Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА МЕТОДОВ СИНТЕЗА ВРЕМЕННЫХ АВТОМАТОВ НА ОСНОВЕ ПРОГРАММИРОВАНИЯ В ОГРАНИЧЕНИЯХ

Автор: Костливцев Никита Алексеевич _____

Направление подготовки: 01.03.02 Прикладная математика и информатика

Квалификация: Бакалавр

Руководитель: Чивилихин Д.С., к.т.н. _____

К защите допустить

Руководитель ОП Парфенов В.Г., проф., д.т.н. _____

«____» _____ 20___ г.

Санкт-Петербург, 2019 г.

Студент Костливцев Н.А. Группа М3439 Факультет ИТиП

Направленность (профиль), специализация
Математические модели и алгоритмы в разработке программного обеспечения

ВКР принята «____» _____ 20___ г.

Оригинальность ВКР _____%

ВКР выполнена с оценкой _____

Дата защиты «____» _____ 20___ г.

Секретарь ГЭК Павлова О.Н. _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

УТВЕРЖДАЮ
Руководитель ОП
проф., д.т.н. Парфенов В.Г. _____
«____» _____ 20___ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студент Костливцев Н.А. **Группа** М3439 **Факультет** ИТиП

Руководитель Чивилихин Д.С., к.т.н., научный сотрудник Университета ИТМО

1 Наименование темы: Разработка методов синтеза временных автоматов на основе программирования в ограничениях

Направление подготовки (специальность): 01.03.02 Прикладная математика и информатика

Направленность (профиль): Математические модели и алгоритмы в разработке программного обеспечения

Квалификация: Бакалавр

2 Срок сдачи студентом законченной работы: «31» мая 2019 г.

3 Техническое задание и исходные данные к работе

а) Изучение пакета программного обеспечения minizinc
б) Изучение существующих методов решения задачи синтеза временных автоматов
в) Разработка методов сведения к программированию в ограничениях
г) Программная реализация разработанных методов
д) Сравнение реализованных решений с существующими методами

4 Содержание выпускной работы (перечень подлежащих разработке вопросов)

а) Постановка задачи и исследование существующих на данный момент решений
б) Описание разработанных идей и их программная реализация
в) Эксперименты над реализованными решениями, сравнение с существующими подходами

5 Перечень графического материала (с указанием обязательного материала)

Графические материалы и чертежи работой не предусмотрены

6 Исходные материалы и пособия

а) *Verwer S.*, *Weerdt M. de*, *Witteveen C.* Efficiently identifying deterministic real-time automata from labeled data // Machine Learning. 2010. P. 295–333;
б) *Verwer S.*, *Weerdt M. de*, *Witteveen C.* The efficiency of identifying timed automata and the power of clocks // Information and Computation. 2011. P. 606–625;

в) *Pastore F.*, *Micucci D.*, *Mariani L.* Timed k-Tail: Automatic Inference of Timed Automata // 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). 2017;
г) MOHA: a Multi-mode Hybrid Automaton Model for Learning Car-following Behaviors / Q. Lin [et al.] // IEEE Transactions on Intelligent Transportation Systems. 2018. P. 790–796;
д) *Zakirzyanov I.*, *Shalyto A.*, *Ulyantsev V.* Finding all minimum-size DFA consistent with given examples: SAT-based approach // Software Engineering and Formal Methods. 2017. P. 117–131.

7 Дата выдачи задания «01» сентября 2018 г.

Руководитель ВКР _____

Задание принял к исполнению _____

«01» сентября 2018 г.

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студент: Костливцев Никита Алексеевич

Наименование темы ВКР: Разработка методов синтеза временных автоматов на основе программирования в ограничениях

Наименование организации, где выполнена ВКР: Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования: Разработка методов синтеза временных автоматов на основе программирования в ограничениях

2 Задачи, решаемые в ВКР:

а) Изучение литературы

б) Разработка метода

в) Эксперименты

3 Число источников, использованных при составлении обзора: 20

4 Полное число источников, использованных в работе: 22

5 В том числе источников по годам:

Отечественных			**Иностранных**		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
1	1	3	4	3	10

6 Использование информационных ресурсов Internet: да, число ресурсов: 8

7 Использование современных пакетов компьютерных программ и технологий:

Пакеты компьютерных программ и технологий	**Раздел работы**
Пакет `minizinc`	2.2

8 Краткая характеристика полученных результатов

Разработан метод, способный синтезировать временные автоматы минимального размера на малом числе входных данных. По сравнению с методом RTI с эвристикой consistent EDSM, на маленьких целевых временных автоматах удалось получить сравнимые с ним результаты качества, а также значительно выиграть по количеству состояний и переходов.

9 Гранты, полученные при выполнении работы

Нет

10 Наличие публикаций и выступлений на конференциях по теме работы

а) *Костливцев Н.А.* Генерация временных автоматов по трассировкам с помощью SAT-решателей // Конгресс молодых ученых. – 2019 – Университет ИТМО.

Студент Костливцев Н.А. _____

Руководитель Чивилихин Д.С. _____

«____» _____ 20___ г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Существует множество систем реального времени, поведение которых существенно зависит от времени. Поведение данных систем часто не поддается описанию вручную. Поэтому его моделируют с помощью вычилительных устройств исходя из примеров поведения. На текущий момент лучшей моделью для описания данного вида систем является временной автомат (timed automaton). Следуя принципу бритвы Окамма, лучшей сгенерированной моделью, описывающей систему, будет являться модель, использующая наименьшее число сущностей, как следствие, наилучшим автоматом, описывающим систему реального времени, будет являться минимальный. Существующие на текущий момент решения генерируют временные автоматы в пределе (learning in the limit). Данный метод может гарантировать лишь существование определенного множества примеров поведения, такого что автомат, построенный по этим примерам будет являться минимальным.

В данной работе предлагается алгоритм, генерирующий по заданным примерам поведения минимальный автомат. Предлагаемый алгоритм основан на сведении к задаче удовлетворения ограничений. Также временные автоматы последнее время используются в машинном обучении [14], поэтому на сегодняшний день разработка новых методов генерации временных автоматов является актуальной задачей.

Целью данной работы является разработка нового метода генерации минимального детерминированного конечного временного автомата. Для достижения цели решались следующие задачи.

а) Изучение литературы по временным автоматам и способам их генерации.
б) Разработка сведения к задаче удовлетворения ограничений
в) Запись ограничений сведения на языке Minizinc
г) Экспериментальное исследование разработанного метода и его сравнение с алгоритмом RTI.

Научная новизна разработанного метода состоит в том, что впервые задача генерации минимального временного автомата была решена с помощью сведения к задаче удовлетворения ограничений.

В главе 1 описаны основные термины, приведены основные известные теоретические результаты и методы синтеза детерминированных временных автоматов.

В главе 2 описан предложенный метод синтеза детерминированных временных автоматов, основанный на сведении к задаче программирования в ограничениях.

В главе 3 описано экспериментальное исследование предложенного метода и его сравнение с существующими решениями.

ГЛАВА 1. ВРЕМЕННЫЕ АВТОМАТЫ И МЕТОДЫ ИХ СИНТЕЗА

1.1. Термины и понятия

В данном разделе приведены определения терминов, которые будут использоваться в следующих главах, а также основные результаты о временных автоматах. Определения и формулировки были взяты из статьи [21].

Определение 1. Таймер – математический объект, хранящий внутри себя значение, увеличивающееся со временем, синхронно со всеми остальными таймерами. Значение таймера x обозначается как $\nu(x) \in \mathbb{N}$. Таймеры можно сбрасывать, то есть устанавливать их значение равным нулю ($\nu(x) = 0$).

Определение 2. Временное ограничение g – предикат на таймерах. Его можно получить одним из трех способов:

— $g := c \leqslant x$;

— $g := c \geqslant x$;

— $g := g_1 \wedge g_2$,

где g – временное ограничение, $c \in C$ – таймер, $x \in \mathbb{N}$ – граница ограничения.

Определение 3. Временной автомат $\mathcal{A}$ – кортеж $\langle \Sigma, Q, q_0, C, F, \Delta \rangle$, где:

— Σ – алфавит (множество символов) автомата;

— Q – конечное множество состояний;

— $q_0 \in Q$ – начальное состояние автомата;

— C – конечное множество таймеров;

— $F \subseteq Q$ – множество принимающих состояний;

— Δ – конечное множество переходов, где переход – это кортеж $\langle q_1, q_2, a, g, R \rangle$, такой что:

  – $q_1, q_2 \in Q$ – начальное и конечное состояние, которые соединяет данный переход;

  – $a \in \Sigma$ – символ, по которому осуществляется переход;

  – g – временные ограничения на таймерах;

  – $R \in 2^C$ – таймеры, которые нужно сбросить после перехода по δ.

Пример 4. На рисунке 1 приведен пример временного автомата. Состояние q_3 является принимающим, все же остальные принимающими не являются. Начальным состоянием временного автомата является q_0. Из него исходит одно ребро в состояние q_1 с символом перехода a. После перехода по данному ребру таймер x сбрасывается. Из состояния q_1 тоже исходит ребро в состояние q_2 с символом перехода b, но уже с ограничением на таймер x. Чтобы пройти

по данному ребру значение таймера x должно быть меньше единицы. Лучше всего использование таймеров демонстрируют два перехода из состояния q_2, так как из-за таймеров, чтобы пройти в состояние q_3, необходимо будет три раза пройти по петле в то же состояние q_2.

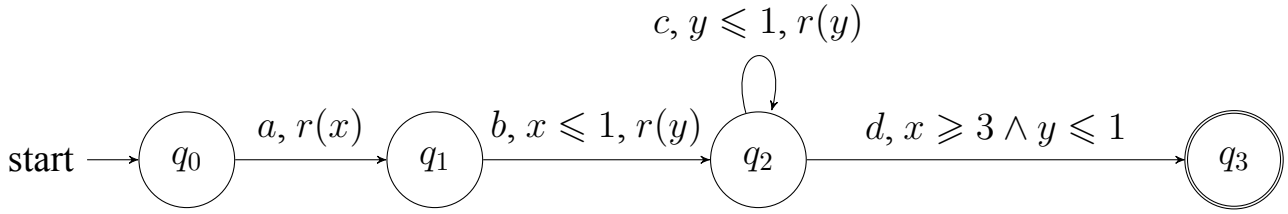

Рисунок 1 – Пример временного автомата

Определение 5. Трассировка τ – последовательность пар $\{(a, t)\}$, где $t \in \mathbb{N}$ – задержка между предыдущим и текущим событиями, $a \in \Sigma$ – символ перехода.

Определение 6. Путь во временном автомате по трассировке $\{(a_i, d_i)\}$ – последовательность из пар $\{(q_i, \nu_i)\}$ таких, что $q_i \in Q$ – состояние на шаге i, ν_i – значения таймеров на шаге i, для которых выполняются следующие ограничения:

— $q_{2i} = q_{2i+1}$;
— $\forall x \in C : \nu_{2i+1}(x) - \nu_{2i}(x) = d_i$;
— $\exists! \langle q^1, q^2, a, g, R \rangle \in \Delta$:
  – $q_{2i+1} = q^1$;
  – $q_{2i+2} = q^2$;
  – $\forall x \in R : \nu_{2i+2}(x) = 0$;
  – $\forall x \notin R : \nu_{2i+2}(x) = \nu_{2i+1}(x)$;
  – $\forall x \in C : \nu_{2i+1}(x)$ удовлетворяет g.

Другими словами, в пути во временном автомате каждая пара из состояния и значений таймеров с четным номером отличается от предыдущей изменением состояния автомата, а также, возможно, сбрасыванием некоторого числа таймеров, каждая же пара с нечетным индексом отличается от предыдущей только увеличением значений таймеров.

Пример 7. Путем для трассировки $\{(a,10)\,(b,1)\,(c,1)\,(c,1)\,(d,1)\}$ в автомате, представленном на рисунке 1, будет являться последовательность:

$$\begin{aligned}\{&(q_0,\{\nu_1=0,\nu_2=0\}),(q_0,\{\nu_1=10,\nu_2=10\}),(q_1,\{\nu_1=0,\nu_2=10\}),\\&(q_1,\{\nu_1=1,\nu_2=11\}),(q_2,\{\nu_1=1,\nu_2=0\}),(q_2,\{\nu_1=2,\nu_2=1\}),\\&(q_2,\{\nu_1=2,\nu_2=0\}),(q_2,\{\nu_1=3,\nu_2=1\}),(q_2,\{\nu_1=3,\nu_2=0\}),\\&(q_3,\{\nu_1=4,\nu_2=1\})\}\end{aligned} \tag{1}$$

Определение 8. Детерминированный временной автомат $\mathcal{A}$ – временной автомат, для которого не существует трассировки, для которой существует два пути в данном временном автомате.

В дальнейшем, если это не оговорено заранее, под понятием временного автомата будет подразумеваться детерминированный временной автомат.

Определение 9. Языком $\mathcal{L}(\mathcal{A})$ временного автомата $\mathcal{A}$ назовем множество всех принимаемых им трассировок.

Определение 10. Класс автоматов $\mathbb{K}$ называется достижимым за полином, если существует полином p, такой что для любых $\mathcal{A}\in\mathbb{K}, q\in Q_{\mathcal{A}}$ существует трассировка τ такая, что порожденный ей путь заканчивается в q и $|\tau|<p(|\mathcal{A}|)$.

Приведем также определения более слабого класса временных автоматов – автоматов реального времени (real-time automata).

Определение 11. Автомат реального времени – временной автомат, такой что $C=\{c_0\}, R_\delta=\{c_0\}$ для любого перехода $\delta\in\Delta$.

Определение 12. Вероятностный детерминированный автомат реального времени – кортеж $\langle\mathcal{A},\mathcal{E},\mathcal{T},\mathcal{H}\rangle$, где

— $\mathcal{A}=\langle\Sigma,Q,q_0,C,\Delta,F\rangle$ – автомат реального времени;
— $\mathcal{E}: Q\times\Sigma\to[0,1]$ – распределение вероятностей на множестве символов перехода;
— $\mathcal{T}: Q\times\mathcal{H}\to[0,1]$ – распределение вероятностей на множестве времен ожидания для данного интервала $h\in\mathcal{H}$;
— $\mathcal{H}$ – конечное множество неперекрывающихся интервалов.

1.2. Теоретические результаты

Следующие результаты были получены авторами статьи [21].

Лемма 13. Класс временных автоматов не является достижимым за полином.

Лемма 13 фактически утверждает, что, чтобы однозначно правильно синтезировать автомат по примерам, необходимо в качестве примеров иметь пример экспоненциальной длины от размера временного автомата.

Определение 14. Класс автоматов $\mathbb{K}$ называется различимым за полином, если существует полином p, такой что для любых $\mathcal{A}, \mathcal{A}' \in \mathbb{K}, \mathcal{L}(\mathcal{A}) \neq \mathcal{L}(\mathcal{A}')$ существует трассировка τ, такая что $\tau \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}^C(\mathcal{A}') \cup \mathcal{L}^C(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$ и $|\tau| < p(|\mathcal{A}| + |\mathcal{A}'|)$

Лемма 15. Класс временных автоматов не различим за полином.

Определение 16. Характеристическим множеством языка $\mathcal{L}_t$ для алгоритма A называется множество трассировок $S_{cs} = \{S_{cs+}, S_{cs-}\}$, где $S_{cs+} \subseteq \mathcal{L}_t$, $S_{cs-} \subseteq \mathcal{L}_t^c$, такое что для любого $S \supseteq S_{cs}, S_+ \subseteq \mathcal{L}_t, S_- \subseteq \mathcal{L}_t^c$ алгоритм A на вход S вернет автомат $\mathcal{A}$ такой, что $\mathcal{L}(\mathcal{A}) = \mathcal{L}_t$.

Определение 17. Класс автоматов $\mathbb{K}$ является эффективно синтезируемым в пределе (identifiable in the limit), если существуют полиномы p, q и алгоритм A такие, что:

— время работы алгоритма на входе S ограничено сверху полиномом $p(\sum\limits_{\tau \in S} |\tau|)$;

— для любого языка $\mathcal{A} \in \mathbb{K}$ существует S_{cs} – характеристическое множество языка $\mathcal{A}$ для алгоритма A, такой что $\sum\limits_{\tau \in S_{cs}} |\tau| < q(|\mathcal{A}|)$.

Теорема 18. Класс временных автоматов не является эффективно синтезируемым.

Доказаны также более сильные варианты данной теоремы.

Теорема 19. Если $coNP \neq PSPACE$, то временной автомат не может быть эффективно синтезирован.

Теорема 20. Если $NP \neq PSPACE$, то временной автомат не может быть эффективно синтезирован.

Хоть и было доказано, что класс временных автоматов в целом не является эффективно синтезируемым, из него возможно выделить подкласс временных автоматов, который бы являлся эффективно синтезируемым. Приведем формулировку теоремы, которая немного проясняет ситуацию с временными автоматами.

Теорема 21. Класс временных автоматов, в котором используется два или больше таймеров, не может быть эффективно синтезирован.

Таким образом, возможно, стоит обратить внимание на подкласс временных автоматов, которые могут использовать всего один таймер? В статье [21] были также доказаны следующие факты.

Лемма 22. Класс временных автоматов с одним таймером является полиномиально достижимым.

Лемма 23. Класс временных автоматов с одним таймером является полиномиально различимым.

Теорема 24. Класс временных автоматов с одним таймером является эффективно синтезируемым.

В качестве доказательства теоремы 24 авторы статьи [21] предоставили алгоритм ID-DTA-1, который эффективно синтезирует временной автомат в пределе. При этом отметим, что задача синтеза минимального автомата с одним таймером все еще является NP-трудной.

1.3. Эффективный синтез временного автомата с одним таймером с помощью алгоритма ID-DTA-1

Впервые данный алгоритм был описан в статье [21]. Алгоритм основан на переборе. Идея алгоритма весьма проста. Он руководствуется правилом: если меня ничто не останавливает от совершения следующего действия, то я его совершу. Пусть имеется некоторый минимальный автомат, который необходимо синтезировать по трассировкам. Алгоритм будет строить автомат итеративно. Построение начинает с одного стартового состояния. За одну итерацию к существующему автомату будет добавляться либо один новый переход, либо новое состояние и переход в него. Значительное правило, которое используется в алгоритме: для любой числовой характеристики всегда существует строгий порядок перебора ее значений, не зависящий от множества переданных алгоритму трассировок. Данное свойство необходимо для доказательства существования характеристического множества.

Каждая итерация алгоритма начинается с определения состояния, из которого будет выходить следующий переход. Процедура заключается в переборе состояний в порядке их появления в текущем автомате. Выбирается первое состояние, для которого существует множество трассировок T_q, путь которых нужно продолжить из состояния q, то есть некоторый префикс путей трассировок заканчивается в данном состоянии q и не существует корректного перехода далее по автомату. Теперь алгоритм выбирает символ перехода для нового реб-

ра. Алгоритм перебирает символы перехода в лексикографическом порядке и выбирает первый символ a, для которого в множестве T_q существует трассировка, для продолжения пути которой по автомату требуется новый переход по символу a из состояния q. Как видно, перебор состояний и перебор символов перехода не зависит от множества трассировок: их алгоритм осуществляет всегда в одном и том же порядке. Единственное, на что влияет множество трассировок – это значение, которое выбирается первым.

Далее для ребра алгоритму необходимо определить ограничения перехода. Для этого необходимо определить его верхнюю и нижнюю границы. Сначала определим верхнюю границу. Верхняя граница перебирается сверху вниз, и выбирается первая, которая при выборе не внесет в автомат недетерминизм. Перебор всех возможных верхних границ слишком дорогостоящая операция из-за того, что границы записываются, как минимум, в бинарном виде, и их перебор может быть экспоненциальным по отношению к размеру автомата. Поэтому алгоритм помнит для каждого состояния q и символа перехода a предыдущую нижнюю границу $c'_{q,a}$. Тогда для нового перехода верхняя граница будет равна $c'_{q,a} - 1$. Изначально $c'_{q,a} = \infty$. Нижнюю границу, наоборот, он перебирает снизу вверх, но по тем же причинам делать напрямик слишком плохая затея. Очевидно, что достичь того же эффекта можно используя множество T_q. При выборе очередной границы множество T_q делится на две части: множество трассировок, которые будут проходить по данному ребру, и множество трассировок, которые, наоборот, проходить не будут. Чем ниже выбирается граница, тем большим становится первое множество, причем трассировки при понижении границы могут переходить только из второго множества в первое. Это означает, что как только в первое множество будет добавлена трассировка, которая ведет к неконсистентности в автомате, дальнейшее понижение границы не будет иметь смысла. Также нужно иметь ввиду, что нижнюю границу всегда можно опускать хотя бы до того момента, пока множества остаются неизменными. Значит нижняя граница точно равна значению таймера, увеличенному на единицу, в пути некоторой трассировки из T_q. Таким образом, алгоритм перебирает значения таймеров (увеличенных на единицу), трассировок в T_q, которым необходимо возникновение нового перехода с символом перехода a, делит эти трассировки на два множества, проверяет для первого множества

условие на консистентность, из всевозможных вариантов значений выбирает наименьшее и присваивает его нижней границе.

Далее определяется, нужно ли сбрасывать таймер после прохождения по синтезируемому переходу. В алгоритме таймер будет во множестве сброса, если при сбросе удастся получить интервал для ограничения не меньше, чем интервал для ограничения без сброса. Осталось выбрать вершину, в которую будет вести данный переход. Здесь все просто: также перебираются состояния в порядке появления их в автомате и выбирается первое, при проведении перехода в которое не образуется неконсистентности. Возможна также ситуация, когда такого состояния не существует. Тогда алгоритм добавляет новое состояние и проводит переход в него. Проверка на консистентность, упомянутая до этого, происходит не совсем обычным образом. Недостаточно просто проверить, в каких состояниях заканчиваются трассировки, ведь бывают трассировки, для которых еще не существует перехода, чтобы продолжить путь по автомату. Опасны две трассировки, которые в некоторый момент выравнивают значения таймеров и дальше переходят по одинаковым символам, ожидая одинаковое время. Таким образом, алгоритм ID-DTA-1 проверяет существование двух трассировок $\tau(a,t)\tau''$ и $\tau'(a,t+\nu-\nu')\tau''$, для которых пути для τ и τ' заканчиваются в состоянии q, имея значения таймеров ν и ν' соответственно.

Теперь разберем, как можно построить для данного алгоритма характеристическое множество, зная минимальный временной автомат, который нужно синтезировать алгоритму ID-DTA-1. Характеристическое множество будет строиться итеративно. Пусть на данной итерации уже имеется некоторое множество трассировок S и текущий временной автомат A, который является частью минимального временного автомата. Зная текущий автомат и характеристическое множество, можно легко определить, какой переход будет добавлен следующим в текущий автомат. Пусть он будет равен $\langle q, q', a, [c'', c'], r\rangle$. Если добавляемый переход не является частью минимального временного автомата, то нужно подкорректировать множество трассировок, чтобы оно запретило данный переход. Конкретнее, в данном переходе могут быть неверные границы перехода g, состояние q', в которое ведет переход, и множество сбросов r.

Предполагая, что минимальный автомат является полным, то есть из каждого состояния можно перейти с любым значением таймера, в минималь-

ном автомате существует переход с выбранной в текущем автомате верхней границей c' для нового перехода. Некорректной может быть лишь нижняя граница. Но в алгоритме граница c'' выбирается наименьшей из возможных, которые не приводят к неконсистентности. Значит нужно предоставить две новые трассировки, которые бы привели к неконсистентности, проходя по новому переходу. Если $t \in r$, то добавим в множество трассировок следующие трассировки: $\tau(a, c - \nu_{min})\tau'$, $\tau(a, c - \nu_{min} - 1)\tau')\tau'$, где τ – трассировка, путь которой заканчивается в состоянии q с наименьшим значением таймера ν_{min}, которая существует и имеет полиномиальную длину (по лемме 22), $c > c''$ - нижняя граница соответствующего перехода в минимальном автомате. Если в минимальном автомате существует разделение на два ребра: одно, которое имеет нижней границей ограничения c, второе – которое имеет верхней границей ограничения $c - 1$, то для множеств $\mathbb{L}_1 = \{\tau''|\tau(a, c - \nu_{min})\tau'' \in \mathbb{L}_t\}$ $\mathbb{L}_2 = \{\tau''|\tau(a, c - \nu_{min})\tau'' \in \mathbb{L}_t\}$, где $\mathbb{L}_t$ – язык минимального автомата, выполняется $\mathbb{L}_1 \neq \mathbb{L}_2$. Иначе можно было бы слить два данных ребра и минимальный автомат бы таковым не являлся. Выберем $\tau' : \tau' \in \mathbb{L}_1, \tau' \notin \mathbb{L}_2$ или наоборот. Трассировка τ' существует и имеет полиномиальную длину по теореме 23. Аналогично, при $t \notin r$ нужно добавить трассировки $\tau(a, c - \nu_{min})(b, t_1)\tau'$ и $\tau(a, c - \nu_{min} - 1)(b, t_1 + 1)\tau'$, чтобы создать неконсистентность при выборе нижней границы, меньшей чем c.

Чтобы устранить ненужный сброс, можно использовать трассировки $\tau(a, c - \nu_{min})\tau'$ и $\tau(a, c - \nu_{min} + 1)\tau'$, а чтобы, наоборот, установить сброс – трассировки $\tau(a, c - \nu_{min})(b, t_1)\tau'$, $\tau(a, c - \nu_{min} + 1)(b, t_1 - 1)\tau'$.

Если состояние q', в которое ведет переход, не равно состоянию q'', в которое ведет соответствующий переход в минимальном автомате, добавим в характеристическое множество следующие трассировки: одну принимаемую, другую отвергаемую: $\tau(a, c-\nu_{min})(b, t_1)\tau''$ и $\tau'(b, t_1')\tau''$, где путь трассировки τ заканчивается в состоянии q, τ' – в состоянии q', $\nu' + t_1' = \nu + t_1 + c - \nu_{min}$, ν, ν' – значения таймеров в конце путей τ и τ' соответственно. Таким образом, если алгоритм проведет переход в q', он столкнется в неконсистентностью временного автомата. Такие трассировки существуют, иначе минимальный автомат не является минимальным хотя бы по числу переходов.

Алгоритм ID-DTA-1 нужно воспринимать просто, как алгоритм, приведенный в доказательство теоремы 24. Его скорость работы и эффективность, а

также, возможно, плохие характеристические множества, не позволяют генерировать хорошие временные автоматы. В поисках эффективного алгоритма авторам приходится еще сильнее ограничить класс автоматов.

1.4. Эффективный синтез автомата реального времени с помощью алгоритма RTI

Данный алгоритм основан на адаптации одного из лучших алгоритмов синтеза конечных детерминированных автоматов Evidence driven state merging (EDSM) [4]. Он начинает синтез автомата с построения префиксного дерева трассировок. Префиксное дерево будет выступать в качестве первоначального автомата. Далее алгоритм пытается находить состояния, которые, как он предполагает, являются одинаковыми в минимальном конечном автомате, и сливать их.

EDSM синтезирует автомат в так называемой красно-синей структуре (red-blue framework). Данная структура представляет собой ни что иное, как разделение множества всех состояний на множество красных состояний, которые являются уже просмотренными, множество синих состояний, которые сейчас рассматриваются, и множество непокрашенных состояний, которые на данном шаге не рассматриваются. Красные состояния окаймлены синими, то есть для каждого синего состояния на текущем шаге есть красное состояние с ребром, которое ведет в него.

На каждом шаге выбирается один из двух вариантов: слить два состояния или покрасить синее состояние в красное. Слить можно либо два синих состояния, либо красное состояние с синим. Предполагается, что между двумя красными состояниями переходы никаким образом поменяться не могут. При слиянии двух состояний множества переходов, выходящих из этих двух состояний, объединяются. Из-за этого получившийся автомат может оказаться недетерминированным: из новой вершины, слитой из двух, могут выходить два ребра с переходом по одному и тому же символу. Данная проблема решается слиянием вершин, в которые ведут данные ребра. Тогда два ребра, из-за которых образуется недетерминизм, также сливаются в одно. Но не всегда два состояния можно слить, иначе бы всегда получался автомат из одной вершины. Предикатом на возможность слияния двух вершин будет то, что сливаемые вершины не должны быть разными по принятию трассировок (одна – принимающей, вторая – отвергающей), а также, если произошла ситуация недетерми-

низма, все потомки, которые необходимо слить, также должны удовлетворять предикату на возможность слияния. Возможна также ситуация, что на текущем шаге лучше покрасить синию вершину в красную, чем сливать вершины.

Нужно упомянуть, что синтез автомата в красно-синей структуре не ограничивает в возможности найти корректный минимальный автомат, ведь в некоторый момент каждая из вершин становится синей и именно в этот момент необходимо корректно определить, нужно ли ее слить с какой-то посещенной вершиной или же наоборот пропустить, чтобы потом слить с какой-нибудь еще непосещенной вершиной. Таким образом, размер автомата, который получится в конце, определяется стратегией, которую выбирают при синтезе автомата. Однако абсолютно правильную стратегию слияний и перекрасок выбрать сложно из-за того, что задача синтеза минимального детерминированного конечного автомата по трассировкам NP-полна [12], то для этого применяют эвристики. В алгоритме EDSM используется эвристика для оценки правильности некоторой операции. Эта оценка является просто натуральным числом. На текущей итерации оценивают правильность всевозможных слияний, всевозможных покрасок и применяют ту операцию, для которой оценка является наивысшей величиной среди остальных. В качестве оценки обычно выступает оценка качества автомата, который получается после применения данной операции. В EDSM обычно применяют следующую эвристическую оценку:

$$pure = \#pos + \#neg \tag{2}$$

В эвристике (2) $\#pos$, $\#neg$ – число слитых вершин, которые обе были принимающими и отвергающими соответственно. При равенстве нескольких наилучших оценок выбирают сначала слияние, а затем покраску. Иными словами, с данной эвристикой у покраски всегда оценка правильности равна нулю. Таким образом, она будет произведена только в случае, если больше не осталось вершин, которые можно было бы слить. В качестве вершины для покраски можно выбирать любую.

Авторы алгоритма RTI [20] адаптировали алгоритм EDSM, чтобы он мог учитывать время. Авторам статьи для этого пришлось добавить в исходный алгоритм еще одну операцию – разделение. Изначально строится такое же префиксное дерево, как и в EDSM, без учета временных составляющих трассировок, и каждому переходу в префиксном дереве выставляются максимальные

ограничения на переход. При таком подходе некоторые трассировки могли попасть в префиксном дереве в одну и ту же вершину, хотя, возможно, одна из них являлась принимаемой, а другая – отвергаемой. Таким образом, временной автомат изначально не всегда является консистентным. Но в конце алгоритма он должен быть таковым. Так как, как и алгоритм EDSM, RTI работает в красно-синей структуре, и между красными состояниями переходы не меняются, то необходимо, чтобы на каждом шаге красная часть автомата была консистентной, а также чтобы остальную часть автомата можно было превратить в консистентную операциями слияния, разделения и перекраски. Изначальную неконсистентность исправляют операциями разделения. Возможно провести разделение только тех ребер, которые ведут из красных состояний в синие. При разделении ребра крайне нежелательно, чтобы два новых ребра вели в то же самое состояние, так как в таком случае это никак не избавит от неконсистентности. Решается эта проблема глубоким копированием всего, куда ведет резделяемое ребро. Это можно сделать потому что, все, до чего можно добраться, проходя по разделяемому ребру, это либо синее, либо непокрашенное состояние. То есть, на самом деле, это префиксное дерево из непокрашенных вершин. Поэтому из него нет ребер в красные состояния, и данную часть автомата можно без опасений скопировать. Разделение еще применяют в качестве вспомогательной операции при слиянии. Заметим, что так как разделение можно применить только на ребрах из красных состояний в синие, ребра, выходящие из синих вершин, являются нетронутыми, а значит имеют изначальные максимальные ограничения на переход. Поэтому, перед тем как слить синее состояние с красным, алгоритм RTI разбивает ребра, выходящие из синего состояния, в тех же пропорциях, в которых находятся ребра, выходящие из красной вершины. В данном случае также применяется глубокое копирование выходящих префиксных деревьев. Проверка на то, что автомат можно достроить до консистентного, точно такая же, как и в алгоритме EDSM (предикат на возможность слияния).

Аналогично, нужно упомянуть, что добавленная операция разделения не повлияет на возможность нахождения минимального временного автомата: необходимо в момент, когда состояние, из которого выходят ребро, стало красным, а значит состояние, в которое входит, стало синим, провести правильное число разделений данного ребра в корректных местах. Но на практике у опера-

ции разделения куда меньший приоритет по сравнению с операцией слияния, так как при слиянии как раз и произойдет, скорее всего, самое правильное разбиение ребер.

В алгоритме RTI точно так же применяется некоторое множество эвристических оценок. Первая из них – это эвристика (2), заимствованная из алгоритма EDSM. Видно, что операция разделения при данной эвристической оценке автомата может только его ухудшить, что не всегда правда.

При равенстве нескольких наилучших оценок операций при применении любой эвристики приоритет отдается сначала слиянию, затем разделению, а затем покраске.

Следующая эвристика – это подправленная эвристика 2. В отличие в нее в данной эвристике также учитывается число новых неконсистентных состояний:

$$consistent = \#pos + \#neg - \#posneg \tag{3}$$

В эвристике (3) $\#pos$, $\#neg$ – также число слитых вершин, которые обе принимающие или отвергающие, $\#posneg$ – число вершин, которые являются неконсистентными, то есть существуют минимум две трассировки, одна из которых принимаемая, другая отвергаемая, которые заканчиваются в данном состоянии.

Следующая эвристика уже не является столь простой, но она по сути является расширением предыдущей эвристики, а именно расширяется понимание $\#posneg$. Ведь интуитивно кажется, что если трассировки, ведущие в неконсистентное состояние имеют большие разбросы по времени, то их легко разделить, и они не должны вносить сильный отрицательный вклад. А вот такие же строки, которые имеют практически одинаковые времена на пути к неконсистентному состоянию, должны давать, наоборот, большой отрицательный вклад. Таким образом, в этой эвристике особый упор отдается времени. Для каждых двух путей до листьев в префиксном дереве (временных строк), для которых одинаковы последовательности событий без времен, считается их похожесть. Так как их последовательности событий без времен одинаковы, они заканчиваются в одном и том же листе префиксного дерева. Похожесть двух строк определяется как вероятность того, что данные строки можно разделить, разделив равновероятно одно из ребер на пути от корня до листа, вы-

брав в качестве времени для разделения временного ограничения равновероятно любую его точку. Эвристика выглядит следующим образом:

$$\begin{aligned} impact = \sum_{q \in Q_r} pure(q) - \sum_{\tau \in \Delta_b} max\{\#impact(\tau, \tau') | \tau \in S^{\delta}_{+} \wedge \tau' \in S^{\delta}_{-}\} \\ + \sum_{\tau \in \Delta_b} max\{\#impact(\tau, \tau') | \tau \in S^{\delta}_{+} \wedge \tau' \in S^{\delta}_{+}\} \\ + \sum_{\tau \in \Delta_b} max\{\#impact(\tau, \tau') | \tau \in S^{\delta}_{-} \wedge \tau' \in S^{\delta}_{-}\} \end{aligned} \quad (4)$$

В эвристике (4) $\#impact(\tau, \tau')$ – похожесть двух временных строк τ и τ', $pure(q)$ – эвристическая оценка EDSM только для вершины q, S^{δ}_{+}, S^{δ}_{-} – принимаемые и отвергаемые трассировки соответственно, которые в своем пути по текущему автомату проходят по ребру δ.

Последняя эвристическая оценка тоже добавляет в изначальную оценку EDSM влияние неконсистентных вершин. В данном случае пытаются учесть величину того, сколько потребуется разделений, чтобы избавить непокрашенные префиксные деревья от неконсистентных вершин. Так как точное число посчитать сложно из-за NP-трудности данной задачи, применяется некоторый жадный алгоритм. Эвристическая оценка имеет следующий вид:

$$splits = \sum_{q \in Q} max(pure(q) - \#split(q), 0) \quad (5)$$

В эвристике (5) $\#splits(q)$ – приблизительное число разделений, необходимых для избавления от неконсистентных состояний в префиксном дереве, выходящем из вершины $q \in Q$.

При тестировании лучше всего себя показали эвристические оценки (3) и (5).

1.5. Подход к синтезу в алгоритме Timed k-Tail

Данный алгоритм Timed k-Tail [15] является адаптацией алгоритма k-Tail [7], применяемого для синтеза детерминированных конечных автоматов исключительно по позитивным примерам поведения. Алгоритм Timed k-Tail нацеливается на генерацию автомата, хорошо описывающего поведение реальных программ. Предполагается, что есть некоторая эталонная программа,

все трассировки которой являются корректными. Нужно построить некоторый автомат, имея только корректные трассировки, полученные при запусках данной программы, чтобы можно было, в частности, валидировать корректность работы аналогичных программ. Трассировки для данного алгоритма тоже подойдут не любые. События для данных трассировок делятся на два вида: начало некоторой функции и конец некоторой функции. Трассировка должна быть правильной скобочной последовательностью, если назначить начало функций в качестве открывающейся скобки, а конец в качестве закрывающейся скобки, причем у разных функций должны быть разные скобки. Алгоритм делится на несколько стадий.

Первая стадия данного алгоритма – нормализация. На вход изначально поступает некоторое множество трассировок выполнения реальной программы. Первое, что происходит в алгоритме, нормализация: трассировки изменяются таким образом, чтобы время старта для каждой трассировки было одинаковым: обычно от каждого времени в трассировке отнимают время первого события в данной трассировке, таким образом выходит, что первое событие каждой трассировки стартует в момент времени ноль. Однако все задержки времени между любой парой событий в трассировке сохраняются неизменными.

Вторая стадия – построение по трассировкам префиксного дерева. В данном алгоритме используется огромное число таймеров, чтобы замерять задержки между началом функции в некоторый момент времени и ее концом. Также будет существовать один нулевой (глобальный) таймер t, который нельзя будет сбрасывать. Видоизменим трассировки. Во-первых, каждая трассировка теперь будет аннотироваться не временем, а ограничениями и множеством таймеров для сброса. В качестве первого ограничения для каждого события j в трассировке i добавим $t = time_{i,j}$, где $time_{i,j}$ – момент времени, в который произошло событие j в трассировке i. Далее для каждого события начала функции добавим во множество сброса новый таймер. Пусть в трассировке i для события j этот таймер будет иметь номер c. Тогда в трассировке i нужно найти событие конца данной функции, пусть оно по номеру равно k, и в его аннотацию добавить еще одно ограничение $c = time_{i,k} - time_{i,j}$. Таким образом замеряется время работы каждого запуска какой-либо функции. Последнее, что нужно сделать, первому событию в каждой из трассировок до-

бавить во множество сброса таймер t, таким образом позволяя не думать о том, какими таймеры могут быть в корне будущего префиксного дерева, все они все равно в будущем будут сброшены. И только после данных операций по видоизмененным трассировкам нужно построить префиксное дерево.

Третья стадия – слияние состояний. На текущем этапе нужно выбрать величину k, присутствующую в названии данного алгоритма. Идея, которая главенствует на данной стадии синтеза автомата, гласит, что «скорее всего» система находится в одном и том же состоянии, если, как минимум, первые k событий, исходящие из двух состояний, совпадают. Но нужно также разобраться с ребрами, которые получаются при слиянии двух состояний. Алгоритм предписывает реберам, выходящим из сливаемых состояний с одинаковыми переходными символами, также слиться, при этом объединяя множества сбросов и множество ограничений.

Четвертая стадия – улучшение таймеров. К текущему моменту у автомата может возникнуть много таймеров, которые меряют одно и то же из-за того, что на предыдущей стадии произошло слияние состояний. На данном этапе нужно избавиться от таких таймеров. Пара таймеров меряет одно и то же, если каждый из них сбрасывается и проверяется на одних и тех же переходах. Находим все таймеры, которые меряют одно и то же, и оставляем из данного множества только один, остальные удаляем из всего автомата.

Пятая стадия – генерация ограничений. Чтобы автомат мог хоть что-то валидировать, необходимо расширить ограничения. На некоторых переходах в ограничениях таймер мог встречаться по несколько раз, например $t = 3 \wedge t = 5$. Понятно, что в текущем состоянии автомат ничего не способен принять. Поэтому на данной стадии заменяют все такие ограничения на одно таким образом, чтобы новое ограничение включило в себя все или почти все предыдущие равенства. Возвращаясь к примеру, логично будет заменить равенства на новое ограничение $3 \leqslant t \leqslant 5$. Замена на самом деле определяется политикой. В статье приводится два вида политик. Первая политика – в качестве нового ограничения выбирать промежуток $[(1-\epsilon)min, (1+\epsilon)max]$, где min, max – минимальное и максимальное значение чисел, встреченных в правых частях изначальных равенств. Вторая политика – в зависимости от параметра γ на основе изначальных равенств, выдать промежуток, куда попадут γ возможных

трассировок, в предположении, что изначальное распределение правых частей равенств было нормальным.

Данный алгоритм интересен своей простотой и нетребовательностью к разметке данных. Также интересен факт, что Timed k-Tail достаточно хорошо показал себя на практике: как оказалось, настолько простая модель способна хорошо, по сравнению с алгоритмом Perfume [1], схватывать поведение систем, использующих время, избегая переобучения (из-за чего Perfume оказался хуже).

1.6. Применение временных автоматов на практике. Алгоритм MONA

Некоторая модификация алгоритма RTI – алгоритм RTI+ [19], умеющий генерировать вероятностные детерминированные автоматы реального времени (определение 12), опираясь только на принимаемые трассировки, используется в алгоритме MONA [14] как одна из составных частей реализации. Опишем алгоритм MONA и применение в нем временных автоматов.

Данный алгоритм был применен на данных из двух открытых множеств траекторий NGSIM [5]: I80 и US101. В данных множествах положение автомобилей обновляется с частотой 10 Гц. Алгоритм состоит из нескольких стадий: сначала происходит крупнозернистая грануляция примеров, затем символизированные примеры подаются на вход RTI+, в результате работы которого появляется автомат, состояния данного автомата кластеризуются алгоритмом MISSI [14], образуя режимы в автомате, далее на данных режимах обучают модели отслеживания машин.

Чтобы можно было применить алгоритм RTI+, сначала необходимо сконвертировать данные в валидные. Для этого по имеющимся данным вычисляются переменные для любых двух подряд идущих транспортных средств: скорость транспортного средства, относительное расстояние до впередиидущего транспортного средства, относительная скорость двух транспортных средств и ускорение первого транспортного средства. Далее к полученным данным применяется алгоритм кластеризации K-means. Разные кластеры теперь обозначают разные символы перехода. Новое событие происходит, когда значительно меняется состояние транспортного средства, следовательно изменяется кластер, к которому оно принадлежит. В качестве символа перехода берется номер нового кластера. Символизированные таким образом примеры подаются на вход алгоритму RTI+.

По полученному автомату и поданным на его вход примерам формируются пути данных примеров по автомату. Отсечем от путей временную составляющую и, используя их, начнем кластеризовать состояния для получения режимов. Извлечем из путей подпути длины не меньше заданной константы L_{min} с встречаемостью чаще заданной константы ϵ и кластеризуем их по похожести. Режимы для состояний определяются путем голосования. То есть режимом для конкретного состояния будет являться номер кластера, в котором подпути чаще всего его использовали.

По результатам тестирования было установлено, что представленный алгоритм значительно превосходит алгоритмы, представленные до него, что является хорошим показателем важности использования временных автоматов в моделях машинного обучения и важности построения новых алгоритмов синтеза временных автоматов.

1.7. Задача выполнимости булевых формул и задача удовлетворения ограничений

Определение 25. Булевой называется переменная, принимающая значения из множества $\{0, 1\}$.

Определение 26. Булеву формулу A можно получить слудующими способами:

— $A := x$, где x – булева переменная;

— $A := B * C$, где B, C – булевы формулы, $*$ – одна из операций $\wedge, \vee$;

— $A := \neg B$, где B – булева формула.

Определение 27. Задача выполнимости булевой формулы (SAT) – по заданной формуле необходимо найти значения ее булевых переменных так, чтобы формула стала истинной, либо доказать, что сделать это невозможно.

Определение 28. Задача удовлетворения ограничений (CSP) – задача поиска решения формулы исчисления предикатов, где используются только двуместные предикаты $\leqslant, \geqslant$, либо доказательства, что сделать это невозможно.

Задача выполнимости булевой формулы и задача удовлетворения ограничений являются известными NP-полными задачами. Одной из первых была доказана NP-полнота задачи удовлетворения булевых формул (теорема Кука-Левина 1971-1973). После этого доказательства NP-трудности и NP-полноты стали достаточно тривиальными: в первом случае нужно свести задачу удовлетворения булевых формул к текущей задаче, а во втором случае – в дополнение

к первому нужно еще и предоставить обратное сведение. Из-за того, что вышеупомянутые задачи чаще всего используются для доказательств NP-полноты, было разработано большое число пакетов программ, позволяющих их решать за хоть сколько-то разумное время.

Современные SAT-решатели основаны на алгоритмах DPLL [10], CDCL [9]. Наиболее известные SAT-решатели: GRASP [13], VSIDS [8], MiniSAT [18]. CSP-решатели в свою очередь в своей реализации задействуют SAT-решатели. Наиболее известные их них: Sugar [17], AbsCon [6], minizinc [16].

Minizinc [16] – это бесплатно распространяемый язык для записи ограничений с открытым исходным кодом. Можно использовать Minizinc для задач удовлетворения ограничений модели и оптимизационных задач. Язык является высокоуровневом и независимым от решателя, вбирая в себя плюсы большой библиотеки уже записанных ограничений и предикатов. Написанные на данном языке ограничения транслируются во более низкоуровневый язык FlatZinc [11], понимаемый большинтвом SAT-решателей.

К задаче выполнимости булевых формул можно свести задачу построения минимального детерминированного конечного автомата [3]. Задача генерации минимального временного автомата представляет собой более сложную задачу из-за использования таймеров и ограничений на них. Тем не менее она легко сводится к задаче удовлетворения ограничений.

Выводы по главе 1

Задача генерации минимального временного автомата является актуальной. В попытках ее решения была доказана теорема об эффективном синтезе временных автоматов, использующих один таймер, и разработаны алгоритмы, базирующиеся на данном доказательстве и на других эвристических подходах. Однако задача остается NP-полной. Логично попробовать метод сведения данной задачи к задаче выполнимости булевых формул или задаче удовлетворения ограничений и оценить результаты.

ГЛАВА 2. ПРЕДЛАГАЕМЫЙ МЕТОД СИНТЕЗА ВРЕМЕННОГО АВТОМАТА НА ОСНОВЕ ПРОГРАММИРОВАНИЯ В ОГРАНИЧЕНИЯХ

2.1. Постановка задачи

На текущий момент существуют методы генерации временных автоматов в пределе (ID-DTA-1, RTI), а также эвристические методы генерации временных автоматов (Timed k-Tail), однако отсутствует алгоритм, гарантированно генерирующий минимальный временной автомат. Целью данной работы является разработка такого алгоритма на основе сведения к задаче CSP. Для этого в данной работе решались следующие задачи.

а) Изучение возможностей программного пакета Minizinc [16], предназначенного для решения задач удовлетворения граничений.
б) Разработка сведения задачи к задаче удовлетворения ограничений.
в) Реализация решения на основе сведения и сравнение с существующим решением.

2.2. Переменные, используемые в решателе задачи удовлетворения ограничений

Будем представлять автомат следующим образом. Для каждого состояния будет иметься информация о каждом переходе: в какое состояние автомата ведет, какой переходный символ использует и для каждого таймера его нижнюю и верхнюю границы. Также для каждого состояния необходимо знать, является ли оно принимающим или же отвергающим. Префиксное дерево будет представляться в виде списка ребер: из какой вершины выходит, в какую вершину идет, какой переходный символ записан на ребре и какая задержка была между предыдущим переходным символом и данным. Общая идея сведения: сопоставить каждой вершине префиксного дерева состояние автомата и значения таймеров таким образом, что для каждому ребру префиксного дерева будет сопоставлен единственный переход в автомате, который имеет тот же самый символ перехода, значения таймеров, сопоставленные состоянию, из которого ведет ребро, удовлетворяет ограничения перехода, а также значения таймеров на вершине, в которое ведет данное ребро, обновляется в соответствии с задержкой на ребре и множеством сбросов на переходе автомата.

Пример 29. Построим для множества принимаемых трассировок A и множества отвергаемых трассировок B префиксное дерево, приведенное на рисун-

ке 2, а также автомат, приведенный на рисунке 3.

$$\begin{aligned} A &= \{\{(a,0)\,(b,0)\},\{(a,1)\,(b,1)\}\} \\ B &= \{\{(a,1)\,(b,0)\}\} \end{aligned} \tag{6}$$

Можно увидеть, что вершине перфиксного дерева p_1 сопоставилось состояние автомата q_1, вершины p_2 – состояние q_2, вершине p_3 – состояние q_2, вершине p_4 – состояние q_3, вершине p_5 – состояние q_3, вершине p_6 – состояние q_3. Состояние q_2 является отвергающим, отвергая все трассировки множества B, а состояние q_3 является принимающим, принимая все трассировки из множества A.

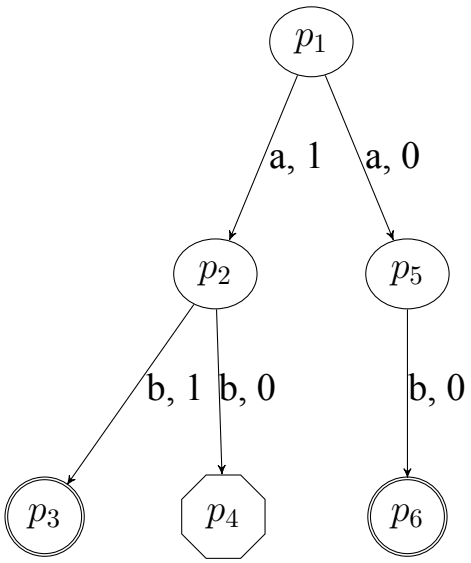

Рисунок 2 – Пример префиксного дерева

Введем переменные, используемые при синтезе временного автомата:

— V – число состояний, которые в данный момент использует решатель системы ограничений;
— T – число таймеров, которые используются во временном автомате;
— E – максимальная степень вершины, которая может использоваться в автомате;
— M – число ребер, используемое в префиксном дереве;
— $N = M + 1$ – число вершин, используемое в префиксном дереве;
— inf – верхняя граница для ограничения на переходе. Обычно при синтезировании автомата с n таймерами в качестве верхней границы выбирается максимальная сумма ожиданий переходов по каждой из трассировок, при синтезировании временного автомата RTA в качестве верхней

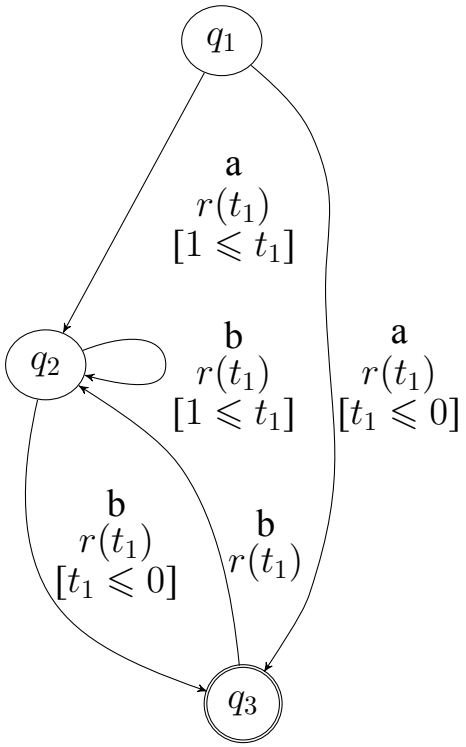

Рисунок 3 – Пример построенного автомата

границы выступает максимальное ожидание среди всех ожиданий каждой из трассировок;

— TC – максимальное число активных таймеров, которые можно использовать во временном автомате;

— TE – максимальное число переходов, которое можно использовать во временном автомате;

— $COL = \{W, G, B\}$ – множество цветов, в которые могут быть покрашены вершины префиксного дерева. W обозначает, что в данном состоянии заканчивается отвергаемая трассировка, B – принимаемая, G в свою очередь означает, что в данном состоянии не заканчивается ни одна из трассировок;

— S – множество используемых в трассировках событий;

— $labels : 1..M \to S$ – события каждого из переходов $m \in 1..M$ в префиксном дереве;

— $next : 1..M \to 1..N$ – вершины, из которого выходят ребра $m \in 1..M$;

— $prev : 1..M \to 1..N$ – вершины, в которые идут ребра $m \in 1..M$;

— $acc : 1..M \to COL$ – цвет вершины $m \in 1..M$.

— $times : 1..M \to 0..inf$ – времена ожидания переходов $m \in 1..M$;

— $table : 1..V \times 1..E \to 1..V$ – таблица переходов состояний: состояния, в которые переходят из состояния $v \in 1..V$ по переходу из данного состояния $e \in 1..E$;

— $symbols : 1..V \times 1..E \to S$ – таблица переходов событий: события, необходимые для перехода из состояния $v \in 1..V$ по переходу из данного состояния $e \in 1..E$;

— $final : 1..V \to \{False, True\}$ – условия для каждого из состояний $v \in 1..V$, что оно является принимающим;

— $mn : 1..V \times 1..E \times 1..T \to 0..inf$ – нижние границы ограничений для каждого из состояний $v \in 1..V$, каждого из переходов из данного состояния $1..E$ и каждого из таймеров $t \in 1..T$;

— $mx : 1..V \times 1..E \times 1..T \to 0..inf$ – верхние границы огранчений для каждого из состояний $v \in 1..V$, каждого из переходов из данного состояния $1..E$ и каждого из таймеров $t \in 1..T$;

— $reset : 1..V \times 1..E \times 1..T \to \{False, True\}$ – условие для каждого из состояний $v \in 1..V$, каждого из переходов из данного состояния $e \in 1..E$, каждого из таймеров $t \in 1..T$, что данный таймер сбрасывается на данном переходе из данного состояния;

— $disTimer : 1..V \times 1..E \times 1..T \to \{False, True\}$ – условие для каждого из состояний $v \in 1..V$, каждого из переходов из данного состояния $e \in 1..E$, каждого из таймеров $t \in 1..T$, что данный таймер не используется на данном переходе из данного состояния;

— $disEdge : 1..V \times 1..E \to \{False, True\}$ – условие, что для каждого из состояний $v \in 1..V$, каждого из переходов из данного состояния $e \in 1..E$, что данное ребро не используется. То есть в автомате переходы, для которых условие выполняется, не существуют;

— $map : 1..N \to 1..V$ – вершины из временного автомата, соответствующие вершинам из префиксного дерева;

— $after : 1..N \times 1..T \to 0..inf$ – значения таймеров $t \in 1..T$ сразу после перехода по ребрам, ведущим в вершины $q \in 1..N$ префиксного дерева. Для корня префиксного дерева можно считать, что существует некоторое невыделяемое на рисунках ребро, по которому в него приходят;

— $before : 1..M \times 1..T \to 0..inf$ – значения таймеров $t \in 1..T$ сразу перед переходом по ребрам $e \in 1..M$ префиксного дерева;

— $parents : 1..V \to 1..V$ – вершины, из которых в первый раз попадают в соответствующие вершины $v \in 1..V$ при обходе автомата алгоритмом BFS;

— $minEdge : 1..V \to 1..E$ – минимальные по номерам ребра, ведущие из родительских вершин в вершины $v \in 1..V$.

2.3. Ограничения, используемые в решателе системы ограничений

Было принято решение разбить ограничения для генерации минимального временного автомата на пять блоков:

а) Ограничения, запрещающие недетерминизм – ограничения, запрещающие существование трассировки, имеющей несколько разных путей в автомате.

б) Основной предикат на переходы – предикат, который сопоставляет вершине префиксного дерева состояние автомата.

в) Ограничения на начальное состояние – ограничение, задающие переменные для корня префиксного дерева.

г) Ограничения на минимальность – ограничения, запрещающие существование характеристик автомата выше заданных.

д) Ограничения на допускающие состояния – ограничения, требующие, чтобы все трассировки из множества заканчивались в нужном состоянии (принимаемые – в допускающем, отвергаемые – в недопускающем).

Приведем множество ограничений, используемых в решателе систем ограничений.

2.3.1. Ограничения, запрещающие недетерминизм

Исходя из определения (8) детерминированного временного автомата, не должно существовать трассировки, которая порождает два или более корректных пути по временному автомату. То есть, не должно существовать трассировки, путь которой заканчивается в некотором состоянии с некоторым набором значений таймеров, и существует два перехода по разным ребрам, удовлетворяя ограничения на данных ребрах. Но такого вида условие было бы слишком тяжелым для решателя систем ограничений. Запишем несколько более сильное, но менее общее, ограничение, запрещающее недетерминизм.

В ограничении, описанном на языке Minizinc в листинге 1, не допускается существование двух переходов по одному символу из одного состояния,

для которых гиперпараллелепипеды, составленные из ограничений каждого таймера, пересекаются. Очевидно, чтобы недопустить пересечение гиперпараллелепипедов необходимо иметь хотя бы один таймер, ограничения которого для первого и второго перехода не пересекаются, то есть нижняя граница ограничения данного таймера на первом переходе больше чем на втором, или наоборот.

Листинг 1 – Ограничение, запрещающее недетерминизм

```
constraint
forall (s  in 1..V,
        e1 in 1..E,
        e2 in 1..E
  where e1 != e2)
(symbols[s, e1] = symbols[s, e2] ->
 (disEdge[s, e1] \/
  disEdge[s, e2] \/
  exists (t in 1..T)
  (mx[s, e1, t] < mn[s, e2, t] \/
   mx[s, e2, t] < mn[s, e1, t]))));
```

2.3.2. Основной предикат на переходы

Основной предикат, записанный на листинге 2 – предикат на то, что переходу в префиксном дереве соответствует данный переход в автомате. Чтобы переходу в префиксном дереве соответствовал переход в автомате, необходимо выполнение следующих условий:

— символ перехода в префиксном дереве равен символу перехода в автомате;
— состоянию, из которого ведет переход префиксного дерева, соответствует состояние, из которого ведет переход автомата;
— состоянию, в которое ведет переход префиксного дерева, соответствует состояние, в которое ведет переход автомата;
— значения каждого из таймеров непосредственно до перехода удовлетворяют ограничениям перехода по каждому из таймеров;
— значения таймеров сразу после перехода в префиксном дереве равны значениям таймеров непосредственно перед переходом в случае, если таймеры не находятся во множестве сброса перехода, иначе равны нулю.

Применим описанный выше предикат дважды: необходимо, чтобы для каждого перехода из префиксного дерева существовал единственный соответ-

Листинг 2 – Основной предикат на переходы

```
predicate check (1..M: pe, 1..E: e) =
 symbols[map[prev[pe]], e] = labels[pe] /\
 map[next[pe]] = table[map[prev[pe]], e] /\
 forall (t in 1..T)
 (before[pe, t] >= mn[map[prev[pe]], e, t] /\
  before[pe, t] <= mx[map[prev[pe]], e, t] /\
  after[next[pe], t] = if reset[map[prev[pe]], e, t] then 0 else
     before[pe, t] endif);
```

ствующий ему переход в автомате, а также необходимо, чтобы для каждого перехода в автомате существовал как минимум один соответствующий переход в префиксном дереве. Это описывают ограничения, приведенные в листинге 3.

Листинг 3 – Использование основного предиката

```
constraint
forall (pe in 1..M)
(exists (e in 1..E)
 (check (pe, e)));

constraint
forall (s in 1..V,
        e in 1..E)
(disEdge[s, e] \/
 exists (pe in 1..M)
 (map[prev[pe]] = s -> check (pe, e)));
```

2.3.3. Ограничения на начальное состояние

Ограниченииями на начальное состояния будут являться:

— корню префиксного дерева соответствует стартовая вершина в автомате;

— значения всех таймеров сразу после перехода в стартовую вершину равны нулю.

Данные ограничения записаны на листинге **??**.

Листинг 4 – Стартовые ограничения

```
constraint
forall (t in 1..T)
(after[1, t] = 0);

constraint
map[1] = 1;
```

2.3.4. Ограничения на минимальность

В первую очередь нас интересует минимальность числа состояний данного автомата. Но состояния автомата перебираются вне решателя систем ограничений. Решатель только пытается удовлетворить переданные ему ограничения. Также необходимо, чтобы в получившемся автомате было минимальное число переходов и минимальное число активных таймеров. Таймер активен на переходе автомата, если либо верхняя граница ограничения для данного таймера не равна inf, либо нижняя граница ограничения для данного таймера не равна нулю. Таким образом, необходимые ограничения приведены в листингах 5 и 6

Листинг 5 – Ограничение на верхнюю границу числа ребер

```
constraint
constraint
sum   (s in 1..V,
       e in 1..E)
(1 - bool2int (disEdge[s, e])) <= TE;
```

Листинг 6 – Ограничение на верхнюю границу числа активных таймеров

```
constraint
sum   (s in 1..V,
       e in 1..E,
       t in 1..T)
(1 - bool2int (disTimer[s, e, t] \/ disEdge[s, e])) <= TC;
```

Ограничение в листинге 5 требует, чтобы число ребер не превосходило TE. Ограничение в листинге 6 требует, чтобы число активных таймеров, не превосходило TC.

2.3.5. Ограничение на допускающие состояния

Так как в решатель передается префиксное дерево, каждое из состояний которого покрашено в один из трех цветов, автомат должен учитывать эту информацию при покраске своих вершин. Состояние автомата может быть принимающим только в случае, когда вершины префиксного дерева, которым оно сопоставлена, имеют цвета либо G, либо B. Отвергающим же состояние имеет право быть только в случае, если вершины префиксного дерева, которым оно сопоставлено, имеют цвета либо G, либо W. Запишем это в ограничениях, приведенных в листинге 7.

Листинг 7 – Ограничения на допускающие состояния

```
constraint
forall (s  in 1..V,
        ps in 1..N)
(map[ps] = s -> (acc[ps] = B -> final[s]));

constraint
forall (s  in 1..V,
        ps in 1..N)
(map[ps] = s -> (acc[ps] = W -> (not final[s])));
```

То есть, если состоянию ps префиксного дерева соответствует состояние s автомата, а также состояние ps покрашено в цвет B, то состояние s обязано быть принимающим. То же самое для цвета W и отвергаемого состояния.

2.3.6. Ограничения нарушения симметрии на основе обхода в ширину

Ограничения (BFS) впервые были представлены в работе [22]. Они позволяют значительно сократить область поиска нужного детерминированного конечного автомата за счет установления нумерации вершин в порядка обхода BFS. Для временных автоматов можно применить ту же самую идею.

Ограничения BFS представляют собой ограничения на порядок состояний и ограничения на порядок переходов. Первое ограничение, записанное в листинге 8 – ограничение на порядок состояний, которое требует, чтобы состояния, в которые можно прийти из меньших но номеру состояний, имели номер меньше, чем состояния, в которые можно прийти из больших по номеру состояний.

Наложим ограничения на $parents$ и $minEdge$. В алгоритме BFS при обходе графа используется очередь. Соответственно, вершины, попавшие в очередь раньше, раньше извлекаются и получают меньший номер. Причем после извлечения в конец очереди добавляются вершины в порядке просмотра ребер, которые еще не были до этого в очереди, но в них можно перейти из только что извлеченной вершины по соответствующему ребру. Тогда для данного состояния a $parents[a]$ должно указывать на состояние a', которое должно является наименьшим по номеру состоянием, из которого можно попасть в вершину, а $minEdge$ должно указывать на наименьшее по номеру ребро, ведущее из a' в a. Таким образом, в листинге 8 второе ограничение требует, чтобы для любого ребра меньше чем $minEdge[a]$ переход из $parents[a]$ ведет в отличную от a

вершину, а также из любого состояния, меньшего по номеру чем $parents[a]$, по любому переходу нельзя попасть в a.

Листинг 8 – BFS-ограничения

```
constraint
forall (s1 in 2..V,
        s2 in 2..V
  where s1 < s2)
(parents[s1] < parents[s2] \/
 parents[s1] = parents[s2] /\
 minEdge[s1] < minEdge[s2]);

constraint
forall (s in 2..V)
(table[parents[s], minEdge[s]] = s /\
 forall (e in 1..E)
 (e < minEdge[s] -> table[parents[s], e] != s) /\
 forall (s1 in 1..V)
 (s1 < parents[s] ->
  forall (e in 1..E)
  (disEdge[s1, e] \/ table[s1, e] != s)));
```

Единственное, что осталось сделать – определить в каком порядке перебирать переходы. Ведь может встретиться два перехода с одинаковыми символами или переходы с одинаковыми ограничениями. Чтобы задать на них полный порядок, обычно перебирают каждый из параметров в некотором порядке и делают выбор минимального перехода лексикографически. В данном случае это можно сделать ограничением, заданным в листинге 9: сравним переходы сначала по символу, затем по нижней границе первого таймера, затем по верхней границе второго таймера, затем по нижней границе второго таймера, затем по верхней границе второго таймера и так далее. Первое ограничение в листинге 9 тоже немного обрезает область поиска: все неактивные ребра будут находиться последними в списке ребер, выходящих из данной вершины. Однако последнее ограничение в листинге 9 является очень трудным для решателя систем ограничений, поэтому порой оно может быть опущено.

2.3.7. Ограничения для автоматов реального времени

Простой способ превратить алгоритм поиска временного автомата с n таймерами в алгоритм поиска автоматов реального времени – передать в решатель в качестве $T = 1$, а также добавить ограничения:

Листинг 9 – Ограничения на порядок переходов

```
constraint
forall (s in 1..V,
        e in 1..(E - 1))
(disEdge[s, e] -> disEdge[s, e + 1]);

constraint
forall (s in 1..V,
        e in 1..(E - 1))
((not disEdge[s, e]) ->
 ((not disEdge[s, e + 1]) ->
  symbols[s, e] < symbols[s, e + 1] \/
  (symbols[s, e] = symbols[s, e + 1] /\
   exists (t in 1..T)
   ((mn[s, e, t] < mn[s, e + 1, t] \/
     (mn[s, e, t] == mn[s, e + 1, t] /\
      mx[s, e, t] < mx[s, e + 1, t])) /\
    forall (t0 in 1..T)
    (t0 < t ->
     mn[s, e, t0] == mn[s, e, t0] /\
     mx[s, e, t0] == mx[s, e, t0])))));
```

Листинг 10 – Ограничения RTA

```
constraint
forall (s in 1..V,
        e in 1..E,
        t in 1..T)
(reset[s, e, t]);
```

2.4. Перебор параметров сведения

С генерацией минимальных временных автоматов дело обстоит несколько сложнее, чем с генерацией минимальных детерминированных конечных автоматов. Первый признак того, что все не очень просто – невозможность как-то разумно ограничить число выходящих переходов из вершины. Ведь в обычном детерминированном конечном автомате можно было просто ограничить данное число мощностью алфавита символов переходов. В данной же задаче по одному и тому же символу перехода из данного состояния может выходить несколько переходов, имея непересекающиеся гиперпараллелепипеды возможных значений таймеров. Далее приходит понимание того, что понятие минимальности само по себе довольно расплывчато. Рассмотрим следующий возможный случай. Пусть есть множество трассировок, в которых задержки во времени задаются сколь угодно точно. Тогда, если данные трассировки по-

лучились случайным образом, очень вероятна ситуация, когда всевозможные задержки во времени будут различными. А это означает, что есть возможность сгенерировать автомат из двух состояний, принимающего и отвергающего, и с огромным количеством переходов. Главная цель данного автомата – направить трассировку по последнему переходу, так как безразлично, как она ведет себя до последнего перехода, ведь в конце все равно возможно будет их различить и направить в правильное состояние. Тогда можно понять, что в реальности можно встретить несколько валидных «минимальных» пар из количества состояний и числа ребер, лежащих на Парето-фронте. Однако в отличие от числа состояний во временном автомате действительно хорошим показателем его минимальности является число ребер.

Основной алгоритм можно представить псевдокодом на листингах 11. В неосновном цикле, со строки 3 до строки 7, производится заполнение и последующая сортировка пар из числа состояний и степени вершины. Алгоритм перебирает число состояний и степень вершины в порядке увеличения числа переходов. Также алгоритм поддерживает внутри себя текущее множество трассировок (строка 9). Постепенно данное множество пополняется. Этот метод постепенного пополнения числа рассматриваемых трассировок известен как Counter Example Guided Refinement (CEGAR) [2]. В втором цикле, со строки 17 до строки 34, идет поиск первой пары из числа состояний и степени вершины, для которой автомат построенный с такими характеристиками, по префиксному дереву, построенному из текущего множества трассировок, являлся бы консистентным со всеми трассировками. Если же построенный автомат не является консистентным со всем множеством трассировок, то в текущее множество добавляется еще одна трассировка, которую построенный автомат не удовлетворил, и которая из таких является минимальной по длине.

После данного цикла идет вызов двух процедур минимизации. Минимизация сначала происходит по числу переходов (листинг 12), а затем по числу активных таймеров (листинг 13). Минимизация числа переходов нужна в том случае, когда состояния имеют различную степень. Действуют два цикла одинаково: пытаются построить автомат с меньшей характеристикой чем имеющаяся и, если то удается, новый автомат опять же проходит проверку на консистентность со всеми трассировками, так как при минимизации нужно не забывать, что главная цель – удовлетворить все трассировки. Если новый автомат

неконсистентен со всеми трассировками, то также в текущее множество добавляется минимальная неудовлетворимая. Данный подход хорош по причине того, что в самом первом цикле алгоритм перебирает элементы из $\mathbb{N}^2$. А значит перед тем, как он найдет нужную пару, придется отсечь много неправильных пар, что хотелось бы делать, имея малое число состояний в префиксном дереве. Причем, в алгоритм редко добавляются абсолютно все трассировки, самые длинные и часто удовлетворимые остаются нетронутыми.

2.5. Реализация

Описанный в предыдущей части алгоритм реализован на языке программирования Kotlin. Реализацию и данные к ней можно найти в репозитории по ссылке https://github.com/slelaron/bachelor.

Выводы по главе 2

В данной главе была описана и разъяснена идея предложенного метода, описано и разъяснено сведение к задаче удовлетворения ограничений (основные сущности и ограничения).

Листинг 11 – Псевдокод предложенного алгоритма

```
1: function Генерация консистентного с трассировками автомата(traces, timersNumber)
2:     order ← []
3:     for states ← [1; ∞] do
4:         for vertexDegree ← [1; ∞] do
5:             order += (states, vertexDegree)
6:         end for
7:     end for
8:     Отсортировать order по величине states * vertexDegree
9:     activeTraces ← []
10:    activeTraces += минимальная трассировка из traces по длине
11:    states ← 0
12:    vertexDegree ← 0
13:    edges ← 0
14:    activeTimers ← 0
15:    automaton ← emptyAutomaton
16:    index ← 1
17:    loop
18:        (statesNumber, vertexDegreeNumber) ← order[index]
19:        states ← statesNumber
20:        vertexDegree ← vertexDegreeNumber
21:        edges ← states * vertexDegree
22:        activeTimers ← edges * timersNumber
23:        prefixTree ← префиксное дерево по трассировкам из activeTraces
24:        automaton ← решить ограничения с параметрами states, vertexDegree, edges, activeTimers, prefixTree
25:        if automaton ≠ NULL then
26:            if automaton конситентен с трассировками из traces then
27:                break
28:            else
29:                activeTraces += минимальная неудовлетворимая трассировка из traces
30:            end if
31:        else
32:            index ← index + 1
33:        end if
34:    end loop
35:    Минимизация числа переходов(automaton, timersNumber, states, vertexDegree, activeTraces, traces)
36:    Минимизация числа активных таймеров(automaton, timersNumber, states, vertexDegree, edges, activeTraces, traces)
37:    return automaton
38: end function
```

Листинг 12 – Псевдокод минимизирующей число переходов процедуры

```
1: procedure Минимизация числа переходов(automaton, timersNumber,
   states, vertexDegree, activeTraces, traces)
2:     edges ← число ребер в automaton
3:     edgesNumber ← edges − 1
4:     loop
5:         edges ← edgesNumber
6:         activeTimers ← edges ∗ timersNumber
7:         prefixTree ← префиксное дерево по трассировкам из
   activeTraces
8:         newAutomaton ← решить ограничения с параметрами states,
   vertexDegree, edges, activeTimers, prefixTree
9:         if newAutomaton ≠ NULL then
10:            if newAutomaton конситентен с трассировками из traces then
11:                automaton ← newAutomaton
12:                edgesNumber ← edgesNumber − 1
13:            else
14:                activeTraces += минимальная неудовлетворимая трасси-
   ровка из traces
15:            end if
16:        else
17:            break
18:        end if
19:    end loop
20: end procedure
```

Листинг 13 – Псевдокод минимизирующей число активных таймеров процедуры

```
1: procedure Минимизация числа активных таймеров(automaton, timersNumber, states, vertexDegree, edges, activeTraces, traces)
2:     activeTimers ← число активных таймеров в automaton
3:     activeTimersNumber ← activeTimers − 1
4:     loop
5:         activeTimers ← activeTimersNumber
6:         prefixTree ← префиксное дерево по трассировкам из activeTraces
7:         newAutomaton ← решить ограничения с параметрами states, vertexDegree, edges, activeTimers, prefixTree
8:         if newAutomaton ≠ NULL then
9:             if automaton конситентен с трассировками из traces then
10:                automaton ← newAutomaton
11:                activeTimersNumber ← activeTimersNumber − 1
12:            else
13:                activeTraces += минимальная неудовлетворимая трассировка из traces
14:            end if
15:        else
16:            break
17:        end if
18:    end loop
19: end procedure
```

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

Как и в [20], экспериментальное исследование разработанного метода было проведено на случайных сгенерированных автоматах. Были сгенерировано тестовое множество для всевозможных комбинаций параметров:

— число состояний: $2, 3, 4, 5, 6, 7, 8$;
— число символов переходов: $2, 3, 4$;
— число делений переходов: $2, 4, 8$;
— число, принимаемое за бесконечность: $10, 20$;

Суммарное число тестов получилось равным $7 \times 3 \times 3 \times 2 = 126$.

В качестве тренировочного и тестового множества для каждого автомата было сгенерировано по 1000 случайных трассировок. Данные трассировки могли закончиться в состоянии, в которое они пришли, с вероятностью $0{,}1$, следовательно средняя длина одной трассировки равна 10. Следующий переходный символ выбирался равновероятно. Задержка выбиралась равновероятно из промежутка $[0..\infty]$, где ∞ – максимальная величина ограничений на таймеры в целевом автомате.

Алгоритм RTI, использующий эвристику consistent EDSM, был протестирован на полном тренировочном множестве. Разработанному же решению было решено подать на вход 20 и 30 случайных трассировок из тренировочного множества соответственно. Будем различать данные способы как Алгоритм-20 и Алгоритм-30 соответственно. Так как сравнение проводилось с алгоритмом RTI, разработанный алгоритм также был ограничен на класс автоматов RTA: в изначальные условия было добавлено ограничение, чтобы сбросы таймера после перехода всегда срабатывали. При каждом запуске генерации автоматов любым из трех способов (Алгоритмом-20, Алгоритмом-30 и RTI) было использовано ограничение по времени, равное 15 минутам.

Данные эксперименты проводились на компьютере с процессором Intel Core i5 5200U (2,2 ГГц) и 4 Гб оперативной памяти.

В таблице 1 приведены полученные результаты. Чтобы данные поместились пришлось отказаться от полных названий заголовков. Расшифровка заголовков приведена далее:

— N – число состояний в целевом автомате;
— A – число переходных символов в целевом автомате;
— S – число разделений ребер целевого автомата;

— E – число ребер в целевом автомате;
— ∞ – максимальная величина ограничений на таймеры в целевом автомате;
— T20/T30/T* – время работы Алгоритма-20, Алгоритма-30 и RTI соответственно;
— N20/N30/N* – число состояний в автомате, полученном Алгоритмом-20, Алгоритмом-30 и RTI соответственно;
— E20/E30/E* – число ребер в автомате, полученном Алгоритмом-20, Алгоритмом-30 и RTI соответственно;
— P20/P30/P* – процент корректно принимаемых трассировок из тестового множества автоматом, полученным Алгоритмом-20, Алгоритмом-30 и RTI соответственно;
— F20/F30/F* – f-мера множества принимаемых трассировок автоматом, полученным Алгоритмом-20, Алгоритмом-30 и RTI соответственно.

Из результатов можно установить, что Алгоритм-20 не укладывается во время, отведенное на генерацию достаточно редко (не завершился в 7% случаев) в отличии от Алгоритма-30 (не завершился в 40% случаев). Зато у последнего гораздо лучшие показатели f-меры и относительного количества корректно принимаемых трассировок. Алгоритм RTI обходит первые два алгоритма по скорости выполнения и конкурирует на маленьких данных с Алгоритмом-30 за процент корректной принимаемости трассировок и f-меру, но зато значительно уступает первым двум алгоритмам в числе состояний и ребер. Можно заметить, что у автоматов, сгенерированных RTI, число ребер почти всегда превышает 100, а число состояний практически всегда превышает 10. Алгоритм-30 для случая восьми состояний завершился всего 6 раз из 18, что уже демонстрирует его низкую скорость выполнения при росте числа состояний. При числе состояний равном двум, Алгоритм-30, если завершается, превосходит алгоритм RTI по f-мере в 73% случаев, равном трем – в 69%, равном четырем – в 41%, равном пяти – в 62%, равном шести – в 41%, равном семи – в 30%, равном восьми – в 100% случаев. Если же принимать незавершение как негативный результат, то статистика будет следующей: при числе состояний равном двум – в 61% случаев, равным трем – в 50%, равным четырем – 27%, равном пяти – в 27%, равном шести – в 27%, равном семи – в 20%, равном восьми – в 33% случаев. Видно, на чем больше число состояний, тем хуже и медленнее

начинает работать Алгоритм-30, что неудивительно. Но тем не менее, RTI не всегда превосходит предложенный алгоритм по f-мере. Причем большим преимуществом разработанного метода является нетребовательность к большому числу трассировок: для маленьких целевых автоматов достаточно 30 трассировок, чтобы можно было показать результаты, сравнимые с RTI.

Таблица 1 – Результаты эмпирических запусков предложенного метода и RTI

N	**A**	**S**	**E**	∞	**T20**	**T30**	**T***	**N20**	**N30**	**N***	**E20**	**E30**	**E***	**P20**	**P30**	**P***	**F20**	**F30**	**F***
2	2	2	6	10	31	247	34	2	2	24	5	6	211	91,6	93,8	91,7	0,92	0,94	0,92
2	2	2	6	20	11	14	9	2	2	2	5	5	5	100,0	100,0	100,0	1,00	1,00	1,00
2	2	4	8	10	10	11	48	2	2	40	5	5	277	90,9	100,0	84,4	0,91	1,00	0,84
2	2	4	8	20	9	13	5	2	2	2	4	4	4	100,0	100,0	100,0	1,00	1,00	1,00
2	2	8	12	10	27	9	4	2	2	2	8	6	7	74,6	80,9	100,0	0,79	0,82	1,00
2	2	8	12	20	14	14	140	2	2	41	6	6	396	85,8	88,9	78,0	0,86	0,89	0,78
2	3	2	8	10	10	15	6	2	2	2	6	6	7	95,7	95,7	100,0	0,96	0,96	1,00
2	3	2	8	20	36	30	37	2	2	17	6	6	164	100,0	100,0	94,5	1,00	1,00	0,95
2	3	4	10	10	23	489	27	2	2	18	7	10	120	61,7	100,0	96,9	0,62	1,00	0,97
2	3	4	10	20	183	127	69	2	2	15	7	7	284	94,8	94,8	87,2	0,95	0,95	0,90
2	3	8	14	10	29	599	27	2	2	25	7	9	187	77,2	100,0	89,0	0,79	1,00	0,89
2	3	8	14	20	18	578	86	2	2	22	8	9	347	80,8	72,0	84,6	0,83	0,72	0,85
2	4	2	10	10	95	803	14	2	2	2	10	9	9	97,0	97,0	100,0	0,97	0,97	1,00
2	4	2	10	20	828	841	218	2	2	40	9	9	738	95,5	85,9	67,8	0,96	0,86	0,68
2	4	4	12	10	377	–	46	2	–	33	9	–	335	59,3	–	84,0	0,62	–	0,84
2	4	4	12	20	–	–	185	–	–	42	–	–	663	–	–	70,6	–	–	0,71
2	4	8	16	10	0	1	34	1	1	12	0	2	189	90,2	91,1	84,9	0,91	0,91	0,88
2	4	8	16	20	861	–	177	3	–	31	12	–	695	43,0	–	69,4	0,42	–	0,69
3	2	2	8	10	19	46	89	4	3	59	8	8	439	61,7	100,0	67,2	0,62	1,00	0,67
3	2	2	8	20	1	0	10	1	1	2	1	0	12	89,3	100,0	98,1	0,84	1,00	0,98
3	2	4	10	10	0	0	4	1	1	2	0	0	5	97,9	97,9	100,0	0,98	0,98	1,00
3	2	4	10	20	47	51	39	4	4	11	8	8	142	84,2	80,1	89,0	0,87	0,81	0,90
3	2	8	14	10	23	799	101	4	2	45	8	9	511	61,8	75,3	70,7	0,63	0,75	0,71
3	2	8	14	20	36	447	137	4	2	25	8	7	480	63,6	80,2	71,0	0,67	0,81	0,72
3	3	2	11	10	38	557	21	2	2	21	8	10	160	77,3	80,9	93,5	0,78	0,83	0,94
3	3	2	11	20	20	68	162	2	3	37	7	9	529	71,9	86,8	72,5	0,75	0,87	0,72
3	3	4	13	10	7	192	69	2	4	40	7	12	393	68,9	90,7	75,0	0,72	0,91	0,75
3	3	4	13	20	4	–	117	2	–	28	6	–	498	70,0	–	76,6	0,69	–	0,76
3	3	8	17	10	45	54	50	2	2	35	7	7	315	91,5	91,5	82,3	0,91	0,91	0,85
3	3	8	17	20	38	898	133	3	4	34	9	12	558	69,6	57,3	73,9	0,70	0,60	0,73
3	4	2	14	10	839	–	33	3	–	26	12	–	292	62,1	–	88,4	0,65	–	0,88

Продолжение таблицы 1

N	A	S	E	∞	T20	T30	T*	N20	N30	N*	E20	E30	E*	P20	P30	P*	F20	F30	F*
3	4	2	14	20	98	842	170	2	3	31	9	12	653	78,1	84,5	65,5	0,78	0,86	0,67
3	4	4	16	10	36	24	72	2	2	29	8	8	340	93,0	93,0	79,6	0,93	0,93	0,82
3	4	4	16	20	–	–	191	–	–	36	–	–	653	–	–	71,0	–	–	0,71
3	4	8	20	10	774	–	110	3	–	46	12	–	533	69,9	–	72,7	0,68	–	0,72
3	4	8	20	20	–	–	72	–	–	18	–	–	355	–	–	84,2	–	–	0,84
4	2	2	10	10	8	10	20	3	3	22	6	6	196	76,6	84,0	90,8	0,80	0,84	0,91
4	2	2	10	20	6	41	111	3	4	26	6	8	411	74,0	85,6	77,8	0,79	0,87	0,80
4	2	4	12	10	11	–	85	2	–	56	5	–	448	76,7	–	69,1	0,79	–	0,69
4	2	4	12	20	22	472	201	2	3	36	6	7	523	78,7	74,7	73,5	0,79	0,76	0,73
4	2	8	16	10	9	84	102	2	3	36	6	8	444	69,7	73,7	74,4	0,68	0,70	0,73
4	2	8	16	20	436	–	95	3	–	51	8	–	403	58,2	–	74,4	0,58	–	0,74
4	3	2	14	10	107	638	47	2	2	27	8	8	282	82,5	90,5	83,9	0,82	0,91	0,84
4	3	2	14	20	5	4	107	1	1	21	3	3	303	91,1	91,1	90,7	0,87	0,87	0,89
4	3	4	16	10	597	178	84	2	4	41	9	12	471	60,9	62,9	70,8	0,64	0,63	0,71
4	3	4	16	20	4	10	83	2	2	26	6	8	413	63,2	81,8	84,4	0,66	0,79	0,83
4	3	8	20	10	13	71	32	2	3	27	6	9	225	96,0	85,0	92,2	0,96	0,86	0,92
4	3	8	20	20	4	5	38	1	1	15	3	3	140	97,0	97,0	97,0	0,96	0,96	0,96
4	4	2	18	10	9	49	97	2	2	36	7	9	536	84,4	71,1	66,8	0,85	0,71	0,71
4	4	2	18	20	28	–	129	2	–	33	8	–	526	86,8	–	74,9	0,88	–	0,76
4	4	4	20	10	10	100	65	2	2	38	8	10	390	82,2	74,1	86,3	0,81	0,76	0,85
4	4	4	20	20	336	–	208	2	–	34	9	–	619	87,7	–	71,5	0,88	–	0,75
4	4	8	24	10	675	–	64	3	–	31	12	–	475	58,6	–	73,4	0,60	–	0,73
4	4	8	24	20	19	–	175	2	–	31	7	–	654	72,3	–	75,6	0,71	–	0,77
5	2	2	12	10	6	6	8	2	2	15	5	4	111	95,8	93,8	93,2	0,96	0,94	0,93
5	2	2	12	20	26	20	125	4	4	31	8	8	508	98,5	98,5	68,6	0,99	0,99	0,68
5	2	4	14	10	8	209	108	3	3	42	6	7	465	60,4	82,5	71,0	0,67	0,83	0,71
5	2	4	14	20	14	637	165	2	3	29	6	9	527	72,9	75,5	67,0	0,73	0,75	0,68
5	2	8	18	10	7	35	34	3	4	28	6	8	206	80,1	83,3	85,1	0,81	0,83	0,86
5	2	8	18	20	7	5	196	3	3	28	6	6	559	84,5	78,7	72,5	0,83	0,78	0,76
5	3	2	17	10	139	–	40	3	–	30	9	–	313	71,8	–	80,9	0,73	–	0,81
5	3	2	17	20	14	453	154	2	2	26	7	9	524	73,3	73,6	76,0	0,70	0,72	0,74
5	3	4	19	10	317	–	109	2	–	40	7	–	475	83,4	–	70,2	0,83	–	0,69
5	3	4	19	20	459	–	261	3	–	35	9	–	648	60,2	–	71,2	0,63	–	0,71
5	3	8	23	10	342	–	116	2	–	41	10	–	543	62,3	–	63,3	0,63	–	0,62
5	3	8	23	20	601	–	227	2	–	40	10	–	765	59,3	–	61,9	0,59	–	0,62
5	4	2	22	10	16	15	73	2	2	32	8	8	408	76,1	79,2	78,3	0,75	0,77	0,79
5	4	2	22	20	439	–	260	3	–	47	12	–	870	51,2	–	61,6	0,51	–	0,61
5	4	4	24	10	793	–	78	2	–	41	10	–	471	41,4	–	77,2	0,42	–	0,78

Продолжение таблицы 1

N	A	S	E	∞	T20	T30	T*	N20	N30	N*	E20	E30	E*	P20	P30	P*	F20	F30	F*
5	4	4	24	20	152	–	252	2	–	37	9	–	807	66,6	–	62,8	0,68	–	0,64
5	4	8	28	10	–	–	121	–	–	45	–	–	563	–	–	68,0	–	–	0,71
5	4	8	28	20	7	8	145	1	1	29	4	4	538	86,6	86,6	82,8	0,80	0,80	0,81
6	2	2	14	10	17	82	132	4	3	42	8	8	527	74,9	71,4	61,1	0,75	0,71	0,63
6	2	2	14	20	34	27	63	4	2	21	8	8	338	76,6	65,9	79,3	0,77	0,66	0,79
6	2	4	16	10	18	223	110	4	5	35	8	10	457	65,2	70,2	72,7	0,65	0,70	0,73
6	2	4	16	20	336	–	205	2	–	35	8	–	643	59,9	–	70,4	0,60	–	0,70
6	2	8	20	10	–	450	114	–	4	40	–	12	438	–	72,9	74,6	–	0,73	0,75
6	2	8	20	20	37	330	362	2	3	36	6	9	708	56,2	60,7	65,0	0,59	0,60	0,64
6	3	2	20	10	140	175	186	2	3	46	8	9	545	65,3	71,8	64,0	0,66	0,72	0,64
6	3	2	20	20	13	–	148	2	–	27	7	–	489	77,8	–	71,5	0,78	–	0,73
6	3	4	22	10	38	55	105	2	2	32	7	7	374	76,4	84,7	82,1	0,78	0,84	0,82
6	3	4	22	20	227	–	181	2	–	24	8	–	468	78,3	–	79,3	0,79	–	0,76
6	3	8	26	10	13	18	109	4	2	49	8	8	468	63,5	66,3	76,5	0,62	0,67	0,76
6	3	8	26	20	144	190	289	2	2	33	8	8	631	64,5	71,0	81,5	0,69	0,73	0,80
6	4	2	26	10	–	–	122	–	–	46	–	–	545	–	–	62,9	–	–	0,63
6	4	2	26	20	600	–	264	2	–	48	10	–	739	53,2	–	69,5	0,59	–	0,72
6	4	4	28	10	–	–	142	–	–	51	–	–	605	–	–	65,2	–	–	0,67
6	4	4	28	20	753	867	320	2	2	43	10	10	771	65,8	67,8	63,9	0,66	0,68	0,66
6	4	8	32	10	785	637	169	2	3	38	10	12	564	61,4	67,9	67,1	0,62	0,70	0,67
6	4	8	32	20	829	413	370	2	3	46	10	12	792	60,8	59,2	67,4	0,62	0,61	0,68
7	2	2	16	10	10	123	13	4	5	5	8	10	10	73,6	93,5	100,0	0,74	0,94	1,00
7	2	2	16	20	8	–	295	3	–	31	6	–	652	67,2	–	63,1	0,69	–	0,63
7	2	4	18	10	22	434	148	4	3	35	8	9	435	64,1	68,8	77,6	0,68	0,69	0,76
7	2	4	18	20	41	707	204	4	3	29	8	9	552	70,6	64,5	71,7	0,72	0,64	0,72
7	2	8	22	10	25	–	114	4	–	39	8	–	427	58,1	–	76,6	0,57	–	0,77
7	2	8	22	20	13	67	217	3	4	33	6	8	608	68,0	75,0	68,1	0,68	0,75	0,68
7	3	2	23	10	21	582	94	3	4	36	9	12	444	72,6	67,0	75,1	0,73	0,67	0,74
7	3	2	23	20	4	5	70	1	1	22	3	3	271	89,3	89,3	90,0	0,84	0,84	0,89
7	3	4	25	10	9	8	109	2	2	38	6	7	489	66,8	71,7	70,6	0,66	0,69	0,70
7	3	4	25	20	94	–	211	2	–	34	8	–	786	56,0	–	55,6	0,56	–	0,56
7	3	8	29	10	76	–	107	3	–	33	9	–	532	69,9	–	72,6	0,69	–	0,73
7	3	8	29	20	529	864	238	2	4	39	10	12	690	64,4	57,9	64,2	0,65	0,58	0,64
7	4	2	30	10	5	6	83	1	1	38	4	4	418	90,2	90,2	83,7	0,86	0,86	0,85
7	4	2	30	20	386	–	275	2	–	37	10	–	779	64,5	–	62,7	0,65	–	0,63
7	4	4	32	10	170	–	100	2	–	40	9	–	584	69,0	–	63,5	0,69	–	0,66
7	4	4	32	20	193	840	314	2	3	43	9	12	804	56,5	62,6	61,7	0,56	0,62	0,62
7	4	8	36	10	–	901	129	–	3	36	–	12	497	–	69,8	69,2	–	0,70	0,71

Продолжение таблицы 1

N	A	S	E	∞	T20	T30	T*	N20	N30	N*	E20	E30	E*	P20	P30	P*	F20	F30	F*
8	2	2	18	10	20	–	86	2	–	38	6	–	454	50,7	–	68,3	0,52	–	0,69
8	2	2	18	20	39	63	86	4	4	23	8	8	358	91,8	91,8	77,5	0,93	0,93	0,78
8	2	4	20	10	11	153	108	4	3	35	8	8	420	77,0	75,8	71,3	0,77	0,76	0,71
8	2	4	20	20	12	391	181	3	3	28	6	9	541	71,2	72,0	67,2	0,71	0,74	0,69
8	2	8	24	10	11	423	130	2	3	32	6	9	443	61,1	74,8	72,6	0,65	0,74	0,73
8	2	8	24	20	118	–	356	2	–	35	7	–	797	58,9	–	60,6	0,59	–	0,61
8	3	2	26	10	140	–	109	2	–	38	8	–	487	50,3	–	68,8	0,52	–	0,70
8	3	2	26	20	720	–	201	2	–	31	8	–	805	54,1	–	64,3	0,54	–	0,64
8	3	4	28	10	–	–	104	–	–	50	–	–	584	–	–	63,4	–	–	0,63
8	3	4	28	20	719	–	184	3	–	28	9	–	707	63,6	–	69,2	0,66	–	0,69
8	3	8	32	10	49	–	72	2	–	30	8	–	395	63,8	–	78,7	0,65	–	0,79
8	3	8	32	20	43	–	264	2	–	32	10	–	854	61,3	–	61,8	0,62	–	0,62
8	4	2	34	10	708	–	82	3	–	43	12	–	608	56,1	–	70,3	0,61	–	0,69
8	4	2	34	20	897	–	250	2	–	34	10	–	828	59,8	–	66,3	0,62	–	0,65
8	4	4	36	10	445	–	112	2	–	54	9	–	688	60,8	–	66,5	0,63	–	0,64
8	4	4	36	20	7	7	132	1	1	28	4	4	429	89,8	89,8	85,1	0,85	0,85	0,84
8	4	8	40	10	583	–	103	2	–	45	9	–	627	61,4	–	59,1	0,61	–	0,62
8	4	8	40	20	235	125	214	2	3	33	9	12	847	52,7	68,2	58,8	0,53	0,68	0,60

Также были проведены более детальные эксперименты на совсем маленьких целевых автоматах. В таблице **??** приведены результаты запусков алгоритма RTI и Алгоритмов-20 и 30 на 27 самых маленьких автоматах. Из таблицы 1 были взяты 27 самых маленьких параметров, в порядке увеличения сначала числа ребер, а затем числа состояний. Для каждого из параметров было сгенерировано 20 автоматов и Алгоритмы-20 и 30 были запущены на данном тестовом множестве для 20 случайно выбранных трассировок из 1000 изначально сгенерированных 20 раз. f-мера, процент принимаемых трассировок были усреднены по запускам и по построенным автоматам. Алгоритм RTI, использующий эвристику consistent EDSM, запускался на каждом из сгенерированных автоматах по одному разу, так как алгоритм RTI полностью детерминирован. Также 20 раз был запущен данный алгоритм RTI на 20 трассировках из 1000 изначально сгенерированных, чтобы понять, насколько сильно влияет число трассировок на алгоритм RTI, насколько сильно он может улучшить таким образом результаты.

Таким образом, из результатов, представленных в таблицах 2 и 3, видно, что RTI и Алгоритм-30 генерирует похожие по f-мере и проценту корректно

принятых трассировок автоматы, с различием в количестве ребер сгенерированных автоматов и во времени исполнения. Также интересен факт, что из 20 запусков существуют достаточно хорошие, которые показывают f-меру и процент принимаемых состояний, которые на голову превосходят средние значения. Так как в 20 запусках задействована лишь небольшая часть тренировочного множества, можно провести хорошую кросс-валидацию и в качестве результата выбрать тот самый наилучший автомат.

Таблица 2 – Структурные характеристики автоматов, получившихся при эмпирическом запуске предложенного метода и RTI на большом множестве примеров

E	**N**	**A**	**S**	∞	**No20**	**No30**	**T20**	**T30**	**T***	**T30***	**N20**	**N30**	**N***	**N30***	**E20**	**E30**	**E***	**E30***
6	2	2	2	10	0,0	0,0	6,3	4,6	11,8	0,0	2,2	2,1	7,6	5,9	4,89	4,99	54,16	27,14
6	2	2	2	20	0,0	0,0	6,0	5,0	45,5	0,1	2,1	2,0	10,6	5,0	4,60	4,72	121,00	27,76
8	2	3	2	10	0,0	0,0	15,6	13,9	15,9	0,0	2,0	2,0	7,9	6,2	6,21	6,38	64,63	35,43
8	2	3	2	20	0,3	1,3	43,6	69,6	78,4	0,1	2,0	2,0	16,7	5,3	6,19	6,40	245,95	33,67
8	2	2	4	10	0,0	0,0	10,3	12,9	19,3	0,0	2,2	2,1	11,9	5,9	5,40	5,45	96,74	27,42
8	2	2	4	20	0,0	0,0	16,2	23,8	48,0	0,1	2,2	2,1	17,1	5,4	5,42	5,64	143,47	29,07
8	3	2	2	10	0,0	1,6	29,9	64,6	28,7	0,0	2,7	2,8	18,0	5,8	5,89	6,31	143,21	27,60
8	3	2	2	20	0,0	0,0	19,7	48,4	68,0	0,1	2,6	2,7	19,7	5,3	5,48	5,86	252,05	29,73
10	2	4	2	10	1,6	3,7	125,8	139,6	25,3	0,0	2,0	2,0	15,3	6,1	8,36	8,68	154,32	39,74
10	2	4	2	20	0,5	3,4	138,6	171,6	110,8	0,1	2,0	2,0	20,7	5,4	8,23	8,52	314,63	39,60
10	2	3	4	10	0,8	1,8	74,6	95,1	23,7	0,0	2,0	2,0	13,9	5,9	6,41	6,76	127,89	32,43
10	2	3	4	20	0,0	0,5	35,6	93,6	64,7	0,1	2,0	2,0	17,7	5,1	6,57	7,05	216,05	30,89
10	3	2	4	10	0,0	1,3	25,3	88,1	38,7	0,0	2,6	2,7	23,8	5,3	5,46	6,09	198,68	24,05
10	3	2	4	20	0,0	2,4	34,2	78,9	153,2	0,1	2,7	2,7	24,2	5,5	6,08	6,72	370,00	30,13
10	4	2	2	10	0,0	2,1	14,2	84,1	47,7	0,0	2,9	3,2	26,4	5,5	6,07	6,97	220,89	25,07
10	4	2	2	20	0,3	7,9	35,1	164,0	151,5	0,1	2,8	2,9	24,9	5,4	6,13	7,06	434,00	30,26
11	3	3	2	10	5,0	11,8	101,2	152,5	45,0	0,0	2,4	2,9	24,4	5,9	7,75	8,99	242,47	32,69
11	3	3	2	20	2,6	13,7	69,4	112,5	144,5	0,1	2,0	2,3	25,5	5,2	6,61	7,51	400,16	32,31
12	2	4	4	10	2,9	10,0	207,9	316,5	38,7	0,0	2,0	2,0	21,8	5,8	8,76	9,37	224,37	37,15
12	2	4	4	20	1,3	4,7	190,8	243,5	118,3	0,1	2,0	2,0	24,7	5,3	8,33	8,93	363,79	37,48
12	2	2	8	10	0,0	1,6	37,4	100,4	37,3	0,0	2,4	2,2	22,2	6,1	6,56	7,21	180,95	28,79
12	2	2	8	20	0,0	3,7	38,2	149,6	114,8	0,1	2,4	2,2	23,9	5,5	6,19	6,80	304,74	30,68
12	4	2	4	10	0,8	7,6	50,3	169,5	57,0	0,0	2,8	3,2	30,1	5,2	6,30	7,43	258,84	22,92
12	4	2	4	20	0,3	6,3	45,4	157,1	169,1	0,1	2,8	2,9	27,5	5,3	6,52	7,44	435,05	27,86
12	5	2	2	10	0,0	9,7	54,3	NaN	73,6	0,0	3,2	3,5	31,6	5,7	7,24	8,31	315,05	26,71
12	5	2	2	20	0,0	7,4	36,7	136,5	152,3	0,2	2,9	3,1	25,0	5,1	6,14	7,12	390,84	27,35
13	3	3	4	10	2,1	15,0	95,2	NaN	57,9	0,1	2,3	2,7	28,1	5,8	7,71	8,98	285,26	31,15

Продолжение таблицы 2

E	N	A	S	∞	No20	No30	T20	T30	T*	T30*	N20	N30	N*	N30*	E20	E30	E*	E30*
13	3	3	4	20	2,9	27,6	111,8	307,7	202,8	0,1	2,2	2,6	30,3	5,3	7,85	9,09	525,74	32,42

Таблица 3 – Качественные характеристики автоматов, получившихся при эмпирическом запуске предложенного метода и RTI на большом множестве примеров

E	N	A	S	∞	P20	P30	P*	P30*	F20	F30	F*	F30*	M20	M30	M30*
6	2	2	2	10	91,2	96,2	97,4	64,8	0,92	0,96	0,98	0,67	1,00	1,00	0,78
6	2	2	2	20	90,7	95,2	95,2	63,1	0,91	0,95	0,95	0,65	0,99	1,00	0,75
8	2	3	2	10	93,7	96,8	97,0	58,9	0,94	0,97	0,97	0,62	1,00	1,00	0,72
8	2	3	2	20	89,1	92,9	88,7	61,6	0,89	0,93	0,89	0,63	0,98	1,00	0,71
8	2	2	4	10	88,9	95,2	95,2	63,2	0,90	0,95	0,95	0,65	0,99	1,00	0,76
8	2	2	4	20	84,4	91,3	94,3	62,9	0,85	0,92	0,94	0,64	0,98	0,99	0,74
8	3	2	2	10	83,2	91,1	91,4	63,0	0,84	0,91	0,91	0,65	0,97	0,99	0,73
8	3	2	2	20	85,3	90,6	87,4	61,1	0,86	0,91	0,88	0,63	0,95	0,99	0,70
10	2	4	2	10	88,4	93,6	92,8	57,9	0,89	0,94	0,93	0,61	0,98	0,99	0,71
10	2	4	2	20	88,0	94,3	86,1	58,9	0,89	0,95	0,87	0,61	0,99	0,99	0,68
10	2	3	4	10	87,1	93,6	94,0	62,9	0,87	0,94	0,94	0,65	0,99	0,99	0,74
10	2	3	4	20	85,1	91,4	90,6	65,7	0,86	0,92	0,91	0,67	0,96	0,99	0,78
10	3	2	4	10	81,0	87,4	88,7	67,7	0,82	0,88	0,89	0,69	0,94	0,97	0,77
10	3	2	4	20	78,4	85,5	81,2	62,6	0,79	0,86	0,82	0,64	0,91	0,96	0,70
10	4	2	2	10	80,9	86,3	87,0	68,9	0,82	0,87	0,87	0,69	0,93	0,96	0,76
10	4	2	2	20	76,7	80,2	77,7	61,1	0,78	0,81	0,79	0,64	0,90	0,91	0,72
11	3	3	2	10	78,8	85,2	86,3	63,7	0,80	0,86	0,87	0,65	0,94	0,95	0,72
11	3	3	2	20	79,5	84,1	81,8	64,4	0,81	0,85	0,82	0,66	0,91	0,94	0,74
12	2	4	4	10	80,9	89,0	90,3	62,2	0,82	0,89	0,91	0,64	0,95	0,97	0,74
12	2	4	4	20	81,0	87,8	83,7	61,8	0,82	0,88	0,84	0,63	0,95	0,97	0,72
12	2	2	8	10	77,8	88,7	90,2	61,3	0,78	0,89	0,90	0,63	0,95	0,99	0,73
12	2	2	8	20	75,8	84,8	86,0	60,7	0,76	0,85	0,86	0,63	0,93	0,96	0,73
12	4	2	4	10	76,8	81,5	85,8	71,4	0,78	0,82	0,86	0,72	0,88	0,91	0,78
12	4	2	4	20	73,8	79,5	77,6	65,8	0,75	0,80	0,78	0,66	0,88	0,91	0,72
12	5	2	2	10	73,3	78,4	80,4	65,3	0,75	0,79	0,81	0,66	0,88	0,91	0,71
12	5	2	2	20	76,8	80,5	79,6	65,2	0,78	0,81	0,80	0,66	0,89	0,91	0,73
13	3	3	4	10	74,5	80,3	84,3	65,7	0,75	0,81	0,84	0,66	0,90	0,93	0,74
13	3	3	4	20	71,2	76,1	76,0	65,0	0,73	0,77	0,76	0,65	0,86	0,88	0,71

Пример 30. Приведем пример генерации автоматов Алгоритмом-20 и RTI для девятого теста из тестового множества, изображенного на рисунке 4. Сге-

нерированный автомат алгоритма RTI находится на рисунке 6 , а Алгоритма-20 – на рисунке 5. Видно, что Алгоритм-20 (как и Алгоритм-30) строя автоматы снизу-вверх, в отличие от алгоритма RTI, который строит их сверху-вниз.

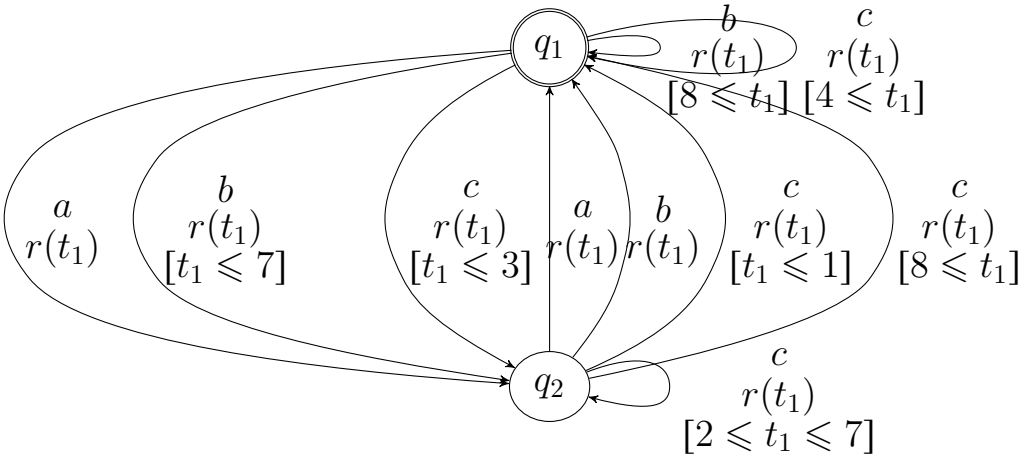

Рисунок 4 – Целевой автомат примера номер девять

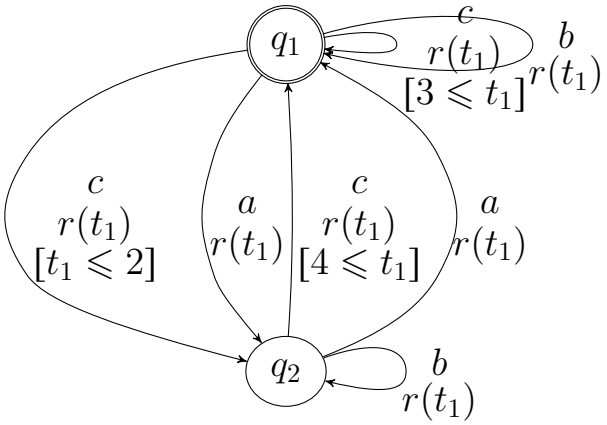

Рисунок 5 – Сгенерированный автомат алгоритмом-20 на примере номер девять

Выводы по главе 3

Проведено экспериментальное исследование предложенного метода. Выявлено, что разработанный алгоритм может использоваться на данных с небольшими целевыми временными автоматами. К сожалению, при числе состояний больше шести разработанный метод, использующий 30 трассировок, завершается очень редко и часто его результаты являются недостаточно хорошими.

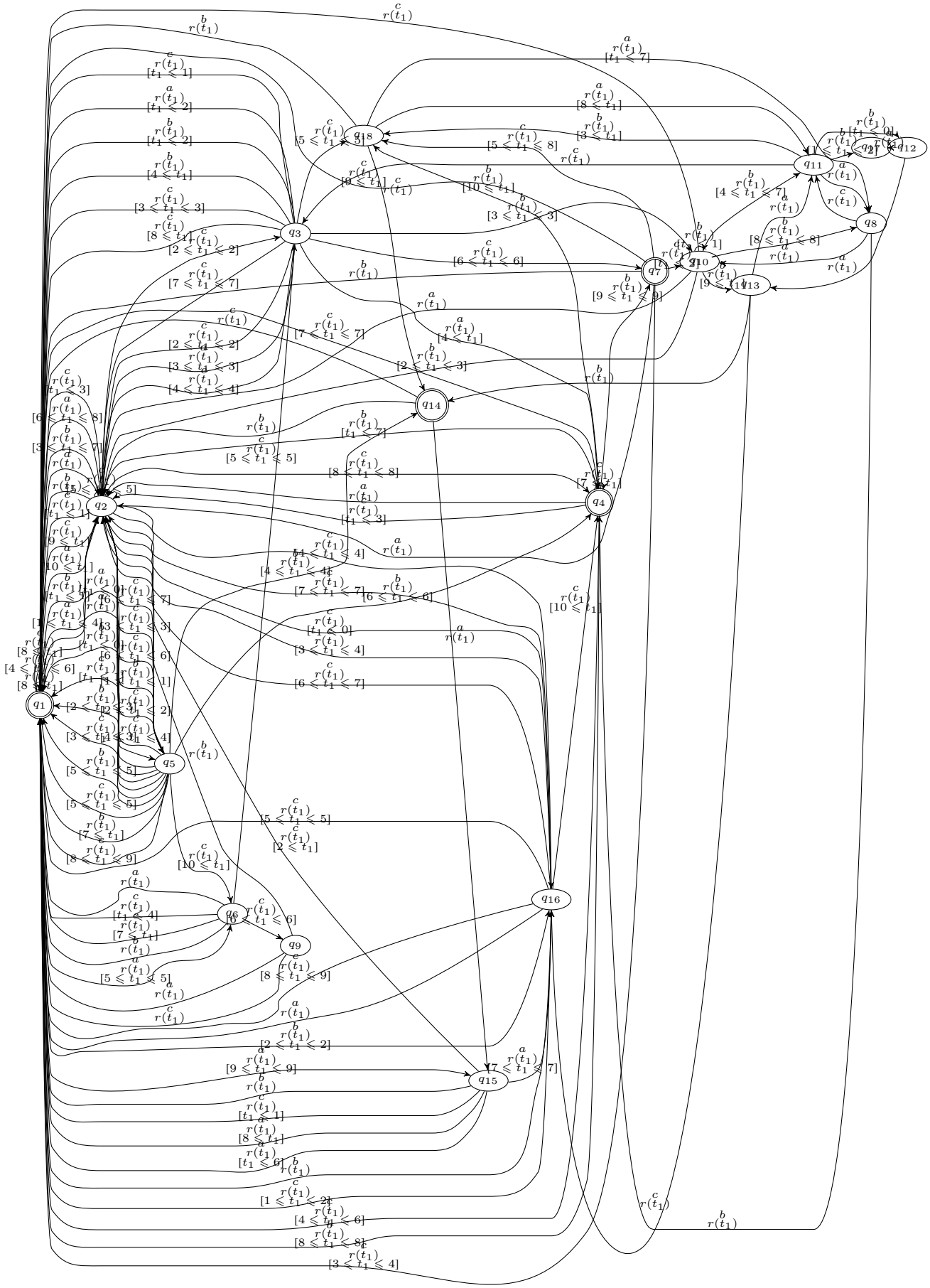

Рисунок 6 – Сгенерированный автомат алгоритмом RTI на примере номер девять

ЗАКЛЮЧЕНИЕ

В результате был разработан метод генерации временных автоматов по трассировкам на основе сведения к задаче удовлетворения ограничений. Помимо этого было произведено экспериментальное исследование, в ходе которого сравнивался разработанный алгоритм, генерирующий автоматы по 20 и 30 трассировкам, с одним из лучших алгоритмов генерации временных автоматов пределе – RTI, использующим эвристику consistent EDSM.

Было выявлено, что разработанный алгоритм может применяться для синтеза временных автоматов, в качестве целевых автоматов которых выступал бы автомат с небольшим числом переходов и состояний. Также данный алгоритм можно применять в ситуации, когда множество размеченных трассировок является достаточно маленьким.

В дальнейшем работа над темой может быть продолжена, остался незакрытым вопрос сведения данной задачи к задаче выполнимости булевой формулы (SAT), использовать можно произвести более грамотные замеры и применить лучшие эвристики для генерации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Behavioral resource-aware model inference // Proceedings of the International Conference on Automated Software Engineering (ASE). — 2014. — С. 19–30.

2 Counterexample-guided abstraction refinement for symbolic model checking // Journal of the ACM. — 2003. — С. 752–794.

3 Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications. — 2010. — С. 66–79.

4 *Lambeau B.*, *Damas C.*, *Dupont P.* State-Merging DFA Induction Algorithms with Mandatory Merge Constraints // Lecture Notes in Computer Science. — 2008. — С. 139–153.

5 *NGSIM*. U.S. Department of Transportation, NGSIM - Next generation simulation. — 2007. — URL: `http://www.ngsim.fhwa.dot.gov`.

6 AbsCon [Электронный ресурс]. — URL: `http://www.cril.univ-artois.fr/en/software/abscon.en.html`.

7 *Biermann A.*, *Feldman J.* On the Synthesis of Finite-State Machines from Samples of Their Behavior // IEEE Transactions on Computers. — 1972. — P. 592–597.

8 Chaff: engineering an efficient SAT solver // Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232). — 2001.

9 Conflict-driven clause learning [Электронный ресурс]. — URL: `https://en.wikipedia.org/wiki/Conflict-driven_clause_learning`.

10 DPLL algorithm [Электронный ресурс]. — URL: `https://en.wikipedia.org/wiki/DPLL_algorithm`.

11 Gecode: FlatZinc [Электронный ресурс]. — URL: `https://www.gecode.org/flatzinc.html`.

12 *Gold M.* Complexity of Automaton Identification from Given Data // Information and Control. — 1978. — P. 302–320.

13 GRASP (SAT solver) [Электронный ресурс]. — URL: `https://en.wikipedia.org/wiki/GRASP_(SAT_solver)`.

14 MOHA: a Multi-mode Hybrid Automaton Model for Learning Car-following Behaviors / Q. Lin [et al.] // IEEE Transactions on Intelligent Transportation Systems. — 2018. — P. 790–796.

15 *Pastore F.*, *Micucci D.*, *Mariani L.* Timed k-Tail: Automatic Inference of Timed Automata // 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). — 2017.

16 *Stuckey P. J.*, *Marriott K.*, *Tack G.* The MiniZinc Handbook [Электронный ресурс]. — 2016.

17 Sugar: a SAT-based Constraint Solver [Электронный ресурс]. — URL: `http://bach.istc.kobe-u.ac.jp/sugar/`.

18 The Minisat Page [Электронный ресурс]. — URL: `http://minisat.se/`.

19 *Verwer S.*, *Weerdt M. de*, *Witteveen C.* A Likelihood-Ratio Test for Identifying Probabilistic Deterministic Real-Time Automata from Positive Data // Grammatical Inference: Theoretical Results and Applications. — 2010. — P. 203–216.

20 *Verwer S.*, *Weerdt M. de*, *Witteveen C.* Efficiently identifying deterministic real-time automata from labeled data // Machine Learning. — 2010. — P. 295–333.

21 *Verwer S.*, *Weerdt M. de*, *Witteveen C.* The efficiency of identifying timed automata and the power of clocks // Information and Computation. — 2011. — P. 606–625.

22 *Zakirzyanov I.*, *Shalyto A.*, *Ulyantsev V.* Finding all minimum-size DFA consistent with given examples: SAT-based approach // Software Engineering and Formal Methods. — 2017. — P. 117–131.