кодом `DFA-Inductor-py` [76]. На рисунке 12 представлена структура программного средства. С помощью средства `DFA-Inductor-py` можно генерировать ДКА с минимальным числом состояний по заданным примерам поведения (`Min-DFA`),

генерировать ДКА по избыточному множеству примеров поведения (CEGAR, подробнее в главе 3), генерировать все неизоморфные ДКА, соответствующие примерам поведения (Find-ALL, подробнее в главе 4). В программном средстве реализованы методы нарушения симметрии как предложенные ранее, так и разработанные в данной диссертации. Для взаимодействия с программными средствами для решения SAT используется библиотека PySAT [47], которая содержит в себе обертки на языке python над популярными программными средствами.

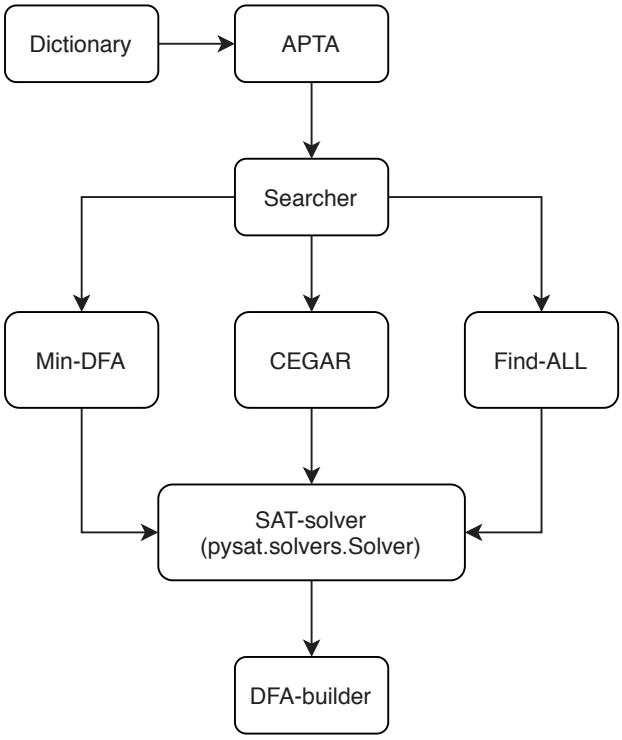


Рисунок 12 – Структура программного средства DFA-Inductor-py

На вход средству подаются примеры поведения в следующем формате. В первой строке файла содержатся два целых числа, разделенных пробелами: число примеров поведения S и размера алфавита L. Далее в следующих S строках описываются сами примеры поведения. Первое число в строке задает к какому типу относится данный пример поведения — если он должен приниматься автоматом, то число равняется 1, если не должен принимать, то 0. Второе число в строке задает длину n данного примера. Следующие n чисел принимают значения от 0 до $L-1$ и являются символами переходов данного примера поведения. Например, множества примеров поведения $S_+ = \{aa, bbb\}$ и $S_- = \{b, abab\}$ представляются

в описанном формате следующим образом:

```
4 2
1 2 1 1
0 1 0
0 4 1 0 1 0
1 3 0 0 0
```

На выход `DFA-Inductor-py` возвращает автомат, соответствующий заданным примерам поведения, в формате `GraphViz` [77].

На рисунке 13 представлена диаграмма классов программного средства `DFA-Inductor-py`. Подробно ознакомиться с кодом программного средства, а также с инструкцией по установке можно по ссылке [76]. Для получения справки по различным параметрам запуска после установки программного средства необходимо в командной строке выполнить команду “`dfainductor --help`”.

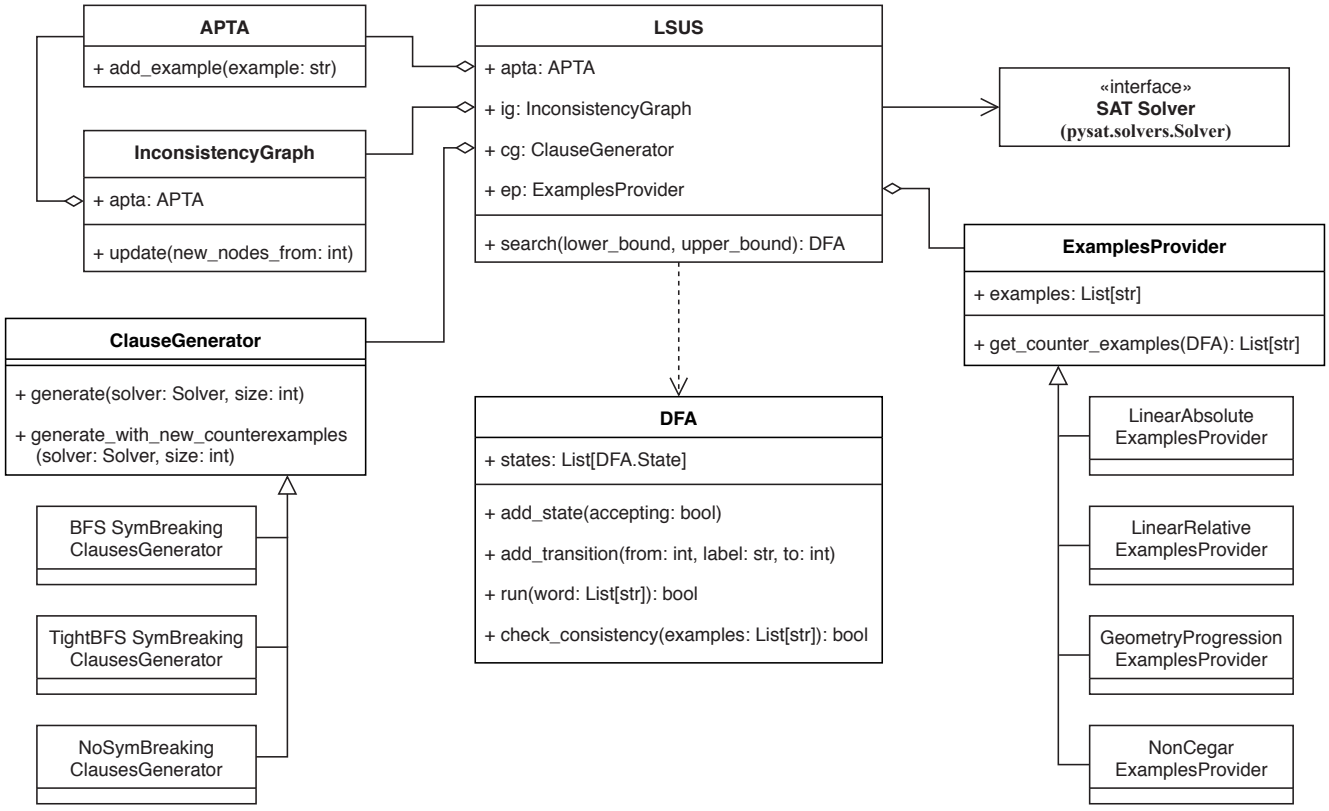


Рисунок 13 – Диаграмма классов программного средства `DFA-Inductor-py`

Реализация разработанных методов генерации детерминированных конечных автоматов. Все разработанные в настоящей главе методы были реализованы в рамках программного средства `DFA-Inductor-py`. Для выбора используемых предикатов нарушения симметрии необходимо при запуске программного средства указать параметр "`--sym-breaking/-b`" и выбрать одно из следующих значений:

- `NOSB` — не использовать предикаты нарушения симметрии;
- `CLIQUE` — использовать предикаты на основе поиска клики в графе несовместимости, предложенные в средстве `DFASAT`;
- `BFS` — использовать BFS-предикаты, предложенные в средстве `DFA-Inductor`;
- `DFS` — использовать DFS-предикаты, предложенные в разделе 2.1 настоящей диссертации;
- `TIGHTBFS` — использовать BFS-предикаты, предложенные в разделах 2.2 и 2.3 настоящей диссертации.

Алгоритм генерации случайных данных для проведения экспериментальных исследований Так как все известные тестовые данные все еще слишком сложны для решения с помощью точных методов генерации ДКА без проведения предварительных эвристических слияний состояний, был разработан алгоритм генерации случайных примеров поведения по случайному ДКА. Данный алгоритм строит множество примеров поведения по следующему набору параметров:

а) размер M детерминированного конечного автомата, по которому будут генерироваться примеры поведения;

б) размер алфавита $A = |\Sigma|$;

в) число примеров поведения S, которые нужно сгенерировать.

Разработанный алгоритм устроен следующим образом. Алфавит заполняется числами от 1 до A: $\Sigma = \{1, \ldots, A\}$. Первым шагом создаются M состояний, пронумерованных уникальными числами от 1 до M — $[d_1, d_2, \ldots, d_M]$. Каждое состояние равновероятно помечается как либо принимающее, либо отвергающее. Затем на

каждом i-ом шаге (начиная с $i = 2$) выбираются случайным образом (равновероятно) одно из состояний $[d_1, \ldots, d_{i-1}]$, состояние d_i, и из первого состояния во второе добавляется переход по случайному символу (также, равновероятно).

Утверждение 1. После M-ого шага получается частично построенный автомат, где все состояния достижимы из стартового состояния d_1.

Доказательство. Докажем по индукции утверждение, что после k-ого шага все состояния $[d_1, \ldots, d_k]$ достижимы из стартового состояния d_1.

База индукции при $k = 2$ верна, так как на втором шаге переходом соединяются одно из состояний $[d_1, \ldots, d_{2-1}] = [d_1] = d_1$ и состояние d_2. Тогда действительно все состояния $[d_1, d_2]$ достижимы из стартового.

Докажем переход индукции. Пусть для некоторого k верно, что после k-ого шага все состояния $[d_1, \ldots, d_k]$ достижимы из стартового состояния d_1. Тогда на $(k+1)$-ом шаге случайно выбирается одно из состояний $[d_1, \ldots, d_k]$ — не уменьшая общности, пусть выбирается состояние d_t — и добавляется переход из него в состояние d_{k+1}. По предположению индукции все состояния $[d_1, \ldots, d_k]$ уже достижимы из стартового, а в состояние d_{k+1} из состояния d_1 можно попасть через состояние d_t по новому добавленному переходу. Переход доказан.

Тогда после M-ого шага все состояния $[d_1, \ldots, d_M]$ достижимы из стартового состояния d_1. □

По доказанному выше утверждению гарантируется, что все состояния автомата достижимы. Далее для каждого состояния добавляются переходы по всем символам, по которым еще нет исходящих переходов, в случайные состояния автомата. Таким образом генерируется случайный полный детерминированный конечный автомат $\mathcal{D}$ с M состояниями над алфавитом Σ.

Затем генерируется S строк над алфавитом Σ, которые с помощью построенного автомата помечаются допуском или недопуском. Распределение длин генерируемых строк смещено в строну более длинных. Такие строки с пометками допуска/недопуска и формируют экземпляр задачи генерации ДКА по заданным примерам поведения.

Экспериментальные исследования разработанных методов Эксперименты проводились на персональном компьютере с процессором *AMD Opteron 6378 @* 2,4 ГГц, 496 ГБ оперативной памяти и операционной системой *Ubuntu*. Для каждого отдельного процесса объем используемой памяти ограничивался 1 Гб.

Первая серия экспериментов посвящена оценке эффективности предикатов нарушения симметрии на основе алгоритма обхода графа в глубину. Для сравнения методов генерации ДКА минимального размера по заданным примерам поведения, основанных на использовании BFS и DFS-предикатов нарушения симметрии, использовались случайные данные, полученные с помощью алгоритма, описанного в разделе 2.4, со следующими параметрами:

а) $M \in [10; 30]$;

б) $A = 2$;

в) $S = 50 \times M$.

Дополнительно в сравнение был включен метод `DFASAT`, использующий в качестве предикатов нарушения симметрии фиксирование нумерации некоторой большой клики графа несовместимости (см. раздел 1.4.2), так как он лежит в основе двух других методов. Для каждой комбинации параметров было сгенерировано по 100 экземпляров задачи. Время работы методов было ограничено одним часом (3600 секунд). Результаты экспериментов представлены в таблице 2 и позволяют сделать вывод, что использование DFS-предикатов нарушения симметрии нецелесообразно, так как метод, их использующий, значительно проигрывает методу, использующему BFS-предикаты. Однако можно заметить, что метод, использующий DFS-предикаты значительно выигрывает в производительности относительно метода `DFASAT`. Тем не менее, исходя из результатов, было решено не продолжать развитие DFS-предикатов и сосредоточиться на улучшении предикатов, использующих алгоритм обхода графа в ширину.

Во втором экспериментальном исследовании сравнивались методы `DFA-Inductor`, использующий оригинальные BFS-предикаты, и `DFA-Inductor-py`, использующий BFS-предикаты, разработанные в настоящей диссертации. Как и ранее дополнительно в сравнение был включен метод

Таблица 2 – Медианное время работы методов генерации ДКА по заданным примерам поведения с использованием BFS-предикатов нарушения симметрии, DFS-предикатов нарушения симметрии и метода DFASAT в секундах.

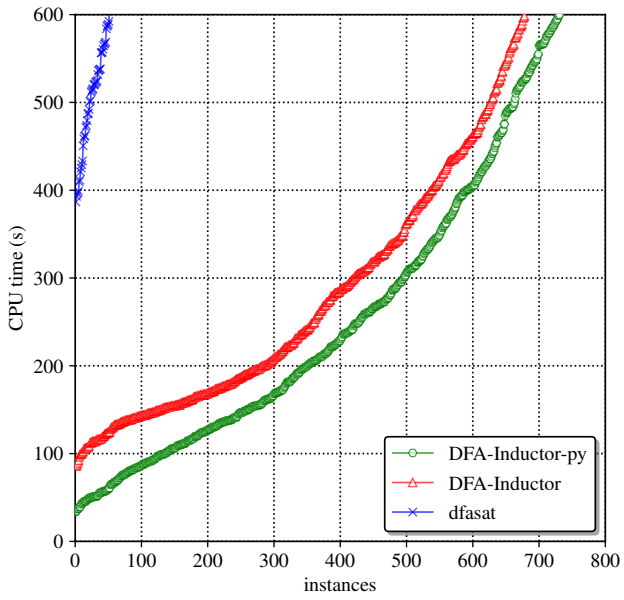
M	DFS	BFS	DFASAT
10	20,9	20,5	23,3
12	40,4	37,6	240,3
14	82,2	62,4	—
16	205,1	114,1	—
18	601,7	181,9	—
20	2501,6	293,7	—
22	—	453,3	—
24	—	625,1	—
26	—	925,8	—
28	—	1314,4	—
30	—	1635,5	—

DFASAT. Результаты сравнения всех трех методов представлены на рисунке 14a и показывают, что метод DFA-Inductor-py способен за одно и то же время решить большее число экземпляров задачи генерации ДКА по заданным примерам поведения, чем метод DFA-Inductor. Так, за одно и тоже время методом DFA-Inductor-py был сгенерирован 731 автомат против 678 сгенерированных автоматов методом DFA-Inductor. Также можно видеть, что метод DFASAT не способен составить конкуренции двум другим подходам.

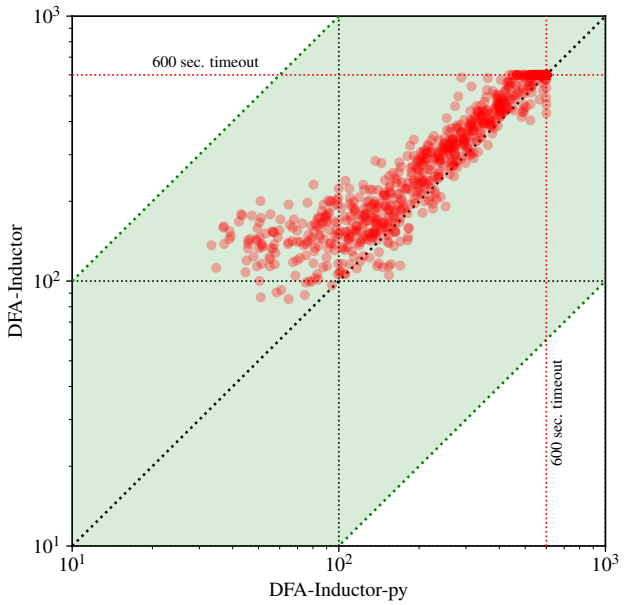
На рисунке 14b представлено детальное сравнение методов DFA-Inductor и DFA-Inductor-py, где видно, что при решении подавляющего числа экземпляров задачи генерации ДКА, второй метод показывает лучшие результаты.

Выводы по главе 2

Во второй главе были предложены новые предикаты нарушения симметрии, основанные на алгоритмах обхода графа в глубину и в ширину и методы, использующие их. Разработанные методы были реализованы в рамках программного средства DFA-Inductor-py.



(a) Сравнение методов `DFA-Inductor`, `DFA-Inductor-py` и `DFASAT`, показывающее, число различных экземпляров задачи, решенное за определенное время



(b) Детальное сравнение методов `DFA-Inductor` и `DFA-Inductor-py`, сравнивающее их производительность в решении каждого экземпляра задачи

Рисунок 14 – Результаты сравнения метода, использующего оригинальные BFS-предикаты (`DFA-Inductor`), метода, использующего новые BFS-предикаты (`DFA-Inductor-py`), и `DFASAT`

Впервые для задачи генерации ДКА по заданным примерам поведения были предложены предикаты нарушения симметрии, основанные на кодировании алгоритма DFS. DFS-предикаты на языке SAT выражаются через $\mathcal{O}\left(M^4 + M^3 \times L^2\right)$ дизъюнктов, где M — размер искомого ДКА, а L — мощность алфавита. Экспериментальные исследования, однако, показали, что метод генерации ДКА, использующий данные предикаты проигрывает аналогичному методу, использующему BFS-предикаты. Таким образом, предикаты нарушения симметрии на основе кодирования алгоритма DFS имеют скорее теоретическую значимость. Данный результат был опубликован в статье [78].

Было предложенно новое кодирование BFS-предикатов нарушения симметрии на языке SAT, которое позволило сократить размер булевой формулы с $\mathcal{O}\left(M^3 + M^2 \times L^2\right)$ до $\mathcal{O}\left(M^2 \times L\right)$ дизъюнктов. Более того, в оригинальном кодировании большинство дизъюнктов состояли из $\mathcal{O}\left(M\right)$ литералов, в то время как в новом более компактном сведении большая часть дизъюнктов состоит из двух

или трех литералов, что облегчает работу программному средству для решения задачи SAT, так как такие дизъюнкты обрабатываются за константное время и не хранятся в памяти [79]. Помимо этого были предложены предикаты нарушения симметрии, учитывающие особенности BFS-дерева. Экспериментальные исследования показали, что время работы метода, использующего все предложенные предикаты нарушения симметрии, основанные на кодировании алгоритма BFS и на особенностях BFS-дерева, значительно меньше времени работы метода DFA-Inductor, являющегося лучшим известным точным методом генерации ДКА по заданным примерам поведения. Так, на тестовых данных за одно и тоже время новым методом было сгенерирован 731 автомата против 678 сгенерированных автоматов при использовании лучшего из известных на момент начала исследования метода. Данные результаты были опубликованы в статьях [31; 80].

Глава 3 Генерация детерминированных конечных автоматов по избыточным примерам поведения

В настоящей главе описывается разработка, реализация и экспериментальные исследования комбинированного метода генерации детерминированных конечных автоматов, на основе сведения к задаче выполнимости булевых формул и с использованием подхода уточнения абстракции по контрпримерам.

3.1 Масштабируемость предложенных методов в зависимости от размера расширенного префиксного дерева

Как было показано в разделе 2.4, предложенный метод генерации детерминированных конечных автоматов с минимальным числом состояний по заданным примерам поведения с использованием предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в ширину является самым эффективным из известных точных методов. Можно заметить, что число ДКА с M состояниями и мощностью алфавита $|\Sigma| = L$ равняется $M \times M^{M \times L} \times 2^M$. Действительно, существуют M различных способов выбрать стартовое состояние, $M^{M \times L}$ различных функций перехода (ее можно представить в виде таблицы размера $M \times L$, где в каждой ячейке находится одно из M состояний) и 2^M различных способа выбрать принимающие состояния. Таким образом, при увеличении размера искомого автомата, экспоненциально растет и пространство поиска. Глава 2 настоящей диссертации посвящена разработке различных предикатов нарушения симметрии, позволяющих быстрее отсекать различные области пространства поиска. Большая часть из предложенных предикатов нарушения симметрии определяют нумерацию искомого автомата и позволяют отсекать большое число решений вне зависимости от имеющихся примеров поведения. Однако, исходное сведение, описанное в разделе 1.4.2, сильно зависит от числа имеющихся примеров пове-

дения — чем больше примеров поведения имеется, тем проще и быстрее будут отыскиваться конфликты при попытках построить различные ДКА.

С другой стороны, булева формула, кодирующая задачу генерации ДКА по примерам поведения на языке SAT, состоит из $\mathcal{O}\left(N \times M^2\right)$ дизъюнктов (N — размер расширенного префиксного дерева, M — размер искомого автомата), а значит ее размер линейно зависит от числа состояний в расширенном префиксном дереве. Число используемых переменных также линейно зависит от размера префиксного дерева — $\mathcal{O}(M^2 + N \times M)$. Несмотря на то, что современные программные средства для решения SAT могут решать задачи с большим числом переменных и большим числом дизъюнктов, работа со слишком большой формулой требует больших временных затрат на работу с ней и больших затрат памяти для хранения всех дизъюнктов. Таким образом, получается, что для программного средства плохо, когда примеров поведения слишком мало, и также плохо, когда примеров поведения слишком много. Ситуацию с недостаточным числом примеров поведения нельзя решить без получения новых примеров поведения, что выбивается за рамки решаемой задачи. В случае же *избыточных данных* логичным решением является попытаться построить автомат по части примеров поведения, однако вопрос выбора этой части не является тривиальным. Среди тысяч или десятков тысяч примеров поведения, может быть единственный пример поведения, описывающий некоторую часть автомата, которую не описывают другие примеры поведения. В следующем разделе приводится описание разработанного метода генерации ДКА по избыточному числу примеров поведения, позволяющего итеративно выбирать “нужные” примеры поведения, отсекая лишние.

3.2 Метод генерации детерминированных конечных автоматов на основе сведения к задаче выполнимости и с использованием подхода уточнения абстракции по контрпримерам

Суть подхода уточнения абстракции по контрпримерам можно описать следующим образом. На начальном шаге генерируется некоторая, возможно случайная модель. Затем на каждом следующем шаге данная модель проходит проверку некоторой проверяющей системы. Если проверка проходит успешно, то искомая модель найдена. Иначе система возвращает один или несколько контрпримеров, которые затем используются для уточнения модели. Процесс повторяется, пока не будет найдена модель, проходящая проверку системы. Данный подход больше похож на метод активного построения модели, в то время как в данной работе рассматривается задача пассивного построения — все примеры поведения известны заранее и никакой дополнительной информации в ходе построения автомата быть получено не может. Однако далее приводится описание того, как метод уточнения абстракции можно применить для построения ДКА.

Как и классический алгоритм CEGAR, предлагаемый метод итеративно уточняет модель, которая в настоящей диссертации является детерминированным конечным автоматом. Предлагается строить ДКА только по части примеров поведения, а затем проверять получившийся автомат на соответствие всем остальным примерам поведения. Первым шагом как и в остальных методах строится расширенное префиксное дерево, однако строится оно не по всем примерам поведения, а лишь по некоторой части. Можно использовать какую-то случайную часть слов из обучающей выборки, либо вообще начать с пустого дерева, состоящего только из корня, который не является ни принимающим, ни отвергающим. Как и ранее, в качестве нижней оценки на размер генерируемого ДКА можно взять любую достоверную оценку, либо с единицы. Затем, на каждом шаге работы алгоритма предлагается с помощью сведения к SAT, описанного в разделе 1.4.2, и с использованием предикатов нарушения симметрии, описанных в главе 2, пытать-

ся генерировать ДКА текущего размера по текущему префиксному дереву. Если такой ДКА не существует, то размер искомого автомата увеличивается на единицу и процесс поиска повторяется, как в методах, описанных ранее. Если же автомат, соответствующий текущему расширенному префиксному дереву найден, то он проверяется на соответствие всему множеству примеров поведения — каждый из неиспользованных примеров поведения обрабатывается автоматом, и затем проверяется совпадает ли статус допуска данного примера с фактическим. Если ДКА соответствует всем примерам поведения, то задача решена. Иначе, среди тех примеров поведения, которым построенный автомат не соответствует, выбирается один или несколько контрпримеров, по которым достраивается префиксное дерево, строится новая булева формула и поиск продолжается. Схема предложенного метода представлена на рисунке 15.

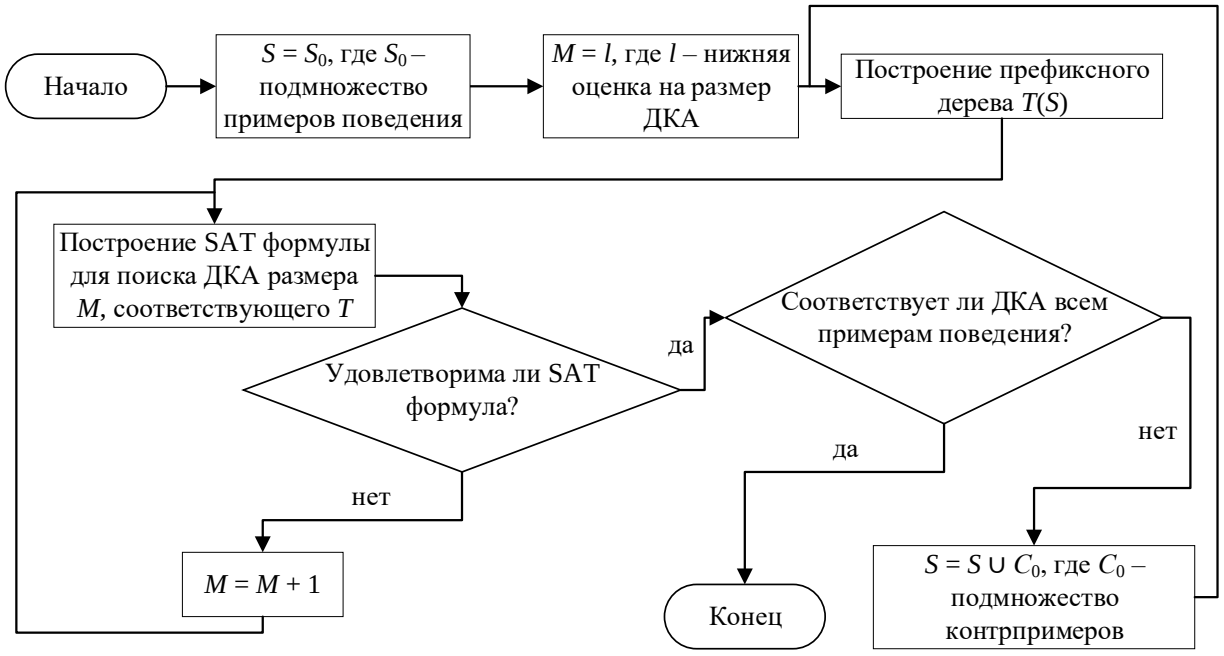


Рисунок 15 – Схема точного метода генерации ДКА по избыточному набору примеров поведения на основе сведения к SAT и с использованием подхода CEGAR

Необходимо заметить, что при добавлении контрпримеров в префиксное дерево, размер искомого автомата не меняется, а значит пространство поиска решения SAT только сужается, а значит нет необходимости начинать поиск выполняющей подстановки заново. Можно использовать инкрементальные программные

средства [81], которые после нахождения некоторой выполняющей подстановки переходят в режим ожидания новых дизъюнктов и затем продолжают поиск решения уже для новой уточненной формулы с того места, где остановились в прошлый раз, сохраняя все новые обученные дизъюнкты.

3.3 Реализация и экспериментальные исследования разработанного метода

В настоящем разделе приводятся описание реализации разработанных методов и экспериментальные исследования, проведенные с ними.

Реализация разработанного комбинированного метода генерации детерминированных конечных автоматов. Предложенный в предыдущем разделе комбинированный метод построения ДКА минимального размера на основе сведения к SAT и с использованием подхода уточнения абстракции по контрпримерам был реализован на языке *python* как модуль программного комплекса `DFA-Inductor-py`. Для использования данного метода необходимо указать параметр "`--cegar-mode/--cegar`". Также, с помощью параметра "`--initial-amount/-init`" можно указать число примеров поведения, используемых для построения расширенного префиксного дерева, а с помощью параметра "`--step-amount/-step`" можно указать число контрпримеров, добавляемых в расширенное префиксное дерево каждый раз, когда был найден ДКА, не соответствующий всем примерам поведения. Любые реализованные предикаты нарушения симметрии и граф несовместимости могут использоваться совместно с данным методом.

Экспериментальные исследования разработанного метода комбинированного метода генерации детерминированных конечных автоматов Эксперименты проводились на персональном компьютере с процессором *QuadCore Intel Core i7-8550U* @ 4 ГГц, 16 ГБ оперативной памяти и операционной системой

ArchLinux 5.5.6. Для проведения экспериментов было сгенерировано 100 тестовых экземпляров с помощью алгоритма, описанного в разделе 2.4. Параметры для генерации автоматов были выбраны следующим образом:

– размеры автоматов, которые нужно построить, — $M \in [15; 25]$;
– число примеров поведения $S = S_+ \cup S_- \in \{50 \times N; 100 \times N; 200 \times N; 500 \times N\}$.

В экспериментах проводилось сравнение разработанного в настоящей главе метода, использующего подход уточнения абстракции по контрпримерам и сведение к SAT, с методом, описанным в главе 2 и основанным только на сведении к SAT. Результаты показали, что при относительно небольшом количестве примеров поведения ($S \in \{50 \times N; 100 \times N\}$) использование комбинированного подхода сокращает время построения вспомогательных структур данных (таких как граф совместимости) и время построения булевой формулы, но увеличивает время работы программного средства для решения SAT и, как следствие, суммарное время решения задачи. Однако в случае, когда количество примеров поведения достаточно велико ($S \in \{200 \times N; 500 \times N\}$), использование предложенного подхода позволяет использовать меньше половины примеров поведения вместо всех, что сокращает как время построения структур данных и булевой формулы, так и время работы программного средства для решения SAT. Так, метод, использующий только часть примеров поведения, работает как минимум в два раза быстрее метода, использующего сразу все примеры поведения. Более того, выигрыш от использования подхода уточнения абстракции по контрпримерам увеличивается с ростом числа примеров поведения. Значительная часть таких экземпляров не были в принципе решены методом, основанным только на сведении к SAT, ввиду чрезмерно большого размера булевой формулы. Таким образом, можно сделать вывод, что использование разработанного метода целесообразно, когда количество примеров поведения велико, и метод, основанный только на сведении к SAT, не применим ввиду слишком большой формулы.

Выводы по главе 3

В третьей главе был предложен точный метод генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам. Разработанный метод был реализован в рамках программного средства `DFA-Inductor-py`.

Размер булевой формулы, получающейся в результате сведения задачи генерации ДКА к задаче выполнимости, зависит как от размера генерируемого автомата, так и от размера расширенного префиксного дерева — $\mathcal{O}\left(N \times M^2\right)$ дизъюнктов, где N — размер префиксного дерева, M — размер генерируемого ДКА. Таким образом, при избыточном числе примеров поведения размер префиксного дерева велик, а значит и булева формула состоит из слишком большого числа дизъюнктов, что слишком сложно для существующих на сегодняшний день программных средств для решения SAT. Экспериментальные исследования показали, что при большом числе примеров поведения $S = |S_+| + |S_-| \geqslant 200 \times M$ выигрыш по скорости работы метода от двух раз и тем больше, чем больше число примеров поведения. Все результаты данной главы опубликованы в статье [82].

Глава 4 Генерация всех неизоморфных детерминированных конечных автоматов, удовлетворяющих заданным примерам поведения

В данной главе рассматривается задача генерации всех неизоморфных детерминированных конечных автоматов минимального размера, соответствующих заданным примерам поведения, которая ранее не имела эффективного решения. Разрабатывается метод, основанный на подходе к решению задачи генерации одного ДКА с использованием программных средстве решения SAT, подробно описанный в разделе 1.4. Рассматриваются два варианта использования программных средств решения SAT: перезапуск неинкрементального средства после нахождения каждого автомата и использование инкрементального средства — если такое средство находит некоторое решение, то оно сохраняет свое текущее состояние и может принимать новые дизъюнкты. Подробнее про инкрементальное решение SAT можно прочитать в [81]. Также в данной главе предлагается переборный метод, который служит базовым для сравнения с методом, основанным на сведении к задаче выполнимости.

4.1 Мотивация и постановка задачи

В классической задаче генерации ДКА по заданным примерам поведения (см. раздел 1.2.3) ищется некоторый детерминированный конечный автомат минимального размера, соответствующий имеющимся примерам. Однако, вполне возможна ситуация, когда одному и тому же набору примеров поведения соответствует более одного ДКА минимального размера M. Как было описано в разделе 1.2.2 для каждого автомата $\mathcal{D}$ размера M существует $\mathcal{O}(M)$ изоморфных ему автоматов. Но изоморфные автоматы задают один и тот же язык, поэтому не представляют никакого интереса. В настоящей главе рассматривается ситуация, когда существуют несколько различных, неизоморфных ДКА минимального размера, соответствующих примерам поведения. На рисунке 16 приведены все

неизоморфные ДКА минимального размера, построенные по заданным примерам поведения.

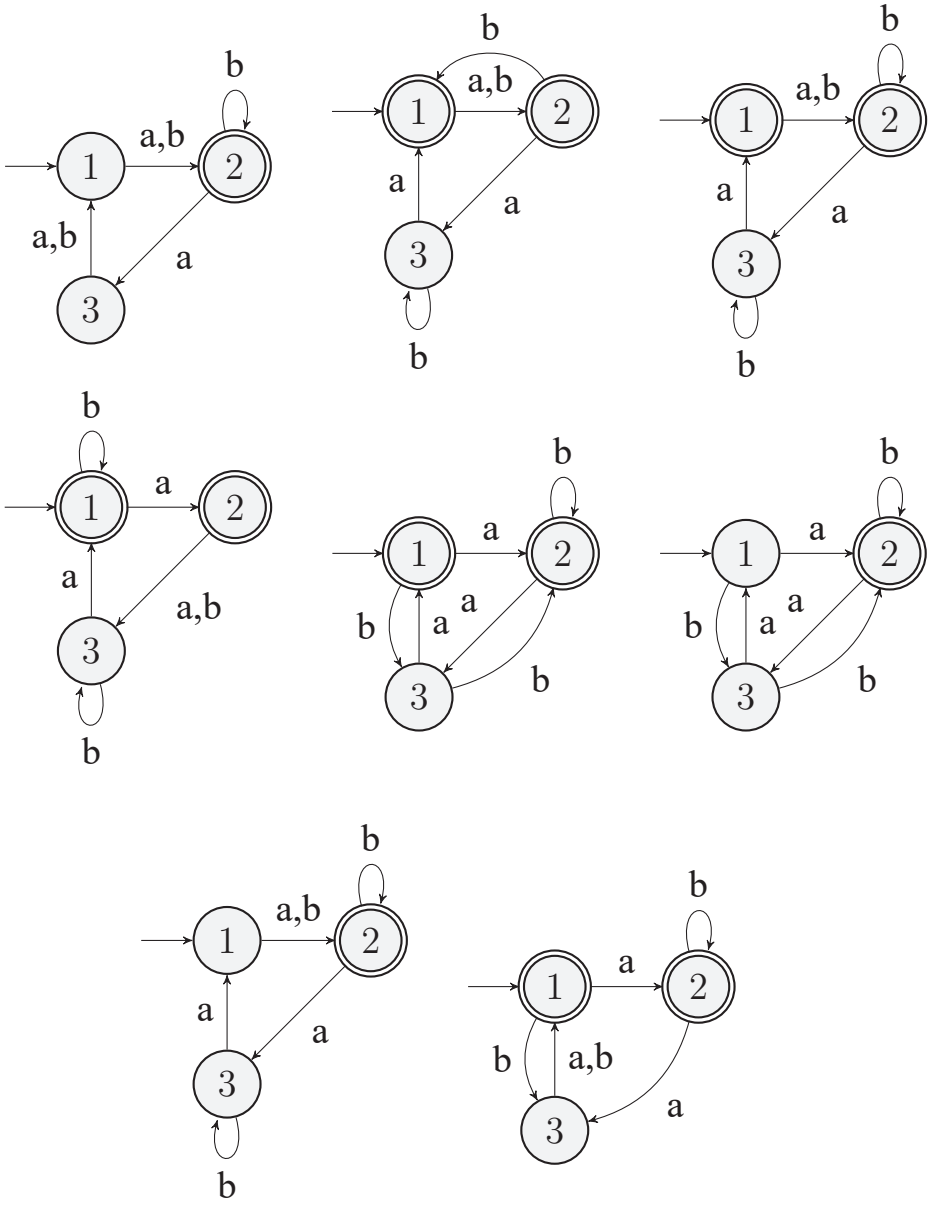


Рисунок 16 – Все неизоморфные ДКА, соответствующие множествам примеров поведения $S_+ = \{a, bb, aaaa\}$ и $S_- = \{aa, bab\}$

Такая ситуация может возникнуть, либо если заданных примеров поведения недостаточно, чтобы точно описать целевой автомат, либо они недостаточного качества. Например, примеров поведения очень много, но большая их часть покрывает лишь часть переходов целевого ДКА. Тогда становится актуальной генерация всех возможных автоматов минимального размера, соответствующих заданным примерам поведения. Имея все возможные автоматы их можно проана-

лизировать, чтобы найти закономерности или чтобы понять, почему имеющиеся примеры поведения допускают несколько решений. Также, нужный автомат среди всех найденных может быть выбран в дальнейшем при помощи дополнительных проверок или спецификаций.

Формально задача генерации всех ДКА формулируется аналогично задаче генерации одного автомата. Пусть примеры поведения представлены двумя множествами слов над алфавитом Σ: S_+ — множество слов, которые должны допускаться искомыми автоматами; S_- — множество слов, которые должны отвергаться искомыми автоматами. Тогда задача генерации всех различных детерминированных конечных автоматов заключается в поиске все неизоморфных автоматов, который будут принимать все слова из S_+ и отвергать все слова из S_-.

Эвристические, метаэвристические и другие неточные методы генерации автоматов по примерам поведения не применимы для решения такой задачи, так как число находимых ими автоматов ничем не ограничено. Точные методы генерации ДКА на основе сведения к SAT применимы для поиска всех различных автоматов, так как способны перебрать все пространство поиска и найти все автоматы минимального размера, но без применения BFS-предикатов нарушения симметрии они допускают факториал изоморфных ДКА, что не позволяет применять их на практике. BFS-предикаты, в свою очередь, как было представлено в разделе 1.4.2, оставляют единственного представителя для каждого класса эквивалентности по изоморфизму. В следующем разделе предлагается первый эффективный метод для решения задачи генерации всех различных ДКА, основанный на сведении к SAT и BFS-предикатах.

Еще одним важным применением разрабатываемого метода является формальная возможность доказать единственность найденного ДКА. Действительно, если при решении задачи генерации всех различных ДКА минимального размера генерируется единственный автомат, то можно сделать вывод, что других автоматов такого размера, соответствующих заданным примерам поведения, не существует. Данный факт может использоваться как косвенное доказательство того,

что обучающие данные подобраны достаточно хорошо и хорошо описывают найденный автомат.

4.2 Метод генерации всех неизоморфных детерминированных конечных автоматов, соответствующих заданным примерам поведения, основанный на сведении к задаче выполнимости

Как было описано в разделе 1.2.2, для каждого ДКА с M состояниями существует $\mathcal{O}(M!)$ изоморфных ему автоматов. Основной идеей предлагаемого метода по генерации всех ДКА является запрет (блокирование) найденных удовлетворяющих подстановок. Очевидно, что без использования предикатов нарушения симметрии, придется блокировать $\mathcal{O}(M)$ изоморфных автоматов для каждого найденного уникального ДКА, что даже при $M = 10$ требует значительных вычислительных затрат. Так как подход к нарушению симметрии с помощью большой клики, описанный в разделе 1.4.2, позволяет зафиксировать только нумерацию k состояний, то метод, предложенный в [18] находит $(C - k)!$ изоморфных автоматов, что в общем случае все еще вычислительно сложно.

Предикаты нарушения симметрии на основе обхода в ширину, как было описано в разделе 1.4.2, в свою очередь допускают единственного представителя каждого класса эквивалентности по изоморфизму. Конкретно, BFS-предикаты допускают BFS-пронумерованный автомат. Тогда, заблокировав единственный найденный автомат, автоматически блокируется весь класс эквивалентности, а значит далее программное средство может найти другой автомат только если он не является изоморфным ранее найденным. Необходимо заметить, что несмотря на то, что идея блокировать найденные решения с целью найти остальные достаточно проста и известна, применять ее на практике невозможно без эффективных предикатов нарушения симметрии. Так, ранее не было описано способов бороться с факториальным числом изоморфных автоматов, а значит и рассматриваемая задача не имела эффективного решения. Использование BFS-предикатов позво-

ляет решить данную задачу путем добавления *блокирующего* дизъюнкта в булеву формулу.

Как было сказано в разделе 1.2.1, детерминированным конечным автоматом называется пятерка $\mathcal{D} = (D, \Sigma, \delta, d_1, D^+)$. При построении автомата по найденной выполняющей подстановке множество состояний D задано неявно через текущее число состояний автомата M, алфавит Σ — через число L, стартовое состояние d_1 всегда имеет номер 1, функция переходов δ определяется с помощью переменных переходов $y_{i,l,j}$, а множество допускающих состояний с помощью переменных допуска z_i. Тогда достаточно из всей найденной выполняющей подстановки запретить только значения переменных переходов и значения переменных допуска. Если с помощью $\varphi(q)$ обозначить значение переменной q в найденной подстановке φ и определить множество $\mathcal{Y} = \{y_{i,l,j} | i, j \in [M] \wedge l \in \Sigma \wedge \varphi(y_{i,l,j}) = 1\}$, то блокирующий дизъюнкт можно определить следующим образом:

$$\bigwedge_{y \in \mathcal{Y}} \neg y \wedge \bigwedge_{i \in [M]} \neg \varphi(z_i).$$

Существует два различных способа использования программных средств для решения SAT. Во-первых, можно перезапускать неинкрементальное средство с новой булевой формулой с добавленным блокирующим дизъюнктом после каждого нахождения выполняющей подстановки. Недостатком данного подхода является то, что процесс поиска решения каждый раз начинается заново, и программное средство каждый раз проделывает полный объем работы по перебору пространства поиска. Второй подход основан на использовании инкрементальных программных средств, которые после нахождения выполняющей подстановки сохраняют свое состояние и способны продолжить поиск решения после добавления некоторого уточнения, выраженного одним или несколькими дизъюнктами. Подробнее про инкрементальное решение SAT можно прочитать в [81].

Необходимо заметить, что если найдено два автомата $\mathcal{D}_1 = \left(D, \Sigma, \delta, d_1, D_1^+\right)$ и $\mathcal{D}_2 = \left(D, \Sigma, \delta, d_1, D_2^+\right)$, которые различаются только множествами допускающих состояний, то можно сделать вывод, что в исходных примерах поведения нет строк, определяющих допуск одного или нескольких состояний автоматов $\mathcal{D}_1$ и

$\mathcal{D}_2$. Если использовать предложенный выше блокирующий дизъюнкт, то будут найдены все такие автоматы, отличающиеся лишь множествами допускающих состояний. Таких автоматов будет 2^k, если k — это число состояний, для которых не хватает информации о допуске. В общем случае, если в исходных данных не хватает информации, чтобы понять является ли состояние допускающим или нет, то не важно, будет ли данное состояние допускающим в найденном автомате. Тогда блокирующий дизъюнкт можно сократить до следующего вида:

$$\bigwedge_{y \in \mathcal{Y}} \neg y.$$

Также необходимо заметить, что так как всегда строится полный автомат, то бывают случаи, когда некоторые переходы найденного ДКА не покрываются переходами расширенного префиксного дерева. Это значит, что существуют некоторые *свободные* переходы, которые не используются при обработке любого из исходных примеров поведения. Свободные переходы могут вести в любое состояние в найденном автомате, так как это не влияет на соответствие найденного ДКА примерам поведения. Тогда существует M^k автоматов, различающих только одним или несколькими из k свободных переходов. Как и в предыдущем случае, если в исходных данных не хватает информации, чтобы определить некоторый переход, то не важно, куда он будет вести. Предлагается зафиксировать все свободные переходы в виде петель — пусть все свободные переходы ведут в то же состояние откуда выходят. Добиться этого можно с помощью дополнительных переменных использования (**u**sed) $\{u_{i,l}\}_{i \in [M], l \in \Sigma}$, которые истинны тогда и только тогда, когда существует переход из некоторой вершины префиксного дерева $\mathcal{T}$, соответствующей состоянию d_i автомата $\mathcal{D}$, по символу l:

$$\bigwedge_{1 \leqslant i \leqslant M} \bigwedge_{l \in \Sigma} u_{i,l} \leftrightarrow \bigvee_{v \in V_l} x_{v,i},$$

где V_l — множество все состояний расширенного префиксного дерева, имеющих исходящее ребро по символу l, то есть $V_l = \{v | \exists w . \tau(v, l) = w\}$.

Тогда, зафиксировать свободные переходы в виде петель можно с помощью добавления следующего ограничения:

$$\bigwedge_{1 \leqslant i \leqslant M} \bigwedge_{l \in \Sigma} \neg u_{i,l} \rightarrow y_{i,l,i}.$$

Данное ограничение выражается через $\mathcal{O}(M \times L)$ дизъюнктов. На рисунке 17 представлено расширенное префиксное дерево, построенное по множествам примеров поведения $S_+ = \{ab, b, ba, bbb\}$ и $S_- = \{abbb\}$. На рисунке 18 представ-

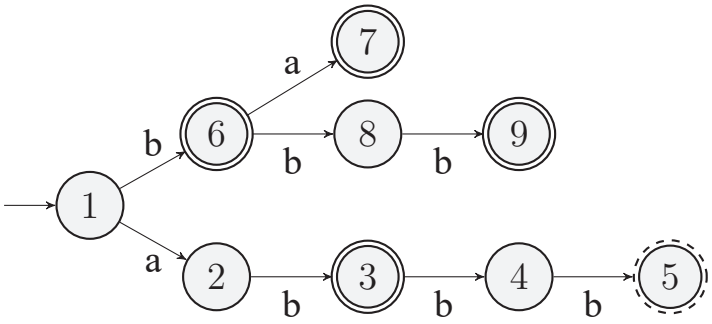


Рисунок 17 – Пример расширенного префиксного дерева для множеств слов $S_+ = \{ab, b, ba, bbb\}$ и $S_- = \{abbb\}$

лен ДКА минимального размера, соответствующий расширенному префиксному дереву, изображенному на рисунку 17. Переход из состояния 2 по символу a в данном автомате не покрывается никакими переходами префиксного дерева, а значит является свободным. С помощью предложенных ранее ограничений можно зафиксировать данный переход в виде петли, что продемонстрировано на рисунке штриховой линией.

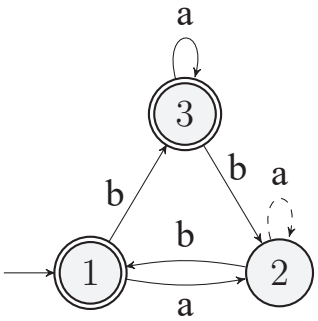


Рисунок 18 – Пример ДКА, построенного по префиксному дереву, представленному на рисунке 17, с зафиксированным свободным переходом из состояния 2 по символу a

4.3 Реализация и экспериментальные исследования разработанных методов

В настоящем разделе приводятся описание реализации разработанных методов и экспериментальные исследования, проведенные с ними.

Алгоритм перебора с возвратами. Так как ранее не предлагалось методов для поиска всех возможных примеров поведения по заданным словарям, то был разработан переборный алгоритм с возвратами, не использующий никаких сторонних программных средств. Данный алгоритм работает следующим образом. Изначально создается псевдоДКА $\mathcal{D}$ с M вершинами (где M — размер искомого ДКА), но без переходов и без выделенных допускающих состояний, который затем будет достраиваться до полноценного ДКА. Для этого на каждой итерации алгоритма поддерживается фронт расширенного префиксного дерева $\mathcal{A}$ $frontier$ — множество ребер префиксного дерева A, для которых все предшествующие ребра на пути от корня дерева до них уже представлены в автомате $\mathcal{D}$, а они сами — еще нет. Изначально фронт состоит из всех исходящих из корня дерева $\mathcal{A}$ ребер. Рекурсивная функция `backtracking` поддерживает фронт в актуальном состоянии. Если фронт не пуст, тогда данная функция пытается достроить текущий ДКА с помощью одного из ребер фронта. После каждого достраивания текущий автомат проверяется на соответствие префиксному дереву, и, если противоречий не найдено, то фронт обновляется. Если же фронт пуст, то все ребра префиксного дерева $\mathcal{A}$ представленны в автомате $\mathcal{D}$, а значит соответствующий автомат найден. Тогда ДКА $\mathcal{D}$ проверяется на полноту, и, если автомат не полон, то недостающие переходы добавляются в виде петель (аналогично тому, как это предлагалось делать в предыдущем разделе с помощью переменных использования) с помощью функции `makeComplete`. Псевдокод разработанного метода представлен в листинге 1. Функция `findNewFrontier` возвращает новый фронт для дополненного ДКА или `NULL`, если ДКА не соответствует расширенному префиксному дереву. Данный алгоритм полного поиска основан на алгоритме из [83].

Листинг 1 – Переборный алгоритм с возвратами для генерации всех ДКА минимального размера

Input: расширенное префиксное дерево $\mathcal{A}$; текущая версия искомого ДКА $\mathcal{D}$; фронт $frontier$.

Output: множество S всех неизоморфных ДКА, соответствующих префиксному дереву $\mathcal{A}$.

```
function BACKTRACKING(𝒜, 𝒟, frontier)
    S ← пустое множество
    edgeedge ← любое ребро из frontier
    for all destination ∈ 1 ... 𝒟.size do
        source ← состояние 𝒟, которому соответствует edge.from
        𝒟' ← 𝒟 ∪ TRANSITION(source, destination, edge.label)
        frontier' ← FINDNEWFRONTIER(𝒜, 𝒟', frontier)
        if frontier' ≠ NULL then
            if frontier' = ∅ then
                S.add (MAKECOMPLETE(𝒟'))
            else
                S.add (BACKTRACKING(𝒜, 𝒟', frontier'))
    return S
```

Реализация разработанных методов генерации всех детерминированных конечных автоматов. Предложенный в предыдущем разделе метод генерации ДКА минимального размера на основе сведения к SAT и с использованием подхода уточнения абстракции по контрпримерам был реализован на языке *python* как модуль программного комплекса `DFA-Inductor-py`. Для использования данного метода необходимо указать параметр "`--find-all/-all`". Также, с помощью параметра "`--find-k/-find`" можно указать число ДКА, которые необходимо сгенерировать. Любые реализованные предикаты нарушения симметрии и граф несовместимости могут использоваться совместно с данным методом. Метод генерации ДКА по избыточному набору примеров поведения, использующий подход уточнения абстракции, также может использоваться совместно с данным методом для генерации всех неизоморфных ДКА по избыточному набору примеров поведения.

Экспериментальные исследования разработанных методов генерации всех детерминированных конечных автоматов. Все эксперименты проводились на сервере с 64-ядерным процессором `AMD Opteron 6378 @ 2.4 ГГц` и операционной системой `Ubuntu 14.04`. Для генерации тестовых данных снова использовался алгоритм, предложенный в разделе 2.4. Использовались следующие параметры: размер искомого автомата $M \in [5; 15]$, мощность алфавита $L = 2$, суммарное число примеров поведения $S \in \{5 \times M; 10 \times M; 25 \times M\}$. Для каждого набора параметров генерировалось по 100 различных автоматов и соответствующих наборов примеров поведения.

Сравнивались три метода генерации всех неизоморфных ДКА по примерам поведения:

- метод, основанный на сведении к SAT, из раздела 4.2 с перезапуском программного средства для решения SAT (столбец REST в таблице);
- метод, основанный на сведении к SAT, из раздела 4.2 с использованием инкрементального программного средства для решения SAT (INC);
- переборный метод с возвратами из раздела 4.3 (BTR).

Результаты экспериментальных исследований представлены в таблице 3. Ограничение по времени было установлено в один час (3600 секунд). Столбец >1 показывает процент экземпляров задачи, где существовало больше одного неизоморфного автомата. Так как в таблице представленно медианное время, то если менее 50 (то есть менее половины) экземпляров были решены, то в соответствующей ячейке стоит прочерк (—).

Результаты экспериментов позволяют сделать несколько выводов. Во-первых, впервые успешно решена задача генерации всех различных ДКА минимального размера по заданным примерам поведения. Во-вторых, оба метода, использующие сведение к SAT, значительно превосходят по производительности переборный метод. В-третьих, использование инкрементального программного средства, как и предполагалось, дает заметное преимущество относительно подхода с перезапуском программного средства, что объясняется сохранением промежуточного состояния инкрементальным программным средством после на-

Таблица 3 – Медианное время нахождения всех различных ДКА с помощью метода на основе сведения к SAT с перезапуском программного средства (REST), метода на основе сведения к SAT с использованием инкрементального программного средства (INC) и переборного метода с возвратами (BTR)

M	$S = 5 \times M$				$S = 10 \times M$				$S = 25 \times M$			
	>1	REST	INC	BTR	>1	REST	INC	BTR	>1	REST	INC	BTR
5	53	2,3	2,0	0,8	40	3,6	3,3	1,3	17	4,1	3,4	1,5
6	56	2,8	2,4	2,1	31	4,7	3,9	1,7	27	5,4	4,3	1,7
7	87	3,9	2,5	4,1	27	3,7	3,0	3,1	13	7,4	6,7	2,5
8	80	4,6	3,7	87,2	34	7,0	6,5	41,7	16	10,1	8,9	11,6
9	91	7,6	3,9	475,1	50	7,7	6,4	121,6	10	13,8	13,0	61,4
10	89	15,7	5,3	2756,2	47	8,6	7,0	974,7	11	18,8	16,1	276,8
11	94	19,9	7,3	—	63	18,5	13,8	3108,0	9	24,5	21,9	1158,4
12	90	28,0	9,9	—	49	22,3	16,7	—	8	33,5	27,2	3289,1
13	92	185,5	18,1	—	57	36,9	22,6	—	12	62,0	51,4	—
14	87	408,5	49,0	—	71	85,1	41,8	—	4	67,0	56,2	—
15	95	571,1	174,1	—	69	193,3	95,7	—	6	29,2	26,2	—

хождения некоторого ДКА. В-четвертых, чем больше примеров поведения дано для генерации ДКА, тем реже случается ситуация, когда существует несколько различных ДКА, соответствующих им. Однако, надо заметить, что помимо количества примеров поведения, их качество не менее важно.

Выводы по главе 4

В четвертой главе была дана постановка задачи генерации всех неизоморфных ДКА, удовлетворяющих заданным примерам поведения, а также предложен метод решения поставленной задачи. Разработанный метод был реализован в рамках программного средства `DFA-Inductor-py`.

Детерминированный конечный автомат минимального размера является максимально точным обобщением имеющихся данных, выраженных с помощью примеров поведения. Однако в случае, когда примеры поведения недостаточно хорошо описывают искомый автомат, может существовать несколько различных неизоморфных удовлетворяющих автоматов минимального размера. В таком случае рациональным решением может быть построение всех таких автоматов. Также, с помощью метода генерации всех ДКА, можно доказать единственность автомата с минимальным числом состояний, соответствующего заданным примерам

поведения. Единственность ДКА в таком случае говорит о том, что имеющиеся данные хорошо описывают найденный автомат.

Без использования предикатов нарушения симметрии на основе кодирования алгоритма BFS или DFS не представляется возможным написать эффективный метод генерации всех неизоморфных ДКА, так как для любого ДКА существует $\mathcal{O}(M!)$ изоморфных автоматов. Использование BFS-предикатов нарушения симметрии позволяет оставить единственного представителя для каждого класса эквивалентности по изоморфизму. Был реализован метод, использующие такие предикаты, для генерации всех неизоморфных ДКА минимального размера. Экспериментальные исследования показали, разработанный метод демонстрирует значительно меньшее время генерации ДКА относительно реализованного алгоритма перебора с возвратами. Также, использование программных средств для решения SAT в итеративном режиме позволяет значительно повысить эффективность разработанного метода, так как между вся накопленная информация сохраняется между запусками программного средства. Все результаты данной главы опубликованы в статье [78].

Заключение

Основные результаты работы заключаются в следующем:

а) Разработаны предикаты нарушения симметрии, основанные на кодировании алгоритмов обхода графа в ширину и в глубину, для сокращения пространства поиска при решении задачи выполнимости и точные методы генерации ДКА по заданным примерам поведения, использующие данные предикаты. Предикаты нарушения симметрии, основанные на кодировании алгоритма обхода графа в глубину имеют скорее теоретическую значимость, так как не продемонстрировали никаких преимуществ, относительно предикатов, основанных на кодировании алгоритма обхода графа в ширину. Новый способ кодирования предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в ширину позволяет использовать асимптотически меньшее число дизъюнктов относительно предложенного ранее. Разработанный точный метод, использующий данные предикаты совместно с предикатами, учитывающими особенности дерева обхода графа в ширину, демонстрирует лучшую производительность относительно существовавших ранее методов.

б) Разработан точный метод генерации ДКА по избыточному набору примеров поведения с использованием сведения к задаче выполнимости и подхода уточнения абстракции по контрпримерам. Размер булевой формулы и число используемых переменных линейно возрастает при увеличении числа примеров поведения. Разработанный метод позволяет генерировать ДКА по избыточному набору примеров поведения, итеративно расширяя множество используемых примеров, достраивая расширенное префиксное дерево и используя программное средство для решения SAT в инкрементальном режиме. Так как множество используемых примеров поведения расширяется контрпримерами к построенным промежу-

точным ДКА, то в итоге для генерации используются только значимые примеры поведения. Таким образом, с помощью данного метода удается точно решать такие задачи, которые раньше не могли быть решены ввиду большого размера булевой формулы.

в) Разработан метод генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, с использованием предикатов нарушения симметрии и программных средств решения задачи выполнимости. Ранее задача генерации всех неизоморфных ДКА минимального размера не имела эффективного решения ввиду того, что изоморфные автоматы, являясь одинаковыми по структуре и по определяемому языку, программным средством для решения SAT считаются различными, что приводит к рассмотрению $\mathcal{O}\left(M!\right)$ изоморфных автоматов с M состояниями. Использование предикатов нарушения симметрии на основе кодирования алгоритма обхода графа в ширину позволяет для каждого класса эквивалентности по изоморфизму оставить для рассмотрения единственного представителя вместо факториала. Таким образом, впервые был предложен метод решения задачи генерации всех неизомофрных ДКА минимального размера. Поиск всех неизоморфных автоматов может быть полезен для дальнейшего их анализа, либо для анализа имеющихся примеров поведения. Другим применением разработанного метода является возможность доказать единственность минимального автомата, соответствующего заданным примерам поведения.

г) Во время работы над диссертацией на языке *Python* было разработано программное средство с открытым исходным кодом `DFA-Inductor-py`, предназначенное для генерации ДКА по заданным примерам поведения. В состав средства входят различные модули, позволяющие решать задачу генерации ДКА по заданным примерам поведения, задачу генерации ДКА по избыточному набору примеров поведения и задачу генерации всех неизоморфных ДКА по заданным примерам поведения. В средстве реализованы различные предикаты

нарушения симметрии — как предложенные ранее другими авторами, так и разработанные в рамках настоящей диссертации.

д) Результаты работы были использованы в учебном процессе на факультете информационных технологий и программирования Университета ИТМО в рамках курса «Проектирование автоматных программ» программы бакалавриата «Математические модели и алгоритмы в разработке программного обеспечения», что подтверждается актом об использовании.

е) Часть результатов работы использовалась при выполнении проекта SAUNA (“Integrated safety assessment and justification of nuclear power plant automaton”), выполненного исследовательской группой “IT in Automation” кафедры электротехники и автоматики университета Аалто, Финляндия, в рамках Финской программы исследований безопасности атомных электростанций — SAFIR2018, что подтверждается письмом руководителя исследовательской группы “IT in Automation” В. В. Вяткина.

ж) Результаты работы также использовались при выполнении под руководством автора диссертации гранта Российского фонда фундаментальных исследований (проект 183700425 «Разработка эффективных методов машинного обучения для построения детерминированных конечных автоматов на основе решения задачи выполнимости», 2018–2020 гг.) и в рамках проектов по программе повышения конкурентоспособности ведущих российских университетов среди ведущих мировых научно-образовательных центров «5-100».

Точный метод генерации ДКА по заданным примерам поведения, использующий предикаты нарушения симметрии на основе алгоритма обхода графа в ширину, демонстрирует лучшую производительность относительно известных ранее методов и позволяет генерировать автоматы большего размера за меньшее время. Метод генерации ДКА по избыточному набору примеров поведения позволяет генерировать автоматы по таким данным, по которым известные ранее ме-

тоды не могли построить ДКА в принципе ввиду большого размера булевой формулы. Метод генерации всех неизоморфных ДКА минимального размера, удовлетворяющих заданным примерам поведения, является первым известным методом, позволяющим сгенерировать все неизоморфные автоматы — существовавшие ранее методы могут быть адаптированы для поиска всех ДКА, но их использование приводит к возникновению комбинаторного взрыва числа рассматриваемых автоматов ввиду отсутствия эффективных предикатов нарушения симметрии. Целью настоящего диссертационного исследования являлось повышение эффективности точных методов генерации детерминированных конечных автоматов по заданным примерам поведения посредством сокращения пространства поиска при решении задачи выполнимости. Таким образом, согласно результатам экспериментальных исследований, цель можно считать успешно достигнутой.

Благодарности. Автор выражает благодарность своему научному руководителю, кандидату технических наук, В. И. Ульянцеву за неоценимую помощь в исследовательской деятельности и в написании настоящей работы, профессору, доктору технических наук, А. А. Шалыто за наставничество, коллегам по международной лаборатории «Компьютерные технологии» кандидату технических наук Д. С. Чивилихину за многочисленные консультации по вопросам исследовательской деятельности и за помощь с переводом текста реферата на английский язык, К. И. Чухареву и Д. М. Суворову за помощь в подготовке иллюстративного материала настоящей диссертации, А. И. Бугровскому и Т. Р. Галимжанову за помощь в документообороте во время процедуры подготовки защиты настоящей диссертационной работы, иностранным коллегам кандидату физико-математических наук А. И. Игнатьеву и доктору Ж. Маркешу-Сильве за организацию стажировки и активное в ней участие, в рамках которой проводилась часть исследовательской деятельности автора.

Также автор выражает благодарность своим родителям Тимуру и Валентине за привитую любовь к знаниям и математике, Лазаревой Екатерине за то, что всегда была рядом, своим друзьям Александру Б., Александру С., Анастасии, Ва-

лентину, Евгению, Марине, Матвею, Наталье, Талгату и Татьяне за оказанную моральную поддержку.

Список литературы

1. *Поликарпова*, *Н. И.*, *Шалыто*, *А. А.* Автоматное программирование. — СПБ: Питер, 2010. — 176 с.
2. *Patil*, *S.*, *Dubinin*, *V.*, *Vyatkin*, *V.* Formal Modelling and Verification of IEC61499 Function Blocks with Abstract State Machines and SMV — Execution Semantics // SETTA. Vol. 9409. — Springer, 2015. — P. 300–315. — (Lecture Notes in Computer Science).
3. *Jongmans*, *S.-S. T. Q.*, *Halle*, *S.*, *Arbab*, *F.* Automata-Based Optimization of Interaction Protocols for Scalable Multicore Platforms // COORDINATION. Vol. 8459. — Springer, 2014. — P. 65–82. — (Lecture Notes in Computer Science).
4. *Heule*, *M.*, *Verwer*, *S.* Software model synthesis using satisfiability solvers // Empirical Software Engineering. — 2013. — Vol. 18, no. 4. — P. 825–856.
5. *Wagner*, *F.*, *Schmuki*, *R.*, *Wagner*, *T.*, *Wolstenholme*, *P.* Modeling Software with Finite State Machines: A Practical Approach. — CRC Press, 2006. — 392 p.
6. *Clarke Jr*, *E. M.*, *Grumberg*, *O.*, *Kroening*, *D.*, *Peled*, *D.*, *Veith*, *H.* Model checking. — Second edition. — MIT press, 2018. — 424 p. — (Cyber Physical Systems Series).
7. *Chivilikhin*, *D.*, *Buzhinsky*, *I.*, *Ulyantsev*, *V.*, *Stankevich*, *A.*, *Shalyto*, *A.*, *Vyatkin*, *V.* Counterexample-guided inference of controller logic from execution traces and temporal formulas // ETFA. — IEEE, 2018. — P. 91–98.
8. *Buzhinsky*, *I.*, *Vyatkin*, *V.* Modular plant model synthesis from behavior traces and temporal properties // ETFA. — IEEE, 2017. — P. 1–7.
9. *Buzhinsky*, *I.*, *Vyatkin*, *V.* Automatic Inference of Finite-State Plant Models From Traces and Temporal Properties // IEEE Transactions on Industrial Informatics. — 2017. — Vol. 13, no. 4. — P. 1521–1530.

10. *Cook*, *J. E.*, *Wolf*, *A. L.* Discovering Models of Software Processes from Event-Based Data // ACM Transactions on Software Engineering Methodology. — 1998. — Vol. 7, no. 3. — P. 215–249.

11. *Bertolino*, *A.*, *Inverardi*, *P.*, *Pelliccione*, *P.*, *Tivoli*, *M.* Automatic synthesis of behavior protocols for composable web-services // ESEC/SIGSOFT FSE. — ACM, 2009. — P. 141–150.

12. *Sivakorn*, *S.*, *Argyros*, *G.*, *Pei*, *K.*, *Keromytis*, *A. D.*, *Jana*, *S.* HVLearn: Automated Black-Box Analysis of Hostname Verification in SSL/TLS Implementations // IEEE Symposium on Security and Privacy. — IEEE Computer Society, 2017. — P. 521–538.

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

14. *Davis*, *M.*, *Logemann*, *G.*, *Loveland*, *D. W.* A machine program for theorem-proving // Communications of the ACM. — 1962. — Vol. 5, no. 7. — P. 394–397.

15. *Marques-Silva*, *J. P.*, *Sakallah*, *K. A.* GRASP — a new search algorithm for satisfiability // ICCAD. — IEEE, 1996. — P. 220–227.

16. The International SAT Competition Web Page. — URL: http : / / www . satcompetition.org/ (дата обр. 15.10.2020).

17. SAT Competition 2020. — URL: https://satcompetition.github.io/2020 (дата обр. 15.10.2020).

18. *Heule*, *M.*, *Verwer*, *S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings. Vol. 6339. — Springer, 2010. — P. 66–79. — (Lecture Notes in Computer Science).

19. *Ульянцев*, *В. И.* Генерация конечных автоматов с использованием программных средств решения задач выполнимости и удовлетворения ограничений : дис. … канд. техн. наук / Ульянцев Владимир Игоревич. — НИУ ИТМО, 2015.

20. *Gold*, *E. M.* Language Identification in the Limit // Information and Control. — 1967. — Vol. 10, no. 5. — P. 447–474.

21. *Трахтенброт*, *Б. А.*, *Барздинь*, *Я. М.* Конечные автоматы (поведение и синтез). — Наука. Гл. ред. физ.-мат. лит., 1970. — 400 с. — (Математическая логика и основания математики).

22. *Lang*, *K. J.* Random DFA's Can Be Approximately Learned from Sparse Uniform Examples // Proceedings of the Fifth ACM Workshop on Computational Learning Theory. — ACM, 1992. — P. 45–52.

23. *Oncina*, *J.*, *García*, *P.* Inferring Regular Languages in Polynomial Update Time // Pattern Recognition and Image Analysis. Vol. 1. — 1992. — P. 49–61. — (Series in Machine Perception and Artificial Intelligence).

24. *Lang*, *K. J.*, *Pearlmutter*, *B. A.*, *Price*, *R. A.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm // ICGI. Vol. 1433. — Springer, 1998. — P. 1–12. — (Lecture Notes in Computer Science).

25. *Lang*, *K. J.* Faster algorithms for finding minimal consistent DFAs : tech. rep. / NEC Research Institute. — 1999.

26. *Cicchello*, *O.*, *Kremer*, *S. C.* Beyond EDSM // ICGI. Vol. 2484. — Springer, 2002. — P. 37–48. — (Lecture Notes in Computer Science).

27. *Lucas*, *S. M.*, *Reynolds*, *T. J.* Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 2005. — Vol. 27, no. 7. — P. 1063–1074.

28. *Gómez*, *J.* An Incremental-Evolutionary Approach for Learning Deterministic Finite Automata // IEEE Congress on Evolutionary Computation. — IEEE, 2006. — P. 362–369.

29. *Chivilikhin*, *D.*, *Ulyantsev*, *V.* Learning Finite-State Machines with Ant Colony Optimization // Swarm Intelligence. Vol. 7461. — Springer Berlin / Heidelberg, 2012. — P. 268–275. — (Lecture Notes in Computer Science).

30. *Coste*, *F.*, *Nicolas*, *J.* Regular Inference as a graph coloring problem // Workshop on Grammatical Inference, Automata Induction, and Language Acquisition (ICML'97). — 1997.

31. *Ulyantsev*, *V.*, *Zakirzyanov*, *I.*, *Shalyto*, *A.* BFS-Based Symmetry Breaking Predicates for DFA Identification // Language and Automata Theory and Applications — 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings. Vol. 8977. — Springer, 2015. — P. 611–622. — (Lecture Notes in Computer Science).

32. *Clarke*, *E. M.*, *Grumberg*, *O.*, *Jha*, *S.*, *Lu*, *Y.*, *Veith*, *H.* Counterexample-Guided Abstraction Refinement // CAV. Vol. 1855. — Springer, 2000. — P. 154–169. — (Lecture Notes in Computer Science).

33. *Мандрыкин*, *М.*, *Мутилин*, *В.*, *Хорошилов*, *А.* Введение в метод CEGAR — уточнение абстракции по контрпримерам // Труды Института системного программирования РАН. — 2013. — Т. 24.

34. *Chivilikhin*, *D.*, *Patil*, *S.*, *Chukharev*, *K.*, *Cordonnier*, *A.*, *Vyatkin*, *V.* Automatic State Machine Reconstruction From Legacy Programmable Logic Controller Using Data Collection and SAT Solver // IEEE Transactions on Industrial Informatics. — 2020. — Vol. 16, no. 12. — P. 7821–7831.

35. *Post*, *E. L.* The Two-Valued Iterative Systems of Mathematical Logic. Vol. 5. — Princeton, NJ : Princeton University Press, 1942. — 122 p. — (Annals of Mathematics Studies).

36. Handbook of Satisfiability. Vol. 185 / ed. by A. Biere, M. Heule, H. van Maaren, T. Walsh. — IOS Press, 2009. — 980 p. — (Frontiers in Artificial Intelligence and Applications).

37. *Ladner*, *R. E.* On the Structure of Polynomial Time Reducibility // Journal of the ACM. — 1975. — Т. 22, № 1. — С. 155–171.

38. *Cook*, *S. A.* The Complexity of Theorem-Proving Procedures // Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA / под ред. M. A. Harrison, R. B. Banerji, J. D. Ullman. — ACM, 1971. — С. 151–158.

39. *Левин*, *Л. А.* Универсальные задачи перебора // Проблемы передачи информации. — 1973. — Т. 9, вып. 3. — С. 115–116.

40. *Carlson*, *J. A.*, *Jaffe*, *A.*, *Wiles*, *A.* The millennium prize problems. — American Mathematical Society, 2006. — 165 p.

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

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

43. *Biere*, *A.* Lingeling Essentials, A Tutorial on Design and Implementation Aspects of the the SAT Solver Lingeling // POSSAT. Vol. 27. — EasyChair, 2014. — P. 88. — (EPiC Series in Computing).

44. *Biere*, *A.* Cadical, Lingeling, Plingeling, Treengeling and YalSAT entering the sat competition 2018 // SAT competition. — 2017. — С. 1.

45. *Soos*, *M.*, *Nohl*, *K.*, *Castelluccia*, *C.* Extending SAT Solvers to Cryptographic Problems // Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Vol. 5584 / ed. by O. Kullmann. — Springer, 2009. — P. 244–257. — (Lecture Notes in Computer Science).

46. *Fleury*, *A. B. K. F. M.*, *Heisinger*, *M.* CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020 // SAT competition. — 2020. — P. 50.

47. *Ignatiev*, *A.*, *Morgado*, *A.*, *Marques-Silva*, *J.* PySAT: A Python Toolkit for Prototyping with SAT Oracles // SAT. — 2018. — P. 428–437.

48. *Selman*, *B.*, *Levesque*, *H. J.*, *Mitchell*, *D. G.* A New Method for Solving Hard Satisfiability Problems // AAAI. — AAAI Press / The MIT Press, 1992. — P. 440–446.

49. *Selman*, *B.*, *Kautz*, *H. A.*, *Cohen*, *B.* Local search strategies for satisfiability testing // Cliques, Coloring, and Satisfiability. Vol. 26. — DIMACS/AMS, 1993. — P. 521–531. — (DIMACS Series in Discrete Mathematics and Theoretical Computer Science).

50. *Schöning*, *U.* A Probabilistic Algorithm for k-SAT and Constraint Satisfaction Problems // FOCS. — IEEE Computer Society, 1999. — P. 410–414.

51. *Hopcroft*, *J. E.*, *Motwani*, *R.*, *Ullman*, *J. D.* Introduction to automata theory, languages, and computation, 2nd Edition. — Addison-Wesley-Longman, 2001. — 560 p. — (Addison-Wesley series in computer science).

52. *Mogensen*, *T. Æ.* Lexical Analysis. — Springer, 2011. — 204 p. — (Undergraduate Topics in Computer Science).

53. *Walsh*, *T.* Symmetry Breaking Constraints: Recent Results // Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada / ed. by J. Hoffmann, B. Selman. — AAAI Press, 2012.

54. *Bugalho*, *M. M. F.*, *Oliveira*, *A. L.* Inference of regular languages using state merging algorithms with search // Pattern Recognition. — 2005. — Vol. 38, no. 9. — P. 1457–1467.

55. *Higuera*, *C. de la*. A bibliographical study of grammatical inference // Pattern Recognition. — 2005. — Vol. 38, no. 9. — P. 1332–1348.

56. *Angluin*, *D.* Learning Regular Sets from Queries and Counterexamples // Information and Computation. — 1987. — Vol. 75, no. 2. — P. 87–106.

57. *Steffen*, *B.*, *Howar*, *F.*, *Merten*, *M.* Introduction to Active Automata Learning from a Practical Perspective // SFM. Vol. 6659. — Springer, 2011. — P. 256–296. — (Lecture Notes in Computer Science).

58. *Neider*, *D.* Applications of automata learning in verification and synthesis : PhD thesis / Neider Daniel. — RWTH Aachen University, 2014.

59. *Banich*, *M. T.*, *Caccamise*, *D.* Generalization of knowledge: Multidisciplinary perspectives. — Psychology Press, 2010. — 380 p.

60. *Егоров*, *К. В.* Генерация управляющих автоматов на основе генетического программирования и верификации : дис. … канд. техн. наук / Егоров Кирилл Викторович. — НИУ ИТМО, 2013.

61. *Чивилихин*, *Д. С.* Генерация конечных автоматов на основе муравьиных алгоритмов : дис. … канд. техн. наук / Чивилихин Даниил Сергеевич. — НИУ ИТМО, 2015.

62. *Dorigo*, *M.*, *Birattari*, *M.* Ant Colony Optimization // Encyclopedia of Machine Learning and Data Mining. — Springer, 2017. — P. 56–59.

63. *Biere*, *A.*, *Cimatti*, *A.*, *Clarke*, *E. M.*, *Zhu*, *Y.* Symbolic Model Checking without BDDs // TACAS. Vol. 1579. — Springer, 1999. — P. 193–207. — (Lecture Notes in Computer Science).

64. *Bouquet*, *P.*, *Magnini*, *B.*, *Serafini*, *L.*, *Zanobini*, *S.* A SAT-Based Algorithm for Context Matching // CONTEXT. Vol. 2680. — Springer, 2003. — P. 66–79. — (Lecture Notes in Computer Science).

65. *Xie*, *Y.*, *Aiken*, *A.* Saturn: A SAT-Based Tool for Bug Detection // CAV. Vol. 3576. — Springer, 2005. — P. 139–143. — (Lecture Notes in Computer Science).

66. *Skvortsov*, *E.*, *Tipikin*, *E.* Experimental Study of the Shortest Reset Word of Random Automata // CIAA. Vol. 6807. — Springer, 2011. — P. 290–298. — (Lecture Notes in Computer Science).

67. *Métivier*, *J.-P.*, *Boizumault*, *P.*, *Crémilleux*, *B.*, *Khiari*, *M.*, *Loudni*, *S.* Constrained Clustering Using SAT // IDA. Vol. 7619. — Springer, 2012. — P. 207–218. — (Lecture Notes in Computer Science).

68. *Narodytska*, *N.*, *Ignatiev*, *A.*, *Pereira*, *F.*, *Marques-Silva*, *J.* Learning Optimal Decision Trees with SAT // IJCAI. — ijcai.org, 2018. — P. 1362–1368.

69. *Walsh*, *T.* SAT v CSP // CP. Vol. 1894. — Springer, 2000. — P. 441–456. — (Lecture Notes in Computer Science).

70. *Luce*, *R. D.*, *Perry*, *A. D.* A method of matrix analysis of group structure // Psychometrika. — 1949. — Vol. 14, no. 2. — P. 95–116.

71. *Karp*, *R. M.* Reducibility Among Combinatorial Problems // Complexity of Computer Computations. — Plenum Press, New York, 1972. — P. 85–103. — (The IBM Research Symposia Series).

72. *Cormen*, *T. H.*, *Leiserson*, *C. E.*, *Rivest*, *R. L.*, *Stein*, *C.* Introduction to Algorithms, Second Edition //. — The MIT Press, McGraw-Hill Book Company, 2001. — Chap. 22.2: Breadth-first search. P. 540–549.

73. *Zhivich*, *M.*, *Cunningham*, *R. K.* The Real Cost of Software Errors // IEEE Security & Privacy. — 2009. — Vol. 7, no. 2. — P. 87–90.

74. *Turing*, *A. M.* On computable numbers, with an application to the Entscheidungsproblem // Proceedings of the London Mathematical Society. S2–42. Issue 1. — 1936. — P. 230–265.

75. *Cormen*, *T. H.*, *Leiserson*, *C. E.*, *Rivest*, *R. L.*, *Stein*, *C.* Introduction to Algorithms, Second Edition //. — The MIT Press, McGraw-Hill Book Company, 2001. — Chap. 22.3: Depth-first search. P. 540–549.

76. *Zakirzyanov*, *I.* DFA-Inductor-py. — URL: https : / / github . com / ctlab / DFA - Inductor-py (дата обр. 02.10.2020).

77. GraphViz — Graph Visualization Software. — URL: https://graphviz.org/ (visited on 10/02/2020).

78. *Zakirzyanov*, *I.*, *Shalyto*, *A.*, *Ulyantsev*, *V.* Finding All Minimum-Size DFA Consistent with Given Examples: SAT-Based Approach // Software Engineering and Formal Methods — SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers. Vol. 10729. — Springer, 2017. — P. 117–131. — (Lecture Notes in Computer Science).

79. *Marques-Silva*, *J. P.*, *Lynce*, *I.*, *Malik*, *S.* Conflict-Driven Clause Learning SAT Solvers // Handbook of Satisfiability. Vol. 185 / ed. by A. Biere, M. Heule, H. van Maaren, T. Walsh. — IOS Press, 2009. — P. 131–153. — (Frontiers in Artificial Intelligence and Applications).

80. *Zakirzyanov*, *I.*, *Morgado*, *A.*, *Ignatiev*, *A.*, *Ulyantsev*, *V.*, *Marques-Silva*, *J.* Efficient Symmetry Breaking for SAT-Based Minimum DFA Inference // Language and Automata Theory and Applications — 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings. Vol. 11417. — Springer, 2019. — P. 159–173. — (Lecture Notes in Computer Science).

81. *Eén*, *N.*, *Sörensson*, *N.* An Extensible SAT-solver // SAT. Vol. 2919. — Springer, 2003. — P. 502–518. — (Lecture Notes in Computer Science).

82. *Закирзянов*, *И. Т.* Построение детерминированных конечных автоматов по примерам поведения с использованием подхода уточнения абстракции по контрпримерам // Научно-технический вестник информационных технологий, механики и оптики. — 2020. — Т. 20, № 3. — С. 394–401.

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

Приложение А Публикации

1. *Ulyantsev*, *V.*, *Zakirzyanov*, *I.*, *Shalyto*, *A.* BFS-Based Symmetry Breaking Predicates for DFA Identification // Language and Automata Theory and Applications — 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings. Vol. 8977. — Springer, 2015. — P. 611–622. — (Lecture Notes in Computer Science).

2. *Zakirzyanov*, *I.*, *Shalyto*, *A.*, *Ulyantsev*, *V.* Finding All Minimum-Size DFA Consistent with Given Examples: SAT-Based Approach // Software Engineering and Formal Methods — SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers. Vol. 10729. — Springer, 2017. — P. 117–131. — (Lecture Notes in Computer Science).

3. *Zakirzyanov*, *I.*, *Morgado*, *A.*, *Ignatiev*, *A.*, *Ulyantsev*, *V.*, *Marques-Silva*, *J.* Efficient Symmetry Breaking for SAT-Based Minimum DFA Inference // Language and Automata Theory and Applications — 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings. Vol. 11417. — Springer, 2019. — P. 159–173. — (Lecture Notes in Computer Science).

4. *Ovsiannikova*, *P.*, *Chivilikhin*, *D.*, *Ulyantsev*, *V.*, *Stankevich*, *A.*, *Zakirzyanov*, *I.*, *Vyatkin*, *V.*, *Shalyto*, *A.* Active Learning of Formal Plant Models For Cyber-Physical Systems // 16th IEEE International Conference on Industrial Informatics, INDIN 2018, Porto, Portugal, July 18-20, 2018. — IEEE, 2018. — P. 719–724.

5. *Закирзянов*, *И. Т.* Построение детерминированных конечных автоматов по примерам поведения с использованием подхода уточнения абстракции по контрпримерам // Научно-технический вестник информационных технологий, механики и оптики. — 2020. — Т. 20, № 3. — С. 394–401.

BFS-Based Symmetry Breaking Predicates for DFA Identification

Vladimir Ulyantsev(✉), Ilya Zakirzyanov, and Anatoly Shalyto

ITMO University, Saint-Petersburg, Russia
{ulyantsev,zakirzyanov}@rain.ifmo.ru, shalyto@mail.ifmo.ru

Abstract. It was shown before that the NP-hard problem of deterministic finite automata (DFA) identification can be translated to Boolean satisfiability (SAT). Modern SAT-solvers can efficiently tackle hard DFA identification instances. We present a technique to reduce SAT search space by enforcing an enumeration of DFA states in breadth-first search (BFS) order. We propose symmetry breaking predicates, which can be added to Boolean formulae representing various DFA identification problems. We show how to apply this technique to DFA identification from both noiseless and noisy data. The main advantage of the proposed approach is that it allows to exactly determine the existence or non-existence of a solution of the noisy DFA identification problem.

Keywords: Grammatical inference · Boolean satisfiability · Learning automata · Symmetry breaking techniques

1 Introduction

Deterministic finite automata (DFA) are models that recognize regular languages [1], therefore the problem of DFA identification (induction, learning) is one of the best studied [2] in grammatical inference. The identification problem consists of finding a DFA with minimal number of states that is consistent with a given set of strings with language attribution labels. This means that such a DFA rejects the negative example strings and accepts the positive example strings. It was shown in [3] that finding a DFA with a given upper bound on its size (number of states) is an NP-complete problem. Besides, in [4] it was shown that this problem cannot be approximated within any polynomial.

Despite this theoretical difficulty, several efficient DFA identification algorithms exist [2]. The most common approach is the evidence driven state-merging (EDSM) algorithm [5]. The key idea of this algorithm is to first construct an augmented prefix tree acceptor (APTA), a tree-shaped automaton, from the given labeled strings, and then to iteratively apply a state-merging procedure until no valid merges are left. Thus EDSM is a polynomial-time greedy method that tries to find a good local optimum. EDSM participated in the Abbadingo DFA learning competition [5] and won it (in a tie). To improve the EDSM algorithm several specialized search procedures were proposed, see, e.g., [6,7]. One of the

A.-H. Dediu et al. (Eds.): LATA 2015, LNCS 8977, pp. 611–622, 2015.
DOI: 10.1007/978-3-319-15579-1_48

most successful approaches is the EDSM algorithm in the red-blue framework [5], also called the Blue-fringe algorithm.

The second approach for DFA learning is based on evolutionary computation; early work includes [8,9]. Later the authors of [10] presented an effective scheme for evolving DFA with a multi-start random hill climber, which was used to optimize the transition matrix of the identified DFA. A so-called smart state labeling scheme was applied to choose the state labels optimally, given the transition matrix and the training set. Authors emphasized that smart selection of state labels gives the evolutionary method a significant boost which allowed it to compete with the EDSM. Authors find that the proposed evolutionary algorithm (EA) outperforms the EDSM algorithm on small target DFAs when the training set is sparse. For larger DFAs with 32 states, the hill climber fails and EDSM then clearly outperforms it.

The challenge of the GECCO 2004 Noisy DFA competition [11] was to learn the target DFA when 10 percent of the given training string labels had been randomly flipped. In [12] Lucas and Reynolds show that within limited time EA with smart state labeling is able to identify the target DFA even at such high noise level. Authors compared their algorithm with the results of the GECCO competition and found that EA clearly outperformed all the entries. Thereby it is the state-of-the-art technique for learning DFA from noisy training data.

In several cases the best solution for noiseless DFA identification is the *translation-to-SAT* technique [13], which was altered to suit the *StaMInA* (State Machine Inference Approaches) competition [14] and ultimately won. The main idea of that algorithm is to translate the DFA identification problem to Boolean satisfiability (SAT). Thus we are able to use highly optimized modern DPLL-style SAT solving techniques [15]. The translation-to-SAT approach was also used to efficiently tackle problems such as bounded model checking [16], solving SQL constraints by incremental translation [17], analysis of JML-annotated Java sequential programs [18], extended finite-state machine induction [19].

Many optimization problems exhibit symmetries – groups of solutions which can be obtained from each other via some simple transformations. To speed up the solution search process we can reduce the problem search space by performing *symmetry breaking*. In DFA identification problems the most straightforward symmetries are groups of isomorphic automata. The idea of avoiding isomorphic DFAs by fixing state numbers in breadth-first search (BFS) order was used in the state-merging approach [20] (function `NatOrder`) and in the genetic algorithm from [21] (*Move To Front* reorganization). Besides, in [13] symmetry breaking was performed by fixing some colors of the APTA vertices from a clique provided by a greedy *max-clique* algorithm was applied in a preprocessing step of translation-to-SAT technique.

In this paper we propose new symmetry breaking predicates [15] which can be added to Boolean formulae representing various DFA identification problems. These predicates enforce DFA states to be enumerated in BFS order. Proposed predicates cannot be applied with the max-clique technique [13] at the same time, but our approach is more flexible. To show the flexibility of the approach, we

draw our attention to the case of noisy DFA identfication. Therefore we propose a modification of the noiseless translation-to-SAT for the noisy case (Section 3). We show that the previously proposed max-clique technique is not applicable in this case while our BFS-based approach is. The main advantage of our approach is that we can determine existence or non-existence of a solution in this case. Experiments showed that using BFS-based symmetry breaking predicates can significantly reduce the time of algorithm execution. Also we show that our strategy outperforms the current state-of-the-art EA from [12] if the number of the target DFA states, noise level and number of strings are small.

2 Encoding DFA Identfication into SAT

The goal of DFA identification is to find a smallest DFA A such that every string from S_+, a set of positive examples, is accepted by A, and every string from S_-, a set of negative examples, is rejected. The size of A is defined as the number of states C it contains. The alphabet $\Sigma = \{l_1, \ldots, l_L\}$ of the sought DFA A is the set of all L symbols from S_+ and S_-. The example of the smallest DFA for $S_+ = \{ab, b, ba, bbb\}$ and $S_- = \{abbb, baba\}$ is shown in Fig. 1. In this work we assume that DFA states are numbered from 1 to C and the start state has number 1.

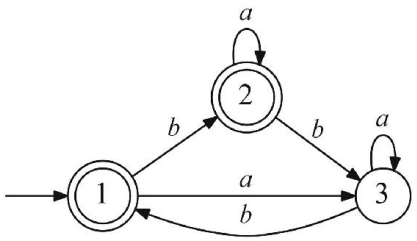
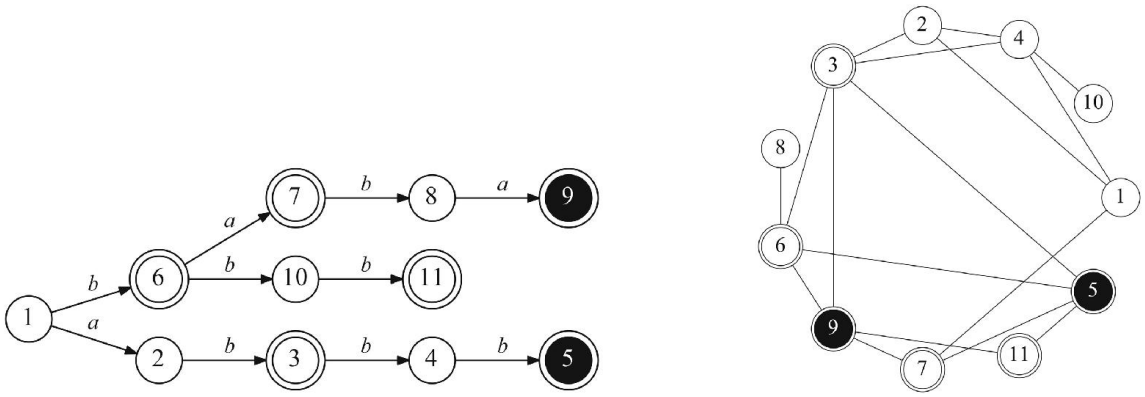


Fig. 1. An example of a DFA

In [13] Heule and Verwer proposed a compact translation of DFA identification problem into SAT. Here we briefly review the proposed technique, since our symmetry breaking predicates supplement it. The first step of both state-merging and translation-to-SAT techniques is augmented prefix tree acceptor (APTA) construction from the given examples S_+ and S_-. APTA is a tree-shaped automaton such that paths corresponding to two strings reach the same state v if and only if these strings share the same prefix in which the last symbol corresponds to v. We denote by V the set of all APTA states; by v_r – the APTA root; by V_+ – the set of accepting states; and by V_- – the set of rejecting states. Moreover, for state v (except v_r) we denote its incoming symbol as $l(v)$ and its parent as $p(v)$. The APTA for S_+ and S_- mentioned above is shown in Fig. 2a.

The second step of the technique proposed in [13] is the construction of the *consistency graph* (CG) for the obtained APTA. The set of nodes of the CG is identical to the set of APTA states. Two CG nodes v and w are connected



(a) An example of an APTA for $S_+ = \{ab, b, ba, bbb\}$ and $S_- = \{abbb, baba\}$

(b) The consistency graph for APTA from Fig. 2a

Fig. 2. An example of APTA and its consistency graph

with an edge (and called inconsistent) if merging v and w in APTA results in an inconsistency: an accepting state is merged with a rejecting state. Let E denote the set of CG edges. The CG for APTA of Fig. 2a is shown in Fig. 2b.

The key part of the algorithm is translating the DFA identification problem into a Boolean folmula in conjunctive normal form (CNF) and using a SAT solver to find a satisfying assignment. For a given set of examples and fixed DFA size C the solver returns a satisfying assignment (that defines a DFA with C states that is compliant with S_+ and S_-) or a message that it does not exist. The main idea of this translation is to use a distinct color for every state of the identified DFA and to find a consistent mapping of APTA states to colors. Three types of variables were used in the proposed compact translation:

1. *color* variables $x_{v,i} \equiv 1$ ($v \in V$; $1 \leqslant i \leqslant C$) iff APTA state v has color i;
2. *parent relation* variables $y_{l,i,j} \equiv 1$ ($l \in \Sigma$; $1 \leqslant i, j \leqslant C$) iff DFA transition with symbol l from state i ends in state j;
3. *accepting color* variables $z_i \equiv 1$ ($1 \leqslant i \leqslant C$) iff DFA state i is accepting.

Direct encoding, described in [13], uses only variables $x_{v,i}$; variables $y_{l,i,j}$ and z_i are auxiliary and are used in compact encoding predicates, which are described below.

The compact translation proposed in [13] uses nine types of clauses:

1. $x_{v,i} \Rightarrow z_i$ ($v \in V_+$; $1 \leqslant i \leqslant C$) – definitions of z_i values for accepting states ($\neg x_{v,i} \vee z_i$);
2. $x_{v,i} \Rightarrow \neg z_i$ ($v \in V_-$; $1 \leqslant i \leqslant C$) – definitions of z_i values for rejecting states ($\neg x_{v,i} \vee \neg z_i$);
3. $x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,C}$ ($v \in V$) – each state v has at least one color;
4. $x_{p(v),i} \wedge x_{v,j} \Rightarrow y_{l(v),i,j}$ ($v \in V \setminus \{v_r\}$; $1 \leqslant i, j \leqslant C$) – a DFA transition is set when a state and its parent are colored ($y_{l(v),i,j} \vee \neg x_{p(v),i} \vee \neg x_{v,j}$);
5. $y_{l,i,j} \Rightarrow \neg y_{l,i,k}$ ($l \in \Sigma$; $1 \leqslant i, j, k \leqslant C$; $j < k$) – each DFA transition can target at most one state ($\neg y_{l,i,j} \vee \neg y_{l,i,k}$);

6. $\neg x_{v,i} \vee \neg x_{v,j}$ $(v \in V;\ 1 \leqslant i < j \leqslant C)$ – each state has at most one color;
7. $y_{l,i,1} \vee y_{l,i,2} \vee \ldots \vee y_{l,i,C}$ $(l \in \Sigma;\ 1 \leqslant i \leqslant C)$ – each DFA transition must target at least one state;
8. $y_{l(v),i,j} \wedge x_{p(v),i} \Rightarrow x_{v,j}$ $(v \in V \setminus \{v_r\};\ 1 \leqslant i, j \leqslant C)$ – state color is set when DFA transition and parent color are set $(\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j})$;
9. $x_{v,i} \Rightarrow \neg x_{w,i}$ $((v, w) \in E;\ 1 \leqslant i \leqslant C)$ – the colors of two states connected with an edge in the consistency graph must be different $(\neg x_{v,i} \vee \neg x_{w,i})$.

Thus, the constructed formula consists of $\mathcal{O}(C^2|V|)$ clauses and, if the SAT solver finds a solution, we can identify the DFA.

To find a minimal DFA, authors use iterative SAT solving. Initial DFA size C is equal to the size of a *large clique* found in the CG. To find that clique, a greedy algorithm proposed in [13] can be applied. Then the minimal DFA is found by iterating over the DFA size C until the formula is satisfied.

The found clique was also used to perform symmetry breaking: in any valid coloring of a graph, all states in a clique must have a different color. Thus, we can fix the state colors in the clique in a preprocessing step. Later we will see that the max-clique symmetry breaking is not compatible with the one proposed in this paper.

To significantly reduce the SAT search space, the authors applied several EDSM steps before translation to SAT. Since EDSM cannot guarantee the minimality of solution, we will omit the consideration of this step in our paper.

3 Learning DFA from Noisy Samples

The translation described in the previous section deals with exact DFA identification. In this section we show how to modify the translation in order to apply it to noisy examples. We assume that not more than K attribution labels of the given training strings were randomly flipped. Solving this problem was the goal of the GECCO 2004 Noisy DFA competition [11] (with K equal to 10 percent of the number of the given training strings). An EA with smart state labeling was later proposed in [12], and since that time it is, to the best of our knowledge, the state-of-the-art technique for learning DFA from noisy training data.

In noisy case we cannot use APTA node consistency: we cannot determine whether an accepting state is merged with a rejecting state because correct string labels are unknown. Thus we cannot use CG and the max-clique symmetry breaking.

The idea of our modification is rather simple: for each labeled state of APTA we define a variable which states whether the label can be flipped. The number of flips is limited by K. Formally, for each $v \in V_{\pm} = V_+ \cup V_-$ we define f_v which is true if and only if the label of state v can (but does not have to) be incorrect (**f**lipped). Using these variables, we can modify the translation proposed in [13] to take into account mistakes in string labels. To do this, we change the z_i definition clauses (items 1 and 2 from list in Section 2): because of mistake possibility they hold in case f_v is false. Thus, new z_i value definitions are expressed in the following way: $\neg f_v \Rightarrow (x_{v,i} \Rightarrow z_i)$ for $v \in V_+$; $\neg f_v \Rightarrow (x_{v,i} \Rightarrow \neg z_i)$ for $v \in V_-$.

To limit the number of corrections to K we use an auxiliary array of K integer variables. This array stores the numbers of the APTA states for which labels can be flipped. Thus, f_v is true if and only if the array contains v. To avoid consideration of isomorphic permutations we enforce the array to be sorted in the increasing order.

To represent the auxiliary array as a Boolean formula we define variables $r_{i,v}$ for $1 \leqslant i \leqslant K$ and $v \in V_{\pm} = \{v_1, \ldots, v_W\}$. $r_{i,v}$ is true if and only if v is stored in the i-th position of the array. To connect variables f_v with $r_{i,v}$ we add so-called channeling constrains: $f_v \Leftrightarrow (r_{1,v} \vee \ldots \vee r_{K,v})$ for each $v \in V_{\pm}$.

We have to state that exactly one $r_{i,v}$ is true for each position i in the auxiliary array. To achieve that we use the order encoding method [22]. We add auxiliary **o**rder variables $o_{i,v}$ for $1 \leqslant i \leqslant K$ and $v \in V_{\pm} = \{v_1, \ldots, v_W\}$. We assume that $o_{i,v}$ for $v \in \{v_1, \ldots, v_j\}$ and $\neg o_{i,v}$ for $v \in \{v_{j+1}, \ldots, v_W\}$ for some j. This can be expressed by the following constraint: $o_{i,v_{j+1}} \Rightarrow o_{i,v_j}$ for $1 \leqslant j < W$. Now we define that $r_{i,v_j} \Leftrightarrow o_{i,v_j} \wedge \neg o_{i,v_{j+1}}$. Also we add clauses $o_{i,v_j} \Rightarrow o_{i+1,v_{j+1}}$ (for $1 \leqslant i < K$ and $1 \leqslant j < W$) to store corrections in increasing order.

The proposed constraints in CNF are listed in Table 1; there are $\mathcal{O}(C|V_{\pm}| + K|V_{\pm}|)$ clauses. Thus, to modify the translation for the noiseless case to deal with noise we can replace the z_i value definition and inconsistency clauses (items 1, 2 and 9 from list in Section 2) with the ones listed in Table 1.

Table 1. Clauses for noisy DFA identification

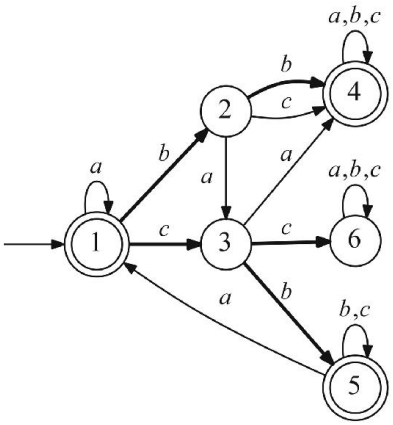
Clauses	CNF representation	Range
$\neg f_v \Rightarrow (x_{v,i} \Rightarrow z_i)$	$\neg x_{v,j} \vee z_j \vee f_v$	$1 \leqslant j \leqslant C$; $v \in V_+$
$\neg f_v \Rightarrow (x_{v,i} \Rightarrow \neg z_i)$	$\neg x_{v,j} \vee \neg z_j \vee f_v$	$1 \leqslant j \leqslant C$; $v \in V_-$
$f_v \Rightarrow (r_{1,v} \vee \ldots \vee r_{K,v})$	$\neg f_v \vee r_{1,v} \vee \ldots \vee r_{K,v}$	$v \in V_{\pm}$
$r_{i,v} \Rightarrow f_v$	$\neg r_{i,v} \vee f_v$	$1 \leqslant i \leqslant K$; $v \in V_{\pm}$
$r_{i,v_j} \Rightarrow o_{i,v_j}$	$\neg r_{i,v_j} \vee o_{i,v_j}$	$1 \leqslant i \leqslant K$; $1 \leqslant j \leqslant W$
$r_{i,v_j} \Rightarrow \neg o_{i,v_{j+1}}$	$\neg r_{i,v_j} \vee \neg o_{i,v_{j+1}}$	$1 \leqslant i \leqslant K$; $1 \leqslant j < W$
$o_{i,v_j} \wedge \neg o_{i,v_{j+1}} \Rightarrow r_{i,v_j}$	$\neg o_{i,v_j} \vee o_{i,b_{j+1}} \vee r_{i,v_j}$	$1 \leqslant i \leqslant K$; $1 \leqslant j < W$
$o_{K,v_W} \Rightarrow r_{K,v_W}$	$\neg o_{K,v_W} \vee r_{K,v_W}$	
$o_{i,v_{j+1}} \Rightarrow o_{i,v_j}$	$\neg o_{i,v_{j+1}} \vee o_{i,v_j}$	$1 \leqslant i \leqslant K$; $1 \leqslant j < W$
$o_{i,v_j} \Rightarrow o_{i+1,v_{j+1}}$	$\neg o_{i,v_j} \vee o_{i+1,v_{j+1}}$	$1 \leqslant i < K$; $1 \leqslant j < W$

4 Symmetry Breaking Predicates

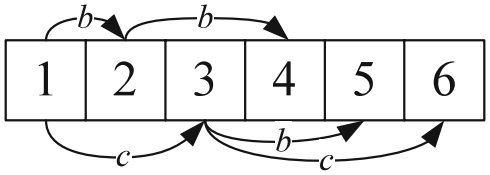
In this section we propose a way to fix automata state enumeration to avoid consideration of isomorphic DFAs during SAT solving. The main idea of our symmetry breaking is to enforce DFA states to be enumerated in breadth-first search (BFS) order. That idea was also used in function `NatOrder` in the state-merging approach described in [20] and the Move To Front reorganization algorithm used in the genetic algorithm [21].

BFS uses the *queue* data structure to store intermediate results as it traverses the graph. First we enqueue the initial DFA state (in this paper state number 1). While the queue is not empty we deque a state i and enqueue any direct child

states j that have not yet been discovered (enqueued before). Since our transitions are labeled with symbols from Σ, we enqueue child states in alphabetical order of symbols l on transitions $i \xrightarrow{l} j$. We call DFA *BFS-enumerated* if its states are enumerated in dequeuing (equals to enqueuing) order. An example of a BFS-enumerated DFA with six states shown in Fig. 3a (BFS-tree transitions that were used to enqueue states are marked bold); BFS enqueues are shown in Fig. 3b. The DFA shown in Fig. 1 is not BFS-enumerated – BFS first dequeues state 3 rather than state 2 (we consider $1 \xrightarrow{a} 3$ before $1 \xrightarrow{b} 2$).



(a) BFS-enumerated DFA with bolded BFS-tree edges



(b) BFS queue. Cells correspond to DFA states, transitions correspond to enqueues

Fig. 3. An example of BFS-numerated DFA and its BFS queue

We propose constraints that enforce DFA to be BFS-enumerated. We assume that translation of a given DFA identification problem to SAT deals with Boolean variables $y_{l,i,j}$ ($l \in \Sigma$; $1 \leqslant i, j \leqslant C$) to set the DFA transition function: $y_{l,i,j} \equiv 1$ iff transition with symbol l from state i ends in state j.

The main idea is to determine each state's parent in the BFS-tree and set constrains between states' parents. We store **p**arents in values $p_{j,i}$ (for each $1 \leqslant i < j \leqslant C$). $p_{j,i}$ is true if and only if state i is the parent of j in the BFS-tree. Each state except the initial one must have a parent with a smaller number, thus

$$\bigwedge_{2 \leqslant j \leqslant C} (p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}).$$

Moreover, in BFS-enumeration states' parents must be ordered. State j must be enqueued before the next state $j+1$, thus the next state's parent k cannot be less than current state's parent i (see Fig. 4):

$$\bigwedge_{1 \leqslant k < i < j < C} (p_{j,i} \Rightarrow \neg p_{j+1,k}).$$

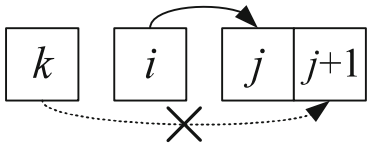


Fig. 4. Part of the queue illustrating the parent ordering predicates. Transitions show parent relations. The dotted transition is not allowed due to BFS-enumeration.

We set parents variables $p_{j,i}$ through $y_{l,i,j}$ using auxiliary variables $t_{i,j}$. In BFS-enumeration state j was enqueued while processing the state with minimal number i among states that have a transition to j:

$$\bigwedge_{1 \leqslant i < j \leqslant C} (p_{j,i} \Leftrightarrow t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j}),$$

where $t_{i,j} \equiv 1$ iff there is a **t**ransition between i and j; we define these auxiliary variables using $y_{l,i,j}$:

$$\bigwedge_{1 \leqslant i < j \leqslant C} (t_{i,j} \Leftrightarrow y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j}).$$

Now to enforce DFA to be BFS-enumerated we have to order children in alphabetical order of symbols on transitions. We consider two cases: alphabet Σ consists of two symbols $\{a, b\}$ and more than two symbols $\{l_1, \ldots, l_L\}$. In the case of two symbols only two states can have the same parent i and they are forced by ordering constraints to have consecutive numbers j and $j + 1$. In this case we force the transition that starts in state i labeled with symbol a to end in state j instead of $j + 1$:

$$\bigwedge_{1 \leqslant i < j < C} (p_{j,i} \wedge p_{j+1,i} \Rightarrow y_{a,i,j}).$$

In the second case we have to introduce a third type of variables in our symmetry breaking predicates. We store the alphabetically **m**inimal symbol on transitions between states: $m_{l,i,j}$ is true if and only if there is a transition $i \xrightarrow{l} j$ and there is no such transition with an alphabetically smaller symbol. We connect these variables with DFA transitions by adding the following channeling predicates:

$$\bigwedge_{1 \leqslant i < j \leqslant C} \bigwedge_{1 \leqslant n \leqslant L} (m_{l_n,i,j} \Leftrightarrow y_{l_n,i,j} \wedge \neg y_{l_{n-1},i,j} \wedge \ldots \wedge \neg y_{l_1,i,j}).$$

Now it remains to arrange consecutive states j and $j+1$ with the same parent i in the alphabetically order of minimal symbols on transitions between them and i (see Fig. 5):

$$\bigwedge_{1 \leqslant i < j < C} \bigwedge_{1 \leqslant k < n \leqslant L} (p_{j,i} \wedge p_{j+1,i} \wedge m_{l_n,i,j} \Rightarrow \neg m_{l_k,i,j+1}).$$

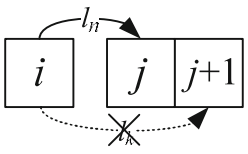


Fig. 5. Illustration of alphabetical ordering predicates. If i is the parent of j and $j+1$, l_n (l_k) is the alphabetically minimal symbol on transitions between i and j (i and $j+1$) then l_k cannot be alphabetically smaller than l_n

Thus we propose symmetry breaking predicates that are composed by listed constraints. Predicates (for three or more symbols case) translated into $\mathcal{O}(C^3 + C^2L^2)$ CNF clauses are listed in Table 2. Our implementation of proposed predicates and all algorithms can be found on the our labaratory github repository (https://github.com/ctlab/DFA-Inductor).

Table 2. BFS-based symmetry breaking clauses

Clauses	CNF representation	Range
$t_{i,j} \Rightarrow (y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j})$	$\neg t_{i,j} \vee y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j}$	$1 \leqslant i < j \leqslant C$
$y_{i,j,l} \Rightarrow t_{i,j}$	$\neg y_{l,i,j} \vee t_{i,j}$	$1 \leqslant i < j \leqslant C; l \in \Sigma$
$m_{l,i,j} \Rightarrow y_{l,i,j}$	$\neg m_{l,i,j} \vee y_{l,i,j}$	$1 \leqslant i < j \leqslant C; l \in \Sigma$
$m_{l_n,i,j} \Rightarrow \neg y_{l_k,i,j}$	$\neg m_{l_n,i,j} \vee \neg y_{l_k,i,j}$	$1 \leqslant i < j \leqslant C; 1 \leqslant k < n \leqslant L$
$(y_{l_n,i,j} \wedge \neg y_{l_{n-1},i,j} \wedge \ldots \wedge \neg y_{l_1,i,j}) \Rightarrow m_{l_n,i,j}$	$\neg y_{l_n,i,j} \vee y_{l_{n-1},i,j} \vee \ldots \vee y_{l_1,i,j} \vee m_{l_n,i,j}$	$1 \leqslant i < j \leqslant C; 1 \leqslant n \leqslant L$
$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$2 \leqslant j \leqslant C$
$p_{j,i} \Rightarrow t_{i,j}$	$\neg p_{j,i} \vee t_{i,j}$	$1 \leqslant i < j \leqslant C$
$p_{j,i} \Rightarrow \neg t_{k,j}$	$\neg p_{j,i} \vee \neg t_{k,j}$	$1 \leqslant k < i < j \leqslant C$
$(t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j}) \Rightarrow p_{j,i}$	$\neg t_{i,j} \vee t_{i-1,j} \vee \ldots \vee t_{1,j} \vee p_{j,i}$	$1 \leqslant i < j \leqslant C$
$p_{j,i} \Rightarrow \neg p_{j+1,k}$	$\neg p_{j,i} \vee \neg p_{j+1,k}$	$1 \leqslant k < i < j < C$
$(p_{j,i} \wedge p_{j+1,i} \wedge m_{l_n,i,j}) \Rightarrow \neg m_{l_k,i,j+1}$	$\neg p_{j,i} \vee \neg p_{j+1,i} \vee \neg m_{l_n,i,j} \vee \neg m_{l_k,i,j+1}$	$1 \leqslant i < j < C; 1 \leqslant k < n \leqslant L$

5 Experiments

All experiments were performed using a machine with an AMD Opteron 6378 2.4 GHz processor running on Ubuntu 14.04. All algorithms were implemented in Java, the *lingeling* SAT-solver was used. Our own algorithm was used for generating problem instances for all experiments based on randomly generated data sets. This algorithm builds a set of strings with the following parameters: size of DFA N which has to be generated, alphabet size A, number of strings S which have to be generated, noise level K (percent of attribution labels of generated strings which have to be randomly flipped).

In the exact case the max-clique method clearly outperforms BFS-based strategy.

For noisy DFA identification we used randomly generated instances. First we considered the case when the target DFA exists and the Boolean formula is satisfiable. We used following parameters: $N \in [5; 10]$, $A = 2$, $S \in \{10N, 25N, 50N\}$. We compared SAT approach without any symmetry breaking predicates, our

solution using BFS-based symmetry breaking predicates and the current state-of-the-art EA from [12]. Each experiment was repeated 100 times. The time limit was set to 1800 seconds. Initial experiments showed that EA clearly outperforms our method when $K > 4\%$. Therefore we set this parameter to $1\% - 4\%$. Results are listed in Table 3. We left only instances which were solved within the time limit. These results indicate that the BFS-based strategy finds the solution faster than the current state-of-the-art EA only when N is small (< 7), noise level is small ($1\% - 4\%$) and the number of strings is also small ($< 50N$). But BFS-based strategy finds the solution extremely faster than SAT approach without symmetry breaking strategy.

Table 3. Mean times of solving noisy DFA identification with count of strings in the each instance set to $10N$, $25N$ and $50N$ respectively

N	K	BFS, sec	SAT, sec	EA, sec
5	2	0.22	0.38	1.22
5	4	0.59	0.9	1.1
6	2	1.05	2.44	2.94
6	4	3.34	7.82	2.85
7	1	4.34	10.83	21.36
7	3	17.22	143.66	19.16
8	1	17.89	31.58	30.29
8	2	163.92	225.31	19.8

N	K	BFS, sec	SAT, sec	EA, sec
5	1	0.54	0.64	2.77
5	2	2.42	4.33	1.80
6	1	6.3	11.95	11.65
6	2	13.3	43.54	4.80
7	1	31.01	114.95	17.24
7	2	286.76	TL	13.11
8	1	239.46	404.32	21.73

N	K	BFS, sec	SAT, sec	EA, sec
5	1	4.2	7.59	6.07
5	2	12.87	22.36	3.05
6	1	20.76	52.5	20.39
6	2	107.94	309.22	11.28

The last experiment considered the case when the target DFA does not exist and the Boolean formula is unsatisfiable. Random dataset was also used here. We tried to find the target DFA using the following parameters: $N \in [5; 7]$, $A = 2$, $S = 50N$, $K \in [1; 2]$ percent. The input set of strings was generated from a $(N+1)$-sized DFA. It should be noted that the EA from [12] cannot determine that an automaton consistent with a given set of strings does not exist. On the other hand, all SAT-based methods are capable of that. Therefore we compared our implementation of compact SAT encoding without using symmetry breaking predicates and the same with BFS-based predicates. Each experiment was repeated 100 times and the time limit was set to 1800 seconds again. Results are listed in Table 4. It can be seen from the table that BFS-based strategy significantly reduces the mean time of determination that an automaton does not exist.

Table 4. Mean times and percent of passed solutions of solving noisy DFA identification when the target DFA does not exist

N	K	BFS, sec	WO, sec	passed BFS, %	passed WO, %
5	1	11.57	257.13	100	100
5	2	46.42	1296.71	100	30
6	1	110.05	TL	100	0
6	2	581.73	TL	100	0
7	1	995.27	TL	89	0
7	2	TL	TL	0	0

6 Conclusions and Future Work

We proposed symmetry breaking predicates which can be added to the Boolean formula representing various DFA identification problems. By adding the predicates we can reduce SAT search space through enforcing DFA states to be enumerated in breadth-first search (BFS) order.

We drew our attention to the case of noisy DFA identfication. We proposed a modification of the noiseless translation-to-SAT [13] for the noisy case. To achieve compact encoding for that case we used the order encoding method. We showed that the previously proposed max-clique technique for symmetry breaking is not applicable in the noisy case while our BFS-based approach is. We showed that the BFS-based strategy can be applied in the noisy case when an automaton which is consistent with a given set of strings does not exists. The current state-of-the-art EA from [12] cannot determine that. In experimental results, we showed that our approach with BFS-based symmetry breaking predicates clearly outperforms algorithm without any predicates. Also we showed that our strategy outperform EA if the number of the target DFA states is small, noise level is small and number of strings is small either.

We plan to translate noisy DFA identification to Max-SAT in order to limit the number of corrections without using an auxiliary array of integer variables. Also we plan to experiment with alternative integer encoding methods. In the future we would like to solve a problem of finding all solutions (instead of a single DFA) using our predicates.

Acknowledgements. Authors would like to thank Daniil Chivilikhin, Igor Buzhinsky and Andrey Filchenkov for useful comments. This work was financially supported by the Government of Russian Federation, Grant 074-U01, and also partially supported by RFBR, research project No. 14-07-31337 mol_a.

References

1. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (2006)
2. De La Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition **38**(9), 1332–1348 (2005)
3. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37**(3), 302–320 (1978)
4. Pitt, L., Warmuth, M.K.: The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of the ACM **40**(1), 95–142 (1993)
5. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 1–112. Springer, Heidelberg (1998)
6. Lang, K.J.: Faster Algorithms for Finding Minimal Consistent DFAs. Technical report (1999)
7. Bugalho, M., Oliveira, A.L.: Inference of regular languages using state merging algorithms with search. Pattern Recognition **38**(9), 1457–1467 (2005)

8. Dupont, P.: Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 236–2445. Springer, Heidelberg (1994)
9. Luke, S., Hamahashi, S., Kitano, H.: Genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1098–1105 (1999)
10. Lucas, S.M., Reynolds, T.J.: Learning DFA: evolution versus evidence driven state merging. In: The 2003 Congress on Evolutionary Computation, CEC 2003, vol. 1, pp. 351–358. IEEE (2003)
11. Lucas, S.: GECCO 2004 noisy DFA results. In: GECCO Proc. (2004)
12. Lucas, S.M., Reynolds, T.J.: Learning deterministic finite automata with a smart state labeling evolutionary algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(7), 1063–1074 (2005)
13. Heule, M.J.H., Verwer, S.: Exact DFA identification using SAT solvers. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 66–79. Springer, Heidelberg (2010)
14. Walkinshaw, N., Lambeau, B., Damas, C., Bogdanov, K., Dupont, P.: STAMINA: a competition to encourage the development and assessment of software model inference techniques. Empirical Software Engineering **18**(4), 791–824 (2013)
15. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS Press (2009)
16. Amla, N., Du, X., Kuehlmann, A., Kurshan, R.P., McMillan, K.L.: An analysis of SAT-based model checking techniques in an industrial environment. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 254–268. Springer, Heidelberg (2005)
17. Lohfert, R., Lu, J.J., Zhao, D.: Solving SQL constraints by incremental translation to SAT. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) IEA/AIE 2008. LNCS (LNAI), vol. 5027, pp. 669–676. Springer, Heidelberg (2008)
18. Galeotti, J.P., Rosner, N., Lopez Pombo, C.G., Frias, M.F.: TACO: Efficient SAT-Based Bounded Verification Using Symmetry Breaking and Tight Bounds. IEEE Transactions on Software Engineering **39**(9), 1283–1307 (2013)
19. Ulyantsev, V., Tsarev, F.: Extended finite-state machine induction using SAT-solver. In: Proc. of ICMLA 2011, vol. 2, pp. 346–349. IEEE (2011)
20. Lambeau, B., Damas, C., Dupont, P.E.: State-merging DFA induction algorithms with mandatory merge constraints. In: Clark, A., Coste, F., Miclet, L. (eds.) ICGI 2008. LNCS (LNAI), vol. 5278, pp. 139–153. Springer, Heidelberg (2008)
21. Chambers, L.D.: Practical handbook of genetic algorithms: complex coding systems, vol. 3. CRC Press (2010)
22. Barahona, P., Hölldobler, S., Nguyen, V.: Efficient SAT-encoding of linear csp constraints. In: 13th International Symposium on Artificial Intelligence and Mathematics-ISAIM, Fort Lauderdale, Florida, USA (2014)

Finding All Minimum-Size DFA Consistent with Given Examples: SAT-Based Approach

Ilya Zakirzyanov[1,2(✉)], Anatoly Shalyto[1], and Vladimir Ulyantsev[1]

[1] ITMO University, Saint Petersburg, Russia
{zakirzyanov,ulyantsev}@rain.ifmo.ru, shalyto@mail.ifmo.ru
[2] JetBrains Research, Saint Petersburg, Russia

Abstract. Deterministic finite automaton (DFA) is a fundamental concept in the theory of computation. The NP-hard DFA identification problem can be efficiently solved by translation to the Boolean satisfiability problem (SAT). Previously we developed a technique to reduce the problem search space by enforcing DFA states to be enumerated in breadth-first search (BFS) order. We proposed symmetry breaking predicates, which can be added to Boolean formulae representing various automata identification problems. In this paper we continue the study of SAT-based approaches. First, we propose new predicates based on depth-first search order. Second, we present three methods to identify all non-isomorphic automata of the minimum size instead of just one—the $\#P$-complete problem which has not been solved before. Third, we revisited our implementation of the BFS-based approach and conducted new evaluation experiments. It occurs that BFS-based approach outperforms all other exact algorithms for DFA identification and can be effectively applied for finding all solutions of the problem.

Keywords: Grammatical inference · Automata identification
Symmetry breaking · Boolean satisfiability

1 Introduction

A variety of models exists in automata theory but a *deterministic finite automaton* (DFA) is the basic one and among the most important ones. DFA is a model that recognizes regular languages [1]. The essence of the DFA identification (induction, learning, synthesis) problem is to find a minimum-size DFA (a DFA with the minimum number of states) that is consistent with a given set of labeled examples—positive-labeled strings that must be accepted by the built DFA and negative-labeled strings that must be rejected. A smaller DFA is simpler and, because of well-known Occam's razor principle, it is a model which better explains the observed examples. Thus the DFA learning problem is to find the regular language that most likely was used to generate a set of labeled examples. This problem is among the best-explored ones in grammatical inference [2].

A. Cerone and M. Roveri (Eds.): SEFM 2017 Workshops, LNCS 10729, pp. 117–131, 2018.
https://doi.org/10.1007/978-3-319-74781-1_9

This problem was shown to be NP-hard in [3]. Nevertheless, several efficient DFA learning approaches were developed, see, e.g., [2]. DFA identification using evolutionary computation methods is one of historically the first and effective approaches, see, e.g., [4,5]. Subsequent research resulted in development of a method for evolving DFA using a multi-start random hill climber, see, e.g., [6].

Later approaches are based on heuristic algorithms. The *evidence driven state-merging* algorithm (EDSM) is the most commonly used and the only one which can handle large-sized target DFA [7]. This algorithm is greedy and works in polynomial time. Despite its efficiency in terms of solving time this approach usually finds only a local optimum but not a global one. The performance of EDSM was several times improved by using specialized search procedures, see, e.g., [8,9]. In [6] Lucas and Reynolds compared the EDSM algorithm and the evolutionary algorithm (EA) mentioned above. They found that the EDSM-based approach outperforms the EA in terms of solving time on almost all instances.

The methods mentioned above are not exact—they cannot guarantee that the found DFA is one of the minimum-sized ones. Heule and Verwer proposed so-called *translation-to-SAT* approach which can be applied to DFA identification [10]. This approach, as it can be obtained from the name, is based on the translation the original problem to well-studied *Boolean satisfiability problem* (SAT). The performance of SAT solvers has significantly improved over the last decade. This computational strength can be used in other problems by *translating* these problems into SAT instances, and subsequently running a modern SAT solver on them. This approach was shown to be very competitive for some problems, see, e.g., [11–14]. The authors have shown that translation-to-SAT is effective for solving DFA identification as well. The SAT-based method is exact as opposed to EA and EDSM algorithms, which is important because of the mentioned Occam's razor principle. The authors also proposed a combined approach, which used a few EDSM steps as a preprocessing step, and won the first prize at the *StaMInA* competition [15]. We do not consider of this step in our paper because EDSM is not an exact algorithm.

There are *symmetries* in many combinatorial problems. *Symmetry breaking predicates* can be added as constraints to SAT formula with purpose of elimination some or all symmetries and thus reduce the search space, see, e.g., [16]. When we talk about DFA the most obvious symmetries are groups of isomorphic automata. Heule and Verwer in [10] proposed simple but effective greedy *maximal clique* (max-clique) algorithm. It allows reducing the amount of isomorphic automata in each group from $n!$ to $(n-k)!$, where n is the size of the DFA and k is the size of the found clique. We proposed symmetry breaking predicates which enforce DFA states to be enumerated in the breadth-first search (BFS) order in [17]. These predicates can be added to a Boolean formula before passing it to a SAT solver. This approach allows to reduce the amount of isomorphic automata in each group from $n!$ to only one representative—the BFS-enumerated one. The results for the exact case still were not very good in our previous paper. However, the BFS-based approach is more flexible than max-clique—we demonstrated its flexibility by developing a modification of the noiseless translation-to-SAT technique for the noisy case (some examples are wrong-labeled).

In this paper we propose new symmetry breaking predicates based on depth-first search (DFS) order. This is the modification of our previous BFS-based approach. BFS-based predicates were not good enough to compete with the max-clique algorithm in DFA identification in our previous paper. Therefore we revisited our implementation of this technique. It occurs that both BFS-based and DFS-based approaches clearly outperform current state-of-the-art DFASAT from [10]. We also propose a method based on these techniques for solving the *problem of finding all automata* (find-all) with the minimum number of states which are consistent with a given set of examples. This problem has not been solved efficiently before. Moreover, none of the existing approaches for the DFA learning are applicable, even with slight modifications, to solve the find-all problem due to their nature. We use two ways of launching SAT solvers: relaunching a non-incremental solver and using an incremental solver. We also developed the heuristic backtracking method (almost similar to the one presented in the paper [18]) as a baseline for comparing it with SAT-based ones.

2 Preliminaries and Previous Work

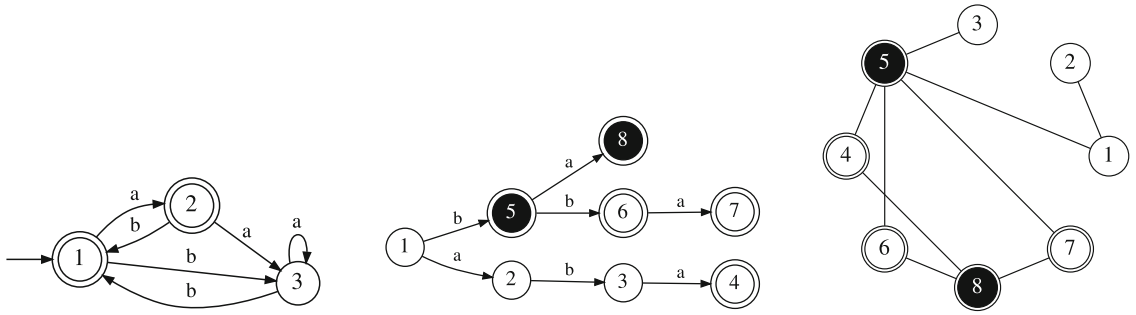
2.1 Encoding DFA Identification into SAT

We assume the reader to be familiar with the theory of languages and automata. The purpose of the DFA identification problem is to find the minimum DFA which is consistent with two given sets of strings: a set of positive examples (S_+) and is a set of negative examples (S_-). In other words, the desired DFA must accept all strings from S_+ and reject all strings from S_-. In this paper it is assumed that DFA states are numbered from 1 to C and the start state has number 1. The example of the minimum DFA for $S_+ = \{aba, bb, bba\}$ and $S_- = \{b, ba\}$ is shown in Fig. 1a.

We briefly describe the current state-of-the-art approach for solving the considered problem. The first step of the technique proposed by Heule and Verwer in [10] is to build an *augmented prefix tree acceptor* (APTA) from the given sets S_+ and S_-. An APTA is a tree-shaped automaton based on a prefix tree for the sets S_+ and S_- but with labeled states. It is called augmented because it may contain states which are not accepting or rejecting. The APTA for S_+ and S_- mentioned above is shown in Fig. 1b.

The second step is to construct the *consistency graph* (CG) for the built APTA. The set of the CG vertices is the same as the APTA vertices set. Two vertices in the CG are adjacent if their merging in the APTA and subsequent determinization process will cause an inconsistency: a situation when an accepting state is merged with a rejecting one. The CG for APTA from Fig. 1b is shown in Fig. 1c.

The third step of the method is to divide the CG vertices set into C disjoint sets. Each set has to contain all vertices equivalent to the corresponding APTA states which will be merged into one state in the resulting DFA. If such a separation can be made, then the automaton with C states consistent with the given sets of strings exists and it can be easily built. C can be iterated from 1 and until



(a) An example of a DFA

(b) An example of an APTA for $S_+ = \{aba, bb, bba\}$ and $S_- = \{b, ba\}$

(c) The consistency graph for the APTA from Fig. 1b

Fig. 1. An example of an APTA and its consistency graph

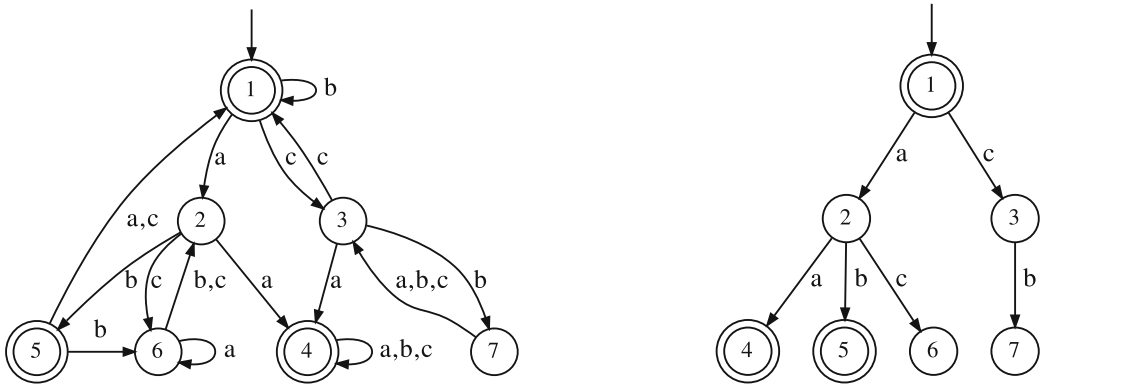
such a partition is found. Thus it is guaranteed that the found C-sized DFA is the minimum DFA consistent with given behavior examples. This can be viewed as a *graph coloring* problem and we need to color CG vertices into the minimum number of colors in such a way that adjacent vertices have different colors.

The next step in the considered algorithm is to translate the graph coloring problem into SAT. Authors proposed so-called *compact encoding* where they use three kinds of Boolean variables to formulate all constraints in CNF: *color variables* $x_{v,i}$ which indicate whether the vertex v in the CG is i-colored; *parent relation variables* $y_{a,i,j}$ which indicate whether there is an a-labeled transition from the i-colored state to the j-colored state in the target DFA; *accepting color variables* z_i which indicate whether the i-colored state in the target DFA is accepting. There are four mandatory and four redundant types of clauses in the proposed compact encoding. The reader can read about them in detail in [10]. The final step of the translation-to-SAT approach is to run an external SAT solver with the built CNF formula. If the formula is satisfiable, then the target DFA can be easily constructed from the found satisfying assignment. Otherwise, the number of colors C is increased.

2.2 Symmetry Breakings Predicates

Large Clique Predicates. Heule and Verwer used symmetry breaking predicates in their algorithm [10]. In the case when the CG cannot be colored into C colors the SAT solver tries to solve the same problem $C!$ times—one time for each permutation of colors. In other words the solver considers $C!$ isomorphic automata. The authors suggested to find some large clique in the CG and to fix the colors of its vertices. It helps to reduce the number of unnecessary considerations because in any valid graph coloring all vertices in a clique obviously have different colors. Thus, assuming that the size of the found clique is k, the solver considers only $(C - k)!$ isomorphic automata. Moreover, the process of iterating over C can be started from k instead of 1.

BFS-based Predicates. We proposed the new approach to symmetry breaking in our previous research [17]. Its main idea is to enforce DFA states to be enumerated in the *breadth-first search* (BFS) order. If some order (say lexicographical) on the transition symbols is fixed then only one representative of each equivalence class with respect to the isomorphic relation is BFS-enumerated due to the uniqueness of such BFS traversal. We call a DFA *BFS-enumerated* if its enumeration corresponds to the order of states processing during the BFS traversal. In other words, if we consider a BFS tree, built for some DFA and if we arrange the children of each state from left to right according to the chosen order on the transition symbols then numbers of states should increase from left to right on the same depth (*layer-order*) and from top to bottom (*depth-order*). In [17] we used the definition based on a BFS-queue which is equivalent to the one described above but less apprehensible. An example of a BFS-enumerated DFA is shown in Fig. 2a, and its BFS tree is shown in Fig. 2b.



(a) An example of a BFS-enumerated DFA

(b) A BFS tree of the DFA from Fig. 2a

Fig. 2. A BFS-enumerated DFA and its BFS-tree

If such predicates are used then while a SAT solver searches for a DFA consistent with the given samples, it is restricted to only BFS-enumerated ones. To implement this we proposed three additional kinds of Boolean variables:

1. **p**arent variables $p_{j,i}$ which are true if and only if state i is the parent of state j in the BFS tree;
2. **t**ransition variables $t_{i,j}$ which are true if and only if there is a transition from state i to state j;
3. **m**inimum symbol variables $m_{l,i,j}$ which are true if and only if there is a l-labeled transition from state i to state j and there are no such transitions labeled with a smaller symbol (according to the choosen order on symbols). These variables are used only in the case of a non-binary alphabet.

BFS-enumeration is enforced with the following seven clauses:

1. $\bigwedge_{1 \leqslant i < j \leqslant C} (t_{i,j} \Leftrightarrow y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j})$—definition of transition variables using variables $y_{l,i,j}$;
2. $\bigwedge_{1 \leqslant i < j \leqslant C} (p_{j,i} \Leftrightarrow t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j})$—definition of parent variables using variables $t_{i,j}$;
3. $\bigwedge_{2 \leqslant j \leqslant C} (p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1})$—each state except the start one holds a parent with a smaller number (depth-order);
4. $\bigwedge_{1 \leqslant k < i < j < C} (p_{j,i} \Rightarrow \neg p_{j+1,k})$—the ordering of children must be the same as the ordering of parents (layer-order for children of different parents);
5. $\bigwedge_{1 \leqslant i < j < C} (p_{j,i} \wedge p_{j+1,i} \Rightarrow y_{a,i,j})$—in case of a binary alphabet this constraint is sufficient to order two children j and $j+1$ of state i (layer-order for children of one parent);
6. $\bigwedge_{1 \leqslant i < j \leqslant C} \bigwedge_{1 \leqslant n \leqslant L} (m_{l_n,i,j} \Leftrightarrow y_{l_n,i,j} \wedge \neg y_{l_{n-1},i,j} \wedge \ldots \wedge \neg y_{l_1,i,j})$—definition of minimum symbol variables using variables $y_{l,i,j}$ which are used in case of a non-binary alphabet;
7. $\bigwedge_{1 \leqslant i < j < C} \bigwedge_{1 \leqslant k < n \leqslant L} (p_{j,i} \wedge p_{j+1,i} \wedge m_{l_n,i,j} \Rightarrow \neg m_{l_k,i,j+1})$—in case of a non-binary alphabet this constraint forces children of a state to be ordered according to the chosen order on symbols (layer-order for children of one parent).

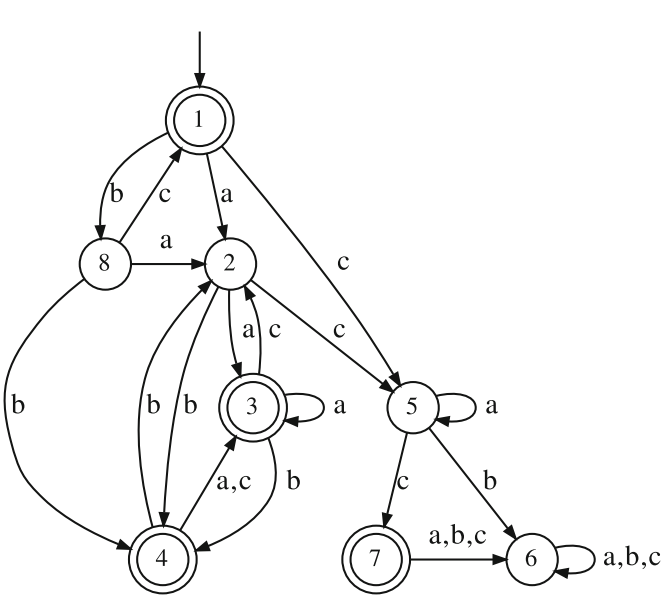
Using variables and clauses described above one can force an automaton to be BFS-enumerated. Unfortunately the implementation of these methods was not perfect when we prepared our previous paper [17] so the results did not show the real improvement caused by the proposed method. We revisited it and performed new evaluation experiments. The results are shown in Sect. 5 and they are quite impressive.

3 DFS-Based Symmetry Breaking Predicates

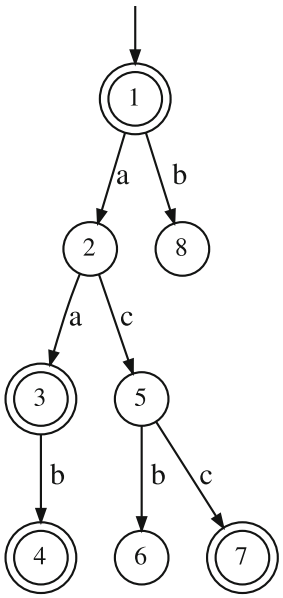
In this section we propose a new way to fix automata states enumeration to avoid consideration of isomorphic automata during SAT solving. This approach is a modification of our BFS-based predicates. It enforces automata states to be enumerated in the *depth-first search* (DFS) order. We describe the method briefly, paying attention only to the differences between DFS- and BFS-based approaches. Detailed information about BFS-based predicates can be found in Sect. 2.2.

During DFS processing it is necessary to find all adjacent unvisited states for each unvisited state of the DFA. Firstly, the DFS algorithm handles the initial DFA state. Then the algorithm processes the children of this state and recursively executes for each of them. We process child states in some particular (e.g., alphabetical) order of symbols l on transitions $i \xrightarrow{l} j$. Thus again only one representative of each equivalence class with respect to the isomorphic relation will be processed. We call a DFA *DFS-enumerated* if its states are numbered in

the order of handling them by DFS traversal with chosen symbol order. Although there is no traversal, we refer to it for the definition and explanation. The set of developed constraints enforces DFS. An example of a DFS-enumerated DFA is shown in Fig. 3a. A DFS tree for this DFA is shown in Fig. 3b.



(a) An example of a DFS-enumerated DFA



(b) A DFS tree of the DFA from Fig. 3a

Fig. 3. A DFS-enumerated DFA and its DFS tree

All variables which were used for the BFS enumeration are also used for the DFS enumeration, but some the constraints must be changed. In the DFS enumeration $p_{j,i}$ variables ($p_{j,i}$ is true if and only if state i is the parent of j in the DFS tree) are defined differently. Due to the greediness of the DFS algorithm, state i is the parent of state j if it has the maximum number among states that have a transition to j:

$$\bigwedge_{1 \leq i < j \leq C} (p_{j,i} \Leftrightarrow t_{i,j} \wedge \neg t_{i+1,j} \wedge \ldots \wedge \neg t_{j-1,j}),$$

where $t_{i,j} \equiv 1$ if and only if there is a transition between i and j (these variables in their turn are defined by using $y_{l,i,j}$ variables).

Moreover, in the DFS enumeration instead of the children ordering constraint we use the following one. If i is the parent of state j and k is a state between i and j ($i < k < j$) then there is no transition from state k to state q, where q is bigger than j:

$$\bigwedge_{1 \leq i < k < j < q < C} (p_{j,i} \Rightarrow \neg t_{k,q}).$$

Indeed, since $i < k < j$, state k has to be considered by the DFS algorithm before state j. Hence if such a transition would exist then state k must have a lower number than state j.

Now, to enforce the DFA to be DFS-enumerated we have to order children according to symbols on transitions (e.g., alphabetically). We consider two cases: alphabet Σ consists of two symbols $\{a, b\}$ and more than two symbols $\{l_1, \ldots, l_L\}$. In the case of two symbols state i can have only two transitions: to state j and to state k (where without loss of generality $j < k$). If the transition from state i to state j is used during the DFS traversal then it must be labeled with a smaller symbol:

$$\bigwedge_{1 \le i < j < k < C} (p_{j,i} \wedge t_{i,k} \Rightarrow y_{a,i,j}),$$

because otherwise state k had to be processed earlier.

In the second case we have to use $m_{l,i,j}$ variables: $m_{l,i,j}$ is true if and only if there is an l-labeled transition from state i to state j and there is no transition from state i to state j with an alphabetically smaller symbol. The idea is similar to the previous case. For state i it remains to arrange its children in the chosen order. For any two transitions from state i to state j and from state i to state k (where without loss of generality $j < k$), if state j is used during the DFS traversal then it must be labeled with a smaller symbol:

$$\bigwedge_{1 \le i < j < k \le C} \bigwedge_{1 \le m < n \le L} (p_{j,i} \wedge t_{i,k} \wedge m_{l_n,i,j} \Rightarrow \neg m_{l_m,i,k}).$$

Thus we proposed the new set of constraints which enforce a DFA to be DFS-enumerated. The predicates (for the case of three or more symbols) translated into $\mathcal{O}(C^4 + C^3L^2)$ (where C is the number of colors and L is the alphabet size) CNF clauses which are listed in Table 1 together with BFS-based predicates, which are translated into $\mathcal{O}(C^3 + C^2L^2)$ clauses.

Table 1. DFS-based and BFS-based symmetry breaking clauses

	Clauses	Range
Both	$t_{i,j} \Rightarrow (y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j})$	$1 \le i < j \le C$
	$y_{i,j,l} \Rightarrow t_{i,j}$	$1 \le i < j \le C$; $l \in \Sigma$
	$p_{j,i} \Rightarrow t_{i,j}$	$1 \le i < j \le C$
	$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$2 \le j \le C$
	$m_{l,i,j} \Rightarrow y_{l,i,j}$	$1 \le i < j \le C$; $l \in \Sigma$
	$m_{l_n,i,j} \Rightarrow \neg y_{l_k,i,j}$	$1 \le i < j \le C$; $1 \le k < n \le L$
	$(y_{l_n,i,j} \wedge \neg y_{l_{n-1},i,j} \wedge \ldots \wedge \neg y_{l_1,i,j}) \Rightarrow m_{l_n,i,j}$	$1 \le i < j \le C$; $1 \le n \le L$
DFS	$p_{j,i} \Rightarrow \neg t_{k,j}$	$1 \le i < k < j \le C$
	$(t_{i,j} \wedge \neg t_{i+1,j} \wedge \ldots \wedge \neg t_{j-1,j}) \Rightarrow p_{j,i}$	$1 \le i < j \le C$
	$p_{j,i} \Rightarrow \neg t_{k,q}$	$1 \le i < k < j < q \le C$
	$(p_{j,i} \wedge p_{k,i} \wedge m_{l_n,i,j}) \Rightarrow \neg m_{l_m,i,k}$	$1 \le i < j < k \le C$; $1 \le m < n \le L$
BFS	$p_{j,i} \Rightarrow \neg t_{k,j}$	$1 \le k < i < j \le C$
	$(t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j}) \Rightarrow p_{j,i}$	$1 \le i < j \le C$
	$p_{j,i} \Rightarrow \neg p_{j+1,k}$	$1 \le k < i < j < C$
	$(p_{j,i} \wedge p_{j+1,i} \wedge m_{l_n,i,j}) \Rightarrow \neg m_{l_m,i,j+1}$	$1 \le i < j < C$; $1 \le m < n \le L$

4 The Find-All Problem

In this section we consider the problem of finding all non-isomorphic DFA (*find-all problem*) with the minimum number of states which are consistent with a given set of strings. We propose a way to modify the SAT-based method of solving regular DFA identification problem in order to apply it to the find-all problem. We consider two ways of using SAT solvers: restarting a non-incremental solver after finding each automaton and using an incremental solver—if such a solver finds a solution, it retains its state and is ready to accept new clauses. The most common interface and technique for incremental SAT solving was proposed in [19]. We also propose the heuristic backtracking method as a baseline for comparing it with SAT-based ones.

4.1 SAT-Based Methods

The main idea of SAT-based methods of solving the find-all problem is to ban satisfying interpretations (variable values) which have already been found. It is obvious that if the proposed symmetry breaking predicates are not used then this approach finds many isomorphic automata—exactly $C!$ for each equivalence class where C is the DFA size. Since max-clique predicates fix k colors only (where k is the clique size), the algorithm of Heule and Verwer finds $(C - k)!$ isomorphic automata which is still bad. The BFS-based and DFS-based symmetry breaking predicates allow us to ban isomorphic DFA from one equality class by banning an accordingly enumerated representative. It must be noted that although the idea to discard satisfying interpretations is classic for such methods, it cannot be used in practice without effective symmetry breaking techniques. There were no known techniques to deal with factorial number of isomorphic automata earlier, and thus the considered problem could not be solved effectively. Proposed symmetry breaking predicates change the situation and bring the solution. It is easy to implement this by adding a *blocking* clause into the Boolean formula. Since we know that $y_{l,i,j}$ variables define the structure of the target DFA entirely, it is enough to forbid only values of these variables from the found interpretation:

$$\neg y_1 \vee \neg y_2 \vee \ldots \vee \neg y_{n|\Sigma|},$$

where y_k is some $y_{l,i,j}$ from the found interpretation for $1 < k < n|\Sigma|$.

There are two different ways of using SAT solvers as it was stated above. First, we can restart a non-incremental SAT solver with the new Boolean formula with the blocking clause after finding each automaton. The second approach is based on incremental SAT solvers: after each found automaton we add the blocking clause to the solver and continue its execution.

It is necessary to mention the case when some transitions of the found DFA are not covered by the APTA. It means that there are some *free* transitions which are not used during processing any given word and each such transition can end in any state, since this does not influence the consistency of the DFA with a given set of strings. But in the case of the find-all problem basically we

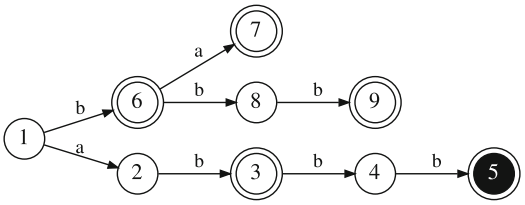
do not wish to find all these automata distinguished only by such transitions. Thus we propose a way to force all free transitions to be self-loops—end in the same state as they start. To achieve that we add auxiliary '**u**sed' variables: $u_{l,i}$ is true if and only if there is an l-labeled APTA edge from the i-colored state:

$$\bigwedge_{l \in \Sigma} \bigwedge_{1 \le i \le C} u_{l,i} \Leftrightarrow x_{1,i} \vee \ldots \vee x_{|V_l|,i},$$

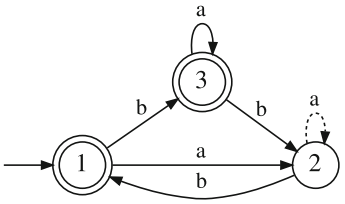
where V_l is the set of all the APTA states which have an outcoming edge labeled with l. To force unused transitions to be self-loop we add the following constraints:

$$\bigwedge_{l \in \Sigma} \bigwedge_{1 \le i \le C} \neg u_{l,i} \Rightarrow y_{l,i,i}.$$

These additional constraints are translated into $\mathcal{O}(C|L|)$ clauses. See Fig. 4 for an example of an APTA for $S_+ = \{ab, b, ba, bbb\}$ and $S_- = \{abbb\}$ and its consistent DFA with an unused transition. If we add the proposed constraints, then this transition will be forced to be a loop as shown by a dashed line in Fig. 4b.



(a) An example of an APTA for $S_+ = \{ab, b, ba, bbb\}$ and $S_- = \{abbb\}$



(b) The DFA is built from the APTA from Fig. 4a with unused a-labeled transition from state 2

Fig. 4. An example of an APTA and its consistent DFA

4.2 Backtracking Algorithm

The solution based on backtracking does not use any external tools like SAT solvers. This algorithm works as follows. Initially there is an empty DFA with n states. Also there is a *frontier*—the set of edges from the APTA which are not yet represented in the DFA. Initially the frontier contains all outcoming edges of the APTA root. The recursive function `Backtracking` maintains the frontier in the proper state. If the frontier is not empty, then the function tries to augment the DFA with one of its edges. Each found DFA is checked to be consistent with the APTA and if the DFA complies with it then an updated frontier is found. If the frontier is empty then the DFA is checked for completeness (a DFA is complete if there are transitions from each state labeled with all alphabet symbols). If it is not complete and there are nodes which have the number of

outcoming edges less than the alphabet size then we add missing edges as self-loops with function `MakeComplete`. Algorithm 1 illustrates the solution. The function `FindNewFrontier` returns the new frontier for the augmented DFA or null if the DFA is inconsistent with the APTA. This algorithm is an exact search algorithm based on the one from [18].

```
Data: augmented prefix tree acceptor APTA, current DFA (initially empty),
      frontier (initially contains all APTA root outcoming edges)
DFAset ← new Set<DFA>
edge ← any edge from frontier
foreach destination ∈ 1..|S| do
    source ← the state of DFA from which edge should be added
    DFA′ ← DFA ∪ transition(source, destination, edge.label)
    frontier′ ← FindNewFrontier(APTA, DFA′, frontier)
    if frontier′ ≠ null then
        if frontier′ = ∅ then
            DFAset.add(MakeComplete(DFA′))
        else
            DFAset.add(Backtracking(APTA, DFA′, frontier′))
        end
    end
end
return DFAset
```

Algorithm 1. Backtracking solution

5 Experiments

All experiments were performed using a machine with an AMD Opteron 6378 2.4 GHz processor running Ubuntu 14.04. All algorithms were implemented in Java, the *lingeling* SAT solver was used [20]. As far as we know all common benchmarks are too hard for solving by exact algorithms without some heuristic non-exact steps. Thus our own algorithm was used for generating problem instances. This algorithm builds a set of strings with the following parameters: size N of DFA to be generated, alphabet size A, the number S of strings to be generated. The algorithm is arranged as follows. First of all N states are generated and uniquely numerated from 1 to N. Each state is equiprobably set to be accepting or rejecting. Next on step i the algorithm picks state i, evenly chooses another state from $[i+1; N]$ and adds a random-labeled transition from the first state to the second. After $N-1$ such steps we have partially built an automaton where all states are reachable from the initial one (1-numbered). In the end the algorithm picks each state one by one and add all missing (in terms of automaton completeness) transitions with destination randomly chosen among all states. Finally S strings are generated by processing the automaton.

The distribution of the words' length is shifted to longer words. These strings with the accepting or rejecting labels form the instance of the DFA identification problem.

For DFA identification we used the following parameters: $N \in [10; 30]$ with step 2, $A = 2$, $S = 50N$. We compared the SAT-based approach with three types of symmetry breaking predicates: the max-clique algorithm from [10] (the current state-of-the-art) and the proposed DFS-based and BFS-based methods. Each experiment was repeated 100 times. The time limit was set to 3600 s. The results are listed in Table 2. It can be seen from the table that both DFS-based and BFS-based strategies clearly outperform the max-clique approach. BFS-based strategy in its turn notably outperforms DFS-based one when target automaton size is larger than 14. These results for the BFS-based approach were not obtained in our previous research due to weaker technical implementation.

Table 2. Median execution times of exact solving DFA identification in seconds

N	DFS	BFS	max-clique
10	20.9	20.5	23.3
12	40.4	37.6	240.3
14	82.2	62.4	TL
16	205.1	114.1	TL
18	601.7	181.9	TL
20	2501.6	293.7	TL
22	TL	453.3	TL
24	TL	625.1	TL
26	TL	925.8	TL
28	TL	1314.4	TL
30	TL	1635.5	TL

The second experiment concerned the find-all problem. A random dataset was also used here. We used the following parameters: $N \in [5; 15]$, $A = 2$, $S \in \{5N, 10N, 25N\}$. We compared the BFS-SAT-based method with the restarting strategy (REST column in the table), the BFS-SAT-based method with the incremental strategy (INC) and the backtracking method (BTR). Each experiment was repeated 100 times as well. The time limit was set to 3600 s. The results are given in Table 3. The first column in each subtable contains the number of instances which have more than one DFA in the solution (> 1). If less than 50 instances were solved then TL is shown instead of a value. It can be seen from the table that SAT-based methods work significantly faster than the backtracking one when the size of the automaton is greater than 8. It happens because the SAT-based methods with BFS-based predicates consider only one DFA for each equivalence class with respect to the isomorphic relation instead

Table 3. Median execution times in seconds of SAT-based restart method, SAT-based incremental method and backtracking method

$\|N\|$	$S = 5\|N\|$				$S = 10\|N\|$				$S = 25\|N\|$			
	>1	REST	INC	BTR	>1	REST	INC	BTR	>1	REST	INC	BTR
5	53	2.3	2.0	0.8	40	3.6	3.3	1.3	17	4.1	3.4	1.5
6	56	2.8	2.4	2.1	31	4.7	3.9	1.7	27	5.4	4.3	1.7
7	87	3.9	2.5	4.1	27	3.7	3.0	3.1	13	7.4	6.7	2.5
8	80	4.6	3.7	87.2	34	7.0	6.5	41.7	16	10.1	8.9	11.6
9	91	7.6	3.9	475.1	50	7.7	6.4	121.6	10	13.8	13.0	61.4
10	89	15.7	5.3	2756.2	47	8.6	7.0	974.7	11	18.8	16.1	276.8
11	94	19.9	7.3	TL	63	18.5	13.8	3108.0	9	24.5	21.9	1158.4
12	90	28.0	9.9	TL	49	22.3	16.7	TL	8	33.5	27.2	3289.1
13	92	185.5	18.1	TL	57	36.9	22.6	TL	12	62.0	51.4	TL
14	87	408.5	49.0	TL	71	85.1	41.8	TL	4	67.0	56.2	TL
15	95	571.1	174.1	TL	69	193.3	95.7	TL	6	29.2	26.2	TL

of $N!$. As we see, the incremental strategy in its turn clearly outperforms the restart strategy. It can be explained as incremental SAT solver saves its state but non-incremental solver does the same actions on each execution.

Our implementation of proposed predicates and algorithms is available on our laboratory github repository[1].

6 Conclusions

We have proposed DFS-based symmetry breaking predicates. They can be added to the Boolean formula before passing it to a SAT solver while solving various DFA identification problems with SAT-based algorithms. Using these predicates allows reducing the problem search space by enforcing DFA states to be enumerated in the depth-first search order.

We have revisited our implementation of the proposed symmetry breaking predicates and compared the translation-to-SAT method from [10] to the same one with proposed symmetry breaking predicates instead of original max-clique predicates. The proposed approach clearly improved the translation-to-SAT technique which was demonstrated with the experiments on randomly generated input data. The BFS-based approach has shown better results than the DFS-based one if the target DFA size is large.

Then, we have proposed a solution for the find-all DFA problem. The proposed approach can efficiently solve the problem that the previously developed methods cannot be applied for. We performed the experiments which have shown

[1] https://github.com/ctlab/DFA-Inductor.

that our approach with the incremental SAT solver clearly outperfoms the Backtracking algorithm.

Acknowledgements. The authors would like to thank Igor Buzhinsky, Daniil Chivilikhin, Maxim Buzdalov for useful comments. This work was financially supported by the Government of Russian Federation, Grant 074-U01.

References

1. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston (2006)
2. De La Higuera, C.: A bibliographical study of grammatical inference. Pattern Recogn. **38**(9), 1332–1348 (2005)
3. Gold, E.M.: Complexity of automaton identification from given data. Inf. Control **37**(3), 302–320 (1978)
4. Dupont, P.: Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 236–245. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58473-0_152
5. Luke, S., Hamahashi, S., Kitano, H.: Genetic programming. In: Proceedings of the genetic and evolutionary computation conference, vol. 2, pp. 1098–1105 (1999)
6. Lucas, S.M., Reynolds, T.J.: Learning DFA: evolution versus evidence driven state merging. In: The 2003 Congress on Evolutionary Computation, 2003. CEC 2003, vol. 1, pp. 351–358. IEEE (2003)
7. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V., Slutzki, G. (eds.) ICGI 1998. LNCS, vol. 1433, pp. 1–12. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054059
8. Lang, K.J.: Faster algorithms for finding minimal consistent DFAs. Technical report (1999)
9. Bugalho, M., Oliveira, A.L.: Inference of regular languages using state merging algorithms with search. Pattern Recogn. **38**(9), 1457–1467 (2005)
10. Heule, M.J.H., Verwer, S.: Exact DFA identification using SAT solvers. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS (LNAI), vol. 6339, pp. 66–79. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15488-1_7
11. Lohfert, R., Lu, J.J., Zhao, D.: Solving SQL constraints by incremental translation to SAT. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) IEA/AIE 2008. LNCS (LNAI), vol. 5027, pp. 669–676. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69052-8_70
12. Galeotti, J.P., Rosner, N., Pombo, C.G.L., Frias, M.F.: TACO: efficient SAT-based bounded verification using symmetry breaking and tight bounds. IEEE Trans. Softw. Eng. **39**(9), 1283–1307 (2013)
13. Ulyantsev, V., Tsarev, F.: Extended finite-state machine induction using SAT-solver. In: Proceedings of ICMLA 2011, vol. 2, pp. 346–349. IEEE (2011)
14. Zbrzezny, A.: A new translation from ECTL* to SAT. Fundamenta Informaticae **120**(3–4), 375–395 (2012)
15. Walkinshaw, N., Lambeau, B., Damas, C., Bogdanov, K., Dupont, P.: STAMINA: a competition to encourage the development and assessment of software model inference techniques. Empirical Software Engineering **18**(4), 791–824 (2013)

16. Crawford, J., Ginsberg, M., Luks, E., Roy, A.: Symmetry-breaking predicates for search problems. KR **96**, 148–159 (1996)
17. Ulyantsev, V., Zakirzyanov, I., Shalyto, A.: BFS-based symmetry breaking predicates for DFA identification. In: Dediu, A.-H., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) LATA 2015. LNCS, vol. 8977, pp. 611–622. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15579-1_48
18. Ulyantsev, V., Buzhinsky, I., Shalyto, A.: Exact finite-state machine identification from scenarios and temporal properties. Int. J. Softw. Tools Technol. Transf. 1–21 (2016)
19. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
20. Biere, A.: Splatz, lingeling, plingeling, treengeling, YalSAT entering the SAT competition 2016. In: Proceedings of SAT Competition, pp. 44–45 (2016)

Active learning of formal plant models for cyber-physical systems

Polina Ovsiannikova*, Daniil Chivilikhin*, Vladimir Ulyantsev*,
Andrey Stankevich*, Ilya Zakirzyanov*, Valeriy Vyatkin*†‡, and Anatoly Shalyto*
*Computer Technologies Laboratory, ITMO University, Saint Petersburg, Russia
Email: polina.ovsyannikova@corp.ifmo.ru, chivdan@rain.ifmo.ru, ulyantsev@rain.ifmo.ru, shalyto@mail.ifmo.ru
†Department of Electrical Engineering and Automation, Aalto University, Finland
Email: vyatkin@ieee.org
‡Department of Computer Science, Computer and Space Engineering, Lulea Tekniska Universitet, Sweden

***Abstract*—As the world becomes more and more automated, the degree of cyber-physical systems involvement cannot be overestimated. A large part of them are safety-critical, thus, it is especially important to ensure their correctness before start of operation or reconfiguration. For this purpose the model checking approach should be used since it allows rigorously proving system correctness by checking all possible states. To ensure the compliance of controller-plant properties with system requirements, the closed-loop verification approach should be chosen, which requires not only a formal model of the controller, but also a formal model of the plant. In this paper we propose an approach for constructing formal models of context-free deterministic plants automatically using active learning algorithms. The case study shows its successful application to plant model generation for the elevator cyber-physical system.**

I. Introduction

Even if cyber-physical system (CPS) functionality does not include immediate interaction with humans, it still can influence them indirectly. Therefore, among other reasons, correct behavior of CPS is required to protect lives. The compliance of system implementation with its specification is commonly checked with simulation and testing. However, even automated testing is limited to checking only a limited number of behaviors defined in test cases. Formal verification using *model checking* [1] is a more comprehensive approach. It examines the entire state space of the system for errors, so that deviations that are inconspicuous during testing can be detected. In a number of works it was argued that in CPS applications closed-loop model checking [2], [3] is preferable to the open-loop approach, which is commonly used for computer programs verification.

In the closed-loop approach, a necessary part of the process is constructing a formal model of the plant. In most cases formal modeling is performed manually which may be very resource consuming. In order to reduce the influence of the human factor and keep the formal model consistent and up-to-date, approaches for automated plant model generation using behavior examples, or traces, were developed.

Approaches to formal model inference can be classified by the ability to interact with the modeled object (simulation model) during the construction process. *Passive learning* approaches build models from data collected before the learning process [4]–[6]. Thus, together with the model inference method, a way of gathering traces should be developed, that provides thorough coverage of system states. Without knowledge about coverage, measuring the similarity between the resulting formal model and the real system is complicated, although research has been done in this area [7]. The second approach to model inference is *active learning* [8] that can be introduced by the well-known L* algorithm [9] and its modifications, e.g. [10], [11]. The essence of active learning algorithms is constant communication of the algorithm with the simulation model during the inference process. This allows refining the model on every iteration and querying the system with the exact requests needed to determine its behavior.

In this paper we introduce the active learning approach for automatic formal models generation of deterministic discrete context-free plants in a black-box scenario. We also show why the most popular active learning algorithm L* and similar ones are not quite helpful when it comes to CPS plant models inferring.

II. Related work: active learning

The most widely known active learning algorithm L* [9] can be used to infer a Deterministic Finite Automaton (DFA) that accepts the same language as the source DFA. This approach is based on the *Nerode congruence theorem* [12], which states that two words u and u' belonging to the same language are equivalent, if and only if there are no *distinguishing suffixes* for them. Suffix v is distinguishing if, when concatenated with word u and u', it makes either uv or $u'v$ belong to language L, but not both.

The essence of the L* algorithm is iterative hypothesis automaton refinement based on counterexamples that are received after comparing the source automaton with the hypothesis one. Consequently, the algorithm consists of the following steps: (1) hypothesis automaton construction, (2) checking whether the hypothesis automaton is equivalent to the source automaton, (3) processing a counterexample if it exists. Having the source DFA `A` and hypothesis DFA `H`, let us describe each step in more detail.

During the first step, a hypothesis automaton `H` is built according to the *Observation Table (OT)* that contains informa-

978-1-5386-4829-2/18/$31.00 ©2018 IEEE

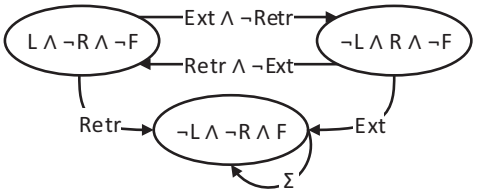


Fig. 1. Moving cylinder plant model

tion about acceptance of particular words by A. This knowledge is collected by sending *Membership Queries* (MQ) – requests to A that contain a sequence to check, on which A responds with True if the sequence belongs to the language or False in the opposite case. To proceed to hypothesis inference, *closedness* and *consistency* OT properties should be satisfied.

On the second step of the algorithm, H is compared to A by means of an *Equivalence Query* (EQ). In the original description of L*, this step is not detailed as the existence of an oracle is assumed: if H is found to be equivalent to A, the oracle reports "YES"; otherwise, it returns a *counterexample* word that is accepted by A, but not by H, or vice versa.

On the last step, a counterexample, if it exists, is processed. All its prefixes are added to the rows of the OT, and then the OT is filled with the results of new MQs and the the first step is repeated. If the oracle reports YES, then H is the resulting DFA, and the algorithm terminates.

In this work the L* algorithm was implemented and tested on the example of inferring a formal model of the plant for a moving cylinder (Fig. 1). The system has two Boolean input variables: Ext (Extend) and Retr (Retract), and three Boolean output variables: L (Left), R (Right) and F (Failure). Thus, in terms of the L* algorithm, there are 2^2 input symbols (all combinations of input variable values) and 2^3 output symbols. The plant is conveniently represented as a Moore machine. Hence, L* was adapted to learn deterministic Moore machines as follows: the cells of OT now store the output symbols generated by the source system in response to MQs instead of labels indicating existence of specific sequences in the source language.

Since the plant is considered as a black box and no oracle is provided, the EQ was implemented in the following way: the system was queried with random sequences that were not equal to the checked ones during OT construction, with their sizes up to the double number of states of the hypothesis automaton.

The constructed model was converted to the input format of the symbolic model checker NuSMV [13]. Once the model was constructed, system specification expressed with Linear Temporal Logic (LTL) properties was checked using NuSMV. The results of model checking are shown in Table I: the model of the system satisfies all considered LTL constraints.

However, since in a black box scenario an oracle is not available and EQ can only be approximated by heuristics, it is impossible to claim that the constructed formal model is equivalent to the source model. Also note that L* was developed for learning regular languages or context-dependent systems – behavior of such systems depends not only on the current input and output variables, but on the *history of interaction*, or the *context*. The absence of context and internal variables in a system means that regardless of the input sequence that has led the system to some particular state, future states will be determined by input symbols only.

Therefore, L* can be applied for learning models of plants with context, though facing the mentioned issue with EQ. Furthermore, additional difficulties are introduced by processing continuous variables – this would require serious modifications of L* while the result would still be inexact due to the heuristic nature of EQ.

However, it can be argued that in a CPS the plant should be independent of the context. Indeed, many systems comply with this assumption. Therefore, the method of inferring context-free systems was developed that does not require an oracle or EQ and where reliability of the resulting model is determined not via automata comparison but by the algorithm termination condition.

III. Proposed approach

Consider a context-free deterministic discrete system with no internal variables nor explicit time dependence. The combination of all input variable values passed to the system in a request is called an *Input Symbol* and is denoted as a tuple $(v(I_1), v(I_2), \ldots, v(I_n))$, where $v(I_1), \ldots, v(I_n)$ are input variable values. Similarly, the combination of all output variable values produced by the system as a response to a request is an *Output Symbol*: $(v(O_1), v(O_2), \ldots, v(O_n))$, where $v(O_1), \ldots, v(O_n)$ are output variable values. Note that since we consider only deterministic context-free plants, the output symbol is regarded as a *system state* since it contains values of all output variables.

Every variable in the model should be discrete. If the system has continuous variables, they can be discretized by splitting their allowed range into a number of contiguous intervals. For example, the real output variable $\mathtt{O} \in [0; 15]$ can be discretized as follows: $\mathtt{O}' \in \{[0; 5), [5; 10), [10; 15]\}$. Then, each continuous value of a variable can be replaced with the index of the interval it belongs to. Since discretization is case-dependent, it is assumed to be provided by the user.

Now note the two following observations. First, the maximum number of states of the resulting automaton is bounded from above by the number of unique output symbols. Second, if the system is in some state and all transitions from this state have been checked with MQs, there is no need to check them again. With these observations in mind, the plant model can be inferred using an active algorithm that resembles classical breadth-first search (BFS), which produces automata of the form shown in Fig. 2.

The core idea of the algorithm is to explore every transition labeled by every input symbol reachable from the chosen initial state. During the learning process only newly discovered states are added to the queue of next states to make transitions from, hence, if a state already exists in the model, it is not

TABLE I
TEMPORAL PROPERTIES FOR THE CYLINDER PLANT MODEL

Name	Temporal property	Comment	Correct value	Obtained value
φ_1	$\mathbf{G}(\mathtt{F} \rightarrow \mathbf{G}\,\mathbf{F}(\mathtt{R} = 0 \ \wedge \ \mathtt{L} = 0))$	If the system experiences failure, it stops working forever	+	+
φ_2	$\mathbf{G}(\mathtt{L} \ \wedge \ \mathtt{R} \ \wedge \ \neg\mathtt{Ext} \rightarrow \mathbf{X}(\mathtt{L}))$	When the cylinder is on the left side and the command is to retract, it will stay on the left	–	–
φ_3	$\mathbf{G}\,\mathbf{F}(\mathtt{R} \ \wedge \ \mathtt{L})$	Infinitely often the cylinder will appear in both sides simultaneously	–	–
φ_4	$\mathbf{G}(\mathtt{R} \ \wedge \ \mathtt{Ext} \rightarrow \mathbf{X}(\mathtt{F}))$	When the cylinder is on the right side and the command is to extend, the failure signal will be produced	+	+
φ_5	$\mathbf{G}(\mathtt{R} \ \wedge \ \neg\mathtt{Ext} \ \wedge \ \neg\mathtt{Retr} \rightarrow \mathbf{X}(\mathtt{R}))$	If the cylinder is on the right side and there are no commands to move anywhere, it does not move (the same for the left side)	+	+
φ_6	$\mathbf{G}(\mathtt{L} \ \wedge \ \mathtt{Ext} \ \wedge \ \neg\mathtt{Retr} \rightarrow \mathbf{X}(\mathtt{R}))$	If the cylinder is on the left side and the command is to extend, it will appear on the right	+	+

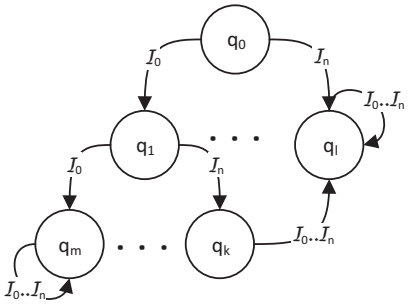


Fig. 2. The example form of the automaton that can be inferred using the proposed approach, where q_l, q_m, and q_k are distinct and unknown in advance states

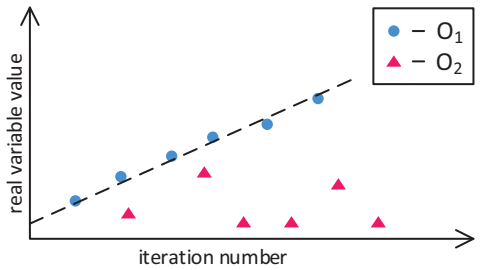


Fig. 3. Real variable $\mathtt{O}_1$ value monotonically increases, correlation coefficient > 0.5, whereas $\mathtt{O}_2$ value fluctuates, correlation coefficient < 0.5

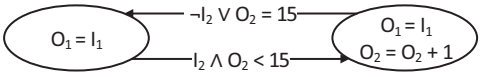


Fig. 4. Example system model with Boolean inputs $\mathtt{I}_1$ and $\mathtt{I}_2$, Boolean output $\mathtt{O}_1$ and continuous output $\mathtt{O}_2$

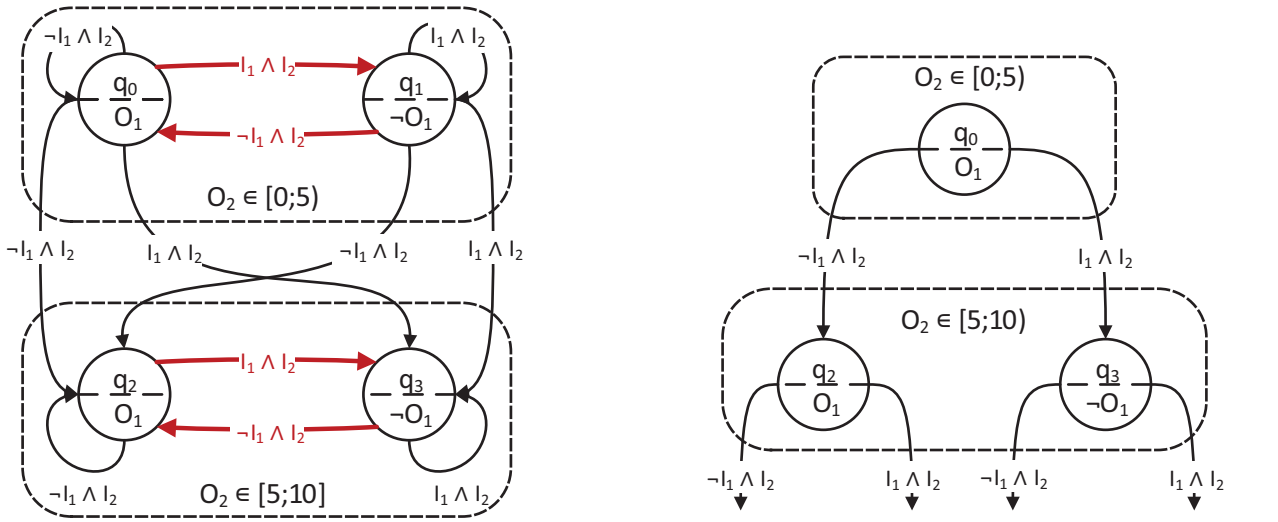
processed again. The algorithm terminates when the queue is empty and there are no new states to process.

The simple algorithm described above is sufficient in the rare case when the plant only uses Boolean variables. In practice, the plant almost always has continuous dynamics and is described with real variables, which makes formal modeling more complicated. Discretization of each real variable into a set of contiguous intervals makes the model discrete, however, as seen on the following example, the above algorithm will not produce a correct model.

Consider a system with one Boolean input variable $\mathtt{I}_1$ and one real output variable discretized with intervals $\mathtt{O}_1' \in \{[0;5),[5;15)\}$. The constructed model automaton will have two states, one for each value of $\mathtt{O}_1$ intervals. In the initial state $\mathtt{O}_1 \in [0;5)$; after a transition triggered by $\mathtt{I}_1 = \mathtt{True}$, the value of $\mathtt{O}_1$ is increased by 1. Following the logic described above, two transitions will be checked, one for $\mathtt{I}_1 = \mathtt{True}$ and one for $\mathtt{I}_1 = \mathtt{False}$. Both queries will result in self-loops in the model, no new states will be generated, and the algorithm will terminate leaving the state in which $\mathtt{O}_1 \in [5;15)$ undiscovered. Still, taking a closer look at the value of $\mathtt{O}_1$ after the first transition may indicate that the variable value is moving towards the next interval and the query simply needs to be repeated several times to reach it. The situation when after some transition a continuous variable changes its value but stays inside the current discretization interval will be called *movement inside an interval*, which means that the continuous variable value changes monotonically.

To detect "movement inside an interval" behavior the same MQ is sent to the system C times and the results are saved. After C repetitions, the Pearson correlation coefficient between the number of iterations and variable value is calculated. If the correlation coefficient is greater than 0.5, we can conclude that the value is changing almost monotonically – in this case we keep querying the system with the same MQ until the variable takes a value from the next interval. If the correlation coefficient is less than 0.5, then there is no change in the system and there is no sense in repeating a particular MQ (see Fig. 3).

However, in the situation when there are several discrete and discretized output variables in the system, even using additional logic for self-loops processing, the resulting model can be wrong – unnecessary cycles can be formed in the resulting model which will cause problems during verification. This problem is illustrated on an example system (Fig. 4) with two Boolean input variables $\mathtt{I}_1$ and $\mathtt{I}_2$ and two output



(a) A part of the model that contains cycles (marked bold red) in the first and in second intervals of variable O_2

(b) A part of the model where cycles from the first and second intervals of O_2 are removed

Fig. 5. Parts of the model before and after removing cycles

variables: Boolean variable O_1 and real variable $O_2 \in [0; 15]$ with discretization $O_2' \in \{[0; 5), [5; 10), [10; 15]\}$.

A part of the resulting automaton is shown in Fig. 5a. Here only transitions with $I_2 = \texttt{True}$ are shown to make the graph easier to read. Having such a model, false negative verification results are possible due to the cycle between q_0 and q_1. For example, the property "when O_2 is in the first interval and I_2 is always true, the system will eventually move to the state where O_2 is in the second interval" is false, while it is satisfied for the original system in Fig. 4.

To resolve this issue, the previous mechanism of detecting "movement inside intervals" situations was enhanced as follows. If during processing of a transition from q_n to q_k induced by some request (MQ) R a "movement inside an interval" situation is detected for some variable, state q_k is saved as a temporary state. Then, firstly, it is necessary to check whether the transition induced by request R from state q_k is a self-loop as it is described above. If so, then no other requests are queried from q_k and request R is repeated until the continuous variable reaches its next interval. Then q_k is registered as a new state. Otherwise, state q_k is marked as a state in which continuous variables will be compared by their concrete values, not by discrete intervals. If a state is marked this way, all states that will be generated after any transition from it will be marked as well if they stay in the same intervals of continuous variables.

Using the described strategy, the part of the model given in Fig. 5a can be redrawn as shown in Fig. 5b. The full pseudocode of the proposed plant model construction algorithm is given in Algorithm 1, where the aforementioned comparison of concrete continuous variable values is implemented via `enableConcreteComparison(state)` function (see line 32). After its execution it becomes impossible to limit the final number of states of the model from above. Also it is worth mentioning that if there are no discretized variables in the system, the loop starting at line 17 of the algorithm can be omitted.

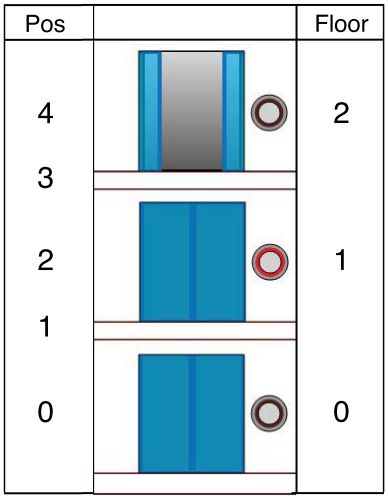


Fig. 6. Interface of the elevator model [7]

IV. Evaluation on a Case Study

The proposed approach was implemented in Java and is available as an open-source tool [14]. The approach was tested on the example of a three-floor elevator simulation model developed in NxtStudio [15] with user interface shown in Fig. 6. This model is almost identical to the one considered in [7] with a difference in buttons logic, which was moved to the controller. The aim of the experiments was to generate the plant model with the proposed method and compare verification results with [7].

The plant has the following Boolean variables:

- input: `moveUp`/`moveDown` – move the car up or down;

Algorithm 1: Proposed plant model synthesis algorithm

Data: set I of input variables, set O of output variables
Result: set of transitions $\mathcal{T}$

$Q_{proc} \leftarrow \varnothing$;
$Q_{next} \leftarrow$ `sendMQ(`$\varnothing$`)`;
$\mathcal{T} \leftarrow \varnothing$;
$\mathcal{I} \leftarrow \prod I_i$;
$\mathcal{O} \leftarrow \prod O_i$;
while $Q_{next} \neq \varnothing$ **do**
  $T \leftarrow \varnothing$;
  for $q \in Q_{next}$ **do**
    for $input \in \mathcal{I}$ **do**
      $(q_s, q_e, s) \leftarrow$ `sendMQ(`$q, input$`)`;
      $T \leftarrow T \cup \{(q_s, q_e, s)\}$;
    end
  end
  $T_{inInterval} \leftarrow \varnothing$;
  for $(q_s, q_e, s) \in T$ **do**
    if `movingInsideInterval(`q_s, q_e`)` **then**
      $T_{inInterval} \leftarrow T_{inInterval} \cup \{(q_s, q_e, s)\}$;
    end
  end
  for $(q_s, q_e, s) \in T_{inInterval}$ **do**
    $q'_e \leftarrow$ `sendMQ(`q_e, s`)`$.end$;
    $c \leftarrow$ `correlation(`q_s, q_e, s`)`;
    if $q'_e = q_e \ \wedge \ c > 0.5$ **then**
      while $q'_e = q_e$ **do**
        $q_e \leftarrow q'_e$;
        $q'_e \leftarrow$ `sendMQ(`q_e, s`)`$.end$;
      end
      $q_e \leftarrow q'_e$;
    else if $q'_e = q_e \ \wedge \ c < 0.5$ **then**
      $q_e \leftarrow q'_e$;
    else
      `enableConcreteComparison(`q_e`)`;
    end
  end
  $\mathcal{T} \leftarrow \mathcal{T} \cup T$;
  $Q_{proc} \leftarrow Q_{proc} \cup \{q_s \mid (q_s, q_e, s) \in T\}$;
  $Q_{next} \leftarrow \{q_e \mid (q_s, q_e, s) \in T, q_e \notin Q_{proc}\}$;
end
return $\mathcal{T}$

- input: `openDoors0..2` – open the doors at the respective floor;
- output: `carAtFloor0..2` – elevator car is at the respective floor;
- output: `doorClosed0..2` – the doors at the respective floor are completely closed.

Also there is a Real output variable `carPos` $\in [30; 419.5)$ discretized with intervals: [30; 30.5), [30.5; 224.5), [224.5; 225.5), [225.5; 418.5), [418.5; 419.5). The `carPos` output variable represents the concrete position of the car, its initial value is 30 (interval [30; 30.5)) that corresponds to the second floor. After each `moveDown`/`moveUp` command the position value increases/decreases by 1, respectively.

Plant model inputs correspond to controller outputs, controller inputs include all plant model outputs except `carPos`, and also include additional Boolean inputs `buttonPressed0..2` – whether the "request elevator" button is pressed at the respective floor. After the car reaches the floor where the button is pressed, the `buttonPressedN` value is set to `False`. To test the proposed algorithm, a special functionality of setting the plant to an arbitrarily chosen state was added to the simulation model.

The proposed algorithm generated the plant model in the form of an automaton with 40 states, which required a total of 135 seconds. Most of this time was spent on processing self-loops and cycles. The generated model was exported in the NuSMV format and composed with the manually prepared controller model to perform closed-loop verification. LTL properties that were verified are the same as the ones checked in [7] and are listed in Table II. All verification results are correct and no additional changes were applied to the resulting formal model.

V. Discussion and conclusion

Active learning approaches to plant model generation are quite perspective since their core idea is to refine the hypothesis model on every iteration and gather only those behavior examples that are required to build a consistent model. In this work a method for generating formal context-free deterministic plant models in a black-box scenario was introduced. It does not require an oracle or equivalence checks between source and hypothesis automata and generates reliable models.

On the other hand, note that model construction starts in some initial state, which means that the algorithm only discovers states reachable from the initial one. Thus, if it is guaranteed that the entire model state space is reachable from the chosen initial state, then the generated formal model for context-free deterministic plant will be reliable. Otherwise, there is a risk to discover only a sub-automaton or, in case of a disconnected state space, to fail to detect some of its parts. Meanwhile, the solution is quite straightforward if the set of possible initial states is known in advance – running the algorithm from each initial state and merging resulting automata will solve the issue. However, if the plant simulation model can be initialized in every possible state, all such states should be explored.

Also it should be noted that the suggested approach relies in its efficiency on the possibility to quickly reset the simulation model to the required state. If for some system this is impossible, at least a reset to the initial state must be available: in this case the method will still work, but each reset will take some time, depending on the simulation model implementation.

Another thing to mention is the time complexity of the algorithm. Since the proposed algorithm resembles BFS, all transitions must be executed for all input symbols from every new state, and the number of transitions will grow exponentially with the number of plant input variables. But, in fact,

TABLE II
ELEVATOR SYSTEM TEMPORAL PROPERTIES VERIFICATION RESULTS

	Temporal property	Comment	Correct value	Obtained value
	Plant temporal properties			
φ_1	$\mathbf{G}(\texttt{carAtFloor1} \wedge \mathbf{G}\ \neg\texttt{motorUp} \wedge \mathbf{G}(\texttt{motorDown} \vee \texttt{carAtFloor0}) \rightarrow \mathbf{G}\ \neg\texttt{carAtFloor2})$	If the car is on the first floor and never moves up and always moves down or stays on floor 0, it will never reach floor 2	+	+
φ_2	$\mathbf{G}(\mathbf{G}\ \neg\texttt{motorUp} \wedge \mathbf{G}(\texttt{motorDown} \vee \texttt{carAtFloor0}) \rightarrow \mathbf{F}\ \texttt{carAtFloor0})$	With similar conditions the car will reach floor 0	+	+
φ_3	$\mathbf{G}(\mathbf{G}\ \neg\texttt{motorDown} \wedge \mathbf{G}(\texttt{motorUp} \vee \texttt{carAtFloor2}) \rightarrow \mathbf{F}\ \texttt{carAtFloor2})$	Analogously, if we the car moves up, it will reach floor 2	+	+
φ_4	$\mathbf{G}\,\mathbf{F}\ \neg\texttt{motorDown}$	Controller cannot always send the "Down" command	−	−
φ_5	$\mathbf{G}(\texttt{carAtFloor1} \wedge \mathbf{G}\ \neg\texttt{motorUp} \wedge \mathbf{G}\,\texttt{motorDown} \rightarrow \mathbf{F}\ \texttt{carAtFloor2})$	Similar to condition 1, but $\mathbf{G}\,\texttt{motorDown}$ is used instead of $\mathbf{G}(\texttt{motorDown} \vee \texttt{carAtFloor0})$	−	−
φ_7	$\mathbf{G}(\texttt{carPos} = 4 \wedge \texttt{motorDown} \wedge \neg\texttt{motorUp} \rightarrow \mathbf{X}\,\texttt{carPos} = 3)$	If the car is on floor 2 and moves down, it will be between floors 1 and 2	+	+
φ_9	$\mathbf{G}(\texttt{carPos} = 2 \wedge \neg\texttt{motorDown} \wedge \texttt{motorUp} \rightarrow \mathbf{X}\,\texttt{carPos} = 3)$	Similarly, floor 1, motorUp	+	+
	Closed-loop model checking			
φ_{11}	$\forall k \in [0..2]\ \mathbf{G}(\texttt{buttonPressed}_\texttt{k} \rightarrow \mathbf{F}\,\texttt{carAtFloor}_\texttt{k})$	If the car is called, it will arrive to the specified floor	−	−
φ_{12}	$\mathbf{G}(\texttt{buttonPressed2} \wedge$ (not always at some floor) $\rightarrow \mathbf{F}\,\texttt{carAtFloor2})$	If the car is called to floor 2 and is not stuck at some floor it will arrive to floor 2	+	+
φ_{14}	$\mathbf{G}(\texttt{buttonPressed0} \wedge$ (not always at some floor) $\rightarrow \mathbf{F}\,\texttt{carAtFloor0})$	The same for floor 0. Because of controller choice the result differs	−	−
φ_{15}	$\mathbf{G}(\texttt{carPos} \in \{1, 3\} \rightarrow \texttt{doorClosed0} \wedge \texttt{doorClosed1} \wedge \texttt{doorClosed2})$	When the car is between floors, all doors are closed	+	+

not every transition ends in a new state. Commonly, some rule exists in input variable changes, therefore, detection of such rules makes it possible not to check all input symbols. Hence, future work will tackle reduction of the number of transitions to be checked during model construction. Another direction of future work is automating discretization intervals construction, which could be done on the basis of inferred model properties analysis and temporal properties verification results.

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Education and Science of the Russian Federation, project RFMEFI58716X0032.

REFERENCES

[1] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.
[2] S. Preusse, *Technologies for Engineering Manufacturing Systems Control in Closed Loop*. Logos Verlag Berlin GmbH, 2013, vol. 10.
[3] V. Vyatkin, H.-M. Hanisch, C. Pang, and C.-H. Yang, "Closed-loop modeling in future automation system engineering and validation," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 39, no. 1, pp. 17–28, 2009.
[4] A. Maier, "Online passive learning of timed automata for cyber-physical production systems," in *12th IEEE International Conference on Industrial Informatics*, 2014, pp. 60–66.
[5] I. Buzhinsky and V. Vyatkin, "Automatic inference of finite-state plant models from traces and temporal properties," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1521–1530, 2017.
[6] G. Giantamidis and S. Tripakis, "Learning moore machines from input-output traces," in *FM 2016: Formal Methods*. Cham: Springer International Publishing, 2016, pp. 291–309.
[7] D. Avdyukhin, D. Chivilikhin, G. Korneev, V. Ulyantsev, and A. Shalyto, "Plant trace generation for formal plant model inference: Methods and case study," in *IEEE 15th International Conference on Industrial Informatics*, 2017, pp. 746–752.
[8] C. de la Higuera, *Grammatical Inference. Learning automata and grammars*. Cambridge University Press, 2010.
[9] D. Angluin, "Queries and concept learning," *Machine Learning*, vol. 2, no. 4, pp. 319–342, 1988.
[10] B. Steffen, F. Howar, and M. Merten, "Introduction to automata learning from a practical perspective," *Formal Methods for Eternal Networked Software Systems*, pp. 256–296, 2011.
[11] R. Rivest and R. Schapire, "Inference of finite automata using homing sequences," *Information and Computation*, vol. 103, pp. 299–347, 1993.
[12] A. Nerode, "Linear automaton transformations," *Proceedings of the American Mathematical Society*, vol. 9, no. 4, pp. 541–544, 1958.
[13] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. International Conference on Computer-Aided Verification*, ser. LNCS, vol. 2404. Copenhagen, Denmark: Springer, 2002.
[14] Proposed algorithm implementation. [Online]. Available: https://github.com/ShakeAnApple/active-learning
[15] NxtControl. [Online]. Available: http://www.nxtcontrol.com

Efficient Symmetry Breaking for SAT-Based Minimum DFA Inference

Ilya Zakirzyanov[1,2(✉)], Antonio Morgado[3], Alexey Ignatiev[3,4], Vladimir Ulyantsev[1], and Joao Marques-Silva[3]

[1] ITMO University, St. Petersburg, Russia
{zakirzyanov,ulyantsev}@corp.ifmo.ru
[2] JetBrains Research, St. Petersburg, Russia
[3] Faculty of Science, University of Lisbon, Lisbon, Portugal
{ajmorgado,aignatiev,jpms}@ciencias.ulisboa.pt
[4] ISDCT SB RAS, Irkutsk, Russia

Abstract. Inference of deterministic finite automata (DFA) finds a wide range of important practical applications. In recent years, the use of SAT and SMT solvers for the minimum size DFA inference problem (MinDFA) enabled significant performance improvements. Nevertheless, there are many problems that are simply too difficult to solve to optimality with existing technologies. One fundamental difficulty of the MinDFA problem is the size of the search space. Moreover, another fundamental drawback of these approaches is the encoding size. This paper develops novel compact encodings for Symmetry Breaking of SAT-based approaches to MinDFA. The proposed encodings are shown to perform comparably in practice with the most efficient, but also significantly larger, symmetry breaking encodings.

Keywords: DFA inference · Boolean satisfiability · Symmetry breaking

1 Introduction

The inference of minimum-size deterministic finite automata (DFA) from (positive and negative) examples of their behavior has been investigated since the early days of computing, with continued improvements until the present day. The importance of topic is illustrated not only by recent improvements to tools for computing minimum-size DFAs [27,30], but also by recent and ever growing list of applications [29]. The problem of computing the minimum-size

IZ was supported by RFBR (project 18-37-00425). AM, AI and JMS were supported by FCT grants ABSOLV (PTDC/CCI-COM/28986/2017), FaultLocker (PTDC/CCI-COM/29300/2017), SAFETY (SFRH/BPD/120315/2016), and SAMPLE (CEECIND/04549/2017). VU was supported by the Government of Russia (Grant 08-08).

C. Martín-Vide et al. (Eds.): LATA 2019, LNCS 11417, pp. 159–173, 2019.
https://doi.org/10.1007/978-3-030-13435-8_12

DFA (MinDFA) witnessed seminal work in the early 70s [6]. Moreover, a number of visible contributions were made in the 90s. These include the use of graph coloring [8], constraint programming techniques [9,22], and state merging approaches [17,18]. Approaches based on SAT and SMT were proposed in the last decade, with promising results [12,13,20,21,25]. Nevertheless, the size of existing propositional encodings do not scale for large DFA inference problems. The use of SMT does not represent a clear improvement, since SMT solving approaches for the MinDFA problem will also encode to propositional logic. This paper revisits SAT encodings for the MinDFA problem as well as recent work on exploiting symmetry breaking [25,30], and proposes a (novel) tighter propositional representation of state-of-the-art symmetry breaking predicates, but it also devises new symmetry breaking constraints which serve to achieve more effective pruning of the search space. The new propositional encoding proposed in this paper enables clear performance gains over the state of the art [13,14,26,30].

The paper is organized as follows. Section 2 introduces the definitions used throughout the paper and briefly overviews related work. Section 3 develops new ideas to encode symmetry breaking predicates. Section 4 compares a new tool for the MinDFA problem with the existing state of the art, showing clear performance gains. Section 5 concludes the paper.

2 Background

2.1 Preliminaries

Throughout the paper we assume that automata are defined over some set of symbols Σ, also known as the *alphabet*. The number of symbols in the alphabet is $L = |\Sigma|$. For earlier DFA inference examples, it was often the case that $\Sigma = B = \{0, 1\}$ [18,22]. For more recent DFA inference examples [28], larger alphabets are often considered.

A *deterministic finite automaton* (DFA) is a tuple $\mathcal{D} = (D, \Sigma, \delta, d_1, D^+, D^-)$, where D is a finite set of states, Σ is the (input) alphabet, $\delta : D \times \Sigma \to D$ is the transition function, d_1 is the initial state, D^+ is the set of accepting states and $D^- = D \setminus D^+$ is the set of rejecting sets. For input strings $\pi \in \Sigma^*$ we define $\hat{\delta}(d_1, \pi)$ inductively as follows [16]: (i) $\hat{\delta}(d_1, \epsilon) = d_1$; (ii) If $\pi = \pi' c$, then $\hat{\delta}(d_1, \pi) = \delta(\hat{\delta}(d_1, \pi'), c)$.

We assume the standard setting of inferring a minimum-size DFA given a set of samples of its behavior [7,15], i.e. the training set, each sample represented by an input string that is either accepted or rejected by some DFA $\mathcal{U} = (U, \Sigma, \mu, u_1, U^+, U^-)$, which is not known. This form of learning is often referred to as *passive learning*, as opposed to *active learning* [2,20], which enables a learning algorithm (aiming to create a target DFA) to formulate queries to some teacher (which knows of the unknown DFA).

A *training set* is a set of pairs $\mathbb{T} = \{(\pi_1, o_1), \ldots, (\pi_R, o_R)\}$, where each pair $(\pi_r, o_r) \in \Sigma^* \times \{0, 1\}$ denotes the output o_r observed given input string π_r. If $o_r = 1$ ($o_r = 0$), then π_r is referred to as a *positive* (*negative*) example. Given

Function MinimumDFA($\mathcal{T}$)
Input : $\mathcal{T}$: APTA
Output: $\mathcal{S}$: minimum size DFA

```
1  M ← FindLowerBound(T)
2  while true do
3      S ← FindConsistentDFA(T, M)
4      if S ≠ ∅ then return S
5      M ← M + 1
```

Algorithm 1: General lower bound refinement algorithm

a training set, we can construct an APTA (*augmented prefix tree acceptor*) [1, 13,24], defined as the DFA $\mathcal{T} = (T, \Sigma, \tau, t_1, T^+, T^-)$, where any input string sharing the same prefix ends up in the same state. Concretely, given input strings $\pi_1 = \pi_a \pi_{b_1}$ and $\pi_2 = \pi_a \pi_{b_2}$ the common prefix π_a will be associated to a unique sequence of states in the APTA. For an APTA $\mathcal{T}$, we have $T^+ \cup T^- \neq T$, and we define $N = |T|$. When clear from the context, the states of $\mathcal{T}$ are referred to by their index, t_i by i, $i = 1, \ldots, N$. In some settings, $\theta(i)$ will be used to denote the distance from the APTA root state t_1 to state t_i.

The *minimum-size DFA* inference problem (MinDFA) is to identify a DFA $\mathcal{S} = (S, \Sigma, \sigma, s_1, S^+, S^-)$, with a minimum number of states, such that for any training pair (π_r, o_r), $\hat{\sigma}(s_1, \pi_r) \in S^+$ iff $o_r = 1$ and $\hat{\sigma}(s_1, \pi_r) \in S^-$ iff $o_r = 0$. For a prospective DFA $\mathcal{S}$, we define $M = |S|$.

Throughout the paper $[R]$ is used to denote the set $\{1, \ldots, R\}$, for some positive integer R. Moreover, we will use integers to refer to either symbols or states. For a given alphabet, by associating states and symbols with integers facilitates imposing a fixed lexicographic order, which will be required later in the paper (see Sect. 3). Additionally, standard SAT definitions are assumed and used [5].

2.2 Minimum Size DFA Inference

This paper focuses on constraint-based exact approaches for the MinDFA problem. Different constraint programming approaches for solving the MinDFA problem have been proposed over the years. More recently, the use of SAT [12–14] and SMT [20,21] has been investigated. A more detailed account of past work is available for example in Neider's PhD thesis [20, Chap. 3].

Algorithm 1 summarizes the most widely used approach for computing a minimum size DFA consistent with a given APTA $\mathcal{T}$ (obtained from the training set). Initially a lower bound on the size of the inferred DFA is computed. An often used heuristic is to compute a maximal clique on states of the APTA that cannot be assigned to the same DFA state [12–14,20–22,26]. Afterwards, starting from the lower bound and for each possible value on the number of states of the DFA, some algorithm decides whether there exists a DFA $\mathcal{S}$ which can be shown consistent with the samples of behavior summarized as the APTA $\mathcal{T}$.

Algorithm 1 is referred to as LSUS (linear-search, UNSAT until SAT) and is used in different settings. Other algorithms can be envisioned. These include binary search, assuming some upper bound is known or can be identified (e.g. with merge-based algorithms). Another alternative is unbounded search with a final binary search step. These algorithms have been used in recent years for solving MaxSAT [19] and for extracting MUSes [4]. The use of propositional encodings can be traced to the work of Grinchtein, Leucker & Piterman [12]. By using two different representations for integers, one in unary and the other in binary, this work proposes two propositional encodings. For the unary representation, the encoding size is in $\mathcal{O}(N \times M^2 + N^2 \times M)$ over $\mathcal{O}(N \times M)$ variables[1]. For the binary representation, the encoding size is in $\mathcal{O}(N \times M \times \log M + N^2 \times M)$ on $\mathcal{O}(N \times \log M)$ variables. More recent work by Heule&Verwer (HV) [13,14] proposed encodings that have been shown effective in practice [28]. The HV encoding builds on the graph coloring analogy proposed in earlier work [8]. The proposed encoding has size $\mathcal{O}(M^3 + N \times M^2)$ over $\mathcal{O}(M^2 + N \times M)$ variables. This encoding is revisited in Sect. 2.3.

2.3 SAT-Based MinDFA

Given an APTA $\mathcal{T}$ and a bound M on the number of states of the inferred DFA $\mathcal{S}$, this subsection provides a derivation of the HV encoding [13,14], based on a different motivation. By careful analysis of this formulation, we achieve a more compact propositional encoding. Instead of relating the MinDFA problem with graph coloring, we formulate it as the problem of matching the N states of the APTA $\mathcal{T}$ to the M states of a target DFA $\mathcal{S}$. The sets of variables of the propositional encoding are as follows:

1. $m_{i,p}$ which is 1 iff state t_i in $\mathcal{T}$ is matched with state s_p in $\mathcal{S}$.
2. $e_{v,p,q}$ which is 1 iff there is a transition from s_p to s_q on symbol l_v in $\mathcal{S}$.
3. a_p which is 1 iff s_p is accepting in $\mathcal{S}$.

The constraints of the proposed encoding are summarized in Table 1. Observe that for encoding the Equals1 constraints, [14] uses a clause to encode an AtLeast1 constraint, and the Pairwise Encoding for encoding an AtMost1 constraint. A simple improvment is to use a more compact encoding, among the many that exist. Concrete examples include sequential counters [23], cardinality networks [3], the ladder encoding [11], sorting networks [10], among several other options. As can be concluded, the proposed encoding grows with $\mathcal{O}(N \times M^2)$. Thus, the encoding is asymptotically (somewhat) tighter than the encoding proposed in [13], in that the encoding of the cardinality constraints changes from $\mathcal{O}(M^3)$ to $\mathcal{O}(M^2)$. This difference can be significant for large values of M. As observed in earlier work [13,14], for some benchmarks [18], the target DFA has hundreds of states, and so an encoding in $\mathcal{O}(M^3)$ is expected to be beyond the memory capacity of existing compute servers. It is straightforward to map the

[1] The encoding size shown is adapted from the results in [20], taking into account that both $|T^+|$ and $|T^-|$ can grow with $N = |T|$. The size of $|\Sigma|$ is assumed constant.

Table 1. Constraints of the SAT encoding

Constraint	Range	
$(\sum_{p=1}^{M} m_{i,p}) = 1$	$i \in [N]$	Each state t_i in $\mathcal{T}$ is matched with exactly one state in $\mathcal{S}$
$m_{i,p} \rightarrow a_p$	$i \in [N]$; $t_i \in \mathcal{T}^+$; $p \in [M]$	Each accepting state t_i in $\mathcal{T}$ is matched with an accepting state in $\mathcal{S}$
$m_{i,p} \rightarrow \neg a_p$	$i \in [N]$; $t_i \in \mathcal{T}^-$; $p \in [M]$	Each rejecting state t_i in $\mathcal{T}$ is matched with a rejecting state in $\mathcal{S}$
$(\sum_{q=1}^{M} e_{v,p,q}) = 1$	$v \in [L]$; $p \in [M]$	There is exactly one transition from s_p on some symbol l_v in $\mathcal{S}$
$m_{i,p} \wedge m_{k,q} \rightarrow e_{v,p,q}$	$i,k \in [N]$; $v \in [L]$; $\sigma(t_i, l_v) = t_k$; $p,q \in [M]$	A transition between t_i and t_k on l_v in $\mathcal{T}$ forces a transition between its mapped nodes on the same l_v in $\mathcal{S}$
$m_{i,p} \wedge e_{v,p,q} \rightarrow m_{k,q}$	$i,k \in [N]$; $v \in [L]$; $\sigma(t_i, l_v) = t_k$; $p,q \in [M]$	A transition between t_i and t_k on l_v in $\mathcal{T}$, with a transition between the mapped state p and a state q on l_v in $\mathcal{S}$, forces a mapping between t_k and q

sets of clauses in the HV formulation [13,14] into the constraints described above. The main difference is that we explicitly use a tighter encoding for the AtMost1 constraints, which are listed as sets of clauses (capturing the well-known pairwise encoding) in [13]. Additionally, the HV formulation [13] considers different sets of redundant constraints to the basic formulation above. A technique that has been proposed for the SAT formulation is the breaking of symmetries of the DFA constructed [26,30]. Symmetry breaking for the SAT formulation is described in depth in Sect. 3, together with new improvements.

3 Efficient Symmetry Breaking

This section revisits recent symmetry breaking for the MinDFA problem, which imposes an order on the states of the DFA [26,30]. Although effective in practice, the existing propositional encoding is not tight, and so unlikely to scale for larger DFAs. Section 3.2 develops a significantly tighter encoding. Section 3.3 devises novel constraints that serve to furhter prune the search space that a SAT solver needs to explore.

3.1 Propositional Formulation for Breaking Symmetries

This section summarizes the recent work on breaking symmetries of the DFA being constructed, by imposing an ordering on the states of the DFA [26,30]. In this section we follow the original formulation [26]. The approach can be

formalized as follows. Assume a target DFA $\mathcal{S} = (S, \Sigma, \sigma, s_1, S^+, S^-)$. The states of the DFA $\mathcal{S}$ are required to be numbered according to the tree induced by a breadth-first search (BFS) of the target DFA. As a result, the formulation of symmetry breaking depends only on the states and transitions of the target DFA $\mathcal{S}$ (independent of the APTA $\mathcal{T}$). In this section we require some fixed (e.g. lexicographic) ordering on the symbols of Σ. Any order of the symbols is valid. The symbols will be numbered from 1 to L, but the numbers respect the fixed ordering.

The propositional variables used in the formulation are as follows:

1. $p_{q,r}$, with $1 \leq r < q \leq M$. $p_{q,r} = 1$ iff state r is the parent of q in the BFS tree.
2. $t_{p,q}$, with $1 \leq p < q \leq M$. $t_{p,q} = 1$ iff there is a transition from p to q in $\mathcal{S}$.
3. $m_{v,p,q}$, with $v \in \Sigma$ and $1 \leq p < q \leq M$. $m_{v,p,q} = 1$ iff there is a transition from state p to state q on symbol l_v and there is no such transition with a lexicographically smaller symbol.

The clauses of the propositional formulation are summarized in Eqs. (1–6).

$$\bigwedge_{2 \leq q \leq M} (p_{q,1} \vee p_{q,2} \vee \cdots \vee p_{q,q-1}) \tag{1}$$

$$\bigwedge_{1 \leq r < s < q < M} (p_{q,s} \rightarrow \neg p_{q+1,r}) \tag{2}$$

$$\bigwedge_{1 \leq r < q \leq M} (t_{r,q} \leftrightarrow e_{1,r,q} \vee \cdots \vee e_{L,r,q}) \tag{3}$$

$$\bigwedge_{1 \leq r < q \leq M} (p_{q,r} \leftrightarrow t_{r,q} \wedge \neg t_{r-1,q} \wedge \cdots \wedge \neg t_{1,q}) \tag{4}$$

$$\bigwedge_{1 \leq r < q \leq M} \bigwedge_{1 \leq v \leq L} (m_{v,r,q} \leftrightarrow e_{v,r,q} \wedge \neg e_{v-1,r,q} \wedge \cdots \wedge \neg e_{1,r,q}) \tag{5}$$

$$\bigwedge_{1 \leq r < q < M} \bigwedge_{1 \leq u < v \leq L} (p_{q,r} \wedge p_{q+1,r} \wedge m_{v,r,q} \rightarrow \neg m_{u,r,q+1}) \tag{6}$$

There are six types of conjunction of clauses considered. (1) relates to the states, and with the exception of the initial state (numbered 1), each clause says that a state must have a parent with smaller number. (2) says that a state q must be enqueued (in the BFS traversal) before the next state $q + 1$, and so the parent r of $q + 1$ cannot be less than the parent s of q. (3) and (4) define the $t_{q,r}$ variables based on the $e_{v,q,r}$ variables and relate them to the parent variables $p_{q,r}$. (5) defines the $m_{v,p,q}$ variables using DFA transitions, and the (6) imposes consecutive states q and $q+1$ with the same parent r to be arranged in the order of the symbols. It is plain to conclude that the size of the encoding grows with $\mathcal{O}(M^3 + M^2L + M^2L^2)$. Observe that the contribution of M^3, which dominates the other components assuming $L \ll M$, results from (2) and (4). Moreover, when $|\Sigma| = 2$, [26] proposes to replace (5) and (6) with

$$\bigwedge_{1 \leq r < q < M} (p_{q,r} \wedge p_{q+1,r} \rightarrow e_{1,r,q}) \tag{7}$$

3.2 A Tighter SAT Encoding

A propositional encoding in $\mathcal{O}(M^3 + M^2L^2 + M^2L)$ is impractical for the larger DFA inference instances [18,28]. This section shows how to modify the symmetry

breaking propositional encoding of Sect. 3.1 such that the encoding size becomes $\mathcal{O}(M^2L)$. The new encoding develops alternative representations for (2) and the (4), but also for (5) and (6). In addition, one needs to require:

$$\sum_{r=1}^{q-1} p_{q,r} = 1 \;\; 1 < q \leq M \tag{8}$$

We first investigate the encoding of (2) and (4). We can view the values of $p_{q,r}$, with $1 \leq r \leq q-1$, as a binary string, with $q-1$ bits, and compare this string with the one of $p_{q+1,r}$, with $1 \leq r \leq q$, and so with q bits. We introduce $p_{q,q} = 0$, and so can also view the values of $p_{q,r}$ as a binary string with q bits (same size).

Observe that (2) encodes the value associated to the binary string of the $p_{q,r}$ variables to be smaller or equal than the value associated to the binary string of the $p_{q+1,r}$ variables. To compare the binary strings, we inspect the bits in order, starting at position q, and moving down to position 1. We consider variables $ng_{q,r}$, such that $ng_{q,r} = 1$ iff the most significant $q-r+1$ bits of the string associated with $p_{q,r}$ are lexicographically *no greater* than those of $p_{q+1,r}$. The value associated to the binary string of the $p_{q,r}$ variables is smaller or equal than the value associated to the binary string of the $p_{q+1,r}$ variables iff $ng_{q,1} = 1$ holds. Since we enforce $p_{q,q} = 0$, then we must have $ng_{q,q} = 1$. Moreover, we also require $ng_{q,1} \leftrightarrow 1$. Thus we obtain:

$$(ng_{q,1} \leftrightarrow 1) \land (ng_{q,q} \leftrightarrow 1) \land \bigwedge_{1 \leq r < q} (ng_{q,r} \leftrightarrow ng_{q,r+1} \land eq_{q,r} \lor p_{q,r} \land \neg p_{q+1,r}) \tag{9}$$

where, $eq_{q,r} \leftrightarrow (p_{q,r} \leftrightarrow p_{q+1,r})$.

Second, a similar approach can be exploited for encoding of (4). We introduce variables $nt_{r,q}$, where $nt_{r,q} = 1$ iff there exists *no* $t_{s,q} = 1$ with $s < r$. Thus, $nt_{r,q}$ can be defined inductively as follows:

$$(nt_{0,q} \leftrightarrow 1) \land \bigwedge_{1 < r < q} (nt_{r,q} \leftrightarrow nt_{r-1,q} \land \neg t_{r,q}) \tag{10}$$

Thus, (4) can be rewritten, using the $nt_{r,q}$ variables as follows:

$$p_{q,r} \leftrightarrow t_{r,q} \land nt_{r-1,q} \tag{11}$$

As can be concluded, by using auxiliary variables $ng_{q,r}$ and $nt_{r,q}$, and Eqs. (9), (10) and (11), we achieve an overall propositional encoding in $\mathcal{O}(M^2L + M^2L^2)$.

However, we can tighten further the propositional encoding for breaking symmetries using a BFS tree. This is achieved by devising alternative encodings for (5) and (6). As shown next, this yields a propositional encoding in $\mathcal{O}(M^2L)$. With respect to (5), we use the additional variables $ne_{v,r,q}$ such that $ne_{v,r,q} = 1$ iff all variables $e_{u,r,q} = 0$ with $u < v$, i.e. there are *no* variables $e_{u,r,q}$ taking value 1, when $u < v$.

$$(ne_{1,r,q} \leftrightarrow \neg e_{1,r,q}) \land \bigwedge_{1 < v < L} (ne_{v,r,q} \leftrightarrow \neg e_{v,r,q} \lor ne_{v-1,r,q}) \tag{12}$$

Thus given (12), (5) can be rewritten as follows:

$$\bigwedge_{1\leq r<q\leq M} \bigwedge_{1\leq v\leq L} (m_{v,r,q} \leftrightarrow e_{v,r,q} \wedge ne_{v-1,r,q}) \tag{13}$$

With respect to (6), we use the additional variables $zm_{v,r,q}$ such that $zm_{v,r,q} = 1$ iff all variables $m_{u,r,q}$ are 0-*valued*, $m_{u,r,q} = 0$, for $u < v$.

$$(zm_{1,r,q} \leftrightarrow \neg m_{1,r,q}) \wedge \bigwedge_{1<v<L} (zm_{v,r,q} \leftrightarrow \neg m_{v,r,q} \wedge zm_{v-1,r,q}) \tag{14}$$

Thus given (14), (6) can be rewritten as follows:

$$\bigwedge_{1\leq r<q\leq M} \bigwedge_{1\leq v\leq L} (p_{q,r} \wedge p_{j+1,r} \wedge m_{v,r,q} \rightarrow zm_{v-1,r,q+1}) \tag{15}$$

One can thus conclude that the resulting propositional encoding size is in $\mathcal{O}(M^2L)$.

3.3 Exploiting BFS-Based Breaking of Symmetries

This section investigates techniques for developing additional constraints when imposing the ordering of states dictated by a BFS tree of the DFA. Figure 1 shows a possible BFS tree illustrating the *largest* state numbers that can be the children of some other state. The additional constraints proposed in this section will relate with Fig. 1.

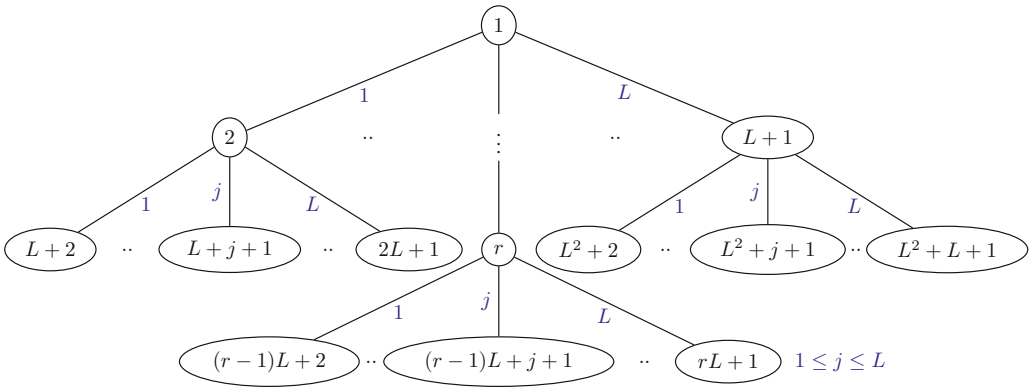


Fig. 1. (Worst case) BFS tree with the largest state numbers that can be the children of some other state. Note that $1 \leq j < L$.

BFS-Induced Properties. Although we have introduced $p_{q,r}$ such that $r < q \leq M$, it is possible to refine the range of q given r.

Property 1. Given a state r, with $1 \leq r \leq M$, in the BFS tree, r can be the parent of states in the range $r + 1$ to $rL + 1$.

Figure 1 illustrates the argument for the upper bound on the number of the children of r. We can conclude that the value of $p_{q,r}$ can be non-zero for $r+1 \leq q \leq rL+1$, which also impacts the possible values of some of the $e_{v,r,q}$ and the $t_{r,q}$ variables.

Property 2. For $q > rL+1$ and $v \in [L]$, then $p_{q,r} = 0$, $e_{v,r,q} = 0$, and $t_{r,q} = 0$.

Given that the BFS tree assumes a fixed ordering not only on the states but also on the input alphabet, it is possible to identify other transitions that must be forced to value 0 (based on the ordering of the symbols). Hence, we have the following.

Property 3. $e_{v,r,rL+2-j} = 0$ for $j \in [L-1]$ and $v \in [L-j]$.

The above observations enable to devise the additional constraints described in the remainder of this section. The constraints are organized as *shape* or *range*, but also result from information from the APTA and the BFS distance.

Shape Constraints. The possible values of $p_{q,r}$ respect a *continuity* property, dictated by the BFS traversal, in that all children of r are consecutively numbered, and there can be *at most* L of these. This continuity property can be encoded using additional variables. Let $lnp_{q,r}$ be assigned value 1 iff r is the parent of $q+1$ but not of q (lnp stands for *left-no-parent*). Thus,

$$\neg p_{q,r} \wedge p_{q+1,r} \rightarrow lnp_{q,r} \tag{16}$$

Moreover, we have the following:

$$(lnp_{q,r} \rightarrow \neg p_{q,r}) \wedge \bigwedge_{r+1<q\leq M} (lnp_{q,r} \rightarrow lnp_{q-1,r}) \tag{17}$$

Thus, $lnp_{q,r}$ is 1 from $q = 1$ until the value of q such that $p_{q+1,r}$ holds.

In a similar fashion, let $rnp_{q,r}$ be assigned value 1 if and only if r is the parent of $q-1$ but not of q (in this case, rnp stands for *right-no-parent*). Thus,

$$p_{q-1,r} \wedge \neg p_{q,r} \rightarrow rnp_{q,r} \tag{18}$$

Similarly to the previous case, one can exploit the $rnp_{q,r}$ variables, and derive the following constraints:

$$\begin{array}{ll} rnp_{q,r} \rightarrow rnp_{q+1,r} & r \leq q < M \\ rnp_{q,r} \rightarrow \neg p_{q,r} & \\ rnp_{q,r} \rightarrow \neg e_{v,q,r} & v \in [L] \end{array} \tag{19}$$

Thus, $rnp_{q,r}$ is 1 from $q = M$ until the value of q such that $p_{q-1,r}$ holds.

Another observation is that r can be the parent of at most L states, due to L outgoing transitions. As a result, we get,

$$\begin{array}{l} p_{q,r} \rightarrow rnp_{q+L,r} \quad \text{if } q+L \leq M \\ p_{q,r} \rightarrow lnp_{q-L,r} \ \text{if } q-L \geq r+1 \end{array} \tag{20}$$

The $lnp_{q,r}$ and $rnp_{q,r}$ variables serve to force $p_{q,r}$ variables to be assigned value 0. However, under some circumstances, we can infer that some $p_{q,r}$ variables must be assigned value 1. For example, for the range of values of q for which both $lnp_{q,r}$ and $rnp_{q,r}$ are 0, the value of $p_{q,r}$ must be 1. Thus,

$$\neg lnp_{q_1,r} \wedge \neg rnp_{q_2,r} \rightarrow p_{q',r} \qquad \begin{array}{c} q_1 < q' < q_2 \\ q_1 < q_2 \leq \min(q_1 + L - 1, rL + 1, M) \\ r + 1 \leq q_1 < \min(rL + 1, M) \end{array} \tag{21}$$

For any q_1, q_2 can range from $q_1 + 1$ to at most $q_1 + L$. Similarly, we can write,

$$p_{q,r} \wedge p_{s,r} \rightarrow p_{s-1,r} \qquad \begin{array}{c} q < s \leq \min(q + L - 1, rL + 1, M) \\ r + 1 \leq q < \min(rL + 1, M) \end{array} \tag{22}$$

As above, for any r, s can range from $r + 1$ to at most $r + L$.

Range Constraints. Given a reference state r, we have shown above that the states of which r can be a parent of range from $r + 1$ until $rL + 1$. Moreover, we also know there is a continuity property, which causes r to be the parent of at most L states, numbered consecutively. This information can be used for constraining the $p_{q,r}$ variables, between states for which r cannot be a parent, as follows,

$$p_{q,r} \rightarrow \neg p_{q+L,r} \qquad q \in \{l \,|\, (l \geq r + 1) \wedge (l + L \leq M) \wedge (l + L \leq rL + 1)\} \tag{23}$$

In addition, we get the following stronger condition by directly forcing the value of $e_{v,r,q}$ variables,

$$p_{q,r} \rightarrow \neg e_{v,r,q+L} \qquad \begin{array}{c} q \in \{l \,|\, (l \geq r + 1) \wedge (l + L \leq M) \wedge (l + L \leq rL + 1)\} \\ v \in [L] \end{array} \tag{24}$$

Furthermore, we can exploit Property 3, and the imposed ordering of the symbols in the BFS to identify a similar extension to (24) as follows,

$$p_{q,r} \rightarrow \neg e_{v,r,q+j} \quad \begin{array}{c} r + 1 \leq q \leq \min(rL + 1, M) \\ j \in \{l \,|\, l \in [L - 1] \wedge (q + l \leq M) \wedge (q + l \leq rL + 1)\} \\ v \in [j] \end{array} \tag{25}$$

Minimum BFS Distance. Given the way the BFS vertices are visited, one can guarantee a minimum BFS shortest path distance for each state. For state q, the shortest BFS path length is given by $D_{\min}(q) = \lceil \log_L (q(L-1)+1) - 1 \rceil$, with $q > 1$, i.e. no matter how the BFS is organized starting at state 1, the shortest path from 1 to q is never less than $D_{\min}(q)$. As a result, if $D_{\min}(q) > \theta(i)$, then $m_{i,q} = 0$. Observe that, under any possible setting in the DFA, the shortest path to q is larger than the distance to state i in the APTA. Thus, to get to q it would require more transitions that those allowed to get from inital state to i.

Exploiting APTA Information. By exploiting the variables and constraints used for breaking symmetries and using a BFS tree on the target DFA, we can devise additional constraints. Observe that, if the depth of a state i in the APTA is some value K, then in the DFA, we *must* be able to move from 1 to q in K of fewer transitions. However, if the shortest path from 1 to q in the DFA exceeds the depth K of vertex of i in the APTA, then it would be impossible to move from state 1 to state q in K or fewer transitions.

We consider the propositional variables $d_{q,j}$, with $q \in [M]$ and $1 \leq j < q$, such that $d_{q,j} = 1$ iff the length of the shortest path in the BFS tree from state 1 to q is j. Moreover, we consider propositional variables $se_{q,j}$, with $q \in [M]$ and $1 \leq j < q$, such that $se_{q,j} = 1$ iff the length of the shortest path in the BFS tree from state 1 to q is *smaller* than or *equal* to j. We can use an inductive definition for $se_{q,j}$ as follows:

$$se_{q,0} \leftrightarrow 0 \quad \text{and} \quad se_{q,j} \leftrightarrow se_{q,j-1} \vee d_{q,j} \tag{26}$$

Similarly to Sect. 3.2, we devise a tight encoding for the definition of the $d_{q,j}$ variables, suitable for larger problem instances. The insight is to introduce additional variables, which are inductively defined. Let $er_{q,r,j}$ be such that $er_{q,r,j} = 1$ iff there *exists* some index $r < q$ such that $p_{q,r} = 1$ and $d_{r,j} = 1$.

$$\begin{aligned} &er_{q,r,j} \leftrightarrow p_{q,r} \wedge d_{r,j} \vee er_{q,r+1,j} \qquad j < r < q-1 \\ &er_{q,q-1,j} \leftrightarrow p_{q,q-1} \wedge d_{q-1,j} \end{aligned} \tag{27}$$

we can now derive constraints on the $m_{i,p}$ variables. Let t_i be a state of the APTA such that the depth of t_i is I. We can define $d_{q,j}$ as follows:

$$d_{q,j} \leftrightarrow \neg se_{q,j-1} \wedge er_{q,j,j-1} \tag{28}$$

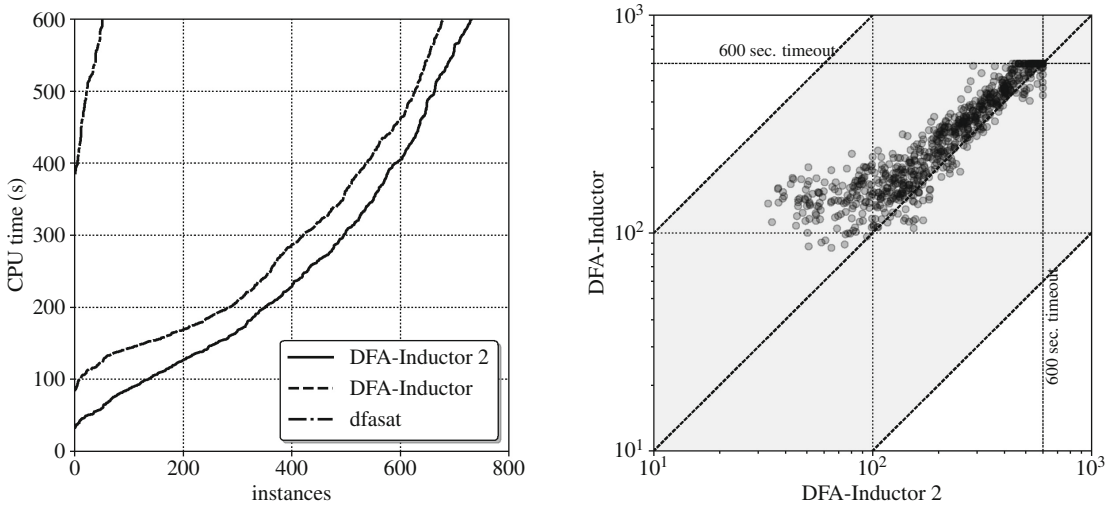
$$\neg se_{q,I} \rightarrow \neg m_{i,q} \tag{29}$$

One can conclude that the modified constraints have an encoding size in $\mathcal{O}(N \times M^2)$.

4 Experimental Results

This section evaluates the ideas described above, namely a compact SAT encoding and symmetry breaking predicates for solving the MinDFA problem. For this, the ideas were implemented on top of a known MinDFA solver called *DFA-Inductor* [26,30] written in Java[2]. The new prototype is referred to as *DFA-Inductor 2*. For comparison, two competitors were considered: the original DFA-Inductor and also *dfasat* [13]. All the selected tools apply the Glucose 4.1[3] SAT solver iteratively and *non-incrementally*, i.e. each call to the oracle is made *from scratch*. All the conducted experiments were performed in Ubuntu Linux on an AMD Opteron 6378 2.40 GHz processor with 496GByte of memory. For

[2] https://github.com/ctlab/DFA-Inductor.
[3] http://www.labri.fr/perso/lsimon/glucose/.



(a) Cactus plot

(b) DFA-Inductor vs. DFA-Inductor 2

Fig. 2. Detailed performance of dfasat, DFA-Inductor, and DFA-Inductor 2

each individual process, the time limit was set to 600 s and the memory limit to 1 GByte. For the comparison, a number of benchmark instances were randomly generated, following the procedure described in [30]. Concretely, starting from a randomly generated APTA of *even* size N, $N \in [20, 36]$, $50 \times N$ samples were generated. The size of the Σ is two. For each even number $N \in [20, 36]$, exactly 100 benchmark instances were created such that given value N, the resulting DFA for each of the corresponding 100 instances is guaranteed to be N. This way, the number of benchmark families defined by values N is 9. Thus, the total number of instances considered is 900. Figure 2a shows a cactus plot depicting the performance of all the selected solvers. As one can observe, dfasat is significantly outperformed by the compact encoding implemented in DFA-Inductor. In total, dfasat is able to solve only 51 benchmark instances (out of 900). Also observe that the symmetry breaking predicates described above further improve the performance of DFA-Inductor (see DFA-Inductor 2 compared to DFA-Inductor in the Fig. 2a). A comparison between DFA-Inductor and DFA-Inductor 2 is detailed in Fig. 2b and also in Table 2. Except for a few outliers, the symmetry breaking predicates of DFA-Inductor 2 are responsible for 20–40% performance improvement on average. Also it is important to note that the harder the problems are, the smaller is the performance gap between the two configurations. Although this can be seen as a drawback, the phenomenon requires further investigation on the use of symmetry breaking with various SAT solvers and a multitude of families benchmark sets. In total, the number of instances solved by DFA-Inductor and DFA-Inductor 2 is 678 and 731, respectively, thus, comprising a gap of 53 benchmark instances. Therefore, symmetry breaking brings more 7.2% instances solved.

Table 2. The effect of applying the symmetry breaking predicates described above. The solver configuration using the proposed symmetry breaking is referred to as *DFA-Inductor 2* and compared to the base configuration, i.e. *DFA-Inductor*. If an instance is timed out, its contribution to the average time of the corresponding benchmark family is assumed to be 600 s. The corresponding values are written in *italic*.

N	DFA-Inductor				DFA-Inductor 2			
	min	*avg*	*max*	*# solved*	*min*	*avg*	*max*	*# solved*
20	86.8	148.3	221.0	**100**	33.3	91.9	228.4	**100**
22	85.5	*147.1*	—	**99**	49.2	*100.4*	—	**99**
24	128.6	181.5	287.8	**100**	80.4	136.8	262.5	**100**
26	158.1	*251.8*	—	**99**	114.8	*209.3*	—	**99**
28	223.4	317.9	534.5	**100**	164.2	*268.9*	—	99
30	307.2	*443.8*	—	91	227.1	*389.2*	—	**95**
32	326.0	*506.5*	—	76	249.2	*447.4*	—	**86**
34	414.5	*591.1*	—	13	392.1	*569.9*	—	**41**
36	—	*600.0*	—	0	448.4	*594.8*	—	**12**

5 Conclusions

This paper proposes a number of novel techniques for encoding and reasoning about symmetries when exploiting SAT oracles for inferring minimum-size DFAs. The experimental results provide evidence of the improvements that can be achieved when compared with the state of the art [26,30], also enabling significant gains over the best exact methods proposed in recent years [13]. The novel symmetry-breaking ideas described in the paper can be applied to other approaches for inferring minimum-size DFAs, including the use of SMT solvers [20], and also in other settings.

References

1. Abela, J., Coste, F., Spina, S.: Mutually compatible and incompatible merges for the search of the smallest consistent DFA. In: ICGI, pp. 28–39 (2004)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
3. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks: a theoretical and empirical study. Constraints **16**(2), 195–221 (2011)
4. Belov, A., Lynce, I., Marques-Silva, J.: Towards efficient MUS extraction. AI Commun. **25**(2), 97–116 (2012)
5. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)

6. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Comput. **21**(6), 592–597 (1972)
7. Bugalho, M.M.F., Oliveira, A.L.: Inference of regular languages using state merging algorithms with search. Pattern Recognit. **38**(9), 1457–1467 (2005)
8. Coste, F., Nicolas, J.: Regular inference as a graph coloring problem. In: IWGI (1997)
9. Coste, F., Nicolas, J.: How considering incompatible state mergings may reduce the DFA induction search tree. In: ICGI, pp. 199–210 (1998)
10. Eén, N., Sörensson, N.: Translating Pseudo-Boolean constraints into SAT. JSAT **2**(1–4), 1–26 (2006)
11. Gent, I.P., Nightingale, P.: A new encoding of all different into SAT. In: Workshop on Modelling and Reformulating Constraint Satisfaction Problems, pp. 95–110 (2004)
12. Grinchtein, O., Leucker, M., Piterman, N.: Inferring network invariants automatically. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 483–497. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_40
13. Heule, M.J.H., Verwer, S.: Exact DFA identification using SAT solvers. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS (LNAI), vol. 6339, pp. 66–79. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15488-1_7
14. Heule, M., Verwer, S.: Software model synthesis using satisfiability solvers. Empir. Softw. Eng. **18**(4), 825–856 (2013)
15. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognit. **38**(9), 1332–1348 (2005)
16. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation - International Edition, 2nd edn. Addison-Wesley, Boston (2003)
17. Lang, K.J.: Faster algorithms for finding minimal consistent DFAs. Technical report, NEC Research Institute (1999)
18. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V., Slutzki, G. (eds.) ICGI 1998. LNCS, vol. 1433, pp. 1–12. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054059
19. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: a survey and assessment. Constraints **18**(4), 478–534 (2013)
20. Neider, D.: Applications of automata learning in verification and synthesis. Ph.D. thesis, RWTH Aachen University (2014)
21. Neider, D., Jansen, N.: Regular model checking using solver technologies and automata learning. In: NFM, pp. 16–31 (2013)
22. Oliveira, A.L., Marques-Silva, J.: Efficient algorithms for the inference of minimum size DFAs. Mach. Learn. **44**(1/2), 93–119 (2001)
23. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005). https://doi.org/10.1007/11564751_73
24. Trakhtenbrot, B.A., Barzdin, Y.M.: Finite Automata: Behavior and Synthesis. North-Holland Publishing Company, Amsterdam (1973)
25. Ulyantsev, V., Tsarev, F.: Extended finite-state machine induction using SAT-solver. In: ICMLA, pp. 346–349 (2011)

26. Ulyantsev, V., Zakirzyanov, I., Shalyto, A.: BFS-based symmetry breaking predicates for DFA identification. In: Dediu, A.-H., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) LATA 2015. LNCS, vol. 8977, pp. 611–622. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15579-1_48
27. Verwer, S., Hammerschmidt, C.A.: flexfringe: a passive automaton learning package. In: ICSME, pp. 638–642 (2017)
28. Walkinshaw, N., Lambeau, B., Damas, C., Bogdanov, K., Dupont, P.: STAMINA: a competition to encourage the development and assessment of software model inference techniques. Empir. Softw. Eng. **18**(4), 791–824 (2013)
29. Wieman, R., Aniche, M.F., Lobbezoo, W., Verwer, S., van Deursen, A.: An experience report on applying passive learning in a large-scale payment company. In: ICSME, pp. 564–573 (2017)
30. Zakirzyanov, I., Shalyto, A., Ulyantsev, V.: Finding all minimum-size DFA consistent with given examples: SAT-based approach. In: Cerone, A., Roveri, M. (eds.) SEFM 2017. LNCS, vol. 10729, pp. 117–131. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74781-1_9

УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
май–июнь 2020 Том 20 № 3 ISSN 2226-1494 http://ntv.itmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
May–June 2020 Vol. 20 No 3 ISSN 2226-1494 http://ntv.itmo.ru/en/
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК УНИВЕРСИТЕТ ИТМО ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

УДК 004.832.25 doi: 10.17586/2226-1494-2020-20-3-394-401

ПОСТРОЕНИЕ ДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ ПО ПРИМЕРАМ ПОВЕДЕНИЯ С ИСПОЛЬЗОВАНИЕМ ПОДХОДА УТОЧНЕНИЯ АБСТРАКЦИИ ПО КОНТРПРИМЕРАМ

И.Т. Закирзянов

Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
Адрес для переписки: ilya.zakirzyanov@gmail.com

Информация о статье
Поступила в редакцию 23.03.20, принята к печати 28.04.20
Язык статьи — русский

Ссылка для цитирования: Закирзянов И.Т. Построение детерминированных конечных автоматов по примерам поведения с использованием подхода уточнения абстракции по контрпримерам // Научно-технический вестник информационных технологий, механики и оптики. 2020. Т. 20. № 3. С. 394–401. doi: 10.17586/2226-1494-2020-20-3-394-401

Аннотация

Предмет исследования. Рассмотрена задача построения детерминированного конечного автомата минимального размера по примерам поведения. Разработан и реализован комбинированный метод решения данной задачи на основе сведения к задаче выполнимости булевых формул и с использованием подхода уточнения абстракции по контрпримерам. **Метод.** Предлагается в качестве исходных данных использовать не все примеры поведения сразу, а начинать с какого-то их подмножества, и строить соответствующий автомат, используя метод на основе сведения к задаче выполнимости. Затем построенный автомат проверяется на соответствие всем остальным примерам поведения. Те примеры, на которых поведение автомата расходится с желаемым, являются контрпримерами. Часть контрпримеров добавляется к исходным примерам поведения и процесс повторяется. **Основные результаты.** Предложенный метод реализован как часть программного комплекса для решения задачи построения детерминированного конечного автомата по примерам поведения на языке Python. Проведено экспериментальное сравнение разработанного метода с методом, основанным на сведении к задаче выполнимости булевых формул, но без использования подхода уточнения абстракции. **Практическая значимость.** Экспериментальное исследование показало, что разработанный метод целесообразно использовать при условии, что число примеров поведения достаточно велико — хотя бы в двести раз превышает количество состояний автомата, — и, как следствие, строящаяся булева формула содержит десятки и сотни миллионов дизъюнктов.

Ключевые слова
построение детерминированного конечного автомата, задача выполнимости, уточнение абстракции по контрпримерам, нарушение симметрии, грамматический вывод, машинное обучение

Благодарности
Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 18-37-00425.

doi: 10.17586/2226-1494-2020-20-3-394-401

DETERMINISTIC FINITE AUTOMATA LEARNING USING COUNTEREXAMPLE GUIDED ABSTRACTION REFINEMENT

I.T. Zakirzyanov

ITMO University, Saint Petersburg, 197101, Russian Federation
Corresponding author: ilya.zakirzyanov@gmail.com

Article info
Received 23.03.20, accepted 28.04.20
Article in Russian

For citation: Zakirzyanov I.T. Deterministic finite automata learning using counterexample guided abstraction refinement. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2020, vol. 20, no. 3, pp. 394–401 (in Russian). doi: 10.17586/2226-1494-2020-20-3-394-401

Abstract

Subject of Research. The paper studies minimum-sized deterministic finite automata inferring problem. A hybrid method is developed and implemented reducing the given problem to Boolean satisfiability (SAT) technique and at the same

time applying a counterexample guided abstraction refinement approach. **Method.** It is proposed to use not all given behavior examples as training data but start with some subset of them and build a consistent automaton, using SAT-based automata inferring method. Then the built automaton is checked against the complete set of behavior examples. The examples not consistent with the automaton are counterexamples. Some subset of counterexamples is added to the current training data and the process is being repeated. **Main Results.** The proposed method is implemented as a part of deterministic finite automata inferring tool in Python language. Experimental comparison of the developed method and the SAT-based method without abstraction refinement is carried out. **Practical Relevance.** Experimental research has shown that the developed method is reasonable for application if the number of behavior examples is large enough, at least, two hundred times exceeds the number of the automaton states, and, therefore, the Boolean formula being created contains tens and hundreds of millions of clauses.

Keywords
deterministic finite automata inference, Boolean satisfiability, counterexample guided abstraction refinement, symmetry breaking, grammatical inference, machine learning

Acknowledgements
The reported study was funded by the RFBR according to the research project No. 18-37-00425.

Введение

Задача построения детерминированного конечного автомата (ДКА) минимального размера по примерам поведения является классической задачей такой области машинного обучения как грамматический вывод. Исследования, посвященные решению данной задачи, проводятся в течение нескольких последних десятилетий. Существует множество прикладных задач, сводящихся к задаче построения ДКА, например, [1, 2]. Первая фундаментальная работа по построению ДКА минимального размера появилась в 70-х годах прошлого века [3]. Следующие важные работы в данной области проводились в 90-х годах: сведение к задаче раскраски графа [4]; использование подходов программирования в ограничениях [5, 6]; подходы на основе алгоритма слияния состояний [7, 8]. В последние годы были предложены подходы, основанные на сведении к задачам выполнимости:

— булевых формул (Boolean Satisfiability problem, SAT) [9–11];
— формул в теориях (Satisfiability Modulo Theories, SMT) [2].

Если предложенные ранее алгоритмы являются эвристическими (неточными), то подходы, основанные на сведении к SAT и SMT, позволяют решать поставленную задачу точно. Ими гарантируется, что если в результате работы алгоритма найден некоторый ДКА, то он минимального размера. Недостатком данных методов изначально являлось их время работы и применимость для задач, где размер автомата представляется хотя бы двузначным числом. В последние годы автором данного исследования и его коллегами были предложены различные подходы к нарушению симметрии и другие способы сокращения пространства поиска, что привело к значительному расширению применимости методов, основанных на сведении к SAT [12–14]. Однако, несмотря на разработанные улучшения, эффективность данных методов сильно падает как при увеличении размера искомого автомата, так и при увеличении размера входных данных — примеров поведения. Данная работа посвящена разработке метода, позволяющего эффективно решать задачу построения ДКА при наличии большого количества примеров поведения.

Предлагается новый комбинированный метод, объединяющий подход, основанный на сведении к SAT, с подходом уточнения абстракции по контрпримерам [15]. Использование подхода уточнения абстракции позволяет среди всех примеров поведения рассматривать только значимые, игнорируя те, которые дублируют уже имеющуюся информацию. Таким образом, автомат, построенный комбинированным методом, будет соответствовать всем примерам поведения, но для его построения будет использоваться только их часть.

Постановка задачи

Детерминированным конечным автоматом $\mathcal{D}$ называется кортеж $(D, \Sigma, \delta, d_1, D^+, D^-)$, где D — конечное множество состояний; Σ — алфавит; δ: $D \times \Sigma \to D$ — функция перехода; d_1 — *стартовое* состояние; D^+ — множество *допускающих* (*принимающих*) состояний; $D^- = D \backslash D^+$ — множество *не допускающих* (*отвергающих*) состояний. Изначально автомат $\mathcal{D}$ находится в стартовом состоянии d_1. В качестве входных данных автомату передается *строка* (*слово*) $s \in \Sigma^*$. Автомат считывает символы строки s по одному, и при считывании очередного символа c_i переходит из текущего состояния d в состояние $\delta(d, c_i)$. Процесс продолжается до тех пор, пока автомат не считает все символы строки s. Если после этого автомат оказался в допускающем состоянии, то говорят, что автомат $\mathcal{D}$ *допускает* (*принимает*) слово s. Иначе говоря, автомат $\mathcal{D}$ *не допускает* (*отвергает*) слово s. Можно индуктивно определить функцию перехода $\hat{\delta}$: $D \times \Sigma^* \to D$ следующим образом: $\hat{\delta}(d_1, \varepsilon) = d_1$ (ε — пустая строка); для $s = s'c$ верно, что $\hat{\delta}(d_1, \varepsilon) = \delta(\hat{\delta}(d_1, s'), c)$.

В данной работе рассматривается задача *пассивного* построения минимального ДКА по имеющимся примерам поведения — множеству строк, которые были приняты или отвергнуты некоторым неизвестным ДКА $\mathcal{U} = (U, \Sigma, \mu, u_1, U^+, U^-)$. Альтернативой является задача активного построения минимального ДКА [2, 16], когда алгоритм построения может делать некоторые запросы к оракулу, который имеет информацию о неизвестном ДКА.

Для простоты будем считать, что примеры поведения представлены двумя множествами слов над алфавитом Σ: S_+ — множество слов, допущенных неизвест-

ным автоматом $\mathcal{U}$; S_- — множество слов, отвергнутых неизвестным автоматом $\mathcal{U}$. Примеры поведения можно представить в виде *расширенного префиксного дерева* (Augmented Prefix Tree Acceptor, APTA) $\mathcal{T} = (T, \Sigma, \tau, t_1, T^+, T^-)$. Все элементы кортежа соответствуют тем, что даны в определении ДКА с единственным отличием, что $T^+ \cup T^- \neq T$. Иными словами, в расширенном префиксном дереве $\mathcal{T}$ присутствуют как принимающие и отвергающие состояния, так и *неопределенные*. Помимо этого, расширенное префиксное дерево обладает всеми свойствами обычных префиксных деревьев: для двух примеров поведения $s_1 = ss_1'$ и $s_2 = ss_2'$ их общий префикс s соответствует уникальной последовательности состояний в APTA.

Таким образом,

$$T^+ = \bigcup_{s \in S_+} \tau(t_1, s)$$

и

$$T^- = \bigcup_{s \in S_-} \tau(t_1, s).$$

Будем считать, что $|T| = N$. Пример расширенного префиксного дерева представлен на рис. 1.

Задача построения ДКА минимального размера заключается в поиске такого ДКА $\mathcal{D} = (D, \Sigma, \delta, d_1, D^+, D^-)$, что $|D|$ — минимально, и для любой строки $s \in S_+$ верно, что $\hat{\delta}(d_1, s) \in D^+$, а для любой строки $s' \in S_-$ верно, что $\hat{\delta}(d_1, s') \in D^-$. Будем считать, что $|D| = M$.

Задача выполнимости булевых формул

Пусть $X = \{x_1, \ldots, x_n\}$ — конечное множество булевых переменных. *Литералом* называется либо переменная x_i, либо ее отрицание $\neg x_i$. *Дизъюнктом* называется дизъюнкция нескольких (возможно одного) литералов, например $(x_1 \vee \neg x_2 \vee x_4 \vee \neg x_6)$. Также дизъюнкт можно представить в виде множества литералов, подразумевая, что они связаны дизъюнкцией — $\{x_1, \neg x_2, x_4, \neg x_6\}$. Булева формула, представленная в конъюнктивно-нормальной форме (КНФ), является конъюнкцией некоторых дизъюнктов, например $(x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_5) \wedge (x_6)$. Иначе булеву формулу в КНФ можно представить как множество дизъюнктов — $\{\{x_1, x_3\}, \{\neg x_2, \neg x_3, x_5\}, \{x_6\}\}$. В данной работе принято, что любая булева формула представлена в КНФ. *Выполняющей подстановкой* $\mathbf{v} = (v_1, \ldots, v_n)$ для некоторой булевой формулы ϕ называется такой булев вектор, что для всех i при подстановке значения v_i вместо переменной x_i в формуле ϕ, получается тождественно истинное выражение. *Задача выполнимости булевых формул* заключается в определении — существует ли выполняющая подстановка $\mathbf{v}$ для данной формулы ϕ [17]. Формула ϕ передается одному из существующих программных средств для решения SAT, которое выносит вердикт: $\mathbb{SAT}$ (satisfiable) — если подстановка $\mathbf{v}$ существует; $\mathbb{UNSAT}$ (unsatisfiable) — иначе. Большинство современных программных средств возвращают также саму подстановку $\mathbf{v}$ в случае вердикта $\mathbb{SAT}$.

Подход к построению минимального ДКА по заданным примерам поведения при помощи сведения к задаче SAT

Среди множества других подходов к решению задачи построения ДКА минимального размера по заданным примерам поведения следует выделить подход, основанный на сведении к SAT. Данный подход в отличие от других является точным — гарантируется, что найденный автомат будет минимально возможным. На рис. 2 представлен наиболее успешный вариант данного подхода. По примерам поведения строится расширенное префиксное дерево $\mathcal{T}$. Затем неким (обычно эвристическим) алгоритмом ищется нижняя оценка l на размер искомого ДКА. После этого, начиная с нижней оценки l, для каждого возможного числа состояний некоторый алгоритм определяет, существует ли ДКА $\mathcal{D}$ такого размера, соответствующий префиксному дереву $\mathcal{T}$. Таким образом гарантируется минимальность найденного автомата $\mathcal{D}$.

Задача поиска ДКА фиксированного размера является NP-полной [18], а значит, может быть сведена к задаче SAT. Такое сведение было предложено в [9]. Далее приводится данное сведение в кратком изложении.

Необходимо решить следующую задачу: для расширенного префиксного дерева $\mathcal{T} = (T, \Sigma, \tau, t_1, T^+, T^-)$ определить, существует ли ДКА $\mathcal{D} = (D, \Sigma, \delta, d_1, D^+, D^-)$ такой, что $|D| = M$. Для решения поставленной задачи авторы [9] предложили задать соответствие между имеющимся префиксным деревом $\mathcal{T}$ и искомым автоматом $\mathcal{D}$.

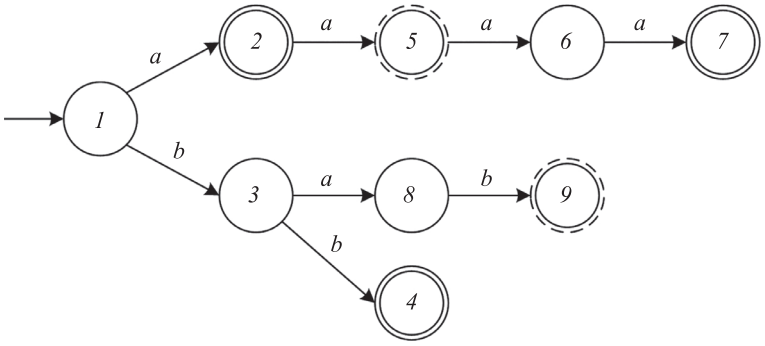


Рис. 1. Расширенное префиксное дерево для $\Sigma = \{a; b\}$, $S_+ = \{a; aaaa; bb\}$ и $S_- = \{aa; bab\}$, где вершины: с двойной сплошной границей — принимающие (*2*, *4*, *7*); с прерывистой внешней границей — отвергающие (*5*, *9*); с одинарной границей — неопределенные (*1*, *3*, *6*, *8*)

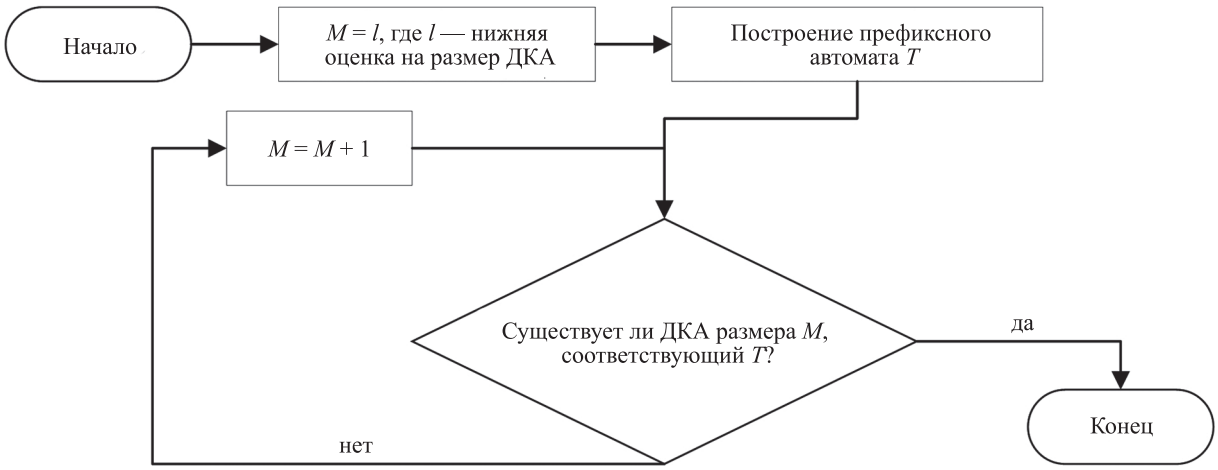


Рис. 2. Точный алгоритм поиска минимального детерминированного конечного автомата по примерам поведения

Для того чтобы закодировать данную задачу на языке SAT, нужно определить три набора булевых переменных:

1) $x_{v,i}$, которая истинна тогда и только тогда, когда вершина t_v в префиксном дереве $\mathcal{T}$ соответствует состоянию d_i в автомате $\mathcal{D}$;
2) $y_{i,l,j}$, которая истинна тогда и только тогда, когда в автомате $\mathcal{D}$ есть переход из состояния d_i в состояние d_j по символу $l \in \Sigma$;
3) z_i, которая истинна тогда и только тогда, когда состояние d_i автомата $\mathcal{D}$ является допускающим.

Для упрощения записи далее используется следующее обозначение: $[R] = \{1, \ldots, R\}$. Тогда рассматриваемое сведение можно представить в следующем виде:

1) $(x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,M})$ для $v \in [N]$ — каждая вершина префиксного дерева соответствует как минимум одному состоянию автомата;
2) $(\neg x_{v,i} \vee \neg x_{v,j})$ для $v \in [N]$; $i, j \in [M]$; $i < j$ — каждая вершина префиксного дерева соответствует не более чем одному состоянию автомата;
3) $(y_{i,l,1} \vee y_{i,l,2} \vee \ldots \vee y_{i,l,M})$ для $i \in [M]$; $l \in \Sigma$ — из каждого состояния автомата существует как минимум один переход по каждому символу, иными словами, автомат полон;
4) $(\neg y_{i,l,j} \vee \neg y_{i,l,h})$ для $i, j, h \in [M]$; $j < h$; $l \in \Sigma$ — из каждого состояния автомата существует не более чем один переход по каждому символу, иными словами, автомат детерминирован;
5) $(\neg x_{v,i} \vee z_i)$ для $t_v \in T^+$; $i \in [M]$ — если вершина t_v является допускающей в префиксном дереве и соответствует состоянию d_i искомого автомата, то состояние d_i также должно быть допускающим;
6) $(\neg x_{v,i} \vee \neg z_i)$ для $t_v \in T^-$; $i \in [M]$ — если вершина t_v является отвергающей в префиксном дереве и соответствует состоянию d_i искомого автомата, то состояние d_i также должно быть отвергающим;
7) $(\neg x_{v,i} \vee \neg x_{w,j} \vee y_{i,l,j})$ для $v, w \in [N]$; $i, j \in [M]$; $l \in \Sigma$; $\tau(t_v, l) = t_w$ — если вершина t_v соответствует состоянию d_i, вершина t_w — состоянию d_j, и в префиксном дереве существует переход из вершины t_v в вершину t_w по символу l, то и в автомате должен быть переход из состояния d_i в состояние d_j по символу l;
8) $(\neg x_{v,i} \vee \neg y_{i,l,j} \vee \neg x_{w,j})$ для $v, w \in [N]$; $i, j \in [M]$; $l \in \Sigma$; $\tau(t_v, l) = t_w$ — аналогично, если вершина t_v соответствует состоянию d_i, если в префиксном дереве есть переход из состояния t_v в состояние t_w по символу l, и в автомате есть переход из состояния d_i в состояние d_j по символу l, то вершина t_w должна соответствовать состоянию d_j;
9) $(x_{1,1})$ — корень префиксного дерева соответствует стартовому состоянию ДКА.

Все приведенные выше наборы дизъюнктов можно объединить в одну формулу и передать программному средству для решения задачи SAT. Если программное средство найдет выполняющую подстановку, то автомат размера M, соответствующий имеющимся примерам поведения, представленным в виде расширенного префиксного дерева $\mathcal{T}$, существует. Этот автомат можно построить, воспользовавшись переменными $y_{i,l,j}$ и z_i.

Представленное выше кодирование содержит $\mathcal{O}(M^3 + N \times M^2)$ дизъюнктов и $O(M^2 + N \times M)$ переменных. Учитывая, что обычно $N \gg M$ ($N \approx 10^3$–10^4; $M \approx 10$ – 100), то можно заключить, что сведение содержит $O(N \times M^2)$ дизъюнктов. Данный подход сам по себе не показывает выдающихся результатов, но в совокупности с идеями, кратко представленными в следующем разделе, его применимость значительно возрастает.

Подходы к сокращению пространства поиска в задаче построения ДКА минимального размера по заданным примерам поведения

В статье [9] авторы предложили дополнительно использовать структуру данных, названную как *граф совместимости* (consistency graph). Несмотря на то, что для данной структуры данных больше подходит название *граф несовместимости*, далее будет использоваться оригинальное название. Основная идея состоит в том, что, имея расширенное префиксное дерево $\mathcal{T} = (T, \Sigma, \tau, t_1, T^+, T^-)$, можно построить граф $\mathcal{G} = (V, E)$ такой, что $V = T$, а E определяется следующим образом. Две вершины в графе соединены ребром тогда и только тогда, когда слияние данных вершин в префиксном дереве в одну и последующее избавление от недетер-

минированности путем слияния потомков, в которые есть переход по одному и тому же символу, приводят к ситуации, когда объединяются принимающее и отвергающее состояния.

Иными словами, если две вершины являются смежными в графе совместимости, то они не могут соответствовать одному и тому же состоянию в ДКА. Это свойство можно выразить с помощью дизъюнктов вида $(\neg x_{v,i} \vee \neg x_{w,i})$ для $v, w \in [N]$; $(v, w) \in E$; $i \in [M]$. Такие дизъюнкты не являются обязательными, но они сильно сокращают пространство поиска программного средства для решения задачи SAT. Минусом использования графа совместимости является то, что в общем случае необходимо добавить $\mathcal{O}(N^2 \times M)$ дизъюнктов, что сильно увеличивает размер формулы.

Помимо этого, было предложено несколько подходов к нарушению симметрии в данной задаче. Лучше всего себя зарекомендовали предикаты нарушения симметрии на основе алгоритма обхода в ширину (Breadth-First Search, BFS), предложенные в [12–14]. Основная идея заключается в том, чтобы путем добавления новых ограничений в сведение зафиксировать нумерацию искомого ДКА в порядке его обхода в ширину. Такие ограничения позволяют запретить программному средству рассматривать $M!$ изоморфных автоматов, оставив по одному представителю для каждого класса эквивалентности по изоморфизму. Последняя модификация данных предикатов может быть выражена через $\mathcal{O}(M^2 \times L)$ дизъюнктов. Подробное описание предикатов нарушения симметрии можно найти в оригинальной работе [12].

В [13] можно найти некоторые другие идеи по сокращению пространства поиска, которые не относятся непосредственно к настоящей работе, но могут быть полезны для ознакомления.

Метод уточнения абстракции по контрпримерам

Главной проблемой методов решения задачи построения минимального ДКА по примерам поведения при помощи сведения к SAT является размер получающейся формулы. Ранее в [12–14] были предложены как более компактные способы кодирования поставленной задачи на языке SAT, так и различные подходы к сокращению пространства поиска. Тем не менее, подход, основанный на сведении к задаче выполнимости, все еще мало применим в случае, когда префиксное дерево большое. Следует заметить, что префиксное дерево увеличивается в размере как при увеличении числа примеров поведения, так и при увеличении их длины. В данном разделе описан новый подход к решению задачи построения минимального ДКА, позволяющий решить вышеуказанную проблему.

В случае, когда стоит задача построить некоторую модель, при этом имея доступ к проверяющей системе, часто применяется метод *уточнения абстракции по контрпримерам* (Counterexample-Guided Abstraction Refinement, CEGAR). Суть данного метода можно описать следующим образом. На начальном шаге генерируется некоторая, возможно случайная модель. Затем на каждом следующем шаге данная модель проходит проверку некоторой проверяющей системы. Если проверка проходит успешно, то искомая модель найдена. Иначе система возвращает один или несколько контрпримеров, которые затем используются для улучшения модели. Процесс повторяется, пока не будет найдена модель, проходящая проверку системы. Данный подход больше похож на метод активного построения модели, в то время как в данной работе рассматривается задача пассивного построения —все примеры поведения известны заранее и никакой дополнительной информации в ходе построения автомата быть получено не может. Однако далее приводится описание того, как метод уточнения абстракции можно применить для построения ДКА.

Для начального шага выбирается некоторое подмножество примеров поведения (возможно пустое), по которому строится расширенное префиксное дерево $\mathcal{T}$, выбирается нижняя оценка на размер искомого ДКА, а затем используется сведение к SAT, описанное ранее. Если программное средство не смогло найти автомат текущего размера, то текущий размер увеличивается на один и процесс повторяется. Если же какой-то автомат, соответствующий дереву $\mathcal{T}$, найден, то оставшиеся неиспользуемые пока примеры поведения проверяются на соответствие текущему ДКА. Множество слов, которые не соответствуют автомату, образуют множество контрпримеров. После этого выбирается некоторое подмножество (или все множество) контрпримеров, которые добавляются в дерево $\mathcal{T}$ и процесс повторяется (рис. 3). Таким образом, вместо построения слишком большой формулы, описывающей все имеющиеся примеры поведения, формула итеративно достраивается и описывает только часть примеров, что должно сократить время на решение SAT.

Другим важным достоинством предлагаемого метода является то, что большинство современных программных средств для решения задачи выполнимости могут работать в так называемом итеративном режиме. В стандартном режиме после вынесения вердикта $\mathbb{SAT}$ или $\mathbb{UNSAT}$ программное средство прекращает работу. В итеративном же режиме, если формула выполнима и вердикт $\mathbb{SAT}$, то программное средство сохраняет свое состояние и ожидает новых ограничений. Можно закодировать новые свойства, используя уже имеющиеся булевы переменные и добавляя новые, и передать получившуюся формулу программному средству. После этого программное средство продолжит решение задачи с того момента, где остановилось. Такой подход позволяет не делать программному средству одну и ту же работу несколько раз. Заметим, что при добавлении новых примеров поведения и расширении префиксного дерева $\mathcal{T}$ никакие старые дизъюнкты не убираются, только добавляются новые, что делает возможным использование программных средств для решения SAT в итеративном режиме.

Важным вопросом является то, сколько контрпримеров добавлять на каждом шаге. Добавление слишком маленького числа контрпримеров может привести к тому, что накладные расходы на построение префиксного дерева, построение и передачу новых дизъюнктов программному средству для решения SAT будут пре-

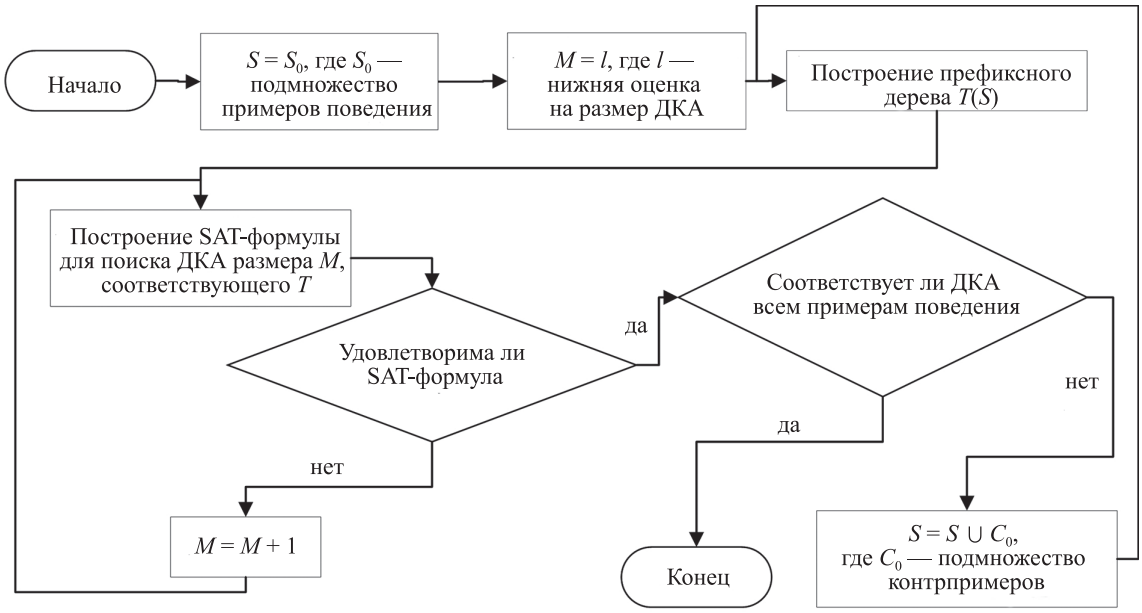


Рис. 3. Комбинированный алгоритм построения минимального детерминированного конечного автомата по примерам поведения с использованием подхода уточнения абстракции по контрпримерам

вышать время работы самого программного средства. Также, чем меньше примеров поведения используется, тем меньше информации у программного средства, тем сложнее ему эффективно перебирать возможные решения. С другой стороны, как было сказано ранее, использование слишком большого числа примеров поведения выражается в виде слишком большой формулы, с которой программному средству в принципе сложно работать. Возможным решением данной дилеммы может быть использование арифметической или геометрической прогрессии для добавления контрпримеров.

Экспериментальные результаты

Предложенный в предыдущем разделе комбинированный метод построения ДКА минимального размера на основе сведения к SAT и с использованием подхода уточнения абстракции по контрпримерам был реализован на языке Python как модуль программного комплекса DFA-Inductor-py. Эксперименты проводились на персональном компьютере с процессором QuadCore Intel Core i7-8550U @ 4 ГГц, 16 ГБ оперативной памяти и операционной системой ArchLinux 5.5.6. Для проведения экспериментов было сгенерировано 100 тестовых экземпляров с помощью алгоритма, разработанного автором ранее и описанного в [14]. Параметры для генерации автоматов были выбраны следующим образом:

— размеры автоматов, которые нужно построить, — $M \in [15; 25]$;
— количество примеров поведения $S = S_+ \cup S_- \in \{50 \times N; 100 \times N; 200 \times N; 500 \times N\}$.

В экспериментах проводилось сравнение разработанного комбинированного метода с предложенным в [13] методом, основанным только на сведении к SAT.

Результаты показали, что при относительно небольшом количестве примеров поведения ($S \in \{50 \times N; 100 \times N\}$) использование комбинированного подхода сокращает время построения вспомогательных структур данных (таких как граф совместимости) и время построения булевой формулы, но увеличивает время работы программного средства для решения SAT и, как следствие, суммарное время решения задачи. Однако в случае, когда количество примеров поведения достаточно велико ($S \in \{200 \times N; 500 \times N\}$), использование предложенного подхода позволяет использовать меньше половины примеров поведения вместо всех, что сокращает как время построения структур данных и булевой формулы, так и время работы программного средства для решения SAT. Выигрыш достигает 30 % на экземплярах, которые оба методы смогли решить за 12 ч. Значительная часть таких экземпляров не были в принципе решены методом, основанным только на сведении к SAT.

Таким образом, можно сделать вывод, что использование разработанного метода целесообразно, когда количество примеров поведения велико, и метод, основанный только на сведении к SAT, не применим ввиду слишком большой формулы.

Заключение

В данной работе предложен новый комбинированный алгоритм построения детерминированного конечного автомата минимального размера по примерам поведения, основанный на сведении к задаче выполнимости булевых формул и с использованием подхода уточнения абстракции по контрпримерам. Экспериментальное исследование показало, что разработанный метод эффективен при условии, что число примеров поведения достаточно велико — хотя бы в двести раз превышает количество состояний, — и, как следствие, строящаяся булева формула содержит десятки и сотни миллионов дизъюнктов. Использование

комбинированного метода позволяет сократить количество используемых примеров поведения без потери точности.

Для дальнейшего исследования остается вопрос об эффективности предложенного метода в других частных, но важных случаях задачи построения минимального детерминированного конечного автомата. Например, когда автомат имеет какую-то особенную структуру, или среди примеров поведения много таких, которые покрывают только одну часть автомата.

Литература

1. Wieman R., Aniche M.F., Lobbezoo W., Verwer S., Van Deursen A. An experience report on applying passive learning in a large-scale payment company // Proc. 33rd IEEE International Conference on Software Maintenance and Evolution (ICSME). Shanghai, China. 2017. P. 564–573. doi: 10.1109/ICSME.2017.71
2. Neider D. Applications of Automata Learning in Verification and Synthesis: PhD thesis. Hochschulbibliothek der Rheinisch-Westfälischen Technischen Hochschule Aachen, 2014. 283 p.
3. Biermann A.W., Feldman J.A. On the synthesis of finite-state machines from samples of their behavior // IEEE Transactions on Computers. 1972. V. C-21. N 6. P. 592–597. doi: 10.1109/TC.1972.5009015
4. Coste F., Nicolas J. Regular inference as a graph coloring problem // Proc. 14th International Conference on Machine Learning (ICML), Workshop on Grammatical Inference, Automata Induction, and Language Acquisition. Nashville, Tennessee, USA. 1997.
5. Coste F., Nicolas J. How considering incompatible state mergings may reduce the DFA induction search tree // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1998. V. 1433. P. 199–210. doi: 10.1007/BFb0054076
6. Oliveira A.L., Silva J.P.M. Efficient algorithms for the inference of minimum size DFAs // Machine Learning. 2001. V. 44. N 1-2. P. 93–119. doi: 10.1023/A:1010828029885
7. Lang K.J. Faster Algorithms for Finding Minimal Consistent DFAs: Technical Report. NEC Research Institute, 1999. 19 p.
8. Lang K.J., Pearlmutter B.A., Price R.A. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1998. V. 1433. P. 1–12. doi: 10.1007/BFb0054059
9. Heule M., Verwer S. Exact DFA identification using SAT solvers // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2010. V. 6339. P. 66–79. doi: 10.1007/978-3-642-15488-1_7
10. Ulyantsev V., Tsarev F. Extended finite-state machine induction using SAT-solver // Proc. 10th International Conference on Machine Learning and Applications and Workshops. Honolulu, Hawaii, USA. 2011. P. 346–349. doi: 10.1109/ICMLA.2011.166
11. Grinchtein O., Leucker M., Piterman N. Inferring network invariants automatically // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2006. V. 4130. P. 483–497. doi: 10.1007/11814771_40
12. Ulyantsev V., Zakirzyanov I., Shalyto A. BFS-based symmetry breaking predicates for DFA identification // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2015. V. 8977. P. 611–622. doi: 10.1007/978-3-319-15579-1_48
13. Zakirzyanov I., Morgado A., Ignatiev A., Ulyantsev V., Silva J.M. Efficient symmetry breaking for SAT-based minimum DFA inference // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2019. V. 11417. P. 159–173. doi: 10.1007/978-3-030-13435-8_12
14. Zakirzyanov I., Shalyto A., Ulyantsev V. Finding all minimum-size DFA consistent with given examples: SAT-based approach // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2018. V. 10729. P. 117–131. doi: 10.1007/978-3-319-74781-1_9
15. Clarke E.M., Grumberg O., Jha S., Lu Y., Veith H. Counterexample-guided abstraction refinement for symbolic model checking // Journal of the ACM. 2003. V. 50. N 5. P. 752–794. doi: 10.1145/876638.876643
16. Angluin D. Learning regular sets from queries and counterexamples // Information and Computation. 1987. V. 75. N 2. P. 87–106. doi: 10.1016/0890-5401(87)90052-6

References

1. Wieman R., Aniche M.F., Lobbezoo W., Verwer S., Van Deursen A. An experience report on applying passive learning in a large-scale payment company. *Proc. 33rd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Shanghai, China, 2017, pp. 564–573. doi: 10.1109/ICSME.2017.71
2. Neider D. *Applications of Automata Learning in Verification and Synthesis*. PhD thesis. Hochschulbibliothek der Rheinisch-Westfälischen Technischen Hochschule Aachen, 2014, 283 p.
3. Biermann A.W., Feldman J.A. On the synthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers*, 1972, vol. C-21, no. 6, pp. 592–597. doi: 10.1109/TC.1972.5009015
4. Coste F., Nicolas J. Regular inference as a graph coloring problem. *Proc. 14th International Conference on Machine Learning (ICML), Workshop on Grammatical Inference, Automata Induction, and Language Acquisition*. Nashville, Tennessee, USA, 1997.
5. Coste F., Nicolas J. How considering incompatible state mergings may reduce the DFA induction search tree. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1998, vol. 1433, pp. 199–210. doi: 10.1007/BFb0054076
6. Oliveira A.L., Silva J.P.M. Efficient algorithms for the inference of minimum size DFAs. *Machine Learning*, 2001, vol. 44, no. 1-2, pp. 93–119. doi: 10.1023/A:1010828029885
7. Lang K.J. *Faster Algorithms for Finding Minimal Consistent DFAs*. Technical Report. NEC Research Institute, 1999, 19 p.
8. Lang K.J., Pearlmutter B.A., Price R.A. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1998, vol. 1433, pp. 1–12. doi: 10.1007/BFb0054059
9. Heule M., Verwer S. Exact DFA identification using SAT solvers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6339, pp. 66–79. doi: 10.1007/978-3-642-15488-1_7
10. Ulyantsev V., Tsarev F. Extended finite-state machine induction using SAT-solver. *Proc. 10th International Conference on Machine Learning and Applications and Workshops*, Honolulu, Hawaii, USA, 2011, pp. 346–349. doi: 10.1109/ICMLA.2011.166
11. Grinchtein O., Leucker M., Piterman N. Inferring network invariants automatically. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006, vol. 4130, pp. 483–497. doi: 10.1007/11814771_40
12. Ulyantsev V., Zakirzyanov I., Shalyto A. BFS-based symmetry breaking predicates for DFA identification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 8977, pp. 611–622. doi: 10.1007/978-3-319-15579-1_48
13. Zakirzyanov I., Morgado A., Ignatiev A., Ulyantsev V., Silva J.M. Efficient symmetry breaking for SAT-based minimum DFA inference. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11417, pp. 159–173. doi: 10.1007/978-3-030-13435-8_12
14. Zakirzyanov I., Shalyto A., Ulyantsev V. Finding all minimum-size DFA consistent with given examples: SAT-based approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 10729, pp. 117–131. doi: 10.1007/978-3-319-74781-1_9
15. Clarke E.M., Grumberg O., Jha S., Lu Y., Veith H. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 2003, vol. 50, no. 5, pp. 752–794. doi: 10.1145/876638.876643

17. Handbook of Satisfiability / ed. by A. Biere, M. Heule, H. van Maaren, T.Walsh. IOS Press, 2009. 980 p. (Frontiers in Artificial Intelligence and Applications, V. 185)
18. Gold E.M. Complexity of automaton identification from given data // Information and Control. 1978. V. 37. N 3. P. 302–320. doi: 10.1016/S0019-9958(78)90562-4

16. Angluin D. Learning regular sets from queries and counterexamples. *Information and Computation*, 1987, vol. 75, no. 2, pp. 87–106. doi: 10.1016/0890-5401(87)90052-6
17. *Handbook of Satisfiability*. Ed. by A. Biere, M. Heule, H. van Maaren, T. Walsh. IOS Press, 2009, 980 p., Frontiers in Artificial Intelligence and Applications, vol. 185.
18. Gold E.M. Complexity of automaton identification from given data. *Information and Control*, 1978, vol. 37, no. 3, pp. 302–320. doi: 10.1016/S0019-9958(78)90562-4

Авторы

Закирзянов Илья Тимурович — аспирант, программист, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, Scopus ID: 56613352900, ORCID ID: 0000-0002-3460-3489, ilya.zakirzyanov@gmail.com

Authors

Ilya T. Zakirzyanov — Postgraduate, Software Engineer, ITMO University, Saint Petersburg, 197101, Russian Federation, Scopus ID: 56613352900, ORCID ID: 0000-0002-3460-3489, ilya.zakirzyanov@gmail.com