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 неизвестным 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 $\mathcal{D}$.

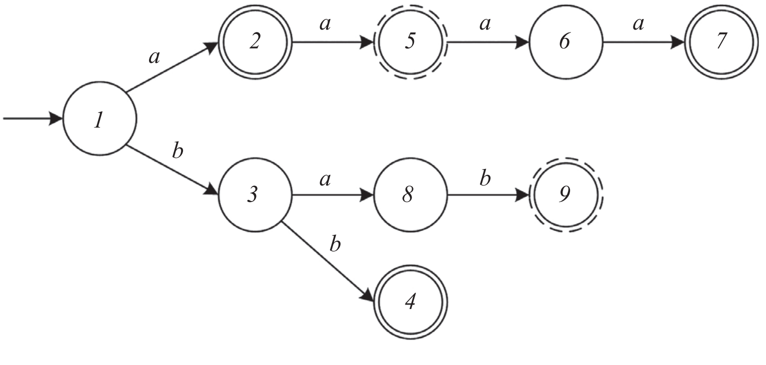

Рис. 1. Расширенное префиксное дерево для $\Sigma = \{a; b\}$, $S_+ = \{a; aaaa; bb\}$ и $S_- = \{aa; bab\}$, где вершины: с двойной сплошной границей — принимающие (*2*, *4*, *7*); с прерывистой внешней границей — отвергающие (*5*, *9*); с одинарной границей — неопределенные (*1*, *3*, *6*, *8*)

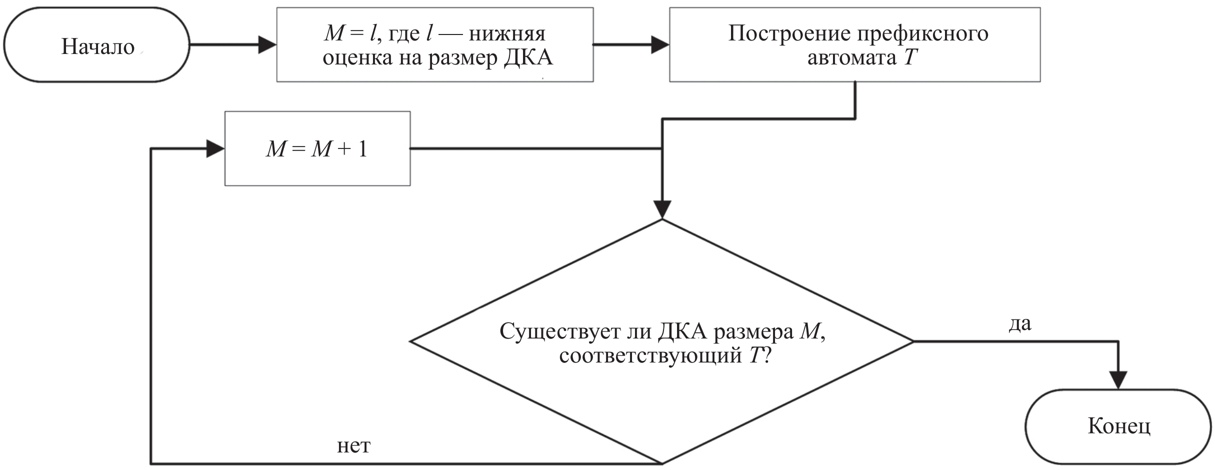

Рис. 2. Точный алгоритм поиска минимального детерминированного конечного автомата по примерам поведения

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 недетерминированности 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-

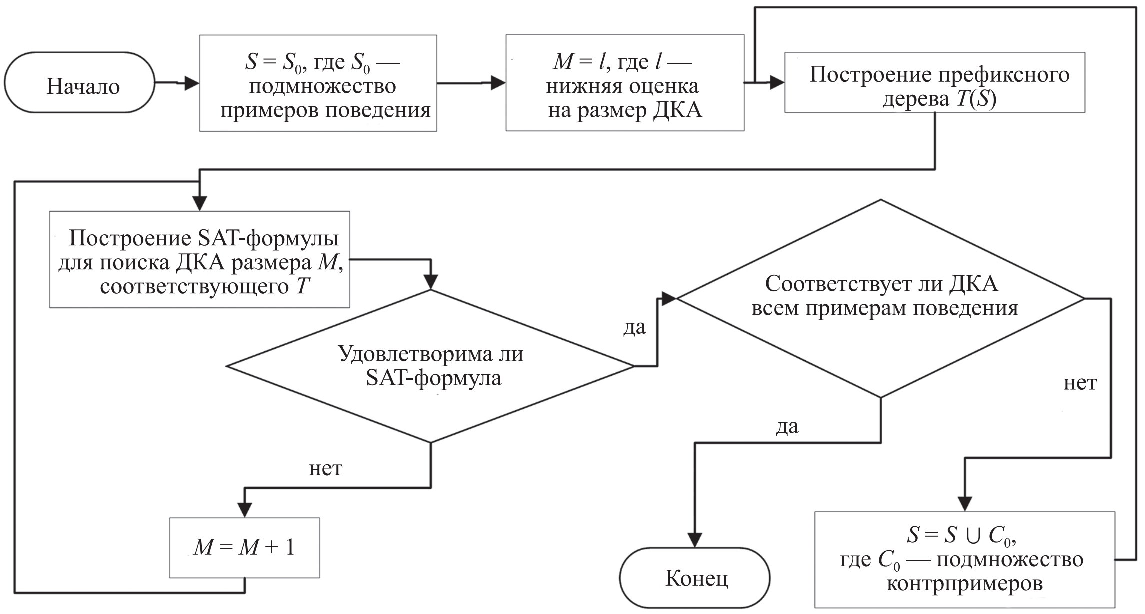

Рис. 3. Комбинированный алгоритм построения минимального детерминированного конечного автомата по примерам поведения с использованием подхода уточнения абстракции по контрпримерам

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 метода 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