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

ПРИМЕР ОФОРМЛЕНИЯ ВКР БАКАЛАВРА

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: Пример оформления ВКР бакалавра

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 решений задачи
в) Разработка методов сведения к программированию в ограничениях
г) Программиная 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

Опубликованные статьи на тему «Синтез временных автоматов» и статей, демонстрирующих их использование

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: Пример оформления ВКР бакалавра

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: 9

4 Полное число источников, использованных в работе: 11

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
0	0	1	4	3	3

6 Использование информационных ресурсов Internet: да, число ресурсов: 1

7 Использование современных пакетов компьютерных программ и технологий:

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

8 Краткая характеристика полученных результатов

Удалось получить алгоритм, способный синтезировать временные автоматы на малом числе входных данных

9 Гранты, полученные при выполнении работы

Нет

10 Наличие публикаций и выступлений на конференциях по теме работы

КМУ: публикация на конференции

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. 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. Данный метод может гарантировать лишь существование определенного множества примеров поведения, при построении на котором автомат будет являться минимальным. В данной работе предлагается алгоритм, генерирующий по заданным примерам поведения минимальный автомат. Также временные автоматы последнее время используются в машинном обучении [5], поэтому на сегодняшний день разработка новых методов генерации временных автоматов является актуальной задачей.

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

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

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

ГЛАВА 1. ВРЕМЕННЫЕ АВТОМАТЫ, МЕТОДЫ ИХ СИНТЕЗА

1.1. Термины и понятия

В данном разделе будут приведены определения терминов, которые будут использоваться в следующих главах, а также основные результаты о временных автоматах. Определения и формулировки были взяты из статьи [10].

Определение 1. 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 2. Временной автомат $\mathcal{A}$ – кортеж $\langle \Sigma, Q, q_0, C, \Delta, F \rangle$, 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 3. 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 таймер x должен 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 3 раз пройти по петле в то же состояние q_2.

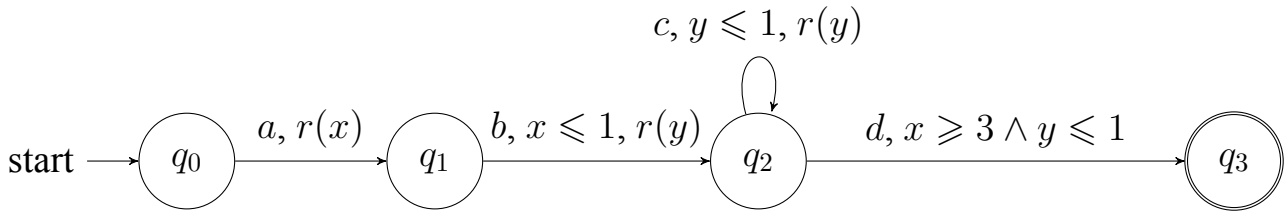

Рисунок 1 – Пример временного автомата

Определение 4. Трассировка τ – последовательность пар $\{(a, t)\}$, где $t \in \mathbb{N}$ – задержка между предыдущим и текущим событиями, $a \in \Sigma$ – символ перехода.

Определение 5. Значение $\nu(x) \in \mathbb{N}$ таймера $x \in C$ – функция от таймера x.

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 $\{(a,10)\,(b,1)\,(c,1)\,(c,1)\,(d,1)\}$ 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.

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 [10].

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 также 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 $\tau \in \mathcal{L}(\mathcal{A}) \Delta \mathcal{L}(\mathcal{A}')$ 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 [10] 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 [10] предоставили алгоритм ID-DTA-1, который эффективно синтезирует временной автомат в пределе.

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

Впервые данный алгоритм был описан в статье [10]. 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 ребра. Алгоритм перебирает символы перехода в лексикографическом порядке и

выбирает первый символ a, для которого во 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. Выбирая очередную границу 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 выбрается 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 ещё 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 минимальном 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, ко-

торые 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 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 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. 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 получаются 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, либо 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 недетерминизма, все потомки, которые необходимо слить, также должны удовлетворять предикату на возможность слияния. Возможна так же 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 её слить с какой-то посещенной вершиной или же наоборот пропустить, чтобы потом слить с какой-нибудь ещё непосещенной вершиной. Таким образом, размер автомата, который получится в конце, определяется стратегией, которую выбирают при синтезе автомата. Так как абсолютно правильную стратегию сливаний и перекрасок выбрать сложно из-за того, что задача синтеза минимального детерминированного конечного автомата по трассировкам NP-полна [3], 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 [9] адаптировали алгоритм EDSM, чтобы он мог использовать 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 ведёт резделяемое ребро. Это можно сделать потому что, всё, 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 ещё 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 операции разделения куда меньший приоритет по сравнению с операцией слияния, так как при слиянии как раз и произойдёт, 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 неё 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, выбрав в качестве времени для разделения временного ограничения равновероятно любую его точку. Эвристика выглядит следующим образом:

$$impact = \sum_{q \in Q_r} pure(q) - \sum_{\tau \in \Delta_b} max\{impact(\tau, \tau') | \tau \in S_+^\delta \wedge \tau' \in S_-^\delta\} + \sum_{\tau \in \Delta_b} max\{impact(\tau, \tau') | \tau \in S_+^\delta \wedge \tau' \in S_+^\delta\} + \sum_{\tau \in \Delta_b} max\{impact(\tau, \tau') | \tau \in S_-^\delta \wedge \tau' \in S_-^\delta\} \quad (4)$$

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 [6] является адаптацией алгоритма k-Tail [2], 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 дан-

ной 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 её 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 ещё 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 добавить во множество сброса таймер t, таким образом позволяя не думать о том, какими таймеры могут быть в корне будущего префиксного дерева, все

они всё 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. Однако качество получаемых временных автоматов тоже не велико.

1.6. Применение временных автоматов на практике. Алгоритм MOHA

Некоторая модификация алгоритма RTI – алгоритм RTI+ [8], умеющий генерировать вероятностные детерминированные автоматы реального времени (определение 12), опираясь только на принимаемые трассировки, используется в алгоритме MOHA [5] как одна из составных частей реализации. Опишем алгоритм MOHA и применение в нем временных автоматов.

Данный алгоритм был применен на данных из двух открытых множеств траекторий NGSIM [1]: I80 и US101. В данных множествах положение автомобилей обновляется с частотой 10Гц. Алгоритм состоит из нескольких стадий: сначала происходит крупнозернистая грануляция примеров, затем символизированные примеры подаются на вход RTI+ в результате которого появляется автомат, состояния данного автомата кластеризуются алгоритмом MISSI [5], образуя режимы в автомате, далее на данных режимах тренируются модели отслеживания машин.

Чтобы можно было применить алгоритм RTI+, сначала данные 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 получния режимов. Извлечем из путей подпути длины не меньше заданной константы L_{min} с встречаемостью чаще заданной константы ϵ и кластреризуем их по по-

хожести. Режим 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 удовлетворимости булевых формул и задача удовлетворения ограничений

Задача удовлетворимости булевых формул и задача удовлетворения ограничений являются известными NP-полными задачами. Одной из первых была доказана NP-полнота задачи удовлетворения булевых формул. 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 вышеупомянутым задачи чаще всего используются для доказательств NP-полноты, то было разработано большое количество пакетов программ, позволяющих их решать за хоть сколько-то разумное время.

В частности к задаче удовлетворения булевых формул можно свести задачу построения минимального детерминированного конечного автомата. 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

— Даны основные определения и факты из предметной области. Произведен краткий анализ фактов.
— Изучены лучшие из существующих решений генерации временных автоматов.
— Произведен обзор существующих решений.

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, а также эвристические методы генерации временных автоматов, однако отсутствует попытка разработки алгоритма, генерирующего минимальный временной автомат на основе сведения к задаче удовлетворения ограничений.

Таким образом в данной работе решались следующие задачи:

— Изучение возможностей программного пакета Minizinc [7], предназначенного для решения задач удовлетворения граничений.
— Разработка сведения задачи к задаче удовлетворения ограничений.
— Реализация решения на основе сведения и сравнение с существующим решением.

2.2. Переменные, используемые в решателе задачи удовлетворения ограничений

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 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 оканчивается 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;
— $next : 1..M \to 1..N$ – состояния, из которого выходят переходы $m \in 1..M$;
— $prev : 1..M \to 1..N$ – состояния, в которые идут переходы $m \in 1..M$;
— $times : 1..M \to 0..inf$ – 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

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 1 не допускается существование двух переходов по одному символу из одного состояния, для которых гиперпаралеллепипеды, со-

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])));
```

ставленные из ограничений каждого таймера, пересекаются. Очевидно, чтобы недопустить пересечение гиперпаралеллепипедов 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.

2.3.2. Основной предикат на переходы

Основной предикат – предикат на то, что переходу в префиксном дереве соответствует данный переход в автомате.

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);
```

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, куда ведет переход префиксного дерева, соответствует состояние, куда ведет переход автомата;

— значения каждого из таймеров сразу до перехода удовлетворяют ограничениям перехода по каждому из таймеров;
— значения таймеров сразу после перехода в префиксном дереве равны значениям таймеров сразу 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 соответствующий ему переход в автомате, а также необходимо, чтобы для каждого перехода в автомате существовал как минимум один соответствующий переход в префиксном дереве. Это описывают следующие ограничения:

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. Стартовые ограничения

Стартовыми ограниченииями будут являться:

— корню префиксного дерева соответствует стартовая вершина в автомате.
— значения всех таймеров сразу после перехода в стартовую вершину равны нулю.

Листинг 4 – Стартовые ограничения

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

constraint
map[1] = 1;
```

2.3.4. Ограничения на минимальность

Основной минимальностью является число состояний данного автомата. Но состояния автомата перебираются вне решателя систем ограничений. Решатель только пытается удовлетворить переданные ему ограничения. Остальные, менее важные минимальности – минимальность на число переходов в данном временном автомата и минимальность количества активных таймеров в автомате. 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 на листингах 5 6

Листинг 5 – Стартовые ограничения-1

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

Листинг 6 – Стартовые ограничения-2

```
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 5 ограничивает число ребер, по которым проходят пути трассировок, которое не должно превосходить TE. Ограничение 6 ограничивает число активных таймеров, которое не должно превосходить TC.

2.3.5. Ограничение на допускающие состояния

Так как в решатель передается префиксного дерево, каждое из состояний которого покрашено в одно из трех цветов, автомат должен учитывать эту информацию при покраске своих вершин. Вершина автомата может быть принимающей трассировки только в случае, если соответствующие ей состояния префиксного дерева имеют цвета либо G, либо B. Отвергающей же имеет право быть только в случае, если соответствующие ей состояния префиксного дерева имеют цвета либо G, либо W. Запишем это в ограничениях:

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 BFS

Ограничения BFS впервые были представлены в работе [11]. 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 вершин и ограничения на порядок переходов. Ограничения на порядок вершин 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. Порядок на ребрах требует, чтобы состояния, в которые ведут меньшие по номеру переходы, имели номер меньше, чем состояния, в которые ведут большие по номеру переходы.

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 по порядке просмотра ребер, которые ещё 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. Следующее ограничение 8 накладывает описанные выше ограниче-

ния: 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-1

```
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)));
```

Теперь необходимо добавить ограничение двух условий, описанных выше.

Листинг 9 – BFS-ограничения-2

```
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]);
```

Единственное, что осталось сделать – определить в каком порядке перебирать переходы. Ведь могут встречаться 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 следующим ограничением:

То есть можно сравнить 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 10 тоже немного обрезает область поиска: все неактивные ребра будут находиться последними в списке ребер, выходящих из данной вершины. Однако, последнее ограничение 10 является очень трудным для решателя систем ограничений, поэтому порой оно может быть опущено.

Листинг 10 – BFS-ограничения-3

```
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])))));
```

2.3.7. Ограничения для автоматов реального времени

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

Листинг 11 – Ограничения RTA

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

2.4. Порядок генерации

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 вы-

ходить несколько переходов, имея непересекающиеся гиперпаралеллепипеды 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 получились 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 всё 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 отличии от количества состояний во временном автомате хорошим показателем его минимальности является количество ребер.

Основной алгоритм можно представить псевдокодом на листингах 12, 13. Алгоритм состоит из трех основных циклов. В цикле неосновном цикле, со строки 3 до строки 7, производится 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. В втором цикле, со строки 17 до строки 36, 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 (пример, 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 два последующих цикла, со строки 38 до строки 54 и со строки 56 до строки 70 соответственно, пытаются минимизировать построенный автомат. Минимизация сначала происходит по числу переходов в данном автомате, а затем по числу активных таймеров. Минимизация числа

переходов нужна в том случае, если 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.

Пример 25. Построим для множества трассировок A префиксное дерево 2.

$$A = \{\{(a,1)\,(c,3)\},\{(b,2)\,,(d,4)\,,(f,6)\},\{(b,2)\,,(c,5)\},\{(b,2)\}\} \qquad (6)$$

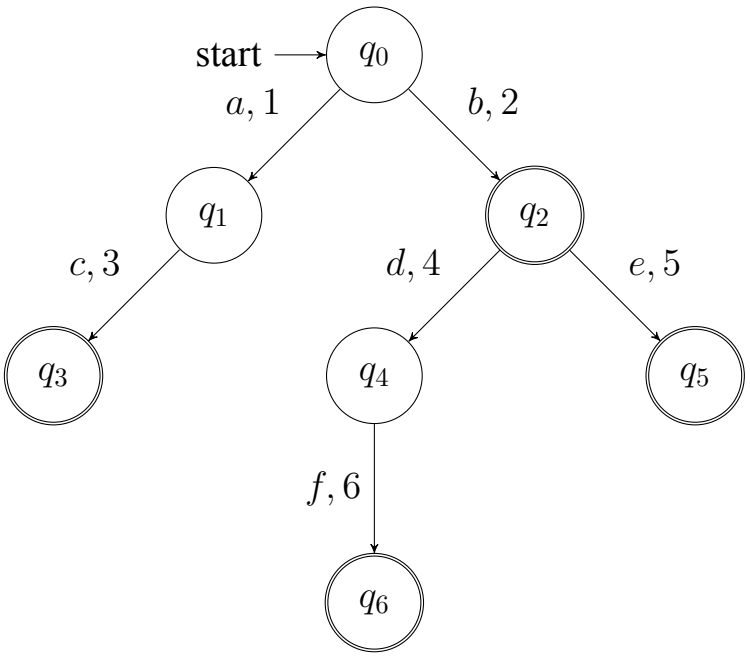

Рисунок 2 – Пример префиксного дерева

Выводы по главе 2

В данной главе была описана и разъеснена идея решения, описано и разъеснено сведение к задаче удовлетворения ограничений (основные сущности и ограничения).

Листинг 12 – Псевдокод алгоритма

```
1: function Основной алгоритм(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:                edges ← число ребер в automaton
28:                activeTimers ← число активных таймеров в automaton
29:                break
30:            else
31:                activeTraces += минимальная неудовлетворимая трасси-
   ровка из traces
32:            end if
33:        else
34:            index ← index + 1
35:        end if
36:    end loop
```

Листинг 13 – Псевдокод алгоритма

```
    edgesNumber ← edges − 1
    loop
        edges ← edgesNumber
        activeTimers ← edges ∗ timersNumber
        prefixTree ← префиксное дерево по трассировкам из activeTraces
        newAutomaton ← решить ограничения с параметрами states, vertexDegree, edges, activeTimers, prefixTree
        if newAutomaton ≠ NULL then
            if newAutomaton конситентен с трассировками из traces then
                automaton ← newAutomaton
                edgesNumber ← edgesNumber − 1
            else
                activeTraces += минимальная неудовлетворимая трассировка из traces
            end if
        else
            activeTimers ← число активных таймеров в automaton
            break
        end if
    end loop
    activeTimersNumber ← activeTimers − 1
    loop
        activeTimers ← activeTimersNumber
        prefixTree ← префиксное дерево по трассировкам из activeTraces
        newAutomaton ← решить ограничения с параметрами states, vertexDegree, edges, activeTimers, prefixTree
        if newAutomaton ≠ NULL then
            if automaton конситентен с трассировками из traces then
                automaton ← newAutomaton
                activeTimersNumber ← activeTimersNumber − 1
            else
                activeTraces += минимальная неудовлетворимая трассировка из traces
            end if
        else
            return automaton
        end if
    end loop
end function
```

ГЛАВА 3. ЭКСПЕРИМЕНТЫ

Тестирования был решено провести на случайных сгенерированных автоматах. Метод генерации был взят из [8] и адаптирован для текущей задачи. Были сгенерировано тестовое множество для всевозможных комбинаций параметров:

— число вершин: $2, 3, 4, 5, 6, 7, 8$;
— число символов переходов: $2, 3, 4$;
— число делений ребер: $2, 4, 8$;
— 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 были сгенерировано по 1000 случайных трассировок. Алгоритм RTI 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 тестирование 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, включался 15-минутный таймер, по истечении которого процесс убивали.

В таблицах 1, 2, 3, 4 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 успевает сгенерировать никакой автомат, удовлетворяющий ограничениям в редких случаях в отличии от Алгоритма-30. Зато у последнего гораздо лучшие показатели f-меры и процента корректной принимаемости 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.

Выводы по главе 3

В главе 3 был описан способ произведения тестирования, произведено тестирование, приведена таблица результатов, дан краткий обзор полученных данных.

Таблица 1 – Результаты

N	A	S	E	∞	T20	N20	E20	P20	F20	T30	N30	E30	P30	F30	T*	N*	E*	P*	F*
2	2	2	6	10	31	2	5	91.6	0.92	247	2	6	93.8	0.94	34	24	211	91.7	0.92
2	2	2	6	20	11	2	5	100.0	1.00	14	2	5	100.0	1.00	9	2	5	100.0	1.00
2	2	4	8	10	10	2	5	90.9	0.91	11	2	5	100.0	1.00	48	40	277	84.4	0.84
2	2	4	8	20	9	2	4	100.0	1.00	13	2	4	100.0	1.00	5	2	4	100.0	1.00
2	2	8	12	10	27	2	8	74.6	0.79	9	2	6	80.9	0.82	4	2	7	100.0	1.00
2	2	8	12	20	14	2	6	85.8	0.86	14	2	6	88.9	0.89	140	41	396	78.0	0.78
2	3	2	8	10	10	2	6	95.7	0.96	15	2	6	95.7	0.96	6	2	7	100.0	1.00
2	3	2	8	20	36	2	6	100.0	1.00	30	2	6	100.0	1.00	37	17	164	94.5	0.95
2	3	4	10	10	23	2	7	61.7	0.62	489	2	10	100.0	1.00	27	18	120	96.9	0.97
2	3	4	10	20	183	2	7	94.8	0.95	127	2	7	94.8	0.95	69	15	284	87.2	0.90
2	3	8	14	10	29	2	7	77.2	0.79	599	2	9	100.0	1.00	27	25	187	89.0	0.89
2	3	8	14	20	18	2	8	80.8	0.83	578	2	9	72.0	0.72	86	22	347	84.6	0.85
2	4	2	10	10	95	2	10	97.0	0.97	803	2	9	97.0	0.97	14	2	9	100.0	1.00
2	4	2	10	20	828	2	9	95.5	0.96	841	2	9	85.9	0.86	218	40	738	67.8	0.68
2	4	4	12	10	377	2	9	59.3	0.62	–	–	–	–	–	46	33	335	84.0	0.84
2	4	4	12	20	–	–	–	–	–	–	–	–	–	–	185	42	663	70.6	0.71
2	4	8	16	10	0	1	0	90.2	0.91	1	1	2	91.1	0.91	34	12	189	84.9	0.88
2	4	8	16	20	861	3	12	43.0	0.42	–	–	–	–	–	177	31	695	69.4	0.69
3	2	2	8	10	19	4	8	61.7	0.62	46	3	8	100.0	1.00	89	59	439	67.2	0.67
3	2	2	8	20	1	1	1	89.3	0.84	0	1	0	100.0	1.00	10	2	12	98.1	0.98
3	2	4	10	10	0	1	0	97.9	0.98	0	1	0	97.9	0.98	4	2	5	100.0	1.00
3	2	4	10	20	47	4	8	84.2	0.87	51	4	8	80.1	0.81	39	11	142	89.0	0.90
3	2	8	14	10	23	4	8	61.8	0.63	799	2	9	75.3	0.75	101	45	511	70.7	0.71
3	2	8	14	20	36	4	8	63.6	0.67	447	2	7	80.2	0.81	137	25	480	71.0	0.72
3	3	2	11	10	38	2	8	77.3	0.78	557	2	10	80.9	0.83	21	21	160	93.5	0.94
3	3	2	11	20	20	2	7	71.9	0.75	68	3	9	86.8	0.87	162	37	529	72.5	0.72
3	3	4	13	10	7	2	7	68.9	0.72	192	4	12	90.7	0.91	69	40	393	75.0	0.75
3	3	4	13	20	4	2	6	70.0	0.69	–	–	–	–	–	117	28	498	76.6	0.76
3	3	8	17	10	45	2	7	91.5	0.91	54	2	7	91.5	0.91	50	35	315	82.3	0.85
3	3	8	17	20	38	3	9	69.6	0.70	898	4	12	57.3	0.60	133	34	558	73.9	0.73
3	4	2	14	10	839	3	12	62.1	0.65	–	–	–	–	–	33	26	292	88.4	0.88
3	4	2	14	20	98	2	9	78.1	0.78	842	3	12	84.5	0.86	170	31	653	65.5	0.67
3	4	4	16	10	36	2	8	93.0	0.93	24	2	8	93.0	0.93	72	29	340	79.6	0.82
3	4	4	16	20	–	–	–	–	–	–	–	–	–	–	191	36	653	71.0	0.71
3	4	8	20	10	774	3	12	69.9	0.68	–	–	–	–	–	110	46	533	72.7	0.72
3	4	8	20	20	–	–	–	–	–	–	–	–	–	–	72	18	355	84.2	0.84

Таблица 2 – Результаты

N	A	S	E	∞	T20	N20	E20	P20	F20	T30	N30	E30	P30	F30	T*	N*	E*	P*	F*
4	2	2	10	10	8	3	6	76.6	0.80	10	3	6	84.0	0.84	20	22	196	90.8	0.91
4	2	2	10	20	6	3	6	74.0	0.79	41	4	8	85.6	0.87	111	26	411	77.8	0.80
4	2	4	12	10	11	2	5	76.7	0.79	–	–	–	–	–	85	56	448	69.1	0.69
4	2	4	12	20	22	2	6	78.7	0.79	472	3	7	74.7	0.76	201	36	523	73.5	0.73
4	2	8	16	10	9	2	6	69.7	0.68	84	3	8	73.7	0.70	102	36	444	74.4	0.73
4	2	8	16	20	436	3	8	58.2	0.58	–	–	–	–	–	95	51	403	74.4	0.74
4	3	2	14	10	107	2	8	82.5	0.82	638	2	8	90.5	0.91	47	27	282	83.9	0.84
4	3	2	14	20	5	1	3	91.1	0.87	4	1	3	91.1	0.87	107	21	303	90.7	0.89
4	3	4	16	10	597	2	9	60.9	0.64	178	4	12	62.9	0.63	84	41	471	70.8	0.71
4	3	4	16	20	4	2	6	63.2	0.66	10	2	8	81.8	0.79	83	26	413	84.4	0.83
4	3	8	20	10	13	2	6	96.0	0.96	71	3	9	85.0	0.86	32	27	225	92.2	0.92
4	3	8	20	20	4	1	3	97.0	0.96	5	1	3	97.0	0.96	38	15	140	97.0	0.96
4	4	2	18	10	9	2	7	84.4	0.85	49	2	9	71.1	0.71	97	36	536	66.8	0.71
4	4	2	18	20	28	2	8	86.8	0.88	–	–	–	–	–	129	33	526	74.9	0.76
4	4	4	20	10	10	2	8	82.2	0.81	100	2	10	74.1	0.76	65	38	390	86.3	0.85
4	4	4	20	20	336	2	9	87.7	0.88	–	–	–	–	–	208	34	619	71.5	0.75
4	4	8	24	10	675	3	12	58.6	0.60	–	–	–	–	–	64	31	475	73.4	0.73
4	4	8	24	20	19	2	7	72.3	0.71	–	–	–	–	–	175	31	654	75.6	0.77
5	2	2	12	10	6	2	5	95.8	0.96	6	2	4	93.8	0.94	8	15	111	93.2	0.93
5	2	2	12	20	26	4	8	98.5	0.99	20	4	8	98.5	0.99	125	31	508	68.6	0.68
5	2	4	14	10	8	3	6	60.4	0.67	209	3	7	82.5	0.83	108	42	465	71.0	0.71
5	2	4	14	20	14	2	6	72.9	0.73	637	3	9	75.5	0.75	165	29	527	67.0	0.68
5	2	8	18	10	7	3	6	80.1	0.81	35	4	8	83.3	0.83	34	28	206	85.1	0.86
5	2	8	18	20	7	3	6	84.5	0.83	5	3	6	78.7	0.78	196	28	559	72.5	0.76
5	3	2	17	10	139	3	9	71.8	0.73	–	–	–	–	–	40	30	313	80.9	0.81
5	3	2	17	20	14	2	7	73.3	0.70	453	2	9	73.6	0.72	154	26	524	76.0	0.74
5	3	4	19	10	317	2	7	83.4	0.83	–	–	–	–	–	109	40	475	70.2	0.69
5	3	4	19	20	459	3	9	60.2	0.63	–	–	–	–	–	261	35	648	71.2	0.71
5	3	8	23	10	342	2	10	62.3	0.63	–	–	–	–	–	116	41	543	63.3	0.62
5	3	8	23	20	601	2	10	59.3	0.59	–	–	–	–	–	227	40	765	61.9	0.62
5	4	2	22	10	16	2	8	76.1	0.75	15	2	8	79.2	0.77	73	32	408	78.3	0.79
5	4	2	22	20	439	3	12	51.2	0.51	–	–	–	–	–	260	47	12	61.6	0.61
5	4	4	24	10	793	2	10	41.4	0.42	–	–	–	–	–	78	41	471	77.2	0.78
5	4	4	24	20	152	2	9	66.6	0.68	–	–	–	–	–	252	37	807	62.8	0.64
5	4	8	28	10	–	–	–	–	–	–	–	–	–	–	121	45	563	68.0	0.71
5	4	8	28	20	7	1	4	86.6	0.80	8	1	4	86.6	0.80	145	29	538	82.8	0.81

Таблица 3 – Результаты

N	A	S	E	∞	T20	N20	E20	P20	F20	T30	N30	E30	P30	F30	T*	N*	E*	P*	F*
6	2	2	14	10	17	4	8	74.9	0.75	82	3	8	71.4	0.71	132	42	527	61.1	0.63
6	2	2	14	20	34	4	8	76.6	0.77	27	2	8	65.9	0.66	63	21	338	79.3	0.79
6	2	4	16	10	18	4	8	65.2	0.65	223	5	10	70.2	0.70	110	35	457	72.7	0.73
6	2	4	16	20	336	2	8	59.9	0.60	–	–	–	–	–	205	35	643	70.4	0.70
6	2	8	20	10	–	–	–	–	–	450	4	12	72.9	0.73	114	40	438	74.6	0.75
6	2	8	20	20	37	2	6	56.2	0.59	330	3	9	60.7	0.60	362	36	708	65.0	0.64
6	3	2	20	10	140	2	8	65.3	0.66	175	3	9	71.8	0.72	186	46	545	64.0	0.64
6	3	2	20	20	13	2	7	77.8	0.78	–	–	–	–	–	148	27	489	71.5	0.73
6	3	4	22	10	38	2	7	76.4	0.78	55	2	7	84.7	0.84	105	32	374	82.1	0.82
6	3	4	22	20	227	2	8	78.3	0.79	–	–	–	–	–	181	24	468	79.3	0.76
6	3	8	26	10	13	4	8	63.5	0.62	18	2	8	66.3	0.67	109	49	468	76.5	0.76
6	3	8	26	20	144	2	8	64.5	0.69	190	2	8	71.0	0.73	289	33	631	81.5	0.80
6	4	2	26	10	–	–	–	–	–	–	–	–	–	–	122	46	545	62.9	0.63
6	4	2	26	20	600	2	10	53.2	0.59	–	–	–	–	–	264	48	739	69.5	0.72
6	4	4	28	10	–	–	–	–	–	–	–	–	–	–	142	51	605	65.2	0.67
6	4	4	28	20	753	2	10	65.8	0.66	867	2	10	67.8	0.68	320	43	771	63.9	0.66
6	4	8	32	10	785	2	10	61.4	0.62	637	3	12	67.9	0.70	169	38	564	67.1	0.67
6	4	8	32	20	829	2	10	60.8	0.62	413	3	12	59.2	0.61	370	46	792	67.4	0.68
7	2	2	16	10	10	4	8	73.6	0.74	123	5	10	93.5	0.94	13	5	10	100.0	1.00
7	2	2	16	20	8	3	6	67.2	0.69	–	–	–	–	–	295	31	652	63.1	0.63
7	2	4	18	10	22	4	8	64.1	0.68	434	3	9	68.8	0.69	148	35	435	77.6	0.76
7	2	4	18	20	41	4	8	70.6	0.72	707	3	9	64.5	0.64	204	29	552	71.7	0.72
7	2	8	22	10	25	4	8	58.1	0.57	–	–	–	–	–	114	39	427	76.6	0.77
7	2	8	22	20	13	3	6	68.0	0.68	67	4	8	75.0	0.75	217	33	608	68.1	0.68
7	3	2	23	10	21	3	9	72.6	0.73	582	4	12	67.0	0.67	94	36	444	75.1	0.74
7	3	2	23	20	4	1	3	89.3	0.84	5	1	3	89.3	0.84	70	22	271	90.0	0.89
7	3	4	25	10	9	2	6	66.8	0.66	8	2	7	71.7	0.69	109	38	489	70.6	0.70
7	3	4	25	20	94	2	8	56.0	0.56	–	–	–	–	–	211	34	786	55.6	0.56
7	3	8	29	10	76	3	9	69.9	0.69	–	–	–	–	–	107	33	532	72.6	0.73
7	3	8	29	20	529	2	10	64.4	0.65	864	4	12	57.9	0.58	238	39	690	64.2	0.64
7	4	2	30	10	5	1	4	90.2	0.86	6	1	4	90.2	0.86	83	38	418	83.7	0.85
7	4	2	30	20	386	2	10	64.5	0.65	–	–	–	–	–	275	37	779	62.7	0.63
7	4	4	32	10	170	2	9	69.0	0.69	–	–	–	–	–	100	40	584	63.5	0.66
7	4	4	32	20	193	2	9	56.5	0.56	840	3	12	62.6	0.62	314	43	804	61.7	0.62
7	4	8	36	10	–	–	–	–	–	901	3	12	69.8	0.70	129	36	497	69.2	0.71

Таблица 4 – Результаты

N	A	S	E	∞	T20	N20	E20	P20	F20	T30	N30	E30	P30	F30	T*	N*	E*	P*	F*
8	2	2	18	10	20	2	6	50.7	0.52	–	–	–	–	–	86	38	454	68.3	0.69
8	2	2	18	20	39	4	8	91.8	0.93	63	4	8	91.8	0.93	86	23	358	77.5	0.78
8	2	4	20	10	11	4	8	77.0	0.77	153	3	8	75.8	0.76	108	35	420	71.3	0.71
8	2	4	20	20	12	3	6	71.2	0.71	391	3	9	72.0	0.74	181	28	541	67.2	0.69
8	2	8	24	10	11	2	6	61.1	0.65	423	3	9	74.8	0.74	130	32	443	72.6	0.73
8	2	8	24	20	118	2	7	58.9	0.59	–	–	–	–	–	356	35	797	60.6	0.61
8	3	2	26	10	140	2	8	50.3	0.52	–	–	–	–	–	109	38	487	68.8	0.70
8	3	2	26	20	720	2	8	54.1	0.54	–	–	–	–	–	201	31	805	64.3	0.64
8	3	4	28	10	–	–	–	–	–	–	–	–	–	–	104	50	584	63.4	0.63
8	3	4	28	20	719	3	9	63.6	0.66	–	–	–	–	–	184	28	707	69.2	0.69
8	3	8	32	10	49	2	8	63.8	0.65	–	–	–	–	–	72	30	395	78.7	0.79
8	3	8	32	20	43	2	10	61.3	0.62	–	–	–	–	–	264	32	854	61.8	0.62
8	4	2	34	10	708	3	12	56.1	0.61	–	–	–	–	–	82	43	608	70.3	0.69
8	4	2	34	20	897	2	10	59.8	0.62	–	–	–	–	–	250	34	828	66.3	0.65
8	4	4	36	10	445	2	9	60.8	0.63	–	–	–	–	–	112	54	688	66.5	0.64
8	4	4	36	20	7	1	4	89.8	0.85	7	1	4	89.8	0.85	132	28	429	85.1	0.84
8	4	8	40	10	583	2	9	61.4	0.61	–	–	–	–	–	103	45	627	59.1	0.62
8	4	8	40	20	235	2	9	52.7	0.53	125	3	12	68.2	0.68	214	33	847	58.8	0.60

ЗАКЛЮЧЕНИЕ

В ходе работы был разработан алгоритм 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 40 трассировкам, с одним из лучших алгоритмов генерации временных автоматов пределе.

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.

Для достижения цели работы был изучен програмный пакет minizinc [7], изучено множество статей на тему генерации временных и детерминированных конечных автоматов [4–6, 8–11].

В дальнейшем работа над темой может быть продолжена, остался незакрытым вопрос сведения данной задачи к задаче удовлетворения булевых переменных, можно произвести более грамотные замеры и применить лучшие эвристики для генерации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

2 *Biermann A.*, *Feldman J.* On the Synthesis of Finite-State Machines from Samples of Their Behavior // IEEE Transactions on Computers. — 1972. — P. 592–597. — URL: https://www.researchgate.net/publication/224483188_On_the_Synthesis_of_Finite-State_Machines_from_Samples_of_Their_Behavior.

3 *Gold M.* Complexity of Automaton Identification from Given Data // Information and Control. — 1978. — P. 302–320. — URL: https://www.researchgate.net/publication/222440911_Complexity_of_Automaton_Identification_from_Given_Data.

4 *Lambeau B.*, *Damas C.*, *Dupont P.* State-Merging DFA Induction Algorightms with Mandatory Merge Constraints // Lecture Notes in Computer Science. — 2008. — P. 139–153. — URL: https://www.info.ucl.ac.be/~pdupont/pdupont/pdf/icgi08.pdf.

5 MOHA: a Multi-mode Hybrid Automaton Model for Learning Car-following Behaviors / Q. Lin [et al.]. — 2018. — URL: https://ieeexplore.ieee.org/document/8384014.

6 *Pastore F.*, *Micucci D.*, *Mariani L.* Timed k-Tail: Automatic Inference of Timed Automata. — 2017. — URL: https://arxiv.org/pdf/1705.08399.pdf.

7 *Stuckey P. J.*, *Marriott K.*, *Tack G.* The MiniZinc Handbook [Электронный ресурс]. — 2016. — URL: https://www.minizinc.org/doc-2.2.3/en/index.html.

8 *Verwer S.* Efficient identification of timed automata: Theory and practice. — 2010. — URL: https://www.narcis.nl/publication/RecordID/oai:tudelft.nl:uuid:61d9f199-7b01-45be-a6ed-04498113a212.

9 *Verwer S.*, *Weerdt M. de*, *Witteveen C.* Efficiently identifying deterministic real-time automatafrom labeled data // Springer. — 2010. — P. 295–333. — URL: https://link.springer.com/article/10.1007/s10994-011-5265-4.

10 *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. — URL: http://wwwis.win.tue.nl/~sverwer.

11 *Zakirzyanov I.*, *Shalyto A.*, *Ulyantsev V.* Finding all minimum-size DFA consistent with given examples: SAT-based approach. — 2017. — URL: http://pages.di.unipi.it/datamod/wp-content/uploads/sites/8/2017/08/Zakirzyanov-Shalyto-Ulyantsev_DataMod2017.pdf.