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: Буздалов Максим Викторович _____

Направление подготовки: 01.03.02 Прикладная математика и информатика

Квалификация: Бакалавр

Руководитель: Шалыто А.А., проф., д.т.н. _____

К защите допустить

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

«____» _____ 20___ г.

Санкт-Петербург, 2019 г.

Студент Буздалов М.В. Группа М3439 Факультет ИТиП

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

Консультанты:

а) Белашенков Н.Р., канд. физ.-мат. наук, без звания _____

б) Беззубик В.В., без степени, без звания _____

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 Буздалов М.В. **Группа** М3439 **Факультет** ИТиП

Руководитель Шалыто А.А., проф., д.т.н., главный 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

Требуется разработать стилевой файл для системы LaTeX, позволяющий оформлять бакалаврские работы и магистерские диссертации на кафедре компьютерных технологий Университета ИТМО. Стилевой файл должен генерировать титульную страницу пояснительной записки, задание, аннотацию и содержательную часть пояснительной записк. Первые три документа должны максимально близко соответствовать шаблонам документов, принятым в настоящий момент на кафедре, в то время как содержательная часть должна максимально близко соответствовать ГОСТ 7.0.11-2011 на диссертацию.

4 Содержание выпускной работы (перечень подлежащих разработке вопросов)

Пояснительная записка должна демонстрировать использование наиболее типичных конструкций, возникающих при составлении пояснительной записки (перечисления, рисунки, таблицы, листинги, псевдокод), при этом должна быть составлена так, что демонстрируется корректность работы стилевого файла. В частности, записка должна содержать не менее двух приложений (для демонстрации нумерации рисунков и таблиц по приложениям согласно ГОСТ) и не менее десяти элементов нумерованного перечисления первого уровня вложенности (для демонстрации корректности используемого при нумерации набора русских букв).

5 Перечень графического материала (с указанием обязательного материала)

Графические материалы и чертежи работой не предусмотрены

6 Исходные материалы и пособия

а) ГОСТ 7.0.11-2011 «Диссертация и автореферат диссертации»;

б) С.М. Львовский. Набор и верстка в системе LaTeX;

в) предыдущий комплект стилевых файлов, использовавшийся на кафедре компьютерных технологий.

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: Буздалов Максим Викторович

Наименование темы ВКР: Пример оформления ВКР бакалавра

Наименование организации, где выполнена ВКР: Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования: Разработка удобного стилевого файла LaTeXдля бакалавров и магистров кафедры компьютерных технологий.

2 Задачи, решаемые в ВКР:

а) обеспечение соответствия титульной страницы, задания и аннотации шаблонам, принятым в настоящее время на кафедре;

б) обеспечение соответствия содержательной части пояснительной записки требованиям ГОСТ 7.0.11-2011 «Диссертация и автореферат диссертации»;

в) обеспечение относительного удобства в использовании — указание данных об авторе и научном руководителе один раз и в одном месте, автоматический подсчет числа тех или иных источников.

3 Число источников, использованных при составлении обзора: 4

4 Полное число источников, использованных в работе: 9

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	1	0	4	1	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 `tabularx` для чуть более продвинутых таблиц	1.8, Приложения А, Б
Пакет `biblatex` и программное средство `biber`	Список использованных источников

8 Краткая характеристика полученных результатов

Получился, надо сказать, практически неплохой стилевик. В 2015–2018 годах его уже использовали некоторые бакалавры и магистры. Надеюсь на продолжение.

9 Гранты, полученные при выполнении работы

Автор разрабатывал этот стилевик исключительно за свой счет и на добровольных началах. Однако значительная его часть была бы невозможна, если бы автор не написал в свое время кандидатскую диссертацию в LaTeX, а также не отвечал за формирование кучи научно-технических отчетов по гранту, известному как «5-в-100», что происходило при государственной финансовой поддержке ведущих университетов Российской Федерации (субсидия 074-U01).

10 Наличие публикаций и выступлений на конференциях по теме работы

По теме этой работы я (к счастью!) ничего не публиковал. Однако покажу, как можно ссылаться на свои публикации из списка литературы:

1 *Буздалов М. В.* Генерация тестов для олимпиадных задач по программированию с использованием генетических алгоритмов // Научно-технический вестник СПбГУ ИТМО. — 2011. — 2(72). — С. 72–77.

2 *Buzdalov M.*, *Shalyto A.* Hard Test Generation for Augmenting Path Maximum Flow Algorithms using Genetic Algorithms: Revisited // Proceedings of IEEE Congress on Evolutionary Computation. — 2015. — P. 2121–2128.

Студент Буздалов М.В. _____

Руководитель Шалыто А.А. _____

«____» _____ 20__ г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пока что не знаю, что здесь написать. Скорее всего, никогда не узнаю.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

В настоящее время широко распространены системы реального времени. Остро стоит вопрос построения модели данной системы реального времени. На практике зачастую сложно создать модель по имеющейся системе. Гораздо более легким подходом оказывается генерация моделей по примерам их поведения. Одной из наиболее востребованных и изученных моделей является Детерминированный Конечный Автомат (ДКА). Данный вид моделей хорошо изучен в теории формальных языков и для него разработано множество алгоритмов.

Примеры поведения моделей обычно представляют собой список событий, которые произошли с системой, аннотированные временем, которое прошло после предыдущего события. Для систем реального времени конкретно данный вид моделей подходит плохо, так как ДКА практически никак не учитывает время. Тем не менее, в данную модель возможно внести время прямолинейным способом, если ввести некоторое нулевое событие, которое описывает атомарную единицу времени. Данный подход плох тем, что теперь, чтобы адаптировать примеры поведения в данного ДКА, необходимо перевести все задержки во времени в унарную систему счисления, что приведет к экспоненциальному росту длины самих примеров и, как следствие, индуциованной по данным примерам модели. Но существует модель, очень близкая по строению к существующей модели Детерминированного Конечного Автомата, хорошо обрабатывающая время, как раз нацеленная на использование в качестве модели для систем реального времени.

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.

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 $\langle q_1, q_2, a, g, R\rangle$, где
  – $q_1, q_2 \in Q$ - 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 20 раз пройти по петле в то же состояние q_2.

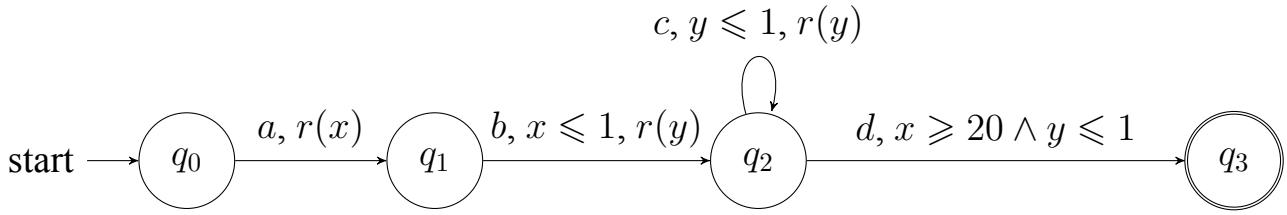

Рисунок 1 – Пример временного автомата

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 $\nu(x)$ таймера $x \in C$ – функция из таймера x в $\mathbb{N}$

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 и 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 количества таймеров, каждая же пара с нечетным индексом отличается от предыдущей только увеличением значений таймеров.

Определение 7. Языком $\mathcal{L}(\mathcal{A})$ временного автомата $\mathcal{A}$ назовем множество всех принимаемых трассировок.

Определение 8. 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 $|\tau| < p(|\mathcal{A}|)$.

1.2. Известные результаты

Лемма 9. Класс временных автоматов не является достижимым за полином.

Лемма 9 фактически утверждает, что, чтобы однозначно правильно синтезировать автомат по примерам, необходимо в качество примеров иметь так же пример экспоненциальной длины от размера временного автомата.

Определение 10. 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 11. Класс временных автоматов не различим за полином.

Определение 12. 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 13. Класс автоматов $\mathbb{K}$ является эффективно синтезируемым в пределе, если существуют полиномы p, q и алгоритм A такие, что

— Время работы алгоритма на входе S ограничено сверху $p(\sum_{\tau \in S} |\tau|)$

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

Теорема 14. Класс временных автоматов не является эффективно синтезируемым

Доказаны также некоторые послабления данной теоремы.

Теорема 15. Если $coNP \neq PSPACE$, то временной автомат не может быть эффективно синтезирован.

Теорема 16. 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 17. Класс временных автоматов, в котором используется два или больше таймеров не может быть эффективно синтезирован.

Таким образом, возможно стоит данный подкласс во временных автоматах, которые могут использовать всего таймер? На самом деле в статье (должен быть реф) были доказаны следующие факты:

Лемма 18. Класс временных автоматов с одним таймером является полиномиально достижимым.

Лемма 19. Класс временных автоматов с одним таймером является полиномиально различимым.

Теорема 20. Класс временных автоматов с одним таймером является эффективно синтезируемым.

Определение 21. Автомат реального времени – временной автомат такой, что $C = \{c_0\}, R_\delta = \{c_0\}$ для любого перехода $\delta \in \Delta$.

В качестве доказательства теоремы 20 авторы статьи (должен быть реф) предоставили интересный алгоритм, который хотелось бы разобрать.

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

В алгоритме временной автомат строится итеративно. Алгоритм выполняется, пока существует трассировка, некоторый префикс которой заканчивается в состоянии q и из этого состояния не существует перехода $\langle q, q', a, g, r\rangle$ по символу a, удовлетворяя ограничения g. На каждой итерации алгоритм перебирает текущие состояния автомата и выбирает минимальное значение таймера v_{min}, с которым можно добраться до текущего состояния, а также максимальное значение таймера c', для которого не существует перехода по символу a, удовлетворяя ограничения g. Если $v_{min} \leqslant c'$, то значит мы не можем перейти по символу a для текущей трассировки с таймером v_{min}. Нужно выбрать нижнюю границу нового перехода. Ее выбираем минимальной, после добавления перехода с такой нижней границей, можно корректно достроить автомат. В качестве возможной нижней границы попробуем подставить всевозможные значения таймеров, которые можно получить, проходя по существующим трассировкам. Корректно достроить автомат можно, если не существует принимающейся и отвергающейся трассировок, префиксы которых приводят в одно и то же состояние, далее идет переход по одинаковому символу с ожиданиями, выравнивающими значения таймеров на следующем состоянии, и далее суффиксы этих трассировок совпадают. Соответственно, алгоритм выбирает минимальную границу из ситуации, когда на данном ребре есть сброс таймера и когда его нет. Осталось выбрать конечное состояние для данного ребра. Выбираем первое состояние, при проведении данного ребра в которое автомат до сих пор сможет корректно достроить автомат. Если такого состояния не существует, добавляем новую вершину и проводим ребро в него. Следующая вершина будет принимающей в случае, если существует трассировка τ, которая заканачивается в нем и $\tau \in S_+$.

1.4. Эффективный синтез автомата реального времени с помощью алгоритмы RTI

Данный алгоритм основан на адаптации текущего state-of-the-art алгоритма синтеза конечных детерминированных автоматов EDSM [7]. Данный алгоритм начинает синтез автомата с построения префиксного дерева трассировок. Данное префиксное дерево будет выступать в качестве первоначального автомата. Далее алгоритм пытается находить состояния, которые, как он пред-

полагает, являются одинаковыми в минимальном конечном автомате и сливает их.

EDSM синтезирует автомат в так называемой красно-синей среде. Данная среда 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 никак не рассматриваются. Красные состояния получаются окаймлены синими, то есть для каждого синего состояния на текущем шаге есть красное состояние с ребром, которое ведет в данное синее.

На данном 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. Предолагается, что между двумя красными состояниями переходы никаким образом поменяться не могут. При слиянии двух состояний множество переходов, выходящих из этих двух состояний, объединяются. Но тогда получившийся автомат может оказаться уже недетерминированным: из новой вершины, слитой из двух, может оказаться 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 сливаются так же 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, так же должны удовлетворять предикату на возможность слияния. Возможна так же ситуация, что на текущем шаге лучше покрасить синию вершину в красную, что сливать вершины.

Нужно упомянуть, что синтез автомата в красно-синей среде 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 какой-нибудь ещё непосещенной вершиной. Таким образом, размер автомата, который получит-

ся 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 конечного детерминированного автомата по трассировкам это NP-полная задача, то для этого применяют эвристики. В алгоритме EDSM используется используется эвристика для оценки правильности некоторой операции. Эта оценка является просто натуральным числом. Тогда на текущей итерации оценивают правильность всевозможных слияний, всевозможных покрасок и применяют ту, 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:

$$pure = \#pos + \#neg \tag{1}$$

В эвристике 1 $\#pos$, $\#neg$ - количество слитых вершин, которые обе были принимающие и отвергающие соответственно. При равенстве нескольких наилучших оценок выбирают сначала слияние, а затем покраску. Иными словами, с данной эвристикой у покраски всегда оценка правильности равна ноль. 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 выставляется 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 – отклоняемой. 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 среде, 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 превра-

тить 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 имеющими 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 так же применяется глубокое копирование выходящих префиксных деревьев. Проверка на то, что автомат можно достроить до консистентного точно такая же, как и в EDSM алгоритме (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 количество 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 1, заимствованная из алгоритма EDSM. В качестве оценки правильности операций разделения тоже можно выбрать ноль. При равенстве нескольких наилучших оценок операций приоритет отдается сначала слиянию, затем разделению, а затем покраске.

Следующая эвристика – это подправленная эвристика 1. В отличие в неё в данной эвристике так же учитывается количество новых неконсистентных состояний.

$$consistent = \#pos + \#neg - \#posneg \tag{2}$$

В эвристике 2 $\#pos$, $\#neg$ – так же количество слитых вершин, которые обе принимающие или отвергающие, $\#posneg$ – количество 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:

$$\begin{aligned} impact = \sum_{q \in Q_r} pure(q) &- \sum_{\tau \in \Delta_b} max\{impact(\tau, \tau') | \tau \in S_+^\delta \wedge \tau' \in S_-^\delta\} \\ &+ \sum_{\tau \in \Delta_b} max\{impact(\tau, \tau') | \tau \in S_+^\delta \wedge \tau' \in S_+^\delta\} \\ &+ \sum_{\tau \in \Delta_b} max\{impact(\tau, \tau') | \tau \in S_-^\delta \wedge \tau' \in S_-^\delta\} \end{aligned} \tag{3}$$

В эвристике 3 $impact(\tau, \tau')$ – похожесть двух временных строк τ и τ', $pure(q)$ – эвритическая 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 количество посчитать сложно из-за NP-трудности данной задачи, применяется некоторый жадный алгоритм. Эвристическая оценка имеет следующий вид:

$$splits = \sum_{q \in Q} max(pure(q) - \#split(q), 0) \tag{4}$$

В эвристике 4 $\#splits(q)$ – приблизительное количество разделений, необходимых для избавления от неконсистентных состояний в префиксном дереве, выходящем из вершины $q \in Q$.

При тестировании лучше всего себя показали эвристические оценки 2 и 4.

1.5. Подход к синтезу в алгоритме Timed-k-Tail

Данный алгоритм Timed-k-Tail[8] является адаптацией алгоритма k-Tail[3], применяемого для синтеза детерминированных конечных автоматов без негативных примеров 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 потупает некоторое множество трассировок выполнения реальной программы. Первое, что происходит в алгоритме, так это изменяются трассировки 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. Но все задержки времени между любой парой событий в трассировке сохраняются неизменными.

Вторая стадия – построение по трассировкам префиксного дерева. В данном алгоритме используется огромное количество 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 . 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 одина-

ковым переходным символом так же слиться, при этом объединяя множества сбросов и множество ограничений.

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

Пытаясь протестировать, у авторов статьи получилась откровенная лажа, поэтому данный алгоритм никак нельзя считать state-of-the-art и вообще воспринимать всерьёз.

1.6. Применение временных автоматов на практике. Алгоритм МОНА

1.7. Дальше можно не читать. Пока ничего нет!!!

Пример ссылок в рамках обзора: [1, 4–6]. Вне обзора: [2].

1.8. Таблицы

В качестве примера таблицы приведена таблица 1.

Есть еще такое окружение `tabularx`, его можно аккуратно растянуть на всю страницу. Приведем пример (таблица 2).

Таблица 1 – Таблица умножения (фрагмент)

–	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68

Таблица 2 – Таблица умножения с помощью `tabularx` (фрагмент)

–	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68

1.9. Рисунки

Пример рисунка (с помощью `TikZ`) приведен на рисунке 2. Под `pdflatex` можно также использовать `*.jpg`, `*.png` и даже `*.pdf`, под `latex` можно использовать Metapost. Последний можно использовать и под `pdflatex`, для чего в стилевике продекларированы номера картинок от 1 до 20.

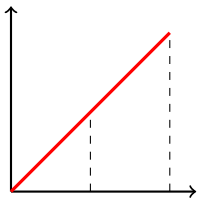

Рисунок 2 – Пример рисунка

1.10. Листинги

В работах студентов кафедры «Компьютерные технологии» часто встречаются листинги. Листинги бывают двух основных видов — исходный код и псевдокод. Первый оформляется с помощью окружения `lstlisting` из пакета `listings`, который уже включается в стилевике и немного настроен. Пример Hello World на Java приведен на листинге 1. Пример большого листинга — в приложении (листинг В.1).

Псевдокод можно оформлять с помощью разных пакетов. В данном стилевике включается пакет `algorithmicx`. Сам по себе он не генерирует фло-

Листинг 1 – Пример исходного кода на Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

атов, поэтому для них используется пакет `algorithm`. Пример их совместного использования приведен на листинге 2.

Листинг 2 – Пример псевдокода

function IsPrime(N)
    for $t \leftarrow [2; \lfloor\sqrt{N}\rfloor]$ **do**
        if $N \bmod t = 0$ **then**
            return false
        end if
    end for
    return true
end function

Наконец, листинги из `listings` тоже можно подвешивать с помощью `algorithm`, пример на листинге 3.

Листинг 3 – Исходный код и флоат `algorithm`

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

ГЛАВА 2. ПРОВЕРКА СКВОЗНОЙ НУМЕРАЦИИ

Листинг 4 должен иметь номер 4.

Листинг 4 – Исходный код и флоат `algorithm`

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Рисунок 3 должен иметь номер 2.

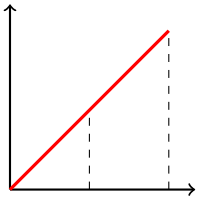

Рисунок 3 – Пример рисунка

Таблица 3 должна иметь номер 3.

Таблица 3 – Таблица умножения с помощью `tabularx` (фрагмент)

–	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68

Выводы по главе 2

В конце каждой главы желательно делать выводы. Вывод по данной главе — нумерация работает корректно, ура!

ЗАКЛЮЧЕНИЕ

В данном разделе размещается заключение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Буздалов М. В.* Генерация тестов для олимпиадных задач по программированию с использованием генетических алгоритмов // Научно-технический вестник СПбГУ ИТМО. — 2011. — 2(72). — С. 72–77.

2 *Bellman R. E.* Dynamic Programming. — Princeton, NJ : Princeton University Press, 1957. — 342 p.

3 *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.

4 *Buzdalov M.*, *Doerr B.*, *Kever M.* The Unrestricted Black-Box Complexity of Jump Functions // Evolutionary Computation. — 2016. — Accepted for publication.

5 *Buzdalov M.*, *Shalyto A.* Hard Test Generation for Augmenting Path Maximum Flow Algorithms using Genetic Algorithms: Revisited // Proceedings of IEEE Congress on Evolutionary Computation. — 2015. — P. 2121–2128.

6 *Doerr B.*, *Doerr C.* Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings [Электронный ресурс]. — 2015. — URL: http://arxiv.org/abs/1504.03212.

7 *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.

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

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.

ПРИЛОЖЕНИЕ А. ПРИМЕР ПРИЛОЖЕНИЯ

В приложениях рисунки, таблицы и другие подобные элементы нумеруются по приложениям с соответствующим префиксом. Проверим это.

Листинг А.1 должен иметь номер А.1.

Листинг А.1 – Исходный код и флоат `algorithm`

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Рисунок А.1 должен иметь номер А.1.

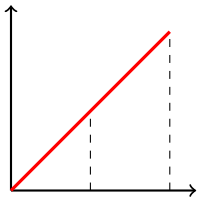

Рисунок А.1 – Пример рисунка

Таблица А.1 должна иметь номер А.1.

Таблица А.1 – Таблица умножения с помощью `tabularx` (фрагмент)

–	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68

Заодно проверим нумерованные и ненумерованные перечисления. Ненумерованные:

— пункт А;
— пункт Б;
— пункт В.

Нумерованные списки нескольких уровней:

а) первый элемент;
б) второй элемент с подэлементами:
    1) первый подэлемент;

2) второй подэлемент;

3) третий подэлемент.

в) третий элемент;

г) четвертый элемент;

д) пятый элемент;

е) шестой элемент;

ж) седьмой элемент;

и) восьмой элемент;

к) девятый элемент;

л) десятый элемент.

ПРИЛОЖЕНИЕ Б. ЕЩЕ ОДИН ПРИМЕР ПРИЛОЖЕНИЯ С НЕИМОВЕРНО ДЛИННЮЩИМ НАЗВАНИЕМ ДЛЯ ТЕСТИРОВАНИЯ ПЕРЕНОСОВ

Проверим на примере таблиц, что нумерация в приложениях — по приложениям. Таблица Б.1 должна иметь номер Б.1.

Таблица Б.1 – Таблица умножения с помощью `tabularx` (фрагмент)

–	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68

ПРИЛОЖЕНИЕ В. ПРИМЕР ОГРОМНОГО ЛИСТИНГА

Листинг В.1 – Пример большого листинга

```
import java.util.*;

public class Example {
    static int[] restoreOutgoing(int[] g, int[] outgoing,
                                 int vertex, int mask) {
        int[] rv = new int[1 + Integer.bitCount(mask)];
        int n = g.length;
        int current = rv.length - 1;
        while (true) {
            rv[current] = vertex;
            if (current == 0) {
                if (vertex != 0) {
                    throw new AssertionError();
                }
                return rv;
            }
            mask ^= 1 << (vertex - 1);
            int prevMask = outgoing[mask] & g[vertex];
            if (prevMask == 0) {
                throw new AssertionError();
            }
            vertex = Integer.numberOfTrailingZeros(prevMask);
            --current;
        }
    }

    static int[] restoreIncoming(int[] g, int[] incoming,
                                 int vertex, int mask) {
        int[] rv = new int[1 + Integer.bitCount(mask)];
        int n = g.length;
        int current = 0;
        while (true) {
            rv[current] = vertex;
            if (current == rv.length - 1) {
                if (vertex != 0) {
                    throw new AssertionError();
                }
                return rv;
            }
```

```
                mask ^= 1 << (vertex - 1);
                int nextMask = incoming[mask] & g[vertex];
                if (nextMask == 0) {
                    throw new AssertionError();
                }
                vertex = Integer.numberOfTrailingZeros(nextMask);
                ++current;
            }
        }
    }
```