Министерство образования и науки Российской Федерации

УДК 004.4’242
ГРНТИ 20.01.01, 28.23.29
Инв. № 310275

УТВЕРЖДЕНО:
Исполнитель: **федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»**
Ректор ФГБОУ ВПО «СПбНИУ ИТМО» **_____/ В.Н. Васильев/** **М.П.**

НАУЧНО-ТЕХНИЧЕСКИЙ ОТЧЕТ

о выполнении 3 этапа Государственного контракта
№ 16.740.11.0455 от 13 мая 2011 г. и Дополнению от 25 октября 2011 г. № 1

Исполнитель: федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»
Программа (мероприятие): Федеральная целевая программа «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг., в рамках реализации мероприятия № 1.2.1 Проведение научных исследований научными группами под руководством докторов наук.
Проект: Разработка метода машинного обучения на основе алгоритмов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов
Руководитель проекта: Шалыто Анатолий Абрамович

Санкт-Петербург
2012 г.

СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

по Государственному контракту 16.740.11.0455 от 13 мая 2011 на выполнение поисковых научно-исследовательских работ для государственных нужд

Организация-Исполнитель: федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики».

Руководитель темы:

доктор технических наук, профессор	А.А. Шалыто

Исполнители темы:

без ученой степени, без ученого звания	В.И. Ульянцев
без ученой степени, без ученого звания	Ф.Н. Царев
без ученой степени, без ученого звания	М.В. Буздалов
кандидат технических наук, доцент	Г.А. Корнеев
без ученой степени, без ученого звания	С.В. Казаков
кандидат технических наук, доцент	А.С. Станкевич

РЕФЕРАТ

Отчет 70 с., 1 ч., 12 рис., 2 табл., 18 источн., 6 прил.

Ключевые слова: машинное обучение, управляющий конечный автомат, автоматное программирование, задача о выполнимости булевой формулы, дерево сценариев, граф совместимости, КНФ-формула

В отчете представлены результаты исследований, выполненных по 3 этапу Государственного контракта № 16.740.11.0455 «Разработка метода машинного обучения на основе алгоритмов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов» (шифр «2011-1.2.1-113-002») от 13 мая 2011 по направлению «Проведение научных исследований научными группами под руководством докторов наук в области информатики» в рамках мероприятия 1.2.1 «Проведение научных исследований научными группами под руководством докторов наук», мероприятия 1.2 «Проведение научных исследований научными группами под руководством докторов наук и кандидатов наук», направления 1 «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы.

Цель работы – разработка метода машинного обучения на основе алгоритмов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов

В Государственном контракте методы, использованные при выполнении отдельных видов работ (этапов), не предусмотрены.

При выполнении первого этапа НИР был использован следующий инструментарий:

1. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036527.
2. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036528.
3. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036529.
4. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036530.
5. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036531.
6. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036532.
7. Компьютер *Aquar7ius Elt E50 S46*. Инв. номер. 110104.7.0036533.
8. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036534.
9. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036535.
10. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036536.
11. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036537.
12. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036538.
13. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036539.
14. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036540.
15. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036541.
16. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036542.
17. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036543.

При выполнении патентных исследований и подготовке научно-технического отчета были использованы следующие нормативные документы:

- Постановление Правительства Российской Федерации от 4 мая 2005 г. № 284 «О государственном учете результатов научно-исследовательских, опытно-конструкторских и технологических работ гражданского назначения»;
- Гражданский кодекс РФ;
- ГОСТ 7.32-2001 «Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления»;

• ГОСТ 15.101.98 «Система разработки и постановки продукции на производство. Порядок выполнения научно-исследовательских работ».

В ходе исполнения обязательств по Государственному контракту № 16.740.11.0455 от 13 мая 2011 г. был создан научно-технический отчет, включающий разработанный алгоритм построения булевой формулы по графу совместимости, программную реализацию алгоритма построения булевой формулы по графу совместимости, алгоритм построения автомата по найденной выполняющей подстановке, программную реализацию алгоритма построения автомата по найденной выполняющей подстановке и публикации результатов НИР.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящем отчете излагаются результаты выполнения этапа 3 «Разработка и программная реализация алгоритма построения булевой формулы по графу совместимости и алгоритма построения автомата по выполняющей подстановке этой формулы» *поисковой научно-исследовательской работы по теме «Разработка метода машинного обучения на основе алгоритмов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов»*, выполняемых в рамках государственного контракта, заключенного между Министерством образования и науки Российской Федерации и государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» в соответствии с решением Конкурсной комиссии Министерства образования и науки Российской Федерации № 1 (протокол от 25.03.2011 № 3/01731000037110000031) по лоту шифр «2011-1.2.1-113-002» «Проведение научных исследований научными группами под руководством докторов наук в области информатики» в рамках федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы».

Целями настоящего этапа являются:

1. Разработка алгоритма построения булевой формулы по графу совместимости.
2. Программная реализация алгоритма построения булевой формулы по графу совместимости.
3. Разработка алгоритма построения автомата по найденной выполняющей подстановке.

4. Программная реализация алгоритма построения автомата по выполняющей подстановке.

Выполнение работ настоящего этапа позволяет в дальнейшем перейти от теоретических исследований и реализации технических решений к экпериментальным исследованиям.

Отчет имеет следующую структуру. В первой главе приводится описание алгоритма построения булевой формулы по графу совместимости. Описываются входные и выходные данные алгоритма, примеры его работы на двух задачах. Во второй главе приводится программная реализация разработанного алгоритма.

В третьей главе приводится описание алгоритма построения автомата по выполняющей подстановке. Описываются входные и выходные данные алгоритма, примеры его работы для двух задач. В четвертой главе приводится программная реализация разработанного алгоритма.

Каждая из глав снабжена выводами, кратко резюмирующими содержание главы. В заключении дается общая оценка работ по этапу.

Машинное обучение [3] – современный раздел информатики, посвященный созданию алгоритмов, которые обучают параметры некоторой модели на заранее подготовленных тестовых примерах, либо в процессе моделирования ее работы в некоторой внешней среде (обучение «на собственных ошибках»).

Задача о выполнимости булевой формулы – одна из задач, имеющих большое значение в информатике. Она состоит в том, чтобы по заданной булевой формуле найти такой набор значений входящих в нее переменных, что результат вычисления формулы – «истина». Эта задача относится к классу *NP*-полных задач, но для ее решения существует большое количество весьма эффективных на практике программных средств. Наиболее современные из этих программных средств, такие как *zChaff* [18],

MiniSat [11], *CryptoMiniSat* [10], *Plingeling* [8], способны обрабатывать формулы длиной в несколько миллионов символов, содержащие десятки и сотни тысяч переменных.

С использованием этих программных средств в настоящее время решаются такие задачи, как проверка эквивалентности формальных моделей программ, верификация моделей программ, формальная верификация микропроцессоров, генерация тестовых шаблонов, разводка печатных плат и т. д.

Автоматное программирование [4] – парадигма программирования, при использовании которой программу предлагается разрабатывать в виде совокупности автоматизированных объектов управления, каждый из которых содержит систему управления (взаимодействующие конечные автоматы) и объект управления.

Объект управления характеризуется множеством вычислительных состояний, а также двумя наборами функций: множеством предикатов, отображающих вычислительное состояние в логическое значение («истина» или «ложь»), и множеством действий, позволяющих изменять вычислительное состояние. Управляющий автомат определяется конечным множеством управляющих состояний, функцией переходов и функцией действий.

Исследования, проводимые в ряде университетов мира (например, в Стэнфордском университете), показывают, что модели, основанные на состояниях, целесообразно применять для построения систем со сложным поведением. Такие системы могут в ответ на одно и то же входное воздействие вырабатывать различные выходные воздействия в зависимости от предыстории работы.

Модели, основанные на состояниях, широко применяются в машинном обучении. Примером таких моделей являются Марковские цепи и скрытые Марковские модели [16]. Они могут применяться в таких задачах, как распознавание речи, и для них разработан ряд алгоритмов обучения. Область их применения ограничивается тем, что

эти модели имеют вероятностный характер. Наиболее близкими к ним моделями, имеющими детерминированный характер, являются конечные автоматы.

Важным достоинством автоматных программ является возможность их автоматической верификации [1], что является очень существенным свойством для систем с повышенными требованиями к надежности. Для этих систем важную роль играет влияние человеческого фактора на процесс разработки. Поэтому актуальной задачей является разработка методов машинного обучения для построения конечных автоматов.

Как правило, в машинном обучении конечные автоматы рассматриваются как распознаватели регулярных языков [14], или как преобразователи слов одного регулярного языка в слова другого. В случае построения управляющих автоматов обычно рассматриваются системы с малым числом входных переменных (одной или двумя), что существенно ограничивает область применения таких алгоритмов.

Кроме этого, большая часть существующих алгоритмов машинного обучения для конечных автоматов основана на принципе обучения «на собственных ошибках» и не являются достаточно эффективными для ряда задач, так как не позволяют эффективно учитывать знания человека. В свою очередь, алгоритмы, основанные на принципе обучения на примерах, обладают недостаточно высокой производительностью для использования на практике – они позволяют работать с автоматами, содержащими пять-десять состояний, а суммарная длина обучающих примеров может составлять не более двухсот-трехсот событий. Построение автомата с помощью существующих алгоритмов может занимать от нескольких минут до нескольких дней.

Для устранения недостатков существующих методов разрабатывается метод машинного обучения, основанный на решении задачи о выполнимости булевой формулы. В качестве входных данных этот метод будет использовать сценарии работы – один из типов обучающих примеров (тестов). Предварительные исследования

показывают, что новый метод будет демонстрировать существенно более высокую производительность (в 10-100 раз более высокую, чем у существующих методов) и сможет строить автоматы с десятками состояний, а размер входных данных сможет составлять тысячи событий.

Изложенное позволяет утверждать, что результаты выполнения научно-исследовательской работы будут превышать мировой уровень разработок в рассматриваемой области.

1. РАЗРАБОТКА АЛГОРИТМА ПОСТРОЕНИЯ БУЛЕВОЙ ФОРМУЛЫ ПО ГРАФУ СОВМЕСТИМОСТИ

В настоящем разделе приводится описание алгоритма построения булевой формулы по графу совместимости [5]. Разработка алгоритма будет осуществляться на основе методов теории графов, теории автоматов, булевой логики.

1.1. Входные и выходные данные алгоритма

Входными данными для алгоритма являются:

- дерево сценариев работы программы;
- построенный по данному дереву граф совместимости;
- число *C* состояний искомого автомата.

Пример дерева сценариев приведен на рис. 1.

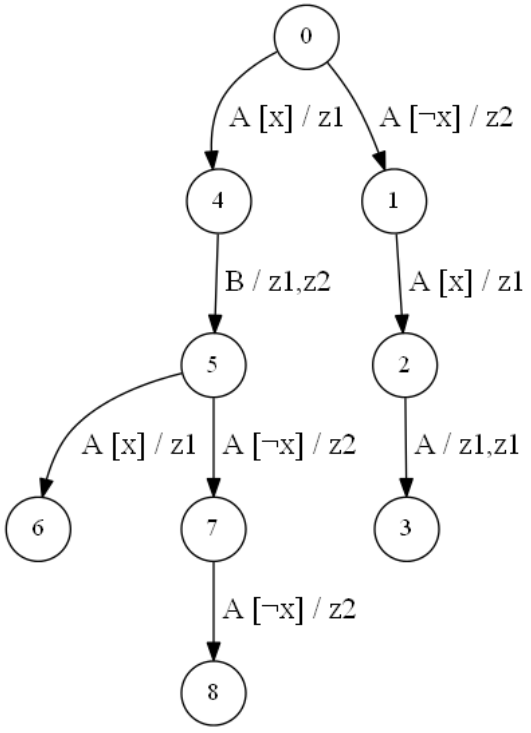

Рис. 1. Пример дерева сценариев

На рис. 2 приведен граф совместимости для данного дерева сценариев (ребра графа изображены полужирными линиями).

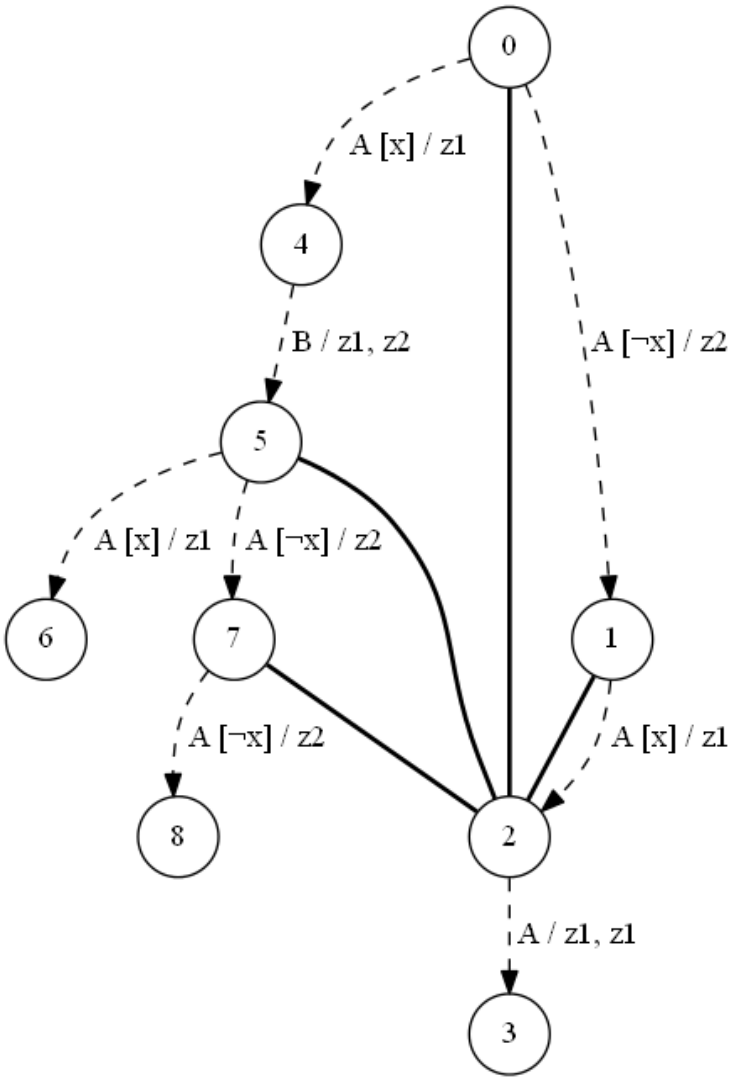

Рис. 2. Пример графа совместимости

Выходными данными алгоритма является КНФ-формула, задающая требования к раскраске построенного графа и выражающая непротиворечивость системы переходов результирующего автомата. Данная формула должна быть представлена в формате *DIMACS* [17] – стандартном формате для записи такого типа формул. Приведем пример записи КНФ-формулы в данном формате.

```
c
c start with comments
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

1.2. Алгоритм построения булевой формулы по графу совместимости

Опишем алгоритм построения булевой КНФ-формулы, задающей требования к раскраске построенного графа и выражающей непротиворечивость системы переходов результирующего автомата. Данное построение аналогично построению КНФ-формулы, используемой в работе [14] для построения автомата-распознавателя.

Напомним, что в настоящей работе на вход алгоритму подается число C состояний результирующего автомата.

В данной формуле будут использоваться следующие логические переменные:

- $x_{v,i}$ (для каждой вершины дерева сценариев v и цвета вершины i – числа от 0 до $C-1$) – верно ли, что вершина v имеет цвет i? Напомним, что вершины одного цвета будут объединены в одно состояние результирующего автомата;
- $y_{a,b,e,f}$ (для каждой пары состояний результирующего автомата a и b, каждого события e, каждой формулы f, встречающейся в сценариях) – верно ли, что в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f?

КНФ-формула состоит из следующих дизъюнктов:

- $(x_{v,0} \vee \ldots \vee x_{v,C\text{-}1})$ (для каждой вершины v дерева сценариев) – накладываем условие существования цвета вершины v;

- $(\neg x_{v,i} \vee \neg x_{v,j})$ (для каждой вершины v дерева сценариев, цвета i и цвета j, где $i < j$) – накладываем условие, что вершина v не покрашена одновременно в цвета i и j;
- $(\neg x_{v,i} \vee \neg x_{u,i})$ (для каждой пары несовместимых вершин u и v дерева сценариев и цвета i) – накладываем ограничения на раскраску, задаваемые графом совместимости;
- $(\neg y_{a,b,e,f} \vee \neg y_{a,d,e,f})$ (для каждой тройки состояний результирующего автомата a, b и d ($b < d$), каждого события e, каждой формулы f) – из состояния a выходит не более одного ребра, помеченного событием e и формулой f;
- $(y_{a,b,e,f} \vee \neg x_{v,a} \vee \neg x_{u,b})$ (для каждого ребра vu дерева сценариев) – если выполняется:
  - ребро из вершины дерева v в вершину u помечено событием e и формулой f;
  - вершина v покрашена в цвет a;
  - вершина u покрашена в цвет b;

  то в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f;
- $(\neg y_{a,b,e,f} \vee \neg x_{v,a} \vee x_{u,b})$ (для каждого ребра vu дерева сценариев) – если выполняется:
  - ребро из вершины дерева v в вершину дерева u помечено событием e и формулой f;
  - вершина v покрашена в цвет a;
  - в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f;

  то вершина u покрашена в цвет b.

1.3. Примеры работы предложенного алгоритма

В настоящем разделе приведены примеры работы предложенного алгоритма построения булевой формулы по графу совместимости. Сначала будет рассмотрен граф, построенный по небольшому набору сценариев. Затем будет рассмотрен граф, построенный по большему набору сценариев, описывающему работу автомата управления часами с будильником.

1.3.1. Пример работы алгоритма на небольшом наборе сценариев

На рис. 1 приведен пример дерева сценариев, построенного по сценариям:

- <*A*, ¬*x*, (*z*2)>, <*A*, *x*, (*z*1)>, <*A*, *true*, (*z*1,*z*1)>;
- <*A*, *x*, (*z*1)>, <*B*, *true*, (*z*1,*z*2)>, <*A*, *x*, (*z*1)>;
- <*A*, *x*, (*z*1)>, <*B*, *true*, (*z*1,*z*2)>, <*A*, ¬*x*, (*z*2)>, <*A*, ¬*x*, (*z*2)>.

Приведем булеву КНФ-формулу, построенную по данному графу совместимости. Результирующий автомат должен содержать два состояния, поэтому параметр *C* равен двум. В формуле будут использоваться логические переменные, приведенные в табл. 1.

Таблица 1. Переменные построенной КНФ-формулы

Описание переменных	Переменные
Переменные вида $x_{v,i}$	$x_{0,0}$, $x_{0,1}$, $x_{1,0}$, $x_{1,1}$, $x_{2,0}$, $x_{2,1}$, $x_{3,0}$, $x_{3,1}$, $x_{4,0}$, $x_{4,1}$, $x_{5,0}$, $x_{5,1}$, $x_{6,0}$, $x_{6,1}$, $x_{7,0}$, $x_{7,1}$, $x_{8,0}$, $x_{8,1}$
Переменные вида $y_{a,b,e,f}$	$y_{0,0,A,true}$, $y_{0,0,A,x}$, $y_{0,0,A,\neg x}$, $y_{0,0,B,true}$, $y_{0,1,A,true}$, $y_{0,1,A,x}$, $y_{0,1,A,\neg x}$, $y_{0,1,B,true}$, $y_{1,0,A,true}$, $y_{1,0,A,x}$, $y_{1,0,A,\neg x}$, $y_{1,0,B,true}$, $y_{1,1,A,true}$, $y_{1,1,A,x}$, $y_{1,1,A,\neg x}$, $y_{1,1,B,true}$

Таким образом, в формуле используется 34 логические переменные. В табл. 2 приведены дизъюнкты каждого вида, описанного в разделе 1.2, рассматриваемой формулы.

Таблица 2. Дизъюнкты построенной КНФ-формулы

Описание дизъюнктов	Дизъюнкты
Дизъюнкт, ставящий в соответствие корню дерева цвет 0	$(x_{0,0})$
9 дизъюнктов вида $(x_{v,0} \vee \ldots \vee x_{v,C\text{-}1})$	$(x_{0,0} \vee x_{0,1})$, $(x_{1,0} \vee x_{1,1})$, $(x_{2,0} \vee x_{2,1})$, $(x_{3,0} \vee x_{3,1})$, $(x_{4,0} \vee x_{4,1})$, $(x_{5,0} \vee x_{5,1})$, $(x_{6,0} \vee x_{6,1})$, $(x_{7,0} \vee x_{7,1})$, $(x_{8,0} \vee x_{8,1})$
9 дизъюнктов вида $(\neg x_{v,i} \vee \neg x_{v,j})$	$(\neg x_{0,0} \vee \neg x_{0,1})$, $(\neg x_{1,0} \vee \neg x_{1,1})$, $(\neg x_{2,0} \vee \neg x_{2,1})$, $(\neg x_{3,0} \vee \neg x_{3,1})$, $(\neg x_{4,0} \vee \neg x_{4,1})$, $(\neg x_{5,0} \vee \neg x_{5,1})$, $(\neg x_{6,0} \vee \neg x_{6,1})$, $(\neg x_{7,0} \vee \neg x_{7,1})$, $(\neg x_{8,0} \vee \neg x_{8,1})$
8 дизъюнктов вида $(\neg x_{v,i} \vee \neg x_{u,i})$	$(\neg x_{2,0} \vee \neg x_{0,0})$, $(\neg x_{2,1} \vee \neg x_{0,1})$, $(\neg x_{2,0} \vee \neg x_{1,0})$, $(\neg x_{2,1} \vee \neg x_{1,1})$, $(\neg x_{2,0} \vee \neg x_{5,0})$, $(\neg x_{2,1} \vee \neg x_{5,1})$,

	$(\neg x_{2,0} \vee \neg x_{7,0})$, $(\neg x_{2,1} \vee \neg x_{7,1})$
8 дизъюнктов вида $(\neg y_{a,b,e,f} \vee \neg y_{a,d,e,f})$	$(\neg y_{0,0,A,true} \vee \neg y_{0,1,A,true})$, $(\neg y_{1,0,A,true} \vee \neg y_{1,1,A,true})$, $(\neg y_{0,0,A,x} \vee \neg y_{0,1,A,x})$, $(\neg y_{1,0,A,x} \vee \neg y_{1,1,A,x})$, $(\neg y_{0,0,A,\neg x} \vee \neg y_{0,1,A,\neg x})$, $(\neg y_{1,0,A,\neg x} \vee \neg y_{1,1,A,\neg x})$, $(\neg y_{0,0,B,true} \vee \neg y_{0,1,B,true})$, $(\neg y_{1,0,B,true} \vee \neg y_{1,1,B,true})$
32 дизъюнкта вида $(y_{a,b,e,f} \vee \neg x_{v,a} \vee \neg x_{u,b})$	$(y_{0,0,A,\neg x} \vee \neg x_{0,0} \vee \neg x_{1,0})$, $(y_{0,1,A,\neg x} \vee \neg x_{0,0} \vee \neg x_{1,1})$, $(y_{1,0,A,\neg x} \vee \neg x_{0,1} \vee \neg x_{1,0})$, $(y_{1,1,A,\neg x} \vee \neg x_{0,1} \vee \neg x_{1,1})$, $(y_{0,0,A,x} \vee \neg x_{1,0} \vee \neg x_{2,0})$, $(y_{0,1,A,x} \vee \neg x_{1,0} \vee \neg x_{2,1})$, $(y_{1,0,A,x} \vee \neg x_{1,1} \vee \neg x_{2,0})$, $(y_{1,1,A,x} \vee \neg x_{1,1} \vee \neg x_{2,1})$, $(y_{0,0,A,true} \vee \neg x_{2,0} \vee \neg x_{3,0})$, $(y_{0,1,A,true} \vee \neg x_{2,0} \vee \neg x_{3,1})$, $(y_{1,0,A,true} \vee \neg x_{2,1} \vee \neg x_{3,0})$, $(y_{1,1,A,true} \vee \neg x_{2,1} \vee \neg x_{3,1})$, $(y_{0,0,A,x} \vee \neg x_{0,0} \vee \neg x_{4,0})$, $(y_{0,1,A,x} \vee \neg x_{0,0} \vee \neg x_{4,1})$, $(y_{1,0,A,x} \vee \neg x_{0,1} \vee \neg x_{4,0})$, $(y_{1,1,A,x} \vee \neg x_{0,1} \vee \neg x_{4,1})$, $(y_{0,0,B,true} \vee \neg x_{4,0} \vee \neg x_{5,0})$, $(y_{0,1,B,true} \vee \neg x_{4,0} \vee \neg x_{5,1})$, $(y_{1,0,B,true} \vee \neg x_{4,1} \vee \neg x_{5,0})$, $(y_{1,1,B,true} \vee \neg x_{4,1} \vee \neg x_{5,1})$, $(y_{0,0,A,x} \vee \neg x_{5,0} \vee \neg x_{6,0})$,

	$(y_{0,1,A,x} \vee \neg x_{5,0} \vee \neg x_{6,1})$, $(y_{1,0,A,x} \vee \neg x_{5,1} \vee \neg x_{6,0})$, $(y_{1,1,A,x} \vee \neg x_{5,1} \vee \neg x_{6,1})$, $(y_{0,0,A,\neg x} \vee \neg x_{5,0} \vee \neg x_{7,0})$, $(y_{0,1,A,\neg x} \vee \neg x_{5,0} \vee \neg x_{7,1})$, $(y_{1,0,A,\neg x} \vee \neg x_{5,1} \vee \neg x_{7,0})$, $(y_{1,1,A,\neg x} \vee \neg x_{5,1} \vee \neg x_{7,1})$, $(y_{0,0,A,\neg x} \vee \neg x_{7,0} \vee \neg x_{8,0})$, $(y_{0,1,A,\neg x} \vee \neg x_{7,0} \vee \neg x_{8,1})$, $(y_{1,0,A,\neg x} \vee \neg x_{7,1} \vee \neg x_{8,0})$, $(y_{1,1,A,\neg x} \vee \neg x_{7,1} \vee \neg x_{8,1})$
32 дизъюнкта вида $(\neg y_{a,b,e,f} \vee \neg x_{v,a} \vee x_{u,b})$	$(\neg y_{0,0,A,\neg x} \vee \neg x_{0,0} \vee x_{1,0})$, $(\neg y_{0,1,A,\neg x} \vee \neg x_{0,0} \vee x_{1,1})$, $(\neg y_{1,0,A,\neg x} \vee \neg x_{0,1} \vee x_{1,0})$, $(\neg y_{1,1,A,\neg x} \vee \neg x_{0,1} \vee x_{1,1})$, $(\neg y_{0,0,A,x} \vee \neg x_{1,0} \vee x_{2,0})$, $(\neg y_{0,1,A,x} \vee \neg x_{1,0} \vee x_{2,1})$, $(\neg y_{1,0,A,x} \vee \neg x_{1,1} \vee x_{2,0})$, $(\neg y_{1,1,A,x} \vee \neg x_{1,1} \vee x_{2,1})$, $(\neg y_{0,0,A,true} \vee \neg x_{2,0} \vee x_{3,0})$, $(\neg y_{0,1,A,true} \vee \neg x_{2,0} \vee x_{3,1})$, $(\neg y_{1,0,A,true} \vee \neg x_{2,1} \vee x_{3,0})$, $(\neg y_{1,1,A,true} \vee \neg x_{2,1} \vee x_{3,1})$, $(\neg y_{0,0,A,x} \vee \neg x_{0,0} \vee x_{4,0})$, $(\neg y_{0,1,A,x} \vee \neg x_{0,0} \vee x_{4,1})$, $(\neg y_{1,0,A,x} \vee \neg x_{0,1} \vee x_{4,0})$, $(\neg y_{1,1,A,x} \vee \neg x_{0,1} \vee x_{4,1})$, $(\neg y_{0,0,B,true} \vee \neg x_{4,0} \vee x_{5,0})$, $(\neg y_{0,1,B,true} \vee \neg x_{4,0} \vee x_{5,1})$,

	$(\neg y_{1,0,B,true} \vee \neg x_{4,1} \vee x_{5,0})$, $(\neg y_{1,1,B,true} \vee \neg x_{4,1} \vee x_{5,1})$, $(\neg y_{0,0,A,x} \vee \neg x_{5,0} \vee x_{6,0})$, $(\neg y_{0,1,A,x} \vee \neg x_{5,0} \vee x_{6,1})$, $(\neg y_{1,0,A,x} \vee \neg x_{5,1} \vee x_{6,0})$, $(\neg y_{1,1,A,x} \vee \neg x_{5,1} \vee x_{6,1})$, $(\neg y_{0,0,A,\neg x} \vee \neg x_{5,0} \vee x_{7,0})$, $(\neg y_{0,1,A,\neg x} \vee \neg x_{5,0} \vee x_{7,1})$, $(\neg y_{1,0,A,\neg x} \vee \neg x_{5,1} \vee x_{7,0})$, $(\neg y_{1,1,A,\neg x} \vee \neg x_{5,1} \vee x_{7,1})$, $(\neg y_{0,0,A,\neg x} \vee \neg x_{7,0} \vee x_{8,0})$, $(\neg y_{0,1,A,\neg x} \vee \neg x_{7,0} \vee x_{8,1})$, $(\neg y_{1,0,A,\neg x} \vee \neg x_{7,1} \vee x_{8,0})$, $(\neg y_{1,1,A,\neg x} \vee \neg x_{7,1} \vee x_{8,1})$

Таким образом, КНФ-формула для данной задачи составлена из 99 описанных дизъюнктов. В приложении А приведена запись данной формулы в формате *DIMACS*.

1.3.2. Пример работы алгоритма для графа совместимости дерева сценариев управления часами с будильником

В разделе 2.2.2 научно-технического отчета [5] о выполнении предыдущего этапа контракта приводятся описание модели часов с будильником, система сценариев работы данных часов, описание дерева сценариев, построенного по заданной системе сценариев работы, описание графа совместимости, построенного по данному дереву сценариев. Построенный граф приведен на рис. 3.

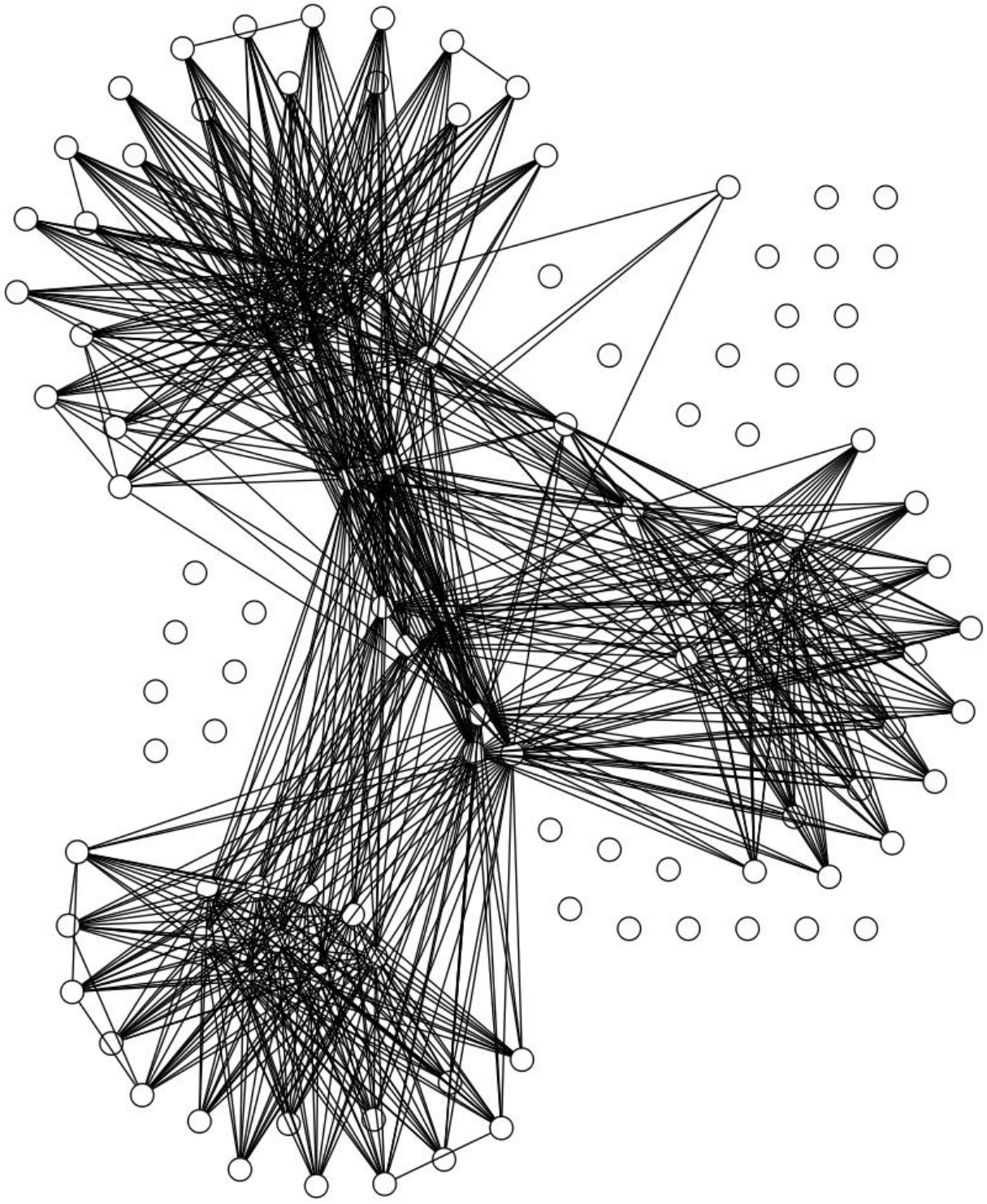

Рис. 3. Граф совместимости для дерева сценариев работы часов с будильником

Дерево сценариев состоит из 116 вершин, а искомый автомат должен содержать три состояния. Поэтому число переменных вида $x_{v,i}$ равно $116 \cdot 3 = 348$. В дереве сценариев встречается семь различных комбинаций входное событие – охранное условие. Поэтому число переменных вида $y_{a,b,e,f}$ равно $7 \cdot 3 \cdot 3 = 63$. Таким образом, в КНФ-формуле используется 348 + 63 = 411 переменных.

Построенная КНФ-формула состоит из 4977 дизъюнктов. Поэтому ограничимся подсчетом числа дизъюнктов каждого вида. Формула состоит из:

- одного дизъюнкта, ставящего в соответствие корню дерева цвет 0;
- 116 дизъюнктов вида $(x_{v,0} \vee \ldots \vee x_{v,C\text{-}1})$;
- 349 дизъюнктов вида $(\neg x_{v,i} \vee \neg x_{v,j})$;
- 2379 дизъюнктов вида $(\neg x_{v,i} \vee \neg x_{u,i})$, что равно произведению числа ребер графа, приведенного на рис. 3, на C, равному трем;
- 63 дизъюнкта вида $(\neg y_{a,b,e,f} \vee \neg y_{a,d,e,f})$;
- 1035 дизъюнктов вида $(y_{a,b,e,f} \vee \neg x_{v,a} \vee \neg x_{u,b})$;
- 1035 дизъюнктов вида $(\neg y_{a,b,e,f} \vee \neg x_{v,a} \vee x_{u,b})$.

Выводы по главе 1

1. Предложен алгоритм построения булевой формулы по графу совместимости. Разработка алгоритма осуществлялась на основе методов теории графов, теории автоматов, булевой логики. Описаны входные и выходные данные алгоритма.
2. Приведен пример работы алгоритма для заданного дерева трех сценариев работы программы и соответствующему ему графу совместимости. Приведена построенная в результате работы алгоритма КНФ-формула, которая использует 34 логические переменные и состоит из 99 дизъюнктов.
3. Приведен пример работы алгоритма для графа совместимости дерева сценариев управления часами с будильником. Построенная в результате работы алгоритма КНФ-формула использует 411 логических переменных и состоит из 4977 дизъюнктов.

2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ БУЛЕВОЙ ФОРМУЛЫ ПО ГРАФУ СОВМЕСТИМОСТИ

Программная реализация разработанного алгоритма построения булевой формулы по графу совместимости выполнялась на языке программирования *Java*. Реализация алгоритма содержится в классе `DimacsCnfBuilder`, содержащий статический метод `getCnf`. Данный метод возвращает строковое представление КНФ-формулы в формате *DIMACS* и принимает как параметр дерево сценариев `tree` типа `ScenariosTree`, описанного в [5], и число `k` состояний искомого автомата.

```
public class DimacsCnfBuilder {
  public static String getCnf(ScenariosTree tree, int k) {
    ...
  }
}
```

Опишем программную реализацию этапов создания переменных, создания дизъюнктов и представления дизъюнктов в формате *DIMACS*. Полная программная реализация класса `DimacsCnfBuilder` содержится в приложении Б.

2.1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ЭТАПА СОЗДАНИЯ ПЕРЕМЕННЫХ

Переменные хранятся в таблице `vars` типа `Map<String, Integer>`, которая ставит в соответствие строковому представлению переменной ее номер.

```
Map<String, Integer> vars = new HashMap<String, Integer>();
```

Добавим в таблицу переменные $x_{v,i}$ для каждой вершины дерева сценариев `node` и цвета вершины `color`. Каждая добавленная переменная означает, верно ли, что вершина `node` имеет цвет `color`.

```
for (Node node : tree.getNodes()) {
  for (int color = 0; color < k; color++) {
    vars.put("x_" + node.getNumber() + "_" + color, vars.size() + 1);
  }
}
```

После этого добавим в таблицу переменные $y_{a,b,e,f}$, рассмотрев все ребра дерева сценариев `t` типа `Transition`. Если в таблице нет переменных, соответствующих комбинации входного события `t.getEvent()` и охранного условия `t.getExpr()`, добавим такие переменные для каждой пары цветов `nodeColor` и `childColor`. Каждая добавленная переменная означает: «Верно ли, что в результирующем автомате существует переход из состояния `nodeColor` в состояние `childColor`, помеченный событием `t.getEvent()` и формулой `t.getExpr()`?».

```
for (Node node : tree.getNodes()) {
  for (Transition t : node.getTransitions()) {
    String key = "y_" + t.getEvent() + "_" +
            t.getExpr().toString() + "_0_0";
    if (!vars.containsKey(key)) {
      for (int nodeColor = 0; nodeColor < k; nodeColor++) {
        for (int childColor = 0; childColor < k; childColor++) {
          String s = "y_" + t.getEvent() + "_" +
            t.getExpr() + "_" + nodeColor + "_" + childColor;
          vars.put(s, vars.size() + 1);
        }
```

```
        }
      }
    }
  }
```

2.2. Программная реализация этапа создания дизъюнктов

Дизъюнкты хранятся в списке `clauses` типа `List<String>`. Каждый дизъюнкт хранится в виде строки, в которой через пробел записаны номера переменных, используемых в данном дизъюнкте, со знаком, соответствующим требуемому значению данной переменной.

```
List<String> clauses = new ArrayList<String>();
```

Добавим дизъюнкт, ставящий в соответствие корню дерева цвет 0 – требующий истинность переменной $x_{0,0}$.

```
String initClause = vars.get("x_0_0") + "";
clauses.add(initClause);
```

Добавим дизъюнкты вида $(x_{v,0} \vee \ldots \vee x_{v,C-1})$ для каждой вершины дерева сценариев `node` и цвета `color`. Каждый из дизъюнктов соответствует тому, что вершина `node` обязательно покрашена хотя бы в один из цветов `color`.

```
for (Node node : tree.getNodes()) {
  String clause = "";
  for (int color = 0; color < k; color++) {
    clause += vars.get("x_" + node.getNumber() + "_" + color) + " ";
  }
  clauses.add(clause);
```

```
}
```

Затем добавим дизъюнкты вида ($\neg x_{v,i} \vee \neg x_{v,j}$) для каждой вершины дерева сценариев `node` и пары цветов `c1`, `c2`, такой, что `c1` меньше `c2`. Каждый из дизъюнктов соответствует тому, что вершина `node` не покрашена одновременно в цвет `c1` и `c2`.

```
for (Node node : tree.getNodes()) {
  for (int c1 = 0; c1 < k; c1++) {
    for (int c2 = 0; c2 < c1; c2++) {
      int v1 = vars.get("x_" + node.getNumber() + "_" + c1);
      int v2 = vars.get("x_" + node.getNumber() + "_" + c2);
      clauses.add(-v1 + " " + -v2);
    }
  }
}
```

Добавим дизъюнкты вида ($\neg x_{v,i} \vee \neg x_{u,i}$) для каждой пары вершин `node` и `other`, соединенных ребром в графе совместимости, и цвета `color`. Каждый из дизъюнктов соответствует тому, что несовместимые вершины `node` и `other` дерева сценариев не покрашены одновременно в цвет `color`. Несовместимые вершины дерева сценариев хранятся в переменной `adjacent` типа `Map<Node, Set<Node>>`. Для определения несовместимых вершин используется метод `getAdjacent` класса `AdjacentCalculator`, который описан в разделе 5.1 отчета [5].

```
Map<Node, Set<Node>> adjacent = AdjacentCalculator.getAdjacent(tree);
for (Node node : tree.getNodes()) {
  for (Node other : adjacent.get(node)) {
```

```
      if (other.getNumber() < node.getNumber()) {
        for (int color = 0; color < k; color++) {
          int v1 = vars.get("x_" + node.getNumber() + "_" + color);
          int v2 = vars.get("x_" + other.getNumber() + "_" + color);
          clauses.add("-" + v1 + " -" + v2);
        }
      }
    }
  }
```

Рассмотрев все ребра дерева сценариев `t`, добавим дизъюнкты вида ($\neg y_{a,b,e,f} \vee \neg y_{a,d,e,f}$). Если дизъюнкты, соответствующие комбинации входного события `t.getEvent()` и охранного условия `t.getExpr()`, еще не добавлены, то добавим их для каждой тройки цветов (`parentColor`, `c1`, `c2`), такой что `c1` больше `c2`. Для данной проверки поддерживается множество `was` типа `Set<String>`. Каждый из добавленных дизъюнктов соответствует тому, что в искомом автомате не должно встречаться больше одного перехода, помеченного входным событием `t.getEvent()` и охранным условием `t.getExpr()`, из состояния `parentColor`:

```
  Set<String> was = new HashSet<String>();
  for (Node node : tree.getNodes()) {
    for (Transition t : node.getTransitions()) {
      String key = t.getEvent() + "_" + t.getExpr();
      if (!was.contains(key)) {
        was.add(key);
        for (int parentColor = 0; parentColor < k; parentColor++) {
          for (int c1 = 0; c1 < k; c1++) {
            for (int c2 = 0; c2 < c1; c2++) {
              int v1 = vars.get("y_" + key + "_" +
                parentColor + "_" + c1);
```

```
                int v2 = vars.get("y_" + key + "_" +
                  parentColor + "_" + c2);
                clauses.add(-v1 + " " + -v2);
              }
            }
          }
        }
      }
```

Наконец, добавим дизъюнкты вида $(y_{a,b,e,f} \vee \neg x_{v,a} \vee \neg x_{u,b})$ и $(\neg y_{a,b,e,f} \vee \neg x_{v,a} \vee x_{u,b})$ для каждого перехода дерева сценариев `t`, соединяющего вершины `node` и `t.getDst()`, и цветов `nodeColor` и `childColor`:

```
    for (Node node : tree.getNodes()) {
      for (Transition t : node.getTransitions()) {
        for (int nodeColor = 0; nodeColor < k; nodeColor++) {
          for (int childColor = 0; childColor < k; childColor++) {
            int nodeVar = vars.get("x_" + node.getNumber() +
                "_" + nodeColor);
            int childVar = vars.get("x_" +
                t.getDst().getNumber() + "_" + childColor);
            int relationVar = vars.get("y_" + t.getEvent() +
                "_" + t.getExpr() + "_" + nodeColor + "_" +
                childColor);

            clauses.add(relationVar + " " + -nodeVar + " " + -childVar);
            clauses.add(-relationVar + " " + -nodeVar + " " + childVar);
          }
        }
      }
```

```
}
```

Таким образом, был составлен список `clauses` строковых представлений дизъюнктов искомой КНФ-формулы.

2.3. Программная реализация этапа представления дизъюнктов в формате *DIMACS*

Построим искомую КНФ-формулу в формате *DIMACS*. Для производительного осуществления операции конкатенации воспользуемся экземпляром `sb` класса `StringBuilder`, включенного в стандартную библиотеку языка *Java*. Сначала, следуя формату, добавим строку с числом переменных и дизъюнктов КНФ-формулы. Затем добавим составленные дизъюнкты, заканчивая каждый из дизъюнктов символом «0». Результатом работы описываемой функции `getCnf` является строковое представление `sb – sb.toString()`.

```
StringBuilder sb = new StringBuilder();
sb.append("p cnf " + vars.size() + " " + clauses.size() + "\n");
for (String s : clauses) {
  sb.append(s + " 0\n");
}
return sb.toString();
```

Выводы по главе 2

1. Реализован алгоритм построения КНФ-формулы по графу совместимости. Программная реализация выполнялась на языке программирования *Java*. Реализация данного алгоритма содержится в статическом методе `getCnf` класса `DimacsCnfBuilder`.
2. Приведены описания программной реализации этапов создания переменных, создания дизъюнктов и представления дизъюнктов в формате *DIMACS*.

3. РАЗРАБОТКА АЛГОРИТМА ПОСТРОЕНИЯ АВТОМАТА ПО НАЙДЕННОЙ ВЫПОЛНЯЮЩЕЙ ПОДСТАНОВКЕ

Разработка алгоритма построения автомата по найденной выполняющей подстановке будет осуществляться на основе методов теории графов и теории автоматов.

3.1. Входные и выходные данные алгоритма

Входными данными алгоритма построения автомата являются выходные данные сторонней программы, решающую задачу о выполнимости булевой формулы. Приведем пример входных данных алгоритма:

```
c Outputting solution to console
c Reading file 'sample-cnf'
c Memory used : 0.00 MB
c CPU time : 0.01 s
s SATISFIABLE
v 1 2 -3 4 -5 0
```

Данные входные данные алгоритма являются примером выходных данных стороннего программного средства для КНФ-формулы, приведенной в разделе 1.1. Строки, начинающиеся с символа «`c`», — комментарии, которые не влияют на работу алгоритма. Строка, начинающаяся с символа «`s`», содержит вывод стороннего средства о том, является ли удовлетворимой заданная КНФ-формула, а строка, начинающаяся с символа «`v`», содержит выполняющую подстановку формулы, если таковая существует.

Выходными данными алгоритма является управляющий автомат типа `Automaton`.

3.2. Алгоритм построения автомата по найденной выполняющей подстановке

Для того чтобы найти выполняющий набор для построенной КНФ-формулы, воспользуемся сторонней программой, решающей задачу *SAT*.

В результате анализа результатов соревнования *SAT-Race* 2010 [15] был произведен выбор программы *Cryptominisat* [10], которая будет использована для получения предварительных результатов.

Подадим на вход выбранной программе построенную КНФ-формулу, записанную в формате *DIMACS*. Если программа не обнаружила выполняющий набор значений переменных, то будем считать, что по данному набору сценариев невозможно построить управляющий автомат с заданным числом состояний.

Если же программа обнаружила выполняющий набор, то построим искомый автомат. Для этого на основании полученных значений переменных $x_{v,i}$ определим цвет каждой вершины дерева сценариев. На рис. 4 представлен пример раскраски дерева сценариев, приведенного на рис. 1.

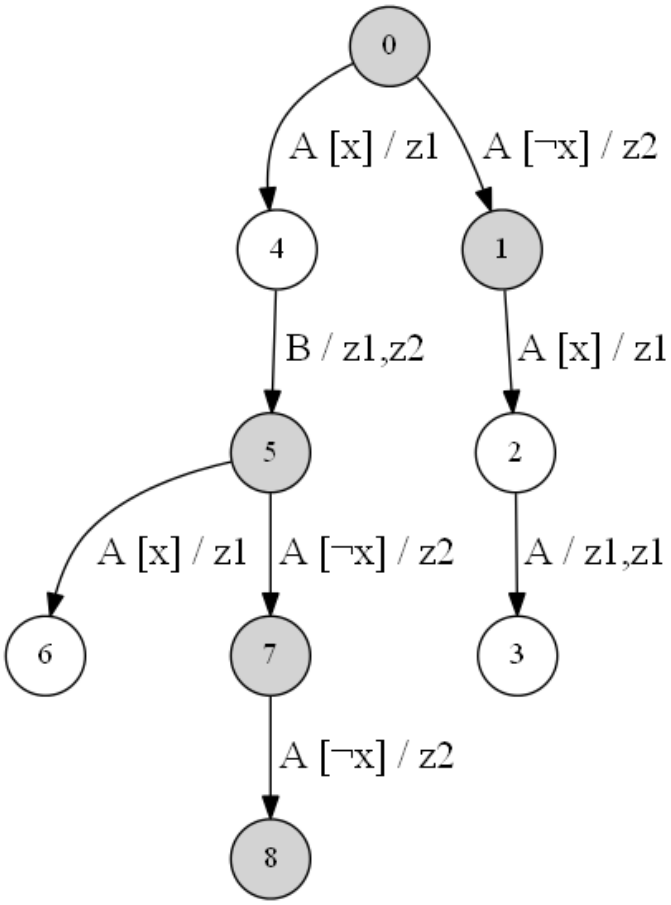

Рис. 4. Пример раскраски дерева сценариев в два цвета

После этого объединим все вершины одного цвета в одно состояние автомата, а начальным состоянием положим состояние, соответствующее цвету корня дерева сценариев. Множество исходящих из состояния переходов построим из объединения множеств ребер, исходящих из вершин заданного цвета.

Например, после объединения вершин дерева, приведенного на рис. 4, получим автомат, приведенный на рис. 5. Отметим, что не исключено существование нескольких автоматов, удовлетворяющих сценариям множества *Sc*.

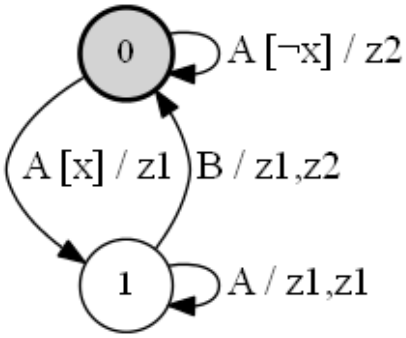

Рис. 5. Автомат, полученный после объединения вершин дерева

3.3. Примеры работы предложенного алгоритма

В настоящем разделе приведены примеры работы предложенного алгоритма построения управляющего автомата по найденной выполняющей подстановке. Сначала будет рассмотрена выполняющая подстановка КНФ-формулы, построенная для небольшого набора сценариев. Затем будет рассмотрена выполняющая подстановка, построенная для набора сценариев управления часами с будильником.

3.3.1. Пример работы алгоритма для небольшого набора сценариев

В разделе 1.3.1 была построена КНФ-формула для небольшого набора сценариев. В приложении А приведена данная формула в формате *DIMACS*. Затем, данная формула была подана на вход стороннему программному средству *cryptominisat*, выход которого приведен в приложении В. После этого по найденной выполняющей подстановке была выполнена раскраска дерева сценариев, приведенного на рис. 4, и построен автомат, приведенный на рис. 5.

Однако данная выполняющая подстановка не является единственной выполняющей подстановкой для построенной КНФ-формулы. На рис. 6 приведен второй вариант раскраски дерева сценариев, который отличается от варианта, приведенного на рис. 4, только цветом вершины 3. Данному варианту раскраски соответствует автомат, приведенный на рис. 7.

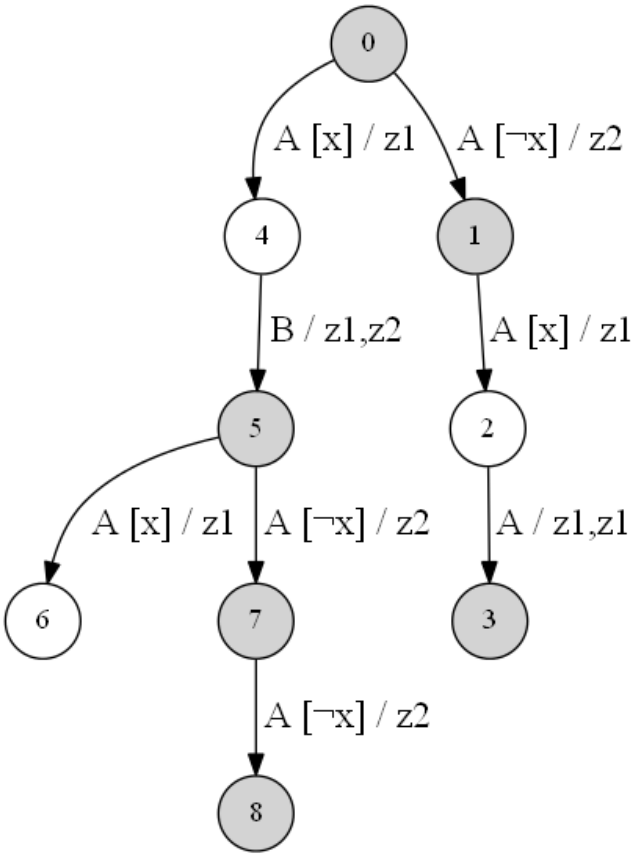

Рис. 6. Второй вариант раскраски дерева сценариев

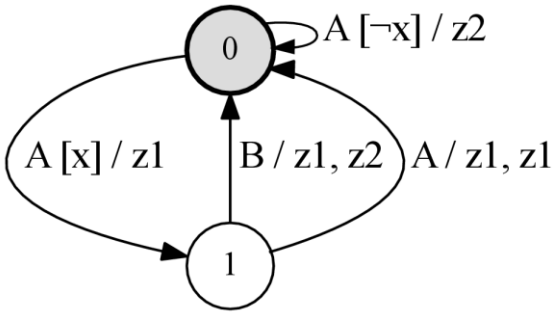

Рис. 7. Второй автомат, удовлетворяющий заданным сценариям

Иных выполняющих подстановок для рассматриваемой КНФ-формулы не существует, и, как следствие, не существует отличных от приведенных на рис. 5,рис. 7 автоматов с двумя состояниями, удовлетворяющих заданным сценариям работы.

3.3.2. Пример работы алгоритма для сценариев управления часами с будильником

Приведем пример работы алгоритма построения управляющего автомата по найденной выполняющей подстановке для задачи построения автомата управления часами с будильником. В данной задаче необходимо построить управляющий автомат с тремя состояниями.

В разделе 1.3.2 была построена КНФ-формула для данной задачи. После этого для нахождения выполняющей подстановки было использовано программное средство *cryptominisat*. Выполняющая подстановка была найдена, после чего получена раскраска в три цвета дерева сценариев, приведенного в [5]. Так как дерево сценариев достаточно велико, разобьем его на части, которые приведены рис. 8 – рис. 11 (корни приведенных частей соответствуют корню дерева).

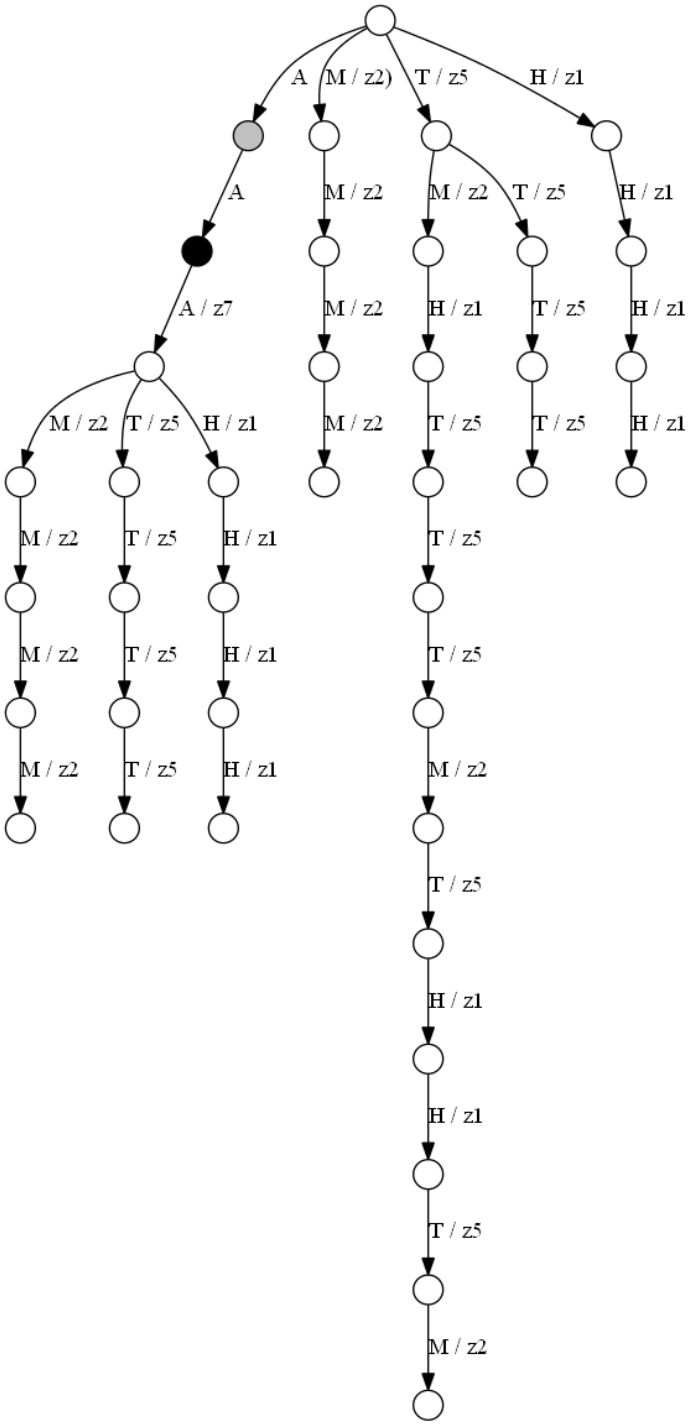

Рис. 8. Первая часть раскрашенного дерева сценариев управления часами с будильником

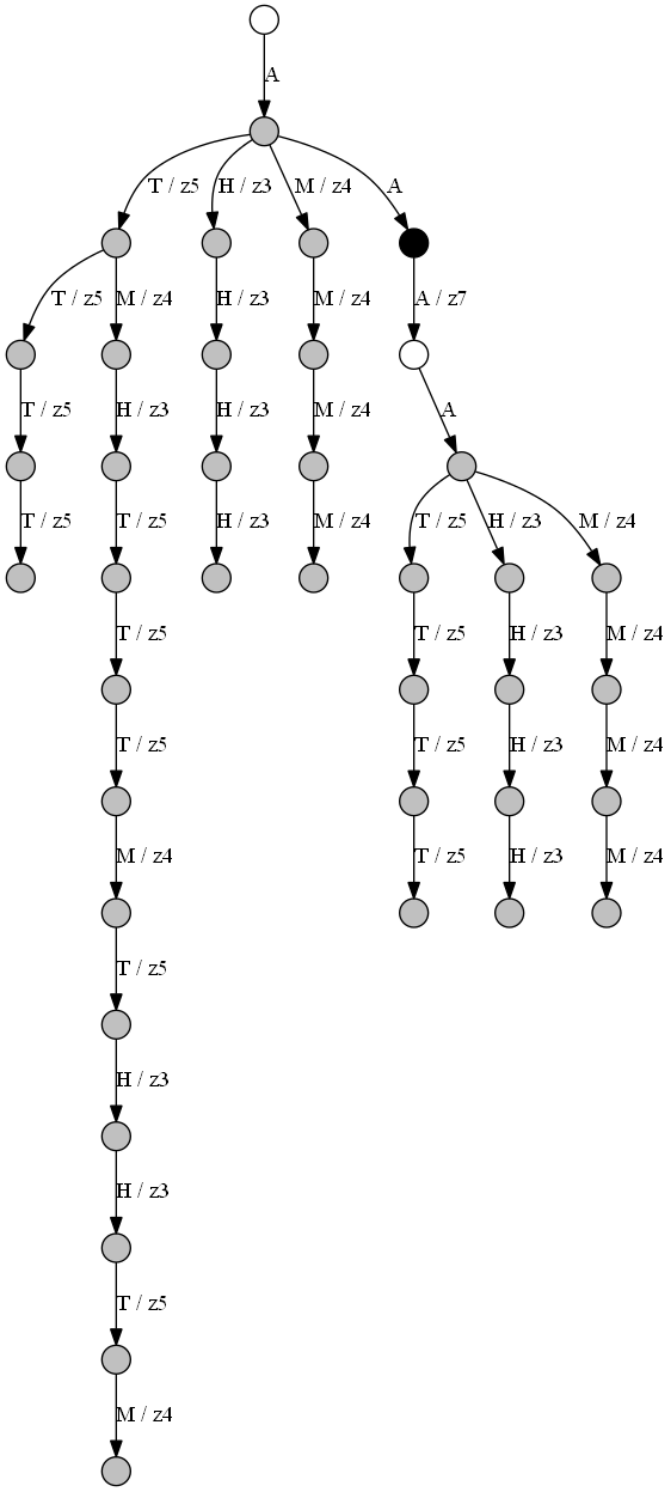

Рис. 9. Вторая часть раскрашенного дерева сценариев управления часами с будильником

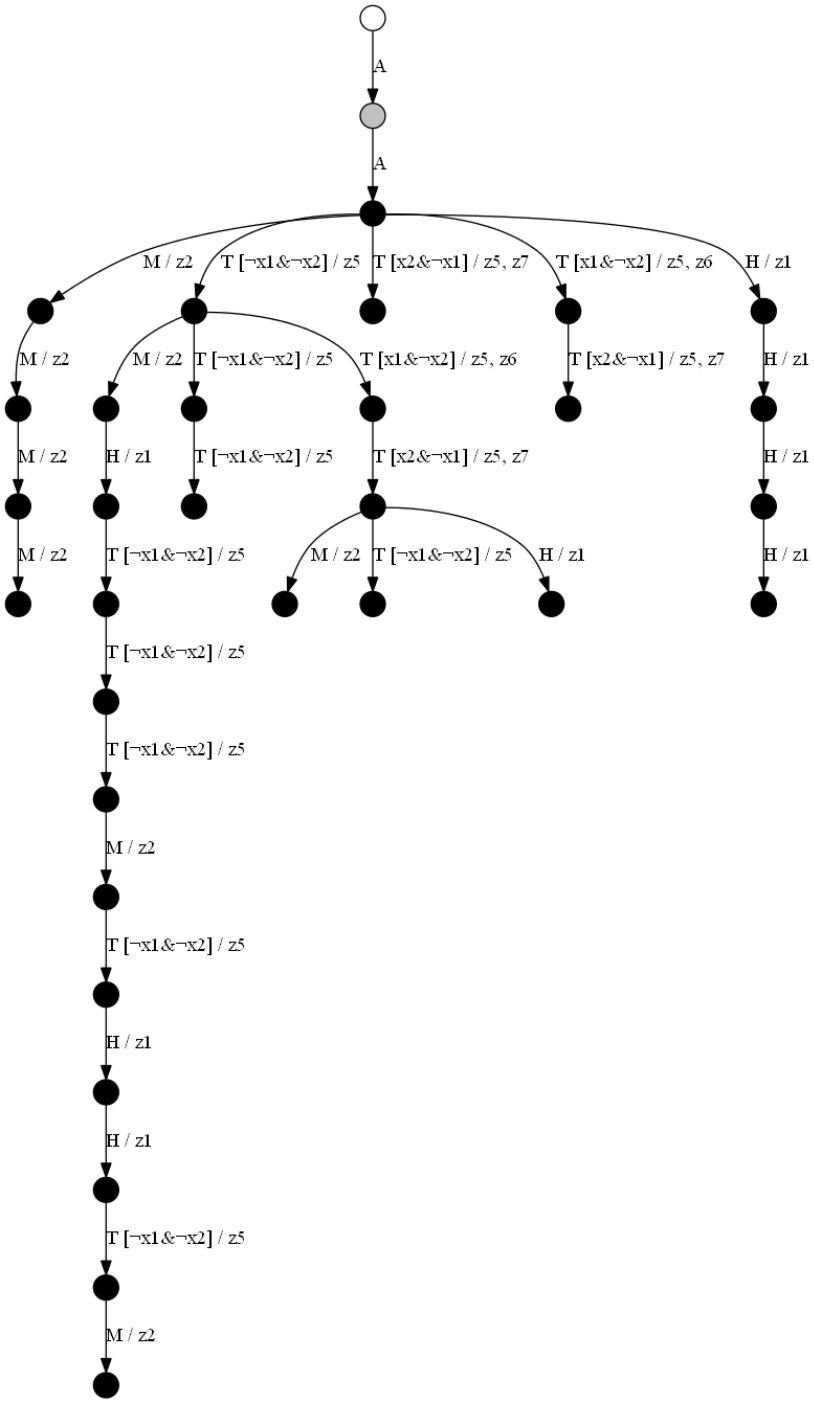

Рис. 10. Третья часть раскрашенного дерева сценариев управления часами с будильником

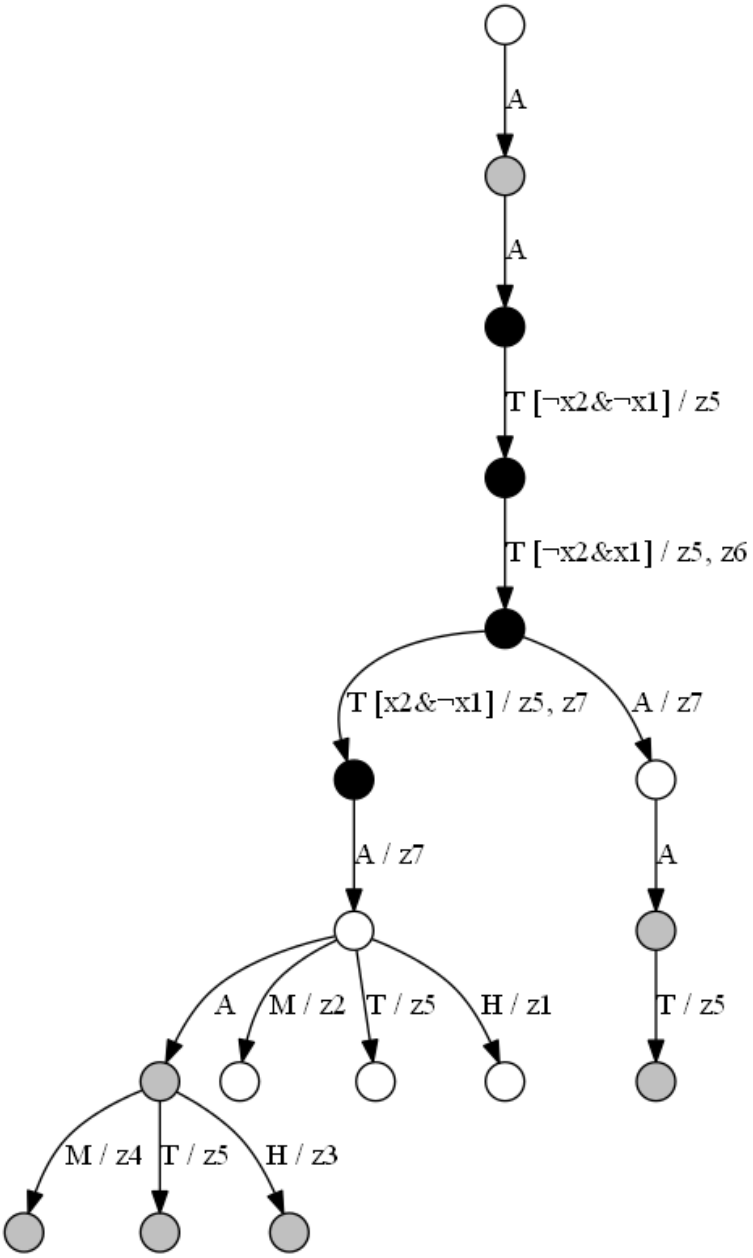

Рис. 11. Четвертая часть раскрашенного дерева сценариев управления часами с будильником

Затем, объединив вершины одного цвета, построим искомый управляющий автомат, приведенный на рис. 12.

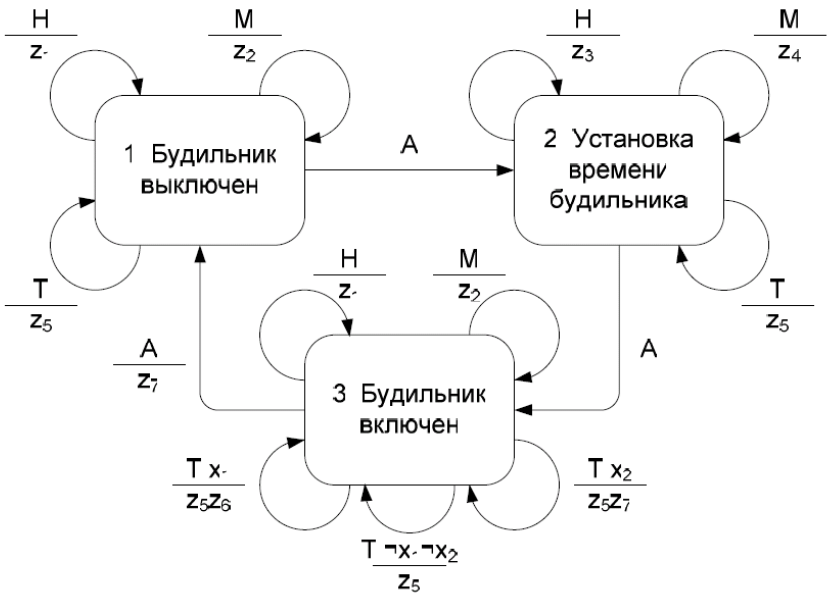

Рис. 12. Автомат управления часами с будильником

Белым вершинам дерева сценариев соответствует состояние автомата «1 Будильник выключен», серым – «2 Установка времени будильника», черным – «3 Будильник включен».

Выводы по главе 3

1. Разработан алгоритм построения управляющего автомата по найденной выполняющей подстановке. Разработка алгоритма осуществлялась на основе методов теории графов и теории автоматов.
2. Приведен пример работы алгоритма для задачи построения управляющего автомата, удовлетворяющего небольшому набору сценариев работы. По найденной выполняющей подстановке произведена раскраска дерева сценариев в два цвета и построение автомата с двумя состояниями. Приведено два решения данной задачи.
3. Приведен пример работы алгоритма для задачи построения автомата управления часами с будильником. По найденной выполняющей подстановке произведена раскраска дерева сценариев, приведенного на рис. 8 – рис. 11, в три цвета. После этого был построен искомый автомат с тремя состояниями, приведенный на рис. 12.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ АВТОМАТА ПО ВЫПОЛНЯЮЩЕЙ ПОДСТАНОВКЕ

Программная реализация разработанного алгоритма построения автомата по выполняющей подстановке выполнялась на языке программирования *Java*. Реализация алгоритма содержится в классе `CryptominisatAutomatonBuilder` (напомним, что для предварительных исследований будет использовано программное средство *cryptominisat*).

Данный класс содержит статический метод `build`, который возвращает управляющий автомат типа `Automaton` и принимает следующие параметры:

- дерево сценариев `tree` типа `ScenariosTree`;
- число `k` состояний искомого автомата;
- экземпляр `cnfPrintWriter` класса `PrintWriter` из стандартной библиотеки *Java*, в который будет произведена запись построенной КНФ-формулы в формате DIMACS;
- экземпляр `solverPrintWriter` класса `PrintWriter`, в который будет произведена запись вывода стороннего программного средства (в настоящей реализации – *cryptominisat*).

```
public class CryptominisatAutomatonBuilder {
  public static Automaton build(ScenariosTree tree,
      int k, PrintWriter cnfPrintWriter, PrintWriter solverPrintWriter)
  throws IOException {
    ...
  }
}
```

Опишем программную реализацию этапов нахождения выполняющей подстановки и построения искомого управляющего автомата по выполняющей

подстановке. Полная программная реализация класса `CryptominisatAutomatonBuilder` содержится в приложении Г.

4.1. Программная реализация этапа нахождения выполняющей подстановки

Вызвав описанный в главе [5] метод `getCnf` класса `DimacsCnfBuilder`, получим строковое представление КНФ-формулы для заданного дерева сценариев `tree` и числа состояний искомого автомата `k`. Выведем, если это необходимо, формулу в заданный `cnfPrintWriter`.

```
String cnf = DimacsCnfBuilder.getCnf(tree, k);
if (cnfPrintWriter != null) {
  cnfPrintWriter.println(cnf);
  cnfPrintWriter.flush();
}
```

Выведем полученную КНФ-формулу во временный файл, который расположим по адресу *«tmp.cnf»*. После этого запустим программное средство *cryptominisat*, которому подадим на вход данный файл. Запуск производится с параметром «*--threads=4*», что соответствует разрешению использовать четыре потока в процессе решения задачи о выполнимости булевой формулы.

```
File tmpFile = new File("tmp.cnf");
PrintWriter tmpPW = new PrintWriter(tmpFile);
tmpPW.print(cnf);
tmpPW.close();
Process p = Runtime.getRuntime().exec("cryptominisat
            --threads=4 tmp.cnf");
```

Обработаем результаты, выведенные программным средством в поток `p.getInputStream()`. Рассмотрев все выведенные строки, найдем строку `ansLine`, начинающуюся с символа «v». Если передан параметр `solverPrintWriter`, то выведем в соответствующий поток данные строки.

```
String ansLine = null;
BufferedReader input = new BufferedReader(new
            InputStreamReader(p.getInputStream()));
String line;
while ((line = input.readLine()) != null) {
  if (solverPrintWriter != null) {
    solverPrintWriter.println(line);
  }
  if (line.charAt(0) == 'v') {
    ansLine = line;
  }
}
input.close();
```

После рассмотрения строк удалим временный файл с КНФ-формулой *«tmp.cnf»* и выйдем из рассматриваемого метода `build`, вернув `null`, если искомая строка `ansLine` не была найдена.

```
tmpFile.delete();
if (ansLine == null) {
  return null;
}
```

4.2. Программная реализация этапа построения управляющего автомата по выполняющей подстановке

Составим массив `nodesColors`, хранящий цвета раскрашенного дерева сценариев. Для этого рассмотрим значения первых `tree.nodesCount() * k` переменных, содержащихся в найденной строке `ansLine`. Данные значения соответствуют значениям переменных $x_{v,i}$. Цвет каждой вершины с номером `nodeNum` равен такому значению `color`, что значение переменной с номером `nodeNum * k + color` истинно.

```
int[] nodesColors = new int[tree.nodesCount()];
String[] sp = ansLine.split(" ");
for (int nodeNum = 0; nodeNum < tree.nodesCount(); nodeNum++) {
  for (int color = 0; color < k; color++) {
    if (sp[1 + nodeNum * k + color].charAt(0) != '-') {
      nodesColors[nodeNum] = color;
    }
  }
}
```

Построим на основании цветов вершин дерева сценариев автомат `ans`. Для этого объединим все вершины одного цвета `color` в одно состояние автомата `state`, а начальным состоянием положим состояние, соответствующее цвету корня дерева сценариев – цвету 0. Множество исходящих из состояния переходов построим из объединения множеств ребер, исходящих из вершин заданного цвета. Наконец, вернем из рассматриваемого метода `build` класса `CryptominisatAutomatonBuilder` построенный автомат `ans`.

```
Automaton ans = new Automaton(k);
for (int i = 0; i < tree.nodesCount(); i++) {
  int color = nodesColors[i];
  Node state = ans.getState(color);
  for (Transition t : tree.getNodes().get(i).getTransitions()) {
    if (!state.hasTransition(t.getEvent(), t.getExpr())) {
      int childColor = nodesColors[t.getDst().getNumber()];
      state.addTransition(t.getEvent(), t.getExpr(),
          t.getActions(), ans.getState(childColor));
    }
  }
}
return ans;
```

Вывод по главе 4

1. Реализован алгоритм построения автомата по выполняющей подстановке. Программная реализация выполнялась на языке программирования *Java*. Реализация данного алгоритма содержится в статическом методе `build` класса `CryptominisatAutomatonBuilder`.
2. Приведены описания программной реализации этапов нахождения выполняющей подстановки и построения искомого управляющего автомата по выполняющей подстановке.

5. ПУБЛИКАЦИИ РЕЗУЛЬТАТОВ НИР

По результатам НИР в журнале «Научно-технический вестник информационных технологий, механики и оптики» 2012. № 1(77) опубликована статья «Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы». Текст статьи приведен в приложении Д.

Кроме этого, по результатам НИР сделаны доклады на конференциях и опубликованы тезисы докладов:

- Ульянцев В.И. Применение методов решения задачи удовлетворения ограничениям для построения управляющих конечных автоматов по сценариям работы / Сборник тезисов докладов конгресса молодых ученых, Выпуск 1. Труды молодых ученых. СПб: НИУ ИТМО, 2012. – 246 с.
- Ulyantsev V., Tsarev F. Extended Finite-State Machine Induction using SAT-Solver / Proceedings of the 14th IFAC Symposium «Information Control Problems in Manufacturing (INCOM'12)». IFAC. 2012, pp. 512–517.

В процессе выполнения НИР также отправлена заявка на получение свидетельства о регистрации программы для ЭВМ (копия заявки приведена в приложении Е).

ЗАКЛЮЧЕНИЕ

В результате исследований на третьем этапе работ по контракту были выполнены следующие работы:

- разработан алгоритм построения булевой формулы по графу совместимости;
- выполнена программная реализация алгоритма построения булевой формулы по графу совместимости;
- разработан алгоритм построения автомата по найденной выполняющей подстановке;
- выполнена программная реализация алгоритма построения автомата по выполняющей подстановке.

Разработан и реализован на языке программирования *Java* алгоритм построения булевой формулы по графу совместимости. Разработка алгоритма осуществлялась с помощью методов теории графов, теории автоматов, булевой логики. Приведены примеры работы данного алгоритма, его входные и выходные параметры.

Разработан и реализован на языке программирования *Java* алгоритм построения автомата по найденной выполняющей подстановке. Разработка алгоритма осуществлялась с помощью методов теории графов и теории автоматов. Приведены примеры работы данного алгоритма, его входные и выходные параметры.

Результаты выполненных работ позволяют утверждать, что научно-технический уровень исследований превышает уровень исследований в рассматриваемой области, проводимых в лучших исследовательских центрах мира.

СПИСОК ЛИТЕРАТУРЫ

1. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. М.: МЦНМО, 2002.
2. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ. М.: Вильямс. 2010.
3. *Николенко С. И., Тулупьев А. Л.* Самообучающиеся системы. М.: МЦНМО, 2009.
4. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2009.
5. Разработка метода машинного обучения на основе алгоритмов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов. Научно-технический отчет о выполнении второго этапа Государственного контракта № 16.740.11.0455 от 13 мая 2011 г.
6. *Скиена С.* Алгоритмы. Руководство по разработке. СПб: БХВ-Петербург, 2011.
7. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
8. *Biere A.* Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical Report 10/1. 2010. FMV Reports Series, Institute for Formal Models and Verification. Johannes Kepler University. Altenbergerstr. 69, 4040 Linz, Austria.
9. *Bouchard C.* Boolean Expression Solver. http://sourceforge.net/projects/bool-expr-solve/.
10. *CryptoMiniSat SAT-solver*. http://www.msoos.org/cryptominisat2.
11. *Een N., Sörensson N.* An extensible SAT-solver. Chalmers University of Technology. Sweden, 2003. http://minisat.se/downloads/MiniSat.pdf.
12. *Gold E. M.* Complexity of automaton identification from given data // Information and Control. 1978, № 37, pp. 302−320.
13. *Graphviz – Graph Visualization Software*. http://www.graphviz.org/.

14. *Heule M., Verwer S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications 10th International Colloquium (ICGI 2010). 2010. Lecture Notes in Computer Science. V. 6339, pp. 66–79.
15. *Presentation of SAT-Race* results at the SAT'10 conference. http://baldur.iti.uka.de/sat-race-2010/downloads/SAT-Race-2010-Presentation.pdf.
16. *Rabiner L.R.* A tutorial on hidden Markov models and selected applications in speech recognition // *Proceedings of the IEEE*. Vol.77. 1989. № 2, pp.257–286.
17. *Satisfiability suggested format DIMACS.* http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf.
18. *zChaff SAT-solver*. http://www.princeton.edu/~chaff/zchaff.html.

ПРИЛОЖЕНИЕ А. КНФ-ФОРМУЛА В ФОРМАТЕ DIMACS

```
c CNF for scenarios tree with 9 nodes, colored in 2 colors
p cnf 34 99
1 0
1 2 0
3 4 0
5 6 0
7 8 0
9 10 0
11 12 0
13 14 0
15 16 0
17 18 0
-2 -1 0
-4 -3 0
-6 -5 0
-8 -7 0
-10 -9 0
-12 -11 0
-14 -13 0
-16 -15 0
-18 -17 0
-15 -9 0
-16 -10 0
-15 -13 0
-16 -14 0
-15 -5 0
-16 -6 0
-15 -1 0
-16 -2 0
-20 -19 0
-22 -21 0
-24 -23 0
-26 -25 0
-28 -27 0
-30 -29 0
-32 -31 0
-34 -33 0
19 -1 -3 0
-19 -1 3 0
20 -1 -4 0
-20 -1 4 0
21 -2 -3 0
-21 -2 3 0
22 -2 -4 0
```

```
-22 -2 4 0
23 -1 -13 0
-23 -1 13 0
24 -1 -14 0
-24 -1 14 0
25 -2 -13 0
-25 -2 13 0
26 -2 -14 0
-26 -2 14 0
27 -3 -5 0
-27 -3 5 0
28 -3 -6 0
-28 -3 6 0
29 -4 -5 0
-29 -4 5 0
30 -4 -6 0
-30 -4 6 0
19 -5 -7 0
-19 -5 7 0
20 -5 -8 0
-20 -5 8 0
21 -6 -7 0
-21 -6 7 0
22 -6 -8 0
-22 -6 8 0
23 -5 -9 0
-23 -5 9 0
24 -5 -10 0
-24 -5 10 0
25 -6 -9 0
-25 -6 9 0
26 -6 -10 0
-26 -6 10 0
23 -9 -11 0
-23 -9 11 0
24 -9 -12 0
-24 -9 12 0
25 -10 -11 0
-25 -10 11 0
26 -10 -12 0
-26 -10 12 0
19 -13 -15 0
-19 -13 15 0
20 -13 -16 0
-20 -13 16 0
21 -14 -15 0
```

```
-21 -14 15 0
22 -14 -16 0
-22 -14 16 0
31 -15 -17 0
-31 -15 17 0
32 -15 -18 0
-32 -15 18 0
33 -16 -17 0
-33 -16 17 0
34 -16 -18 0
-34 -16 18 0
```

ПРИЛОЖЕНИЕ Б. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ БУЛЕВОЙ ФОРМУЛЫ ПО ГРАФУ СОВМЕСТИМОСТИ

```
package algorithms;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import structures.Node;
import structures.ScenariosTree;
import structures.Transition;

public class DimacsCnfBuilder {

  public static String getCnf(ScenariosTree tree, int k) {
    Map<String, Integer> vars = new HashMap<String, Integer>();
    for (Node node : tree.getNodes()) {
      for (int color = 0; color < k; color++) {
        vars.put("x_" + node.getNumber() + "_" + color,
                vars.size() + 1);
      }
    }

    for (Node node : tree.getNodes()) {
      for (Transition t : node.getTransitions()) {
        String key = "y_" + t.getEvent() + "_" +
                t.getExpr().toString() + "_0_0";
        if (!vars.containsKey(key)) {
          for (int nodeColor = 0; nodeColor < k;
                nodeColor++) {
            for (int childColor = 0; childColor < k;
                childColor++) {
              String s = "y_" + t.getEvent() + "_" +
                t.getExpr() + "_" + nodeColor + "_" +
                childColor;
              vars.put(s, vars.size() + 1);
            }
          }
        }
      }
    }

    List<String> clauses = new ArrayList<String>();
    String initClause = vars.get("x_0_0") + "";
    clauses.add(initClause);
```

```
for (Node node : tree.getNodes()) {
  String clause = "";
  for (int color = 0; color < k; color++) {
    clause += vars.get("x_" + node.getNumber() + "_" +
            color) + " ";
  }
  clauses.add(clause);
}

for (Node node : tree.getNodes()) {
  for (int c1 = 0; c1 < k; c1++) {
    for (int c2 = 0; c2 < c1; c2++) {
      int v1 = vars.get("x_" + node.getNumber() + "_" + c1);
      int v2 = vars.get("x_" + node.getNumber() + "_" + c2);
      clauses.add(-v1 + " " + -v2);
    }
  }
}

Map<Node, Set<Node>> adjacent =
            AdjacentCalculator.getAdjacent(tree);
for (Node node : tree.getNodes()) {
  for (Node other : adjacent.get(node)) {
    if (other.getNumber() < node.getNumber()) {
      for (int color = 0; color < k; color++) {
        int v1 = vars.get("x_" + node.getNumber() + "_" + color);
        int v2 = vars.get("x_" + other.getNumber() + "_" + color);
        clauses.add("-" + v1 + " -" + v2);
      }
    }
  }
}

Set<String> was = new HashSet<String>();
for (Node node : tree.getNodes()) {
  for (Transition t : node.getTransitions()) {
    String key = t.getEvent() + "_" + t.getExpr();
    if (!was.contains(key)) {
      was.add(key);
      for (int parentColor = 0; parentColor < k;
            parentColor++) {
        for (int c1 = 0; c1 < k; c1++) {
          for (int c2 = 0; c2 < c1; c2++) {
            int v1 = vars.get("y_" + key + "_" +
              parentColor + "_" + c1);
            int v2 = vars.get("y_" + key + "_" +
              parentColor + "_" + c2);
            clauses.add(-v1 + " " + -v2);
          }
        }
```

```
            }
          }
        }
      }

      for (Node node : tree.getNodes()) {
        for (Transition t : node.getTransitions()) {
          for (int nodeColor = 0; nodeColor < k; nodeColor++) {
            for (int childColor = 0; childColor < k;
                  childColor++) {
              int nodeVar = vars.get("x_" + node.getNumber() +
                  "_" + nodeColor);
              int childVar = vars.get("x_" +
                  t.getDst().getNumber() + "_" + childColor);
              int realtionVar = vars.get("y_" + t.getEvent() +
                  "_" + t.getExpr() + "_" + nodeColor + "_" +
                  childColor);

              clauses.add(realtionVar + " " + -nodeVar + " " +
                  -childVar);
              clauses.add(-realtionVar + " " + -nodeVar + " "
                  + childVar);
            }
          }
        }
      }

      StringBuilder sb = new StringBuilder();
      sb.append("c CNF for scenarios tree with " +
                  tree.nodesCount() + " nodes, colored in " +
                  k + " colors\n");
      sb.append("p cnf " + vars.size() + " " + clauses.size() + "\n");
      for (String s : clauses) {
        sb.append(s + " 0\n");
      }

      return sb.toString();
    }
}
```

ПРИЛОЖЕНИЕ В. ПРИМЕР ВЫХОДНЫХ ДАННЫХ ПРИЛОЖЕНИЯ CRYPTOMINISAT.EXE

```
c Outputting solution to console
c This is CryptoMiniSat Jan 20 2011
c compiled with non-gcc compiler
c Reading file 'sample-cnf'
c -- header says num vars:             34
c -- header says num clauses:          99
c -- clauses added:       0 learnts,        99 normals,      0 xors
c -- vars added         34
c Parsing time:  0.01 s
c  N st  0    0   34     0    46    0     92   0   no data   no data
c asymm  cl-useful: 0/0/0 lits-rem:0 time: 0.00
c Flit:     0 Blit:      0 bXBeca:    0 bXProp:    3 Bins:      0 BRemL:
0 BRemN:      0 P:  0.0M T:  0.00
c Replacing        3 vars Replaced       34 lits Time:     0.00 s
c bin-w-bin subsume rem            0 bins  time:  0.00 s
c subs with bin:        0  lits-rem:       0  v-fix:    0  time:  0.00 s
c  Subs  w/  non-existent bins:       0 l-rem:       0 v-fix:      0 done:
9 time:  0.00 s
c Removed useless bin:   0  fixed:  0  props:   0.00M  time:  0.00 s
c lits-rem: 0  cl-subs: 0  v-elim: 0  v-fix:    0  time:  0.00 s
c Finding binary XORs  T:     0.00 s  found:       0
c Finding non-binary XORs:     0.00 s (found:   0, avg size: -1.$)
c calculated reachability. Time: 0.00
c Calc default polars -  time:  0.00 s pos: 0 undec: 26 neg:       8
c types(t): F = full restart, N = normal restart
c types(t): S = simplification begin/end, E = solution found
c restart types(rt): st = static, dy = dynamic
c  t rt  Rest     Confl      Vars   NormCls    BinCls   Learnts    ClLits
LtLits
c  B st     0         0         9         0         4         0         8
0   no data   no data
c  E st     1         0         0         0         4         0         8
0   no data   no data
c Verified 0 clauses.
c Verified 0 clauses.
c num threads              : 1
c restarts                 : 1
c dynamic restarts         : 0
c static restarts          : 1
c full restarts            : 0
c total simplify time      : 0.00
c learnts DL2              : 0
```

```
c learnts size 2           : 46
c learnts size 1           : 22          (64.71     % of vars)
c filedLit time            : 0.00        (0.00      % time)
c v-elim SatELite          : 0           (0.00      % vars)
c SatELite time            : 0.00        (0.00      % time)
c v-elim xor               : 0           (0.00      % vars)
c xor elim time            : 0.00        (0.00      % time)
c num binary xor trees     : 1
c binxor trees' crown      : 3           (3.00      leafs/tree)
c bin xor find time        : 0.00
c OTF clause improved      : 0           (-1.#J     clauses/conflict)
c OTF impr. size diff      : 0           (-1.#J      lits/clause)
c OTF cl watch-shrink      : 0           (-1.#J     clauses/conflict)
c OTF cl watch-sh-lit      : 0           (-1.#J      lits/clause)
c tried to recurMin cls    : 0           (-1.#J      % of conflicts)
c updated cache            : 0           (-1.#J      lits/tried recurMin)
c clauses over max glue    : 0           (-1.#J     % of all clauses)
c conflicts                : 0           (0.00      / sec)
c decisions                : 7           (0.00      % random)
c bogo-props               : 164         (10933.33  / sec)
c conflict literals        : 0           (-1.#J     % deleted)
c Memory used              : 0.00        MB
c CPU time                 : 0.01        s
s SATISFIABLE
v 1 -2 -3 4 5 -6 -7 8 9 -10 11 -12 13 -14 -15 16 17 -18 -19 20 -21 22 23
-24 -25 -26 27 -28 29 -30 31 -32 33 -34 0
```

ПРИЛОЖЕНИЕ Г. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ПОСТРОЕНИЯ АВТОМАТА ПО ВЫПОЛНЯЮЩЕЙ ПОДСТАНОВКЕ

```
package algorithms;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

import structures.Automaton;
import structures.Node;
import structures.ScenariosTree;
import structures.Transition;

public class CryptominisatAutomatonBuilder {
  public static Automaton build(ScenariosTree tree, int k)
                throws IOException {
    return build(tree, k, null);
  }

  public static Automaton build(ScenariosTree tree, int k,
                PrintWriter cnfPrintWriter) throws
                IOException {
    return build(tree, k, cnfPrintWriter, null);
  }

  public static Automaton build(ScenariosTree tree, int k,
                PrintWriter cnfPrintWriter, PrintWriter
                solverPrintWriter)
      throws IOException {

    String cnf = DimacsCnfBuilder.getCnf(tree, k);
    if (cnfPrintWriter != null) {
      cnfPrintWriter.println(cnf);
      cnfPrintWriter.flush();
    }

    File tmpFile = new File("tmp.cnf");
    PrintWriter tmpPW = new PrintWriter(tmpFile);
    tmpPW.print(cnf);
    tmpPW.close();

    Process p = Runtime.getRuntime().exec("cryptominisat
                --threads=4 tmp.cnf");

    String ansLine = null;
    BufferedReader input = new BufferedReader(new
```

```
                InputStreamReader(p.getInputStream()));
    String line;
    while ((line = input.readLine()) != null) {
      if (solverPrintWriter != null) {
        solverPrintWriter.println(line);
      }
      if (line.charAt(0) == 'v') {
        ansLine = line;
      }
    }
    input.close();
    tmpFile.delete();

    if (ansLine != null) {
      int[] nodesColors = new int[tree.nodesCount()];
      String[] sp = ansLine.split(" ");
      for (int nodeNum = 0; nodeNum < tree.nodesCount();
                nodeNum++) {
        for (int color = 0; color < k; color++) {
          if (sp[1 + nodeNum * k + color].charAt(0) != '-') {
            nodesColors[nodeNum] = color;
          }
        }
      }

      Automaton ans = new Automaton(k);
      for (int i = 0; i < tree.nodesCount(); i++) {
        int color = nodesColors[i];
        Node state = ans.getState(color);
        for (Transition t :
                tree.getNodes().get(i).getTransitions()) {
          if (!state.hasTransition(t.getEvent(),
                t.getExpr())) {
            int childColor =
                nodesColors[t.getDst().getNumber()];
            state.addTransition(t.getEvent(), t.getExpr(),
                t.getActions(), ans.getState(childColor));
          }
        }
      }
      return ans;
    }

    return null;
  }
}
```

ПРИЛОЖЕНИЕ Д. ТЕКСТ СТАТЬИ

УДК 004.4'242

ПРИМЕНЕНИЕ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ О ВЫПОЛНИМОСТИ БУЛЕВОЙ ФОРМУЛЫ ДЛЯ ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ ПО СЦЕНАРИЯМ РАБОТЫ

В.И. Ульянцев, Ф.Н. Царев

Предлагается метод построения управляющих конечных автоматов по сценариям работы. Метод основан на сведении указанной задачи к задаче о выполнимости булевой формулы. Работоспособность метода проверяется на задаче построения автомата управления часами с будильником. На этой задаче построение соответствующего управляющего автомата корректно, а время работы алгоритма составляет меньше секунды на персональном компьютере.
Ключевые слова: конечные автоматы, машинное обучение, задача о выполнимости булевой формулы.

Введение

В последнее время все в более широком кругу задач начинает применяться автоматное программирование, в рамках которого поведение программ описывается с помощью детерминированных конечных автоматов [1]. Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение автоматов затруднительно. К задачам этого класса относятся, в частности, задачи об «Умном муравье» [2–4], об управлении моделью беспилотного летательного аппарата [5]. Для построения автоматов в задачах такого типа успешно применяются генетические алгоритмы [6], в том числе на основе обучающих примеров [7]. Недостатком генетических алгоритмов является то, что время их работы весьма велико, и его достаточно трудно оценить аналитически.

Целью настоящей работы является разработка метода построения управляющего конечного автомата, лишенного указанных недостатков.

Постановка задачи

Управляющим конечным автоматом будем называть детерминированный конечный автомат, каждый переход которого помечен событием, последовательностью выходных воздействий и охранным условием, представляющим собой логическую формулу от входных переменных. Автомат получает события от так называемых поставщиков событий (в их роли могут выступать внешняя среда, интерфейс пользователя и т.д.) и генерирует выходные воздействия для объекта управления. При поступлении события автомат выполняет переход в соответствии с охранными условиями и значениями входных переменных. При выполнении перехода генерируются выходные воздействия, которыми он помечен, и автомат переходит в соответствующее состояние. Отметим, что состояния такого автомата не делятся на допускающие и недопускающие.

Формальное определение управляющего автомата дано в [1]. Пример управляющего автомата приведен на рис. 1.

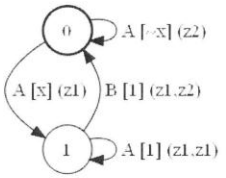

Рис. 1. Пример управляющего автомата

Для данного автомата множество входных событий равно {A, B}, охранные условия зависят от единственной логической входной переменной x, множество выходных воздействий равно {z1, z2}. Далее состояние автомата с номером 0 будем считать начальным.

В настоящей работе в качестве исходных данных для построения управляющего конечного автомата используется множество сценариев работы. Сценарием работы будем называть последовательность $T_1 \ldots T_n$ троек $T_i = \langle e_i, f_i, A_i \rangle$, где e_i – входное событие; f_i – булева формула от входных переменных, задающая охранное условие; A_i – последовательность выходных воздействий. В дальнейшем тройки T_i будем называть элементами сценария.

Будем говорить, что автомат, находясь в состоянии state, удовлетворяет элементу сценария T_i, если из state исходит переход, помеченный событием e_i, последовательностью выходных воздействий A_i и охранным условием, тождественно равным f_i как булева формула. Автомат удовлетворяет сценарию работы $T_1 \ldots T_n$, если он удовлетворяет каждому элементу данного сценария, находясь при этом в состояниях пути, образованного соответствующими переходами.

В работе [8] разработан метод построения автомата-распознавателя по заданному набору слов. Данный метод основан на сведении поставленной задачи к задаче о выполнимости булевой формулы

(boolean satisfiability problem, SAT) – по набору слов строится логическая формула. Выполняющая подстановка для нее ищется с помощью сторонней программы. Затем на основании полученных значений переменных логической формулы строится искомый автомат-распознаватель. Результаты вычислительных экспериментов показывают более высокую скорость работы этого метода по сравнению с методами, основанными на объединении состояний [9, 10].

В настоящей работе решается задача построения управляющего конечного автомата с заданным числом состояний C по заданному множеству сценариев работы Sc, которым автомат должен удовлетворять. Рассмотрим основные этапы работы предлагаемого алгоритма решения поставленной задачи.

Построение дерева сценариев

Деревом сценариев назовем дерево, каждый переход которого помечен событием, булевой формулой и последовательностью выходных воздействий. Опишем алгоритм построения дерева сценариев по заданному множеству сценариев Sc.

Изначально дерево сценариев состоит из единственной вершины – корня дерева. Затем по очереди добавим в дерево все сценарии работы из Sc. Для каждого из сценариев будем добавлять его элементы в дерево в порядке возрастания их номеров, начиная с первого. При этом будем хранить указатель на текущую вершину дерева v и номер i первого необработанного элемента сценария.

В начале процесса добавления v указывает на корень дерева сценариев, а $i = 1$. На каждом шаге проверяется существование исходящего из вершины v ребра, помеченного событием e_i и логической формулой, совпадающей с f_i как булевой функцией. Если такое ребро не существует, то создается новая вершина дерева u, и в нее направляется ребро, помеченное тройкой $<e_i, f_i, A_i>$. После этого u становится текущей вершиной, а значение i увеличивается на единицу.

Если такое ребро существует, то производится сравнение последовательности A_i и последовательности выходных воздействий A', которой помечено рассматриваемое ребро. Если $A_i = A'$, то текущей становится вершина, в которую ведет рассматриваемое ребро дерева, а значение i увеличивается на единицу. Если же указанные последовательности не совпадают, то заданное множество сценариев Sc является противоречивым, поэтому работа алгоритма прерывается, и пользователю выводится соответствующее сообщение.

После завершения добавления всех сценариев в дерево производится проверка охранных условий. Для каждой вершины перебираются все пары исходящих из нее ребер. Если существует такая пара ребер, что они помечены одним и тем же событием, а их охранные условия имеют общий выполняющий набор значений входных переменных, то множество сценариев предполагает недетерминированное поведение. Исходя из этого, работа алгоритма прерывается, и пользователю выводится соответствующее сообщение.

На рис. 2 приведен пример дерева сценариев. Сценариям данного дерева удовлетворяет автомат, приведенный на рис. 1.

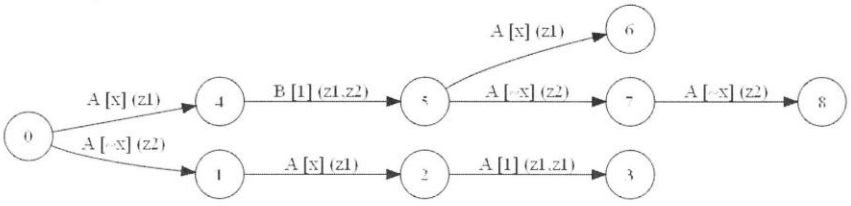

Рис. 2. Пример дерева сценариев

Построение графа совместимости вершин дерева сценариев

Для построения управляющего автомата необходимо «раскрасить» вершины дерева сценариев в заданное число цветов (равное числу состояний, которое задается в качестве одного из параметров алгоритма). При этом вершины одного цвета будут объединены в одно состояние результирующего автомата, а множество исходящих из состояния переходов будет строиться из объединения множеств ребер исходящих из вершин заданного цвета.

Для задания ограничений на раскраску построим так называемый граф совместимости вершин дерева сценариев. Множество вершин этого графа совпадает с множеством вершин дерева сценариев, поэтому в дальнейшем вершины графа и дерева различаться не будут. Ребра графа определяются следующим образом. Вершины графа совместимости u и v соединены ребром (далее такие вершины будем называть несовместимыми), если существует последовательность пар $<e_1, values_1>\ldots<e_k, values_k>$ событий и наборов значений входных переменных, которая различает соответствующие вершины дерева. Будем говорить, что указанная последовательность различает вершины u и v, если выполняется совокупность следующих условий:

- из вершины u существует путь P_u, ребра которого помечены соответственно событиями $e_1 \ldots e_k$ и такими охранными условиями $f_1 \ldots f_k$, что наборы значений входных переменных $values_1 \ldots values_k$ являются соответственно их выполняющими подстановками;
- аналогичный путь P_v существует из вершины v;
- для последних ребер путей P_u и P_v верно хотя бы одно из двух условий:
  - пометки этих ребер различаются в части выходных воздействий;
  - у охранных условий этих ребер есть общий выполняющий набор значений входных переменных, но они не совпадают как булевы функции.

Опишем алгоритм построения графа совместимости. Напомним, что множество вершин этого графа совпадает с множеством вершин дерева сценариев. Основной идеей данного алгоритма является метод динамического программирования [11].

Для каждой вершины дерева сценариев v найдем все несовместимые с ней вершины. Обозначим как $S(v)$ множество вершин, несовместимых с v. Будем вычислять значения функции $S(v)$, начиная с листьев дерева сценариев. Для каждого из листьев u множество $S(u)$ пусто по определению несовместимых вершин.

Покажем, как вычислить значение $S(v)$, если оно уже вычислено для всех значений «детей» вершины v. Переберем все вершины дерева – вершина u входит в множество $S(v)$, если существует пара ребер ux (помечено событием e, формулой f_1 и последовательностью действий A_1) и vy (помечено также событием e, формулой f_2 и последовательностью действий A_2) такая, что выполняется одно из трех:

- формулы f_1 и f_2 имеют общий выполняющий набор значений входных переменных, но не совпадают как булевы функции. Тогда $<e, values>$, где как $values$ обозначена выполняющая подстановка f_1, – последовательность, различающая u и v;
- формулы f_1 и f_2 совпадают как булевы функции, а последовательности A_1 и A_2 не совпадают. Тогда вершины u и v различает такая же последовательность;
- формулы f_1 и f_2 совпадают как булевы функции, и вершина x входит во множество $S(y)$, посчитанное заранее. Тогда существует последовательность $<e_1, values_1> \ldots <e_k, values_k>$, различающая вершины x и y, а вершины v и u различает последовательность $<e, values> <e_1, values_1> \ldots <e_k, values_k>$.

Время работы этого алгоритма составляет $O(n^2)$ (где за n обозначено число вершин в дереве сценариев), так как каждая пара ребер дерева сценариев в процессе работы алгоритма будет рассмотрена не более одного раза. Такое время работы достижимо, если заранее для каждой пары формул вычислено, равны ли они как булевы функции и имеют ли общий выполняющий набор значений входных переменных. В худшем случае время работы этапа обработки формул составляет $O(2^{2m}n^2)$, где за m обозначено максимальное число входных переменных, использующихся в одном охранном условии. На практике число m не превышает четырех.

Построение булевой КНФ-формулы, задающей требования к раскраске построенного графа

Опишем алгоритм построения конъюнктивной нормальной формы (КНФ) булевой формулы, задающей требования к раскраске построенного графа и выражающей непротиворечивость системы переходов результирующего автомата. Данное построение аналогично построению КНФ-формулы, используемой в работе [8] для построения автомата-распознавателя. Напомним, что в настоящей работе на вход алгоритму подается число C состояний результирующего автомата.

В данной формуле будут использоваться следующие логические переменные:

- $x_{v,i}$ (для каждой вершины дерева сценариев v и цвета вершины i – числа от 1 до C) – верно ли, что вершина v имеет цвет i. Напомним, что вершины одного цвета будут объединены в одно состояние результирующего автомата;
- $y_{a,b,e,f}$ (для каждой пары состояний результирующего автомата a и b, каждого события e, каждой формулы f, встречающейся в сценариях) – верно ли, что в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f.

КНФ-формула состоит из следующих дизъюнктов:

- $(x_{v,1} \vee \ldots \vee x_{v,C})$ (для каждой вершины v дерева сценариев) – накладываем условие существования цвета вершины v;
- $(\sim x_{v,i} \vee \sim x_{v,j})$ (для каждой вершины v дерева сценариев, цвета i и цвета j, где $i < j$) – накладываем условие, что вершина v не покрашена одновременно в цвета i и j;
- $(\sim x_{v,i} \vee \sim x_{u,i})$ (для каждой пары несовместимых вершин u и v дерева сценариев и цвета i) – накладываем ограничения на раскраску, задаваемые графом совместимости;
- $(\sim y_{a,b,e,f} \vee \sim y_{a,d,e,f})$ (для каждой тройки состояний результирующего автомата a, b и d ($b < d$), каждого события e, каждой формулы f) – из состояния a выходит не более одного ребра, помеченного событием e и формулой f;
- $(y_{a,b,e,f} \vee \sim x_{v,a} \vee \sim x_{u,b})$ (для каждого ребра vu дерева сценариев) – если выполняется:
  - ребро из вершины дерева v в вершину u помечено событием e и формулой f;

- вершина v покрашена в цвет a;
- вершина u покрашена в цвет b;

то в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f;

- $(\sim y_{a,b,e,f} \vee \sim x_{v,a} \vee x_{u,b})$ (для каждого ребра vu дерева сценариев) – если выполняется:
  - ребро из вершины дерева v в вершину дерева u помечено событием e и формулой f;
  - вершина v покрашена в цвет a;
  - в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f;

то вершина u покрашена в цвет b.

Запуск сторонней программы, решающей задачу о выполнимости булевой КНФ-формулы

Чтобы найти выполняющий набор для построенной КНФ-формулы, воспользуемся сторонней программой (SAT-solver), решающей задачу SAT. В результате анализа результатов соревнования SAT-Race 2010 [12] была выбрана программа cryptominisat [13]. Эта программа признана победителем в указанном соревновании. Подадим на вход выбранной программе построенную КНФ-формулу, записанную в формате DIAMAX (http://www.satlib.org/ubcsat/satformat.pdf).

Если программа не обнаружила выполняющий набор значений переменных, то будем считать, что по данному набору сценариев невозможно построить управляющий автомат с заданным числом состояний. Если же программа обнаружила выполняющий набор, то построим искомый автомат. Для этого на основании полученных значений переменных $x_{v,i}$ определим цвет каждой вершины дерева сценариев. На рис. 3 приведен пример раскраски дерева сценариев, приведенного на рис. 2.

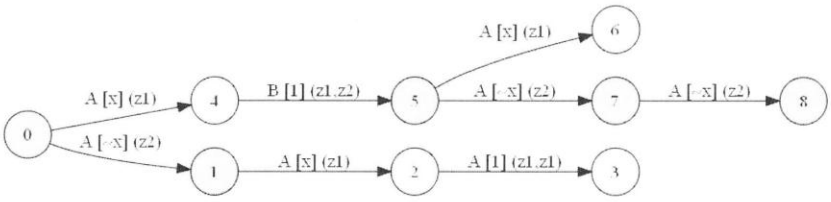

Рис. 3. Пример раскраски дерева сценариев в два цвета

После этого объединим все вершины одного цвета в одно состояние автомата, а начальным состоянием положим состояние, соответствующее цвету корня дерева сценариев. Множество исходящих из состояния переходов построим из объединения множеств ребер, исходящих из вершин заданного цвета.

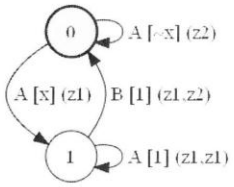

Рис. 4. Автомат, полученный после объединения вершин дерева

Например, после объединения вершин дерева, приведенного на рис. 3, получим автомат, изображенный на рис. 4. Заметим, что данный автомат изоморфен автомату, приведенному на рис. 1. Отметим, что не исключено существование нескольких автоматов, удовлетворяющих сценариям множества Sc.

Экспериментальное исследование

Экспериментальное исследование проводилось на задаче построения автомата управления часами с будильником [1]. Было задано 38 сценариев, аналогичных тестам, приведенным в работе [7]. На основе этих сценариев был построен автомат, изоморфный автомату, построенному вручную в работе [1]. Его построение заняло менее секунды на персональном компьютере с процессором Intel Core 2 Quad Q9400, что позволяет говорить о достаточно высокой производительности разработанного метода.

Заключение

В работе предложен метод построения управляющих конечных автоматов по сценариям работы. Этот метод основан на сведении указанной задачи к задаче о выполнимости булевой формулы. Работоспособность метода проверена на задаче построения автомата управления часами с будильником.

На этой задаче соответствующий управляющий автомат был построен корректно, а время работы алгоритма составляло меньше секунды на персональном компьютере с процессором Intel Core 2 Quad Q9400.

Исследование выполнено в рамках ФЦП «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы».

Литература

1. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2011. – 176 с.
2. Angeline P.J., Pollack J. Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. – 1993. – P. 154–163.
3. Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A. The Genesys System. – 1992 [Электронный ресурс]. – Режим доступа: http://www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html, св. Яз. англ. (дата обращения 25.11.2011).
4. Chambers L. Practical Handbook of Genetic Algorithms. Complex Coding Systems. – Boca Raton: CRC Press, 1999. – V. III – 592 p.
5. Царев Ф.Н., Шалыто А.А. Гибридное управление беспилотными летательными объектами на основе автоматного программирования // 1-я Российская мультиконференция по проблемам управления. Сборник докладов четвертой научной конференции «Управление и информационные технологии». – СПб ГЭТИ «ЛЭТИ», 2006. – С. 138–144.
6. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы. – М.: Физматлит, 2006. – 368 с.
7. Царев Ф.Н. Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. – 2010. – № 5. – С. 31–36.
8. Heule M., Verwer S. Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications. 10th International Colloquium. ICGI 2010. – 2010. – P. 66–79.
9. Oncina J., Garcia P. Inferring regular languages in polynomial update time / Pattern Recognition and Image Analysis. Series in Machine Perception and Artificial Intelligence. – Singapore: World Scientific, 1992. – V. 1. – P. 49–61.
10. Oliveira A.L., Marques-Silva J.P. Efficient search techniques for the inference of minimum sized finite state machines // Proseeding of 5th Symposium on String Processing and Informarion Retrieval. – 1998. – P. 81–89.
11. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. – 2-е изд. – М.: МЦНМО, 2010. – 1296 с.
12. Presentation of SAT-Race results at the SAT'10 conference [Электронный ресурс]. – Режим доступа: http://baldur.iti.uka.de/sat-race-2010/downloads/SAT-Race-2010-Presentation.pdf, свободный. Яз. англ. (дата обращения: 25.11.2011).
13. Soos M. CryptoMiniSat [Электронный ресурс]. – Режим доступа: http://www.msoos.org/cryptominisat2, св. Яз. англ. (дата обращения: 25.11.2011).

Ульянцев Владимир Игоревич – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, студент, ulyantsev@rain.ifmo.ru

Царев Федор Николаевич – Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, аспирант, fedor.tsarev@gmail.com

УДК 004.4'242

ПРИМЕНЕНИЕ АДАПТИВНОГО ГЕНЕТИЧЕСКОГО АЛГОРИТМА ДЛЯ ГЕНЕРАЦИИ КЛЕТОЧНЫХ АВТОМАТОВ

А.В. Тихомиров, А.А. Шалыто

Рассматривается улучшенный алгоритм генерации произвольных клеточных автоматов на основе тестовых наборов при помощи адаптивного генетического алгоритма. Описаны основные отличия и преимущества по сравнению со стандартным генетическим алгоритмом. Произведена апробация на нескольких обучающих примерах.

Ключевые слова: клеточные автоматы, генетические алгоритмы.

Введение

Настоящая работа развивает идеи, описанные в [1]. Для генерации различных клеточных автоматов используется адаптивный генетический алгоритм (ГА), который устраняет многие проблемы стандартного ГА. Алгоритм используется для моделирования многих физических процессов, например: диффузия энергии, разнообразные химические реакции, рост кристаллов и т.д. [2–5].

ПРИЛОЖЕНИЕ Е. КОПИЯ ЗАЯВКИ НА РЕГИСТРАЦИЮ ПРОГРАММЫ ДЛЯ ЭВМ

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (НИУ ИТМО)

Кронверкский пр., д. 49,
г. Санкт-Петербург, 197101
Тел. (812) 232-97-04. Факс (812) 232-23-07
e-mail: od@mail.ifmo.ru
http://www.ifmo.ru

24.05.2012 № 83-05-2/100

В отдел регистрации программ для ЭВМ,
баз данных, топологий ИМС и передачи прав на них
Федерального государственного учреждения «Федеральный институт промышленной собственности Федеральной службы по интеллектуальной собственности, патентам и товарным знакам (ФГУ ФИПС)
Бережковская наб., 30, корп. 1, Москва,
Г-59, ГСП-5, 123995

Направляю Вам на регистрацию программу для ЭВМ **Программное средство для построения графа совместимости вершин дерева сценариев работы программы** правообладателем исключительного права на которую является федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Комплектность заявки указана в приложении.

Приложение:

1.	Заявление (форма РП)	на	1	л.	в	1	экз.
2.	Дополнение к заявлению (форма РП/Доп)	на	1	л.	в	1	экз.
3.	Распечатка исходного текста программы	на	14	л.	в	1	экз.
4.	Реферат	на	1	л.	в	2	экз.
5.	Платежный документ об уплате госпошлины	на	1	л.	в	1	экз.

Всего на 18 листах + 1 платежный документ
Свидетельство прошу выслать по почте

Проректор Никифоров В.О.

«24» мая 2012 г.